Kernel-Mapping Recommender System Algorithms

Mustansar Ali Ghazanfar[a], Adam Prügel-Bennett[a], Sandor Szedmak[b]

[a]*School of Electronics and Computer Science, University of Southampton, Highfield Campus, SO17 1BJ, United Kingdom. Email: mag208r@ecs.soton.ac.uk; Phone: +44 (023) 80594473; fax: +44 (023) 80594498*

[b]*Intelligent and Interactive Systems, University of Innsbruck, 6020 Innsbruck, Austria, Email: sandor.szedmak@uibk.ac.at*

Abstract

Recommender systems apply machine learning techniques for filtering unseen information and can predict whether a user would like a given item. In this paper, we propose a new algorithm that we call the Kernel-Mapping Recommender (KMR), which uses a novel structure learning technique. This paper makes the following contributions: we show how (1) user-based and item-based versions of the KMR algorithm can be built; (2) user-based and item-based versions can be combined; (3) more information—features, genre, etc.—can be employed using kernels and how this affects the final results; and (4) to make reliable recommendations under sparse, cold-start, and long tail scenarios. By extensive experimental results on five different datasets, we show that the proposed algorithms outperform or give comparable results to other state-of-the-art algorithms.

Keywords: Recommender Systems, Structure Learning, Linear Operation, Maximum Margin, Kernel.

1. Introduction

In this paper, we proposed a new class of kernel-based methods for solving the recommendation problem that gives state-of-the-art performance. The main idea is to find a multi-linear mapping between two vector spaces. The first vector space might, for example, have vectors encoding information about the items that we wish to rate, while the second vector space may contain a probability density function describing how a particular user will rate an item. Learning an appropriate mapping can be expressed as a quadratic optimisation problem. As the problem involves a linear mapping, the solution to the optimisation problem involves inner products in the two vector spaces. This allows us to use the kernel trick. Directly solving the optimisation problem using quadratic programming would be too slow for most recommendation datasets. Instead, we find an approximate solution iteratively, following an idea first developed by [19]. This allows us to train the recommender in linear time. The method described here is a specialisation of a general structure learning framework developed by Szedmak and used in [53] for handling incomplete data sources.

The approach we have adopted is easily adapted to different sources of information. We can, for example, use either rating information from other users or textual information about the items. Similarly, we are able to build either an item-based or a user-based version of the algorithm. Because we have chosen to build a mapping to a space of functions approximating the probability density of the ratings, we have an intuitive interpretation of the recommendations produced by the algorithm. This gives us flexibility in how we make our final recommendation, which we can exploit to improve the final prediction for different datasets.

A main requirement of recommender systems is to provide high quality predictions of the rating that a user would give to an item, based on their previous rating history. Thus in testing recommender systems, a dataset is used where some sets of ratings are treated as unseen while the other ratings are used for learning. To obtain accurate results, datasets are usually selected with users that have made a relatively high number of ratings. In real applications, however, the datasets are often highly skewed; for example, a large number of users may have made only a small number of ratings, and a large number of items may have received very few ratings. These are important scenarios in practical systems as giving reasonable recommendations to new users can be crucial in attracting more users. Similarly, giving a sensible rating to a new item may be necessary for those items to be taken up by the community sufficiently to collect more ratings. Often, recommender system algorithms that have been optimised to give good recommendations on dense datasets perform poorly on these skewed datasets. We have generated highly skewed datasets to test our algorithm under these scenarios. In particular we consider the *new-item cold-start problem*, the *new user cold-start problem* [20, 1] and the *long tail problem* [36]. We find that the standard algorithm we developed performs poorly for these skewed datasets; however, we show that by using the flexibility of our approach we can easily modify the algorithm so that it performs well under these scenarios.

Recommender systems have been a very active topic of research for around twenty years. This, in part, has been spurred on by the Netflix competition to improve the performance (in terms of the root mean square error) of a baseline algorithm by 10%. One lesson to emerge from this was that a highly effective way to achieve a very high recommendation performance on a static dataset is to combine a large number of different algorithms. Although such systems are interesting, they are not very flexible and may not be ideal algorithms for most real ap-

plications with rapidly changing users and items. Our algorithm relies on a single coherent method (albeit with several variants) that has not been designed for a specific dataset. We have thus compared our approach with other general purpose recommenders. The best general purpose collaborative filtering algorithms that we are aware of are by [26] and [27]. These achieve a considerable gap in performance advantage over other algorithms. The proposed algorithm achieves similar performance in terms of mean absolute error to these approaches, although it is out-performed by [26] on a dataset with 1 000 000 ratings and by [27] on a dataset of 10 000 000 ratings. The proposed approach is however very different. The other two approaches are based on matrix factorisation, although [26] also uses kernel functions. There has been considerable work on developing matrix factorisation techniques which are at the heart of many of the most competitive algorithms for this problem. Part of the interest of the proposed algorithm is that it takes a very different viewpoint from the matrix factorisation approaches, yet still has very competitive performance.

The rest of the paper has been organised as follows. In the next section we briefly outline related work. Section 3 outlines the proposed algorithm using an item-based approach. In Section 4 we describe extensions to the basic algorithm. Section 5 presents details of the datasets we use for evaluation, the metrics we use and the procedure for tuning parameters of the algorithm. This is followed in Section 6 by a presentation of results from our experimental evaluation. We conclude in Section 7. Some of the details and more extensive results are given in appendices.

2. Related work

There are two main types of recommender systems: collaborative filtering and content-based filtering recommender systems. Collaborative filtering (CF) recommender systems [17, 50, 56, 42, 21, 39, 14] recommend items by taking into account the taste (in terms of preferences of items) of users, under the assumption that users will be interested in items that users similar to them have rated highly. Examples of these systems include GroupLens system [21], Ringo (www.ringo.com), etc. Collaborative filtering systems are classified into two subcategories: memory-based CF and model-based CF. Memory-based approaches [50] make a prediction by taking into account the entire collection of previous items rated by a user, for example, the GroupLens recommender systems [21]. Model-based approaches use rating patterns of users in the training set, group users into different classes, and use ratings of predefined classes to generate recommendations for an *active user* (i.e. the user for whom the recommendations are computed) on a *target item* (i.e. the item a system wants to recommend). Examples include item-based CF [44], Singular Value Decomposition (SVD) based models [46, 59, 24], matrix factorisation [51, 62, 4, 60, 54, 22, 43, 55, 23], nuclear norm regularisation [30], Bayesian networks [7], and clustering methods [36, 47, 61]. Content-based filtering recommender systems [25, 34, 57, 38] recommend items based on the content information of an item, under the assumption that users will like similar items to the ones they liked before. In these systems, an item of interest is defined by its associated features; for instance, NewsWeeder [25], a newsgroup filtering system uses the words of text as features. Other well-known types of recommender systems include knowledge-based systems [8, 6], Ontology-based systems [33], and hybrid systems [9, 40].

Hybrid recommender systems have been proposed elsewhere [31, 9, 37, 10, 8, 12, 15, 13, 40], which combine individual recommender systems to avoid certain limitations of individual recommender systems. In the proposed approach we can add more information (about items) in the forms of additional kernels, which can be thought of as combining collaborative filtering with content-based filtering. A related approach has been proposed in [3], where the authors employed a unified approach for integrating the user-item ratings information with user/item attributes using kernels. They learned a prediction function using an on-line perceptron learning algorithm. They claimed that adding more kernels increases the performance, which is in contrast with our findings[1].

In [53], the authors proposed a structured learning algorithm for learning from incomplete dataset. The idea of the structure learning has been used in [2], where the authors employed it for enzyme prediction. We show how the structure learning approach can also be used to solve the recommender system problem effectively.

Recommendations can be presented to an active user in the following two different ways [18]: by predicting ratings of items a user has not seen before and by constructing a list of items ordered by their preferences. In this paper, we focus on both of them.

3. Kernel-Mapping Recommender

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 rated some, but not all, of the items. We denote these ratings by $(r_{iu}|(i,u) \in \mathcal{D})$, where $\mathcal{D} \subset \mathcal{I} \times \mathcal{U}$ is 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 $|\mathcal{D}| = T \ll |\mathcal{I} \times \mathcal{U}| = N \times M$. It is not unusual in practical systems to have $T/(N \times M) \approxeq 0.01$. The set of possible ratings made by the users can be thought of as elements of an $M \times N$ rating matrix R. 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 recommendation algorithm that predicts an unseen rating r_{iu}, i.e. for $(i,u) \notin \mathcal{D}$.

In this section we describe an item-based recommender. In the next section we show how we can adapt the approach to a user-based recommender. To perform the recommendation task we consider building the additive and multiplicative models for the *residual ratings*. The residue in the additive model is given by

$$\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r}, \tag{1}$$

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

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

The multiplicative model can be expressed as follows:

$$\hat{r}_{iu} = \frac{r_{iu}\tilde{r}}{\tilde{r}_i\tilde{r}_u}, \tag{2}$$

where $\tilde{r}$, $\tilde{r}_i$ and $\tilde{r}_u$ are the geometric means for all the ratings, the ratings for item i, and the rating of user u, respectively. We found the additive model to be (marginally) better than the multiplicative one, and hence this work is based on the additive model.

3.1. Item-based KMR

We use a technique developed by Szedmak and co-workers for learning structured data [53]. In the following we outline how this approach is adapted for solving the collaborative filtering problem. We assume that we have some information about the items which 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 'vectors' in some other Hilbert space. In this paper, we consider these objects to lie in the function space $L_2(\mathbb{R})$. In particular we represent each residual $\hat{r}_{iu}$, by a normal distribution with mean $\hat{r}_{iu}$ and variance σ^2. That is,

$$\boldsymbol{\psi}(\hat{r}_{iu}) = \mathcal{N}(x|\hat{r}_{iu}, \sigma). \tag{3}$$

The motivation of this choice is to model possible errors in the rating either due to the discretisation of the rating scale or the variability in assigning a rating (e.g. due to the mood of the user on the day they made the rating).

The method developed by Szedmak is to seek a linear mapping between these two spaces which can be used for making predictions. More specifically, in our application, we look for a linear mapping $\mathbf{W}_u$ from the space of $\boldsymbol{\phi}$ vectors to the space of $\boldsymbol{\psi}$ vectors. We will use the mapping $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$ to make a prediction for the rating of a new item j by the user u. To learn the mappings $\mathbf{W}_u$ we will minimise the Frobenius norm of $\mathbf{W}_u$ subject to the constraints

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

where $\zeta_i \geq 0$ is a slack variable and where we have a constraint for each pair $(i,u) \in \mathcal{D}$. This ensures that $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$ is aligned with $\boldsymbol{\psi}(\hat{r}_{iu})$ for the ratings in the training set. We can write the training problem for the mappings $\mathbf{W}_u$ as a quadratic programming problem

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}\sum_{u\in\mathcal{U}}\|\mathbf{W}_u\|^2 + C\sum_{i\in\mathcal{I}}\zeta_i \\ \text{with respect to} & \mathbf{W}_u,\ u\in\mathcal{U},\ \zeta_i,\ i\in\mathcal{I} \\ \text{subject to} & \langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1-\zeta_i \\ & \zeta_i \geq 0,\ i\in\mathcal{I},\ u\in\mathcal{D}_i. \end{array} \tag{4}$$

Note that minimisation will be achieved when the vectors $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ are as uniformly aligned as possible with the vector $\boldsymbol{\psi}(\hat{r}_{iu})$. Having learned the mappings $\mathbf{W}_u$ we can then make predictions for a new item j using $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$. This outputs a function which informally we can think of as an estimate for the probability density of the residue $\hat{r}_{ju}$. However, $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$ does not need to be, and typically is not, positive everywhere or normalised. Thus, it is not itself a probability density. We discuss later different methods for interpreting $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$.

To solve this constrained optimisation problem, we define the Lagrangian

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

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

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

subject to the constraints that $\alpha_{iu} \geq 0$ for all $(i,u) \in \mathcal{D}$ and $\lambda_i \geq 0$ for all $i \in \mathcal{I}$. For a general linear mapping, $\mathbf{W}_u$, we have that

$$\frac{\partial}{\partial\mathbf{W}_u}\langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle = \boldsymbol{\psi}(\hat{r}_{iu})\otimes\boldsymbol{\phi}(\boldsymbol{q}_i),$$

where $\otimes$ is the tensor-product of the two vectors. This is clearly the case when the Hilbert spaces are finite dimensions so that the mapping $\mathbf{W}_u$ can be represented by a matrix, but this can be extended for linear mappings between more general Hilbert spaces. Using this result we find

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}_u} = \mathbf{W}_u - \sum_{i\in\mathcal{D}_u}\alpha_{iu}\boldsymbol{\psi}(\hat{r}_{iu})\otimes\boldsymbol{\phi}(\boldsymbol{q}_i).$$

That is, the Lagrangian is minimised with respect to $\mathbf{W}_u$ when $\mathbf{W}_u = \sum_{i\in\mathcal{D}_i}\alpha_{iu}\boldsymbol{\psi}(\hat{r}_{iu})\otimes\boldsymbol{\phi}(\boldsymbol{q}_i)$. Taking derivatives with respect to ζ_i we find

$$\frac{\partial\mathcal{L}}{\partial\zeta_i} = C - \sum_{u\in\mathcal{D}_i}\alpha_{iu} - \lambda_i.$$

Setting these derivatives to 0 we find that the Lagrangian is maximised with respect to ζ_i when

$$\sum_{u\in\mathcal{D}_i}\alpha_{iu} = C - \lambda_i \leq C$$

where the inequality arises because $\lambda_i \geq 0$.

After substituting back the expressions containing only the Lagrange multipliers into the Lagrangian we obtain the dual problem of (4) which is a maximisation problem with respect

to the variables α_{iu}

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

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

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

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

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

then 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. With our choice of mapping the rating residual, $\hat{r}$, to $\psi(\hat{r}) = \mathcal{N}(x|\hat{r},\sigma)$, we note that

$$K_{\hat{r}}(\hat{r},\hat{r}') = \langle\boldsymbol{\psi}(\hat{r}),\boldsymbol{\psi}(\hat{r}')\rangle = \mathcal{N}(\hat{r}-\hat{r}'|\sqrt{2}\sigma),$$

which is inexpensive to compute. We could build more complex kernels for $K_{\hat{r}}(\hat{r},\hat{r}')$, by mapping $\psi(\hat{r})$ into another extended feature space, although we would then lose the interpretation of $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as an approximation to the density function for $\hat{r}_{iu}$.

3.1.1. Learning the Lagrange multipliers

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 $f(\boldsymbol{\alpha})$ about the current position $\boldsymbol{\alpha}^{(t)}$

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

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

$$\tau = \operatorname*{argmax}_{\tau\in[0,1]} f\left(\boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\alpha}-\boldsymbol{\alpha}^{(t)}\right)$$

Note that we can compute the unconstrained maximum for τ

$$\tau_{\max} = \frac{(\boldsymbol{b}-\mathbf{M}\boldsymbol{\alpha}^{(t)})^\mathsf{T}(\boldsymbol{\alpha}^*-\boldsymbol{\alpha}^{(t)})}{(\boldsymbol{\alpha}^*-\boldsymbol{\alpha}^{(t)})^\mathsf{T}\mathbf{M}(\boldsymbol{\alpha}^*-\boldsymbol{\alpha}^{(t)})}.$$

By truncating $\tau_{\max}$ if necessary to ensure that it lies in the interval $[0,1]$ we can ensure that $\boldsymbol{\alpha}^{(t+1)}$ is the maximum value of $\boldsymbol{\alpha}$ along the line segment from $\boldsymbol{\alpha}^{(t)}$ to $\boldsymbol{\alpha}^*$. Since this segment includes the current point, $\boldsymbol{\alpha}^{(t)}$, we are guaranteed that no step decreases the objective function.

We note that in the linear programming problem we have an objective function of the form

$$\hat{f}(\boldsymbol{\alpha}) = \sum_{i\in\mathcal{I}}\sum_{u\in\mathcal{D}_i}\alpha_{iu}g_{iu} + \text{const},$$
$$g_{iu} = \frac{\partial f(\boldsymbol{\alpha}^{(t)})}{\partial\alpha_{iu}} = -\sum_{i'\in\mathcal{D}_u}\alpha_{iu}^{(t)}\ K_{\hat{r}}(\hat{r}_{iu},\hat{r}_{i'u})\ K_{\boldsymbol{q}}(\boldsymbol{q}_i,\boldsymbol{q}_{i'}) + 1$$

which decouples for every set of Lagrange multipliers $\mathcal{A}_i = \{\alpha_{ui}|u\in\mathcal{D}_i\}$. The linear constraints $Z(\boldsymbol{\alpha})$ also decouple into a set of constraints for each set of Lagrange multipliers $\mathcal{A}_i$. Thus, the linear programming problem becomes a series of linear programming problems for each $i \in \mathcal{I}$

$$\begin{aligned}&\text{maximise} && \sum_{u\in\mathcal{D}_i}\alpha_{iu}g_{iu}\\ &\text{subject to} && \sum_{u\in\mathcal{D}_i}\alpha_{iu}\le C \quad\text{and}\quad \forall u\in\mathcal{D}_i\ \ \alpha_{iu}\ge 0.\end{aligned}$$

This linear programming problem is trivial to solve (see figure 1). If g_{iu} has positive components then a maximum will occur when we set $\alpha_{iu^*} = C$ where $g_{iu^*} \ge g_{iu}$ for all $u \in \mathcal{D}_i$ and $\alpha_{iu} = 0$ otherwise.

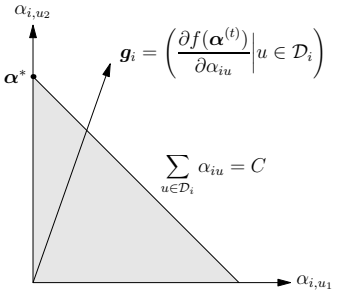

Figure 1: Schematic showing the linear programming problem. The feasible region is shown as a shaded triangle. The vector $\boldsymbol{g}_i$ shows the direction of the objective function. The maximum occurs as the vertex corresponding to the largest component of $\boldsymbol{g}_i$.

Since finding the largest component of g_{iu} can be computed in linear time we are able to perform one step of the optimisation procedure in $\Theta(|\mathcal{D}|)$ time. Note that at each step we have

to compute a vector matrix products involving the matrix $\mathbf{M}$. These products involve the sum over all $u \in \mathcal{U}$ and the sum over all $i \in \mathcal{D}_u$. However $|\mathcal{D}_u|$ does not grow with the number of users, thus these products can also be computed efficiently. Note, that solving the optimisation problem this way makes it feasible to obtain recommendation for databases with up to 10 million ratings.

3.1.2. Predicting unseen ratings

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

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

where $\psi(\hat{r}) = \mathcal{N}(\hat{r}, \sigma)$. We have a choice in how to obtain a single prediction from this function. Our *standard predictor* will be to find the maximum argument of $p_{iu}(\hat{r})$

$$\hat{r}_{iu} = \operatorname*{argmax}_{\hat{r}} p_{iu}(\hat{r}).$$

This works well when we have a sufficient number of ratings for the user and the item. However as we will see it gives poor predictions in scenarios where we have a small amount of training data. Recall that we argued earlier $\mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i)$ can be regarded as an approximation for the probability density of $\hat{r}_{iu}$. It will not generally be positive everywhere, but by removing the negative part of the function we can treat the remaining function as a probability density. In this case, we can consider the *mean*, *mode*, or *median* as approximations for the most likely value of $\hat{r}_{iu}$. Under conditions where we lack sufficient data we find that using a combination of the mean, mode and median together with the standard (max) prediction gives a considerable improvement in accuracy. In particular, we consider a predictor

$$\hat{r}_{M^4} = w_{max}\hat{r}_{max} + w_{mean}\hat{r}_{mean} + w_{mode}\hat{r}_{mode} + w_{median}\hat{r}_{median}$$

where $\hat{r}_m$ with $m \in \{max, mean, mode, median\}$ are the standard predictors and the predictors using the mean, mode and median, while w_m are a set of weights that are learnt from a validation set. We consider the weights to be constrained so that $w_m \geq 0$ and they sum to 1. In the results shown later we denote those results obtained using this predictor by the superscript M^4.

3.2. A small scale example

Suppose a recommender system has four users (i.e. $\mathcal{U} = \{u_1, u_2, u_3, u_4\}$) and three items (i.e. $\mathcal{I} = \{i_1, i_2, i_3\}$). The information about each item is a column vector of the user-item rating matrix, shown in Table 1. The users', items', and overall averages are:

$$\begin{aligned} &\bar{r}_{u_1} = 3.000, \quad \bar{r}_{u_2} = 3.666, \quad \bar{r}_{u_3} = 3.666, \quad \bar{r}_{u_4} = 2.000, \\ &\bar{r}_{i_1} = 4.000, \quad \bar{r}_{i_2} = 4.000, \quad \bar{r}_{i_3} = 1.333, \quad \bar{r} = 3.200, \end{aligned}$$

After applying the additive model (Equation 1), the user-item rating matrix can be represented in the residual form as shown in Table 2. The input feature kernel, $K_{\boldsymbol{q}}$, using the poly-Gaussian kernel (refer to Section 5.5.2) is shown in Equation 5.

Table 1: Example: a subset of the user-item rating matrix in a movie recommender system. We have four users (rows) and three movies (columns). The case, where a user has not rated a particular movie is shown by ⊘ symbol. The rating scale, consisting of integer values between 1 and 5, captures the extreme like (5) and extreme dislike (1) behavior of a user. The rating we want to predict is shown by "?" symbol.

	i_1	i_2	i_3
u_1	5	⊘	1
u_2	5	5	1
u_3	5	4	2
u_4	1	3	?

Table 2: Example: the matrices of rating residues $\hat{r}_{iu}$.

	i_1	i_2	i_3
u_1	1.200	⊘	−0.1333
u_2	0.533	0.533	−0.800
u_3	0.533	−0.466	0.200
u_4	−1.800	0.200	?

$$K_{\boldsymbol{q}} = \begin{bmatrix} 1.000 & 0.067 & 0.003 \\ 0.067 & 1.000 & 0.300 \\ 0.003 & 0.300 & 1.000 \end{bmatrix} \tag{5}$$

We can compute the residual kernel, $K_{\hat{r}}$, based on the inner products between Gaussian densities functions with expected values $\hat{r}$ and $\hat{r}'$, and sharing the common standard deviation σ.

$$K_{\hat{r}}(\hat{r}, \hat{r}') = \langle \boldsymbol{\psi}(\hat{r}), \boldsymbol{\psi}(\hat{r}') \rangle = \frac{1}{2\sigma\sqrt{\pi}} e^{-(\hat{r}-\hat{r}')^2/4\sigma^2}$$

Assume that $\sigma = 0.5$ then we have

$$K_{\text{residual}} = \begin{bmatrix} K_{\hat{r},u_1} & & & \\ & K_{\hat{r},u_2} & & \\ & & K_{\hat{r},u_3} & \\ & & & K_{\hat{r},u_4} \end{bmatrix},$$

where

$$K_{\hat{r},u_2} = \begin{bmatrix} 0.564 & 0.564 & 0.149 \\ 0.564 & 0.564 & 0.149 \\ 2.140 & 2.140 & 0.564 \end{bmatrix} \quad K_{\hat{r},u_1} = \begin{bmatrix} 0.564 & 0.149 \\ 2.140 & 0.564 \end{bmatrix}$$
$$K_{\hat{r},u_3} = \begin{bmatrix} 0.564 & 0.208 & 0.404 \\ 1.534 & 0.564 & 1.099 \\ 0.788 & 0.290 & 0.564 \end{bmatrix} \quad K_{\hat{r},u_4} = \begin{bmatrix} 0.564 & 0.149 \\ 2.140 & 0.564 \end{bmatrix}.$$

The optimal values for the design variables, $\boldsymbol{\alpha}$, are learnt using the conditional gradient method, and are shown in Table 3. After learning the $\boldsymbol{\alpha}$ parameters, the mapping $\mathbf{W}_u$, can be defined for each user (recall $\mathbf{W}_u = \sum_{i \in \mathcal{D}_i} \alpha_{iu} \boldsymbol{\psi}(\hat{r}_{iu}) \otimes \boldsymbol{\phi}(\boldsymbol{q}_i)$). To

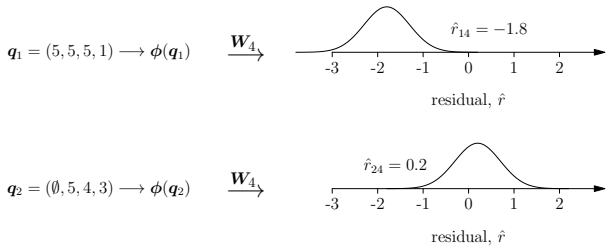

Figure 2: Schematic showing the aim of the algorithm. Information, $\boldsymbol{q}_i$ (in this case a rating vector) about an item i, is first mapped to a vector in an extended feature space $\boldsymbol{\phi}(\boldsymbol{q}_i)$. We then try to find the best linear mapping, $\mathbf{W}_4$, for user u_4, to the 'vector', $\psi(\hat{r}_{iu_4})$, describing the residual.

Table 3: The optimal values of design variable, α, for each user and item.

	i_1	i_2	i_3
u_1	1.000	⊘	1.000
u_2	0.993	0.993	0.999
u_3	0.993	0.768	0.769
u_4	1.000	1.000	?

make a prediction for the rating r_{iu}, where $(i, u) \notin \mathcal{D}$

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

In this case, we have $u = u_4$ and $i = i_3$, so

$$\begin{aligned}\mathbf{W}_{u_4}\phi(\boldsymbol{q}_{i_3}) &= \alpha_{i_1u_4}\boldsymbol{\psi}(\hat{r}_{i_1u_4}) K_{\boldsymbol{q}}(\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_3}) + \alpha_{i_2u_4}\boldsymbol{\psi}(\hat{r}_{i_2u_4}) K_{\boldsymbol{q}}(\boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3}) \\ &= 1.000\boldsymbol{\psi}(\hat{r}_{i_1u_4})0.003 + 1.000\boldsymbol{\psi}(\hat{r}_{i_2u_4})0.300, \\ &= 0.003\boldsymbol{\psi}(\hat{r}_{i_1u_4}) + 0.300\boldsymbol{\psi}(\hat{r}_{i_2u_4}), \\ &= 0.003\mathcal{N}(\hat{r}_{i_1u_4}, \sigma) + 0.300\mathcal{N}(\hat{r}_{i_2u_4}, \sigma),\end{aligned}$$

which is an unnormalised probability density function of mixture of two Gaussians. The optimal rating then can be derived by

$$\begin{aligned}p_{i_3u_4}(\hat{r}) &= \langle \psi(\hat{r}), \mathbf{W}_{u_4}\phi(\boldsymbol{q}_{i_3}) \rangle \\ &= \arg\max_{\hat{r}} \langle \psi(\hat{r}), 0.003\boldsymbol{\psi}(\hat{r}_{i_1u_4}) + 0.300\boldsymbol{\psi}(\hat{r}_{i_2u_4}) \rangle, \\ &= \arg\max_{\hat{r}} \langle 0.003K_{\hat{r}}(\hat{r}, \hat{r}_{i_1u_4}) + 0.300K_{\hat{r}}(\hat{r}, \hat{r}_{i_2u_4}) \rangle, \\ &= \arg\max_{\hat{r}} \langle 0.003\mathcal{N}(\hat{r}|\hat{r}_{i_1u_4}, \sqrt{2}\sigma) + 0.300\mathcal{N}(\hat{r}|\hat{r}_{i_2u_4}, \sqrt{2}\sigma) \rangle\end{aligned}$$

Taking the optimum solution (refer to Figure 3), $r_{i_3u_4}$, the prediction for the residual is 0.2. Hence, user u_4 would rate item i_3 with rating of $\hat{r}_{i_3u_4} + \hat{r}_i + \hat{r}_u - \hat{r} = 0.2 + 2.0 + 1.333 - 3.2 = 0.33$.

4. Extensions to the basic algorithm

In this section we describe extensions to the basic algorithm which are relevant to practical recommender systems.

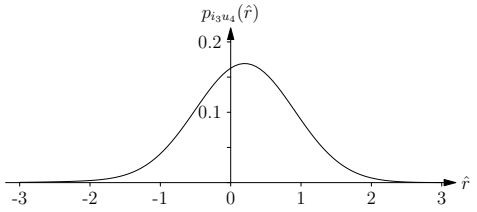

Figure 3: Plotting the probability density function of mixture of two Gaussians with $\hat{r}$={−5:0.2:5}. The optimal solution is found to be 0.2.

4.1. User-based KMR

Depending on the dataset characteristics (e.g. number of items rated by the active user, number of users which have rated the target item, etc.) different models can be trained along the rows or columns of the data matrix. A related algorithm is proposed, which solves the problem from the user point of view, hence it is named as the user-based KMR (KMR_{ub}). 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.

4.2. Combining user- and item-based KMR

User- and item-based versions provide complementary roles in generating predictions as they focus on different types of relationships in a dataset. Let $p^{ub}_{iu}(\hat{r})$ and $p^{ib}_{iu}(\hat{r})$ be the predictions made by the user- and item-based versions respectively. We have considered three different ways of combining user- and item-based predictions.

- *Using the simple linear combination:* In this approach, the user- and item-based versions are linearly combined, where the parameter ρ is learned from a validation set.

$$p_{iu}(\hat{r}) = \rho p^{ub}_{iu}(\hat{r}) + (1 - \rho)p^{ib}_{iu}(\hat{r}) \tag{6}$$

  We denote the resulting hybrid recommender system by KMR^{Linear}_{Hybrid}.

- *Switching on number of ratings:* Here, we take into account the information about user and item profiles. The rationale behind this approach is the intuition that if we have a large number of ratings for an item compared to the number of ratings made by the active user, then the user-based version is likely to give better results than the item-based version and vice-versa. Rather than using the raw number of ratings, we normalise by the number of ratings given by the power user, u_p (i.e. the user that has rated the most number of items) and by the power item i_P (i.e. the item with the most number of ratings). That is, we used

$$p_{iu}(\hat{r}) = \begin{cases} p^{ub}_{iu}(\hat{r}): & \text{if } \frac{|U_i|}{|U_{i_p}|} - \frac{|I_u|}{|I_{u_p}|} > \theta_{Cnt} \\ p^{ib}_{iu}(\hat{r}): & \text{otherwise.} \end{cases} \tag{7}$$

  We denote the resulting hybrid recommender system by KMR^{Cnt}_{Hybrid}.

- *Switching on uncertainty in prediction:* Here we use a different strategy for switching between the user- and item-based predictors. We try to estimate the uncertainty in the prediction by examining the "variance" in $\boldsymbol{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ and $\boldsymbol{W}_i\boldsymbol{\phi}(\boldsymbol{q}_u)$. Since they are not real probability distributions, we must first exclude the regions where the functions go negative and normalise the output so that we can treat them as densities and compute their variance. We denote the variance by Var_{ub} and Var_{ib} for the user- and item-based versions, respectively. We then switch the recommendation according to

$$p_{iu}(\hat{r}) = \begin{cases} p^{ub}_{iu}(\hat{r}): & \text{if } \text{Var}_{ub} - \text{Var}_{ib} > \theta_{Var} \\ p^{ib}_{iu}(\hat{r}): & \text{otherwise.} \end{cases} \tag{8}$$

  We denote the resulting hybrid recommender system by KMR^{Var}_{Hybrid}.

4.3. Combining kernels

In many applications there are multiple sources of information that can be used to make a recommendation. We can easily accommodate different sources of information by combining kernels. To illustrate this we will test our algorithm on datasets consisting of film ratings where we have three types of information available (refer to Section 5.2 for details)

- The ratings of other users from which we can construct a kernel K_{rat}
- "Demographic" information obtained from genre about the films from which we can construct a kernel K_{demo}
- "Feature" information obtained from a textual description of the films from which we construct a kernel K_{feat}.

These kernels can be combined *linearly*

$$K = \beta_{rat}K_{rat} + \beta_{demo}K_{demo} + \beta_{feat}K_{feat}, \tag{9}$$

where the parameters β_{rat}, β_{demo} and $\beta_{feat} = 1 - \beta_{rat} - \beta_{demo}$ can be tuned by measuring the generalisation performance on a validation set. This way of combining kernels can be viewed as a concatenation of the feature vectors

$$\begin{aligned} \boldsymbol{\phi} &= (\sqrt{\beta_{rat}}\boldsymbol{\phi}_{rat},\ \sqrt{\beta_{demo}}\boldsymbol{\phi}_{demo},\ \sqrt{\beta_{feat}}\boldsymbol{\phi}_{feat}) \\ &= \sqrt{\beta_{rat}}\boldsymbol{\phi}_{rat} \oplus \sqrt{\beta_{demo}}\boldsymbol{\phi}_{demo} \oplus \sqrt{\beta_{feat}}\boldsymbol{\phi}_{feat}, \end{aligned}$$

where $\oplus$ represents the direct sum. Alternatively we can combine the kernels *non-linearly*

$$K = K_{rat} \cdot K_{demo} \cdot K_{feat}, \tag{10}$$

where the $\cdot$ denotes the point-wise product of the kernel matrices. This corresponds to taking a tensor product of the feature vectors

$$\boldsymbol{\phi} = \boldsymbol{\phi}_{rat} \otimes \boldsymbol{\phi}_{demo} \otimes \boldsymbol{\phi}_{feat}. \tag{11}$$

5. Experimental setup

In this section we describe the datasets we used and the setup of the experiments for benchmarking the proposed algorithms.

5.1. Datasets

As is common in the field of recommender systems we used data from film recommendation sites to test the proposed algorithm. These provide some of the largest available datasets allowing us to test the scaling performance of the algorithm. In addition, as these datasets are very commonly used in the literature, it allows us to benchmark our algorithm against competitor algorithms. We used the following datasets:

- FilmTrust (denoted by FT) obtained by crawling (on 10th March 2009) the FilmTrust website (http://trust.mindswap.org/FilmTrust/). Only users and movies having more than five ratings were used. This has been used before in [12, 15].
- MovieLens which we split into three groups
  - Small MovieLens (denoted by SML) with 100 000
  - 1 million rating dataset (denoted by ML)
  - 10 million rating dataset (denoted by ML10)

  This has been widely used [44, 59, 15, 12, 26].
- Random sub-sample of 20 000 users from the Netflix dataset (denoted by NF). This dataset has been very widely used (e.g see [22, 4, 5]), in part because of the prize offered for achieving a level of improvement over a benchmark. We have not attempted to compare our algorithm against the state-of-the-art Netflix algorithms for two reasons. Firstly they have been highly tuned to that particular dataset, while we have concentrated on developing a general purpose recommendation algorithm. Secondly, the full Netflix dataset is so large that it is difficult to process on a normal desktop machine without spending significant time on optimising memory management.

5.2. Feature extraction and selection

To test the recommendation algorithm using textual information we also obtained information about each movie. This was used to construct two additional information vectors; a "feature" vector and a "demographic" vector. We downloaded information about each movie in the MovieLens (SML dataset) and FilmTrust dataset from IMDB[2]. For the ML10 dataset, we used the tags and genre information that is provided with this dataset. After stop word[3] removal and stemming[4], we constructed a vector of keywords, tags, directors, actors/actresses, producers, writers, and user reviews given to a movie in IMDB. We used *TF-IDF* (Term Frequency-Inverse Document Frequency) approach for determining the weights of words in a document (i.e. movie). The document frequency (DF) thresholding feature selection technique was used to reduce the feature space by eliminating useless noise words having little (or no) discriminating power in a classifier, or having low signal-to-noise ratio.

To construct the demographic vector, we take the genre information about movies as employed in [59, 15] with the exception that we used the hierarchy of genre as shown in Figure 4. To determine the weight of a genre in the genre vector,

Table 4: Characteristics of the datasets used in this work. FT, SML, ML, ML10, and NF represent the FilmTrust, MovieLens 100k, MovieLens 1M, MovieLens 10M, and Netflix dataset respectively. Average rating represents the average rating given by all users in the dataset.

Characteristics	**Dataset**				
	(FT)	**(SML)**	**(ML)**	**(ML10)**	**(NF)**
Number of users	10 16	943	6 040	71 567	20 000
Number of movies	314	1 682	3 706	10 681	17 766
Number of ratings	25 730	100 000	1 000 209	10 000 054	4 260 735
Rating scale	1.0-10.0	1 -5	1-5	1.0-5.0	1 -5
Sparsity	0.919	0.934	0.955	0.987	0.988
Max number of ratings from a user	133	737	2 314	7 359	17 653
Max number of ratings for a movie	842	583	3 428	34 864	9 667
Average rating	7.601	3.529	3.581	3.512	3.591

we used a simple weighting scheme as employed in QuickStep, an Ontology-based recommender system [32]. To compute an inner product between demographic vectors the immediate super class is assigned 50% of a subject's value, the next super class is assigned 25%, and so on until the most general subject in the Ontology is reached. By making a hierarchy of the genre and assigning different weights to sub- and super-classes, we hope to enrich an item's profile.

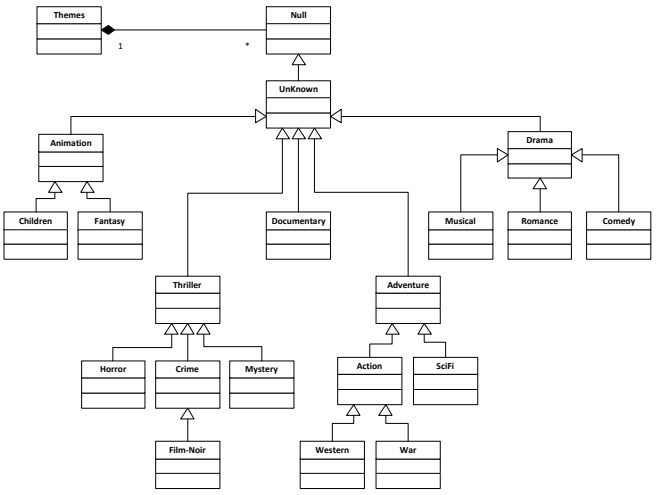

Figure 4: Hierarchy of genres based on [48]. All the super classes of a genre get a share when a genre receives some interest. For instance if a rated movie falls into "crime" genre, then the "crime" subject will get a weight of q, the immediate super class, "Thriller" will get a weight of $q/2$; and the next super class "Unknown" will get a weight of $q/4$.

5.3. Metrics

In the majority of the paper, we have used the *Mean Absolute Error (MAE)* as our measure of performance as this is the most commonly used measure and *de facto* standard for benchmarking recommender systems. In practice, however, recommender systems are commonly used for helping users in selecting high quality items. Thus, arguably, a more appropriate measure of accuracy is to study an algorithm's ability to predict highly rated items. There are a number of metrics that are more specifically designed to measure how well a recommender classifies good quality (relevant) items. These include the *ROC-sensitivity* and $F1$ *measure*. The details of all these metrics are given in Appendix B. Furthermore, we also give tables of results for these last two measures in that appendix.

5.4. Evaluation methodology

We performed 5-fold cross validation by randomly dividing the dataset into a test and training set and reported the average results. We further subdivided the training set into a test and training set for measuring the sensitivity of the parameters. For learning the parameters, we conducted 2-fold cross validation on the training set.

5.5. Learning system parameters

There are a number of parameters that need to be learned. Below, we discuss the training of these parameters.

5.5.1. Number of iterations

The algorithm we develop uses an iterative technique to learn the Lagrange multipliers, $\boldsymbol{\alpha}$. As we increase the number of iterations the mean absolute error improves. The speed of convergence will depend on the dataset and the type of information we are using (e.g. user-based or item-based). Figures 5 and 6, show the mean absolute error and the time taken to learn the Lagrange multipliers versus the number of iterations for the FT and SML datasets respectively.

We note that for the FT dataset, the performance of the item-based version suffers badly when the number of iterations are very small. However, the performance of the user-based version is quite good even after a few iterations. Hence, if one has a constraint on the time required to build the model, then it is better to switch to the user-based version rather than the item-based version for the dataset. In contrast, in the SML dataset, the convergence of all the methods was relatively quick. The convergence clearly depends on the number of users/items and the user/item profile length (e.g. rating profile, feature profile length, etc.). It is not obvious *a priori* how many iterations are needed to get good rating predictions. Based on our initial experiments, we chose the number of iterations to be 400 for the SML dataset, 300 for FT, 400 for ML, and 600 for ML10 and NF.

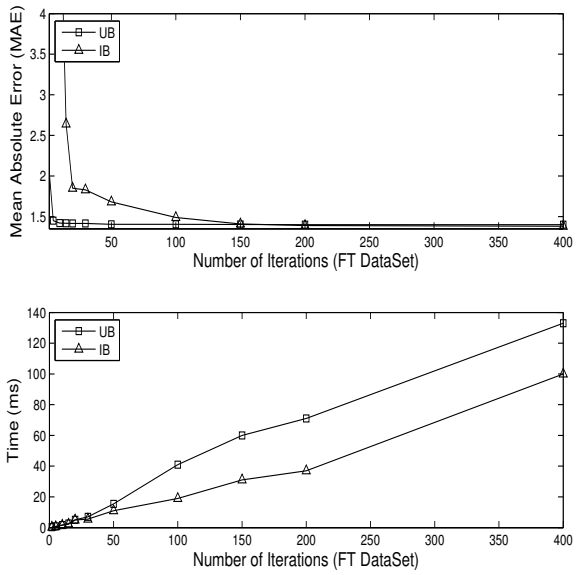

Figure 5: The number of iterations and time required to converge the proposed algorithms for the FT dataset.

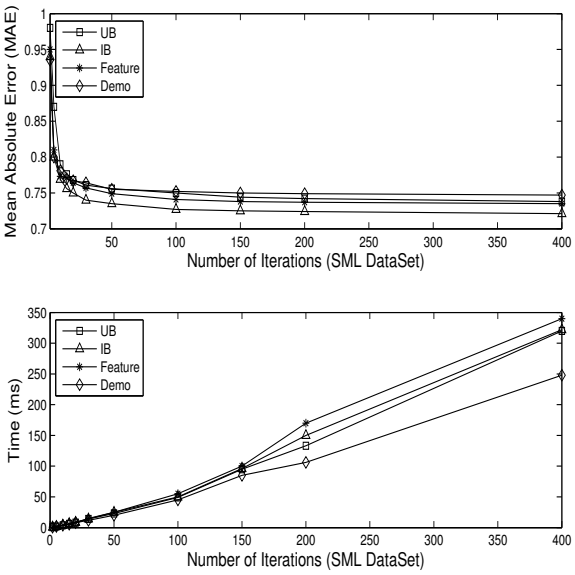

Figure 6: The number of iterations and time required to converge the proposed algorithms for the SML dataset.

5.5.2. The optimal kernel parameters

We trained linear, polynomial, and poly-Gaussian kernels and chose the one giving the most accurate results. The polynomial kernel is of the form

$$K(\boldsymbol{x}, \boldsymbol{y}) = \Big(\langle \boldsymbol{x}, \boldsymbol{y} \rangle + R\Big)^d$$

For the rating-based version, the best polynomial kernel parameters (d, R) are found to be, for user-based and item-based versions respectively: $(3, 0.5)$ and $(4, 0.5)$ for the SML dataset; $(6, 0.4)$ and $(6, 0.4)$ for the FT dataset; and $(6, 0.1)$ and $(9, 0.1)$ for the ML dataset. For the feature-based version, the best polynomial kernel parameters were found to be $(5, 0.5)$ for the SML dataset and $(5, 0.1)$ for the FT dataset.

We did not tune the parameters for the ML10 and NF datasets, as it was computationally expensive. We fixed them to $(14, 0.5)$ for user- and item-based versions for both datasets and $(12, 0.5)$ for the feature-based version for the ML10 dataset.

For the demographic-based version, we found the best kernel was the poly-Gaussian kernel (which is a simple extension of the Gaussian one) given by

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{y}||^q}{\tau}\right), \tag{12}$$

where the best parameters (q, τ) were found to be $(0.1, 0.1)$ for the SML dataset; and $(0.2, 0.1)$ for the FT dataset. Again we did not tune parameters for ML10 dataset and they were fixed to $(0.5, 0.1)$.

The other parameter in setting up the kernel was the standard deviation, σ, used in mapping $\psi(\hat{r}) = \mathcal{N}(x|\hat{r}, \sigma)$. We experimented with learning this parameter for each user, but found this computationally very expensive. We then tried grouping the users according to the variance in their ratings into 100 groups and tuned σ for each group. Although this gave improved performance, it was not found to be statistically significant. We therefore just used a single parameter σ which we tuned using a validation set.

The parameter C (that punishes the slack variables in the Lagrange formulation) was fixed to 20, after initial experimentation. In the extension of the basic recommender there are other parameters, such as the weights for combining kernels and various thresholds for switching between recommenders. The tuning of these parameters are described in Appendix A.

6. Results

In this section, we describe the results obtained from our experiments. In the tables we have denoted the proposed algorithm by KMR^{sup}_{sub}, where the subscript denotes the variant of the algorithm and the occasional superscript describes the variant in more detail where necessary. The main variants are item-based (*ib*), user-based (*ub*), feature-based (F) that use feature vectors rather than rating vectors, demographic (D) that use demographic vectors rather than rating vectors, and hybrid (*Hybrid*) that uses a mixture of user-based and item-based predictions. For the hybrid algorithm we use the superscript to denote the different mechanisms for combining user-based and item-based predictions. When we use combinations of information, e.g. item-based ratings and features, we use KMR_{ib+F} to denote the case when we add the kernels and $KMR_{ib\otimes F}$ when we multiply the kernels. Finally, for the datasets with a limited amount of ratings, instead of using the standard approach to predicting a new rating, we combined the standard approach (value of $\hat{r}$ that maximises the predictor $p(\hat{r})$) with the mean, mode and median of $\boldsymbol{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ (for the item-based approach). We denote this version of the algorithm with a superscript M^4.

We compare the proposed algorithms with other algorithms described in the literature. We chose several other algorithms

Table 6: A comparison of different algorithms in terms of NMAE (Normalised MAE) for the ML dataset. The proposed algorithms outperform URP [29], Attitude [28], MatchBox [52], MMMF [41], ImputedSVD [16], and Item [35]. They give the comparable results to E-MMF [11] and NLMF [26]. Our results and the best results are shown in bold font.

Algorithm	**NMAE**
URP	0.4341 ± 0.0023
Attitude	0.4320 ± 0.0055
MatchBox	0.4206 ± 0.0055
ImputedSVD	0.4192 ± 0.0025
MMMF	0.4156 ± 0.0037
Item	0.4096 ± 0.0029
E-MMF	**0.4029 ± 0.0027**
NLMF Linear	0.4052 ± 0.0011
NLMF RBF	**0.4026 ± 0.0020**
KMR_{ib}	0.4125 ± 0.0036
KMR_{ub}	0.425 ± 0.0038
KMR^{Var}_{Hybrid}	**0.4065 ± 0.0031**

Table 7: A comparison of different algorithms in terms of RMSE for the ML10 dataset. NLMF represents the non-linear matrix factorisation technique as proposed in [26] and M^3F-TIB represents the Mixed Membership Matrix factorisation model as proposed in [27]. Our results and the best results are shown in bold font.

Algorithm	**RMSE**
NLMF	0.8740 ± 0.02
M^3F-TIB	**0.8447 ± 0.009**
KMR_{ib}	0.8721 ± 0.011
KMR_{ub}	0.8994 ± 0.015
KMR^{Var}_{Hybrid}	**0.8612 ± 0.011**

based on the number of citations given in the literature; the algorithm classification space (i.e. memory-based or model-based approaches); and whether the algorithm claims to give state-of-the-art results.

6.1. Direct comparison

We compared the proposed algorithms with three different algorithms: user-based collaborative Filtering (CF) with Default Voting (DV) proposed in [7] (which provides a useful baseline for comparing algorithms), item-based CF proposed in [44] (shown by *Item-Based CF*), and a SVD based approach proposed in [46] (shown by *SVD*). To provide as fair a comparison as possible, we tuned all parameters of the algorithms.

Table 5 shows that the KMR-based algorithms outperform all the aforementioned algorithms. The percentage decrease in error of KMR_{ib}, KMR_{ub}, and KMR^{Var}_{Hybrid} over the baseline approach is found to be 2.68%, 1.48%, and 2.96% for the FT5 dataset; 4.16%, 2.01%, and 4.69% for the SML dataset; 6.09%, 2.83%, and 6.80% for the ML dataset; 5.90%, 4.28%, and 6.64% for the ML10 dataset; and 4.07%, 3.65%, and 4.35% for the NF dataset. The ROC-sensitivity and F1 measure on the same dataset are shown in Tables A.11 and A.12, respectively.

6.2. Indirect comparison

In this section, we compare our results with other algorithms indirectly, i.e. we take the result from the respective papers without re-implementing them, which might make the comparison less than ideal. We conducted the weak generalisation test procedures of [28] using the All-But-One protocol—for each user in the training set a single rating is withheld for the test set. We averaged the results over the three random train-test splits as used in [26, 28, 41].

A comparison in terms of Normalised MAE (NMAE)—see Appendix B—of the algorithms is given in Table 6. In Table 6, URP represents the algorithm proposed in [29], Attitude represents the algorithm proposed in [28], MatchBox[5] is proposed in [52], MMMF represents the maximum margin matrix factorisation algorithm proposed in [41], ImputedSVD is proposed in [16], Item is proposed in [35], E-MMF represents the ensemble maximum margin matrix factorisation technique proposed in [11], and NLMF represents the non-linear matrix factorisation technique (with linear and RBF versions) as proposed in [26].

Table 6 shows that the NLMF and E-MMF perform better than the rest. The proposed hybrid algorithm gives slightly poorer results to them with NMAE = 0.4065. The percentage increase in the NMAE was 0.96 and 0.89 for NLMF and E-MMF respectively. It is worth mentioning that the E-MMF is an ensemble of about 100 predictors, which makes this algorithm unattractive. From this table, we may conclude that the proposed algorithm is comparable to the state-of-the-art algorithm for the MovieLens ($1M$) dataset.

To the best of our knowledge, the best results for the MovieLens $10M$ dataset that have been reported in the literature are those proposed in [26] and [27]. They claimed their proposed algorithm gives RMSE accuracy of 0.8740 ± 0.02 and 0.8447 ± 0.009, respectively. We followed their experimental setup and the results have been shown in Table 7. Table 7 shows that the proposed algorithms outperform [26]'s results. The percentage improvement is found to be 1.46% in the case of KMR^{Var}_{Hybrib}. The M^3F-TIB algorithm gave the best results outperforming our best algorithm KMR^{Var}_{Hybrid} with 1.92% decrease in error. Actually the M^3F-TIB integrates two complementary algorithms—discrete mixed membership modeling and continuous latent factor modeling (i.e. matrix factorisation)—into a common framework using the Bayesian approach, which illustrates the power of carefully combining different algorithms.

Unfortunately, no NMAE (or MAE) was provided for M^3F-TIB technique [27] over Movielens 1M dataset, which makes it harder to compare different algorithms' results with M^3F-TIB. Considering these results, we conclude that the proposed approach appears to be competitive with the current state-of-the-art approaches.

6.3. Combining different kernels

As discussed in Section 4.3, there can be different sources of information that can be used for making recommendations. The proposed framework allows these different sources to be exploited by combining different kernels built from different

Table 5: A comparison of the proposed algorithm with others in terms of MAE. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Algorithm	**Best MAE**				
	SML	**ML**	**ML**10	**FT**	**NF**
User-based CF	0.746 ± 0.001	0.706 ± 0.000	0.678 ± 0.000	1.419 ± 0.008	0.713 ± 0.001
Item-based CF	0.764 ± 0.001	0.715 ± 0.000	0.675 ± 0.001	1.429 ± 0.006	0.719 ± 0.001
Hybrid CF	0.752 ± 0.001	0.702 ± 0.001	0.667 ± 0.000	1.427 ± 0.002	0.717 ± 0.001
SVD	0.774 ± 0.001	0.730 ± 0.001	0.691 ± 0.001	1.483 ± 0.005	0.725 ± 0.001
KMR_{ib}	0.715 ± 0.001	0.663 ± 0.001	0.638 ± 0.001	1.381 ± 0.002	0.684 ± 0.001
KMR_{ub}	0.731 ± 0.001	0.686 ± 0.001	0.649 ± 0.001	1.398 ± 0.002	0.687 ± 0.001
KMR^{Var}_{Hybrid}	**0.711 ± 0.001**	**0.658 ± 0.001**	**0.633 ± 0.001**	**1.377 ± 0.002**	**0.682 ± 0.001**

Table 8: Comparing the performance found with different combinations of kernel for the SML dataset. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Algorithm	**MAE**	**ROC**	**Precision**	**Recall**	**F**1
KMR_{ib}	0.715 ± 0.001	**0.708 ± 0.002**	**0.562 ± 0.002**	**0.546 ± 0.005**	**0.533 ± 0.003**
KMR_{D}	0.748 ± 0.001	0.692 ± 0.002	0.546 ± 0.003	0.532 ± 0.004	0.505 ± 0.004
KMR_{F}	0.729 ± 0.001	0.693 ± 0.002	0.552 ± 0.003	0.526 ± 0.005	0.506 ± 0.003
KMR_{ib+F+D}	0.733 ± 0.001	0.705 ± 0.002	0.550 ± 0.002	0.540 ± 0.003	0.517 ± 0.002
KMR_{ib+F}	0.721 ± 0.001	0.706 ± 0.003	0.561 ± 0.002	0.545 ± 0.005	0.522 ± 0.002
KMR_{ib+D}	0.732 ± 0.001	0.705 ± 0.002	0.556 ± 0.003	0.542 ± 0.005	0.517 ± 0.003
KMR_{F+D}	0.735 ± 0.001	0.699 ± 0.002	0.544 ± 0.002	0.516 ± 0.005	0.501 ± 0.002
$KMR_{ib\otimes F\otimes D}$	0.736 ± 0.001	0.697 ± 0.003	0.551 ± 0.002	0.542 ± 0.016	0.510 ± 0.003
$KMR_{ib\otimes F}$	**0.714 ± 0.001**	0.698 ± 0.002	0.555 ± 0.004	0.532 ± 0.005	0.510 ± 0.003
$KMR_{ib\otimes D}$	0.727 ± 0.001	0.705 ± 0.002	0.554 ± 0.002	0.542 ± 0.003	0.518 ± 0.003
$KMR_{F\otimes D}$	0.739 ± 0.001	0.695 ± 0.002	0.551 ± 0.002	0.540 ± 0.003	0.509 ± 0.003

information vectors. In particular, we consider the rating information, feature information, and demographics information as described in Section 5.2.

Table 8 shows the performance of different combinations of kernels for the SML dataset. We have shown not only the Mean Absolute Error (MAE), but also a number of measures of the ability to classify films as either highly rated or poorly rated (refer to Appendix B for details). We observe reasonable performance using just rating information, demographic information and feature information. Interestingly, for this dataset, combining kernels does not give significantly better performance than using a kernel based on a single source of information. A plausible explanation of this observation is that our error rates are close to the optimum that can be achieved (there is a limit on the performance of any system due to the fickleness of the users making the ratings); or, at least, we are close to the optimum given the way we have represented the problem. On other datasets where, for example, ratings for some users are very sparse, demographic and feature information can be much more significant. The other striking feature of Table 8 is that multiplying kernels together seems to be more successful than adding different kernels.

Similar results (not shown) were observed in the case of FT and ML10 datasets. We also attempted to linearly combine the predictions from different kernels, but again this gave no improvement.

6.4. Combining the user- and item-based versions

The methods of combining the user- and item-based versions (mentioned in Section 4.2) did not give any significant improvement over the individual results for the whole dataset. To check the performance for imbalanced datasets, we (randomly) selected 200 users and 300 movies from the SML dataset, and 200 users and 50 movies from the FT dataset; and randomly withheld x% of their ratings. We checked the performance for two cases: for Case 1, the value of x was chosen uniformly at random to lie between 0 to 50 (i.e. $x \in [0, 50]$), whereas for Case 2, the value of x lies between 0 to 100 (i.e. $x \in [0, 100]$). The latter case creates a relatively imbalanced subset of the dataset as compared to the former one.

Table 9 shows the performance of user-based, item-based, and different methods used to combine the individual versions. We use the average of user- and item-based versions as a baseline. We observe that linearly combining the individual recommender systems does not give significant improvement over the baseline and the same is true for the second method (discussed in 4.2). However, KMR^{Var}_{Hybrid} does significantly improve the performance, with p-value in the case of pair-t test compared with the baseline recommender found to be less than 10^{-6} for both datasets. Similar results were observed for other datasets as well. What is evident from Table 9 is that user- and item-based versions of the algorithm are complementary and can improve the performance, if combined in a systematic way, for the imbalanced dataset.

Table 9: Combining the user-based and item-based versions under imbalanced datasets. The Case 2 produces a relatively sparse subset of the dataset compared to Case 1. The best results are shown in bold font.

Approach	MAE			
	Case 1		Case 2	
	FT	SML	FT	SML
KMR_{ib}	1.969 ± 0.002	0.882 ± 0.002	1.996 ± 0.002	0.941 ± 0.002
KMR_{ub}	1.525 ± 0.001	0.831 ± 0.002	1.751 ± 0.001	0.903 ± 0.002
$(KMR_{ib} + KMR_{ub})/2$	1.675 ± 0.002	0.829 ± 0.002	1.763 ± 0.002	0.901 ± 0.002
KMR^{Linear}_{Hybrid}	1.524 ± 0.002	0.826 ± 0.002	1.715 ± 0.002	0.895 ± 0.002
KMR^{Cnt}_{Hybrid}	1.516 ± 0.002	0.825 ± 0.001	1.704 ± 0.002	0.903 ± 0.001
KMR^{Var}_{Hybrid}	**1.463 ± 0.002**	**0.765 ± 0.001**	**1.545 ± 0.002**	**0.802 ± 0.001**

6.5. Sparse, skewed, and imbalanced datasets

In practical applications recommender systems often have access to limited and highly skewed information. Examples of these are

New user cold-start scenario where new users have relatively few ratings.

New-item cold-start scenario where new items have relatively few ratings.

Long tail scenario where the majority of items have only a few ratings.

Imbalanced sparse datasets where the majority of users/items have only a few ratings.

In the datasets that we have used so far our test set consists of randomly chosen ratings and these are overwhelmingly in the dense region of the rating matrix. That is, the users that we tested typically have rated many items and the items have been rated by many users. Thus, the results we have described so far are not strongly influenced by problems of limited and skewed information. However, these problems are often vital for a recommender system to prosper. For example, to attract new users it is highly beneficial to be able to give them good quality recommendations before they have made many ratings. Similarly, to introduce new items into the system it is useful to make sensible recommendations even if the item has only gained a few ratings.

We have tested the four scenarios outlined above by modifying the datasets we have been using to exaggerate the sparseness or skewness of the data. We found that in all cases the standard predictor that we have been using up to now gives very poor performance. However, we could very substantially improve the performance by combining the standard predictor with predictions using the mean, median and mode of $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as described in Section 3.1.2. In the tables shown below we denote the modified predictor with a superscript M^4.

We concentrate on the new-user cold-start scenario as the results are representative of all four scenarios. The only major difference is in the new-item cold-start scenario where the feature-based and demographic-based recommenders also perform well as they are less influenced by a lack of ratings. Results for the new item cold-start, long tail, and sparse data scenarios are given in Appendix C.

6.5.1. New user cold-start scenario

To test the performance of the proposed algorithms under the new user cold-start scenario, we selected 100 random users, and kept their number of ratings in the training set to 2, 5, 10, 15, and 20. Keeping the number of ratings less than 20 ensures that a user is new, and it captures well the new user cold-start problem. The corresponding *MAE*, represented by *MAE*2, *MAE*5, *MAE*10, *MAE*15, and *MAE*20 is shown in Table 10. Using the standard predictor provides very poor performance. We can substantially improve the performance by combining the standard predictor with predictions using the mean, median and mode of $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as described in Section 3.1.2.

Recall that we learn the weights for combining the standard predictor with the predictor using the mean, mode and median. The value of the weights depend on the dataset. Figure 7 shows how the weights that have been learned change in the new user cold-start scenario as we increase the number of ratings in the training set. (The new user cold-start scenario is taken as an example; similar results were observed in both the new item cold-start and long tail scenarios). The x-axis shows the number of ratings given by users (selected as cold-start users) and the y-axis shows the weights associated with different predictors. We observe that the contribution of the mode, mean, and median predictors decreases with the increase in the number of ratings, and finally become zero when the maximum number of ratings are available, whereas, the contribution of the standard (ratings-based) predictor increases with the increase in the number of ratings, and becomes 1 when the maximum number of ratings are available.

7. Conclusion and future work

Recommender systems is a major research area in machine learning and data mining. A number of approaches have been proposed to solve the recommender system problem including: content-based filtering, Ontology-based approaches, supervised classification techniques, unsupervised clustering techniques, memory-based collaborative filtering, model-based approaches spanning a number of algorithms including singular value decomposition, matrix factorisation techniques, and principal component analysis. All these algorithms suffer from potential problems such as accuracy, scalability, sparsity and imbalanced dataset problems, cold-start problems, and long tail

Table 10: Comparing MAE observed in different approaches under **new user cold-start scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the KMR algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. The best results are shown in bold font.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	3.841 ± 0.002	3.542 ± 0.002	2.872 ± 0.002	2.683 ± 0.002	2.504 ± 0.002
KMR_{ub}	2.102 ± 0.002	1.984 ± 0.002	1.672 ± 0.002	1.547 ± 0.002	1.374 ± 0.001
KMR_D	3.623 ± 0.002	3.321 ± 0.002	2.091 ± 0.002	1.955 ± 0.002	1.896 ± 0.002
KMR_F	3.652 ± 0.002	3.452 ± 0.002	1.944 ± 0.002	1.836 ± 0.002	1.757 ± 0.002
$KMR_{ib}^{M^4}$	0.858 ± 0.002	0.851 ± 0.002	0.809 ± 0.001	0.790 ± 0.001	0.784 ± 0.001
$KMR_{ub}^{M^4}$	**0.843 ± 0.002**	**0.841 ± 0.002**	**0.795 ± 0.001**	**0.776 ± 0.001**	**0.774 ± 0.001**
$KMR_F^{M^4}$	0.860 ± 0.002	0.856 ± 0.002	0.814 ± 0.002	0.801 ± 0.002	0.783 ± 0.002
$KMR_D^{M^4}$	0.866 ± 0.002	0.865 ± 0.002	0.815 ± 0.002	0.795 ± 0.002	0.786 ± 0.002
$KMR_{ib+F}^{M^4}$	0.859 ± 0.002	0.857 ± 0.002	0.810 ± 0.002	0.786 ± 0.002	0.779 ± 0.002

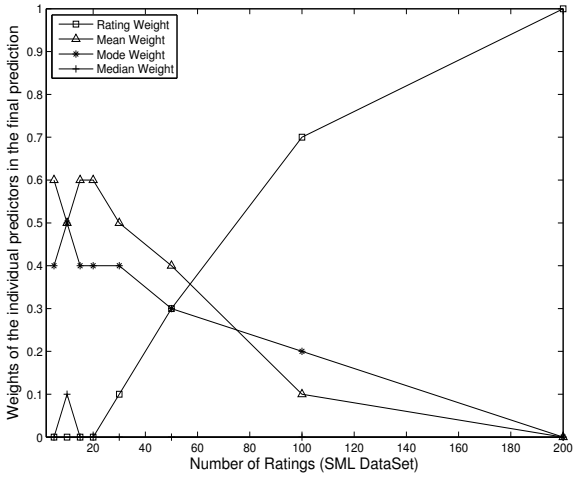

Figure 7: Weight learning over the validation set for the new user cold-start problem (SML dataset). "Number of Ratings" represents the number of ratings given by an active user in the training set.

problems in one way or the other. Against this background, we propose a new class of kernel-based recommendation algorithms that give state-of-the-art performance and eliminates the recorded problems with the recommender systems making the recommendation generation techniques applicable to a wider range of practical situations and real-world scenarios.

The proposed algorithm is competitive with what we believe to be the recommender with the best performance proposed by [26] and [27]. Interestingly both the proposed algorithm and the recommender proposed in [26] use kernel-based methods though in a very different way. Although kernel-based techniques are known to give excellent performance, recommender systems are challenging because of the size of the datasets. By carefully choosing the constraints we have been able to create a kernel-based learning machine that can be trained in linear time in the number of data points.

The algorithm we have developed is very flexible, thus we can easily adapt it so that it is either user-based or item-based. In addition it can use other information such as text-based features and these features can be easily combined. The best algorithm on the large datasets switches between the user-based and item-based information depending on the reliability of the predictions as measured by the spread in the prediction of the algorithms.

One interesting feature of the proposed approach is that we map the residues in the ratings onto a density function which encodes the uncertainty in the residue. For unseen residues we have interpreted the mapping $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as an approximation to a density function for the residue. Even though this function is not itself a density function (it becomes negative in some regions and is not normalised), nevertheless, it is very useful to consider the positive part of the function as a density function from which we can measure the mean, mode, median and variance. These measurements help in improving the performance, particularly in the case of sparse data.

One of the current drawbacks of the proposed algorithm is that the training occurs in one step. Thus, when new data are added it is costly to retrain the system. For practical recommender systems this is a significant problem as ratings are typically being added continuously. We are currently investigating using a perceptron-like algorithm for updating the Lagrange multipliers.

Acknowledgments

The work reported in this paper has formed part of the Instant Knowledge Research Programme of Mobile VCE, (the Virtual Centre of Excellence in Mobile & Personal Communications), www.mobilevce.com. The programme is co-funded by the UK Technology Strategy Board's Collaborative Research and Development programme. Detailed technical reports on this research are available to all Industrial Members of Mobile VCE.

Appendix A. Parameter learning

In this section, we describe how we tuned the other parameters of the system.

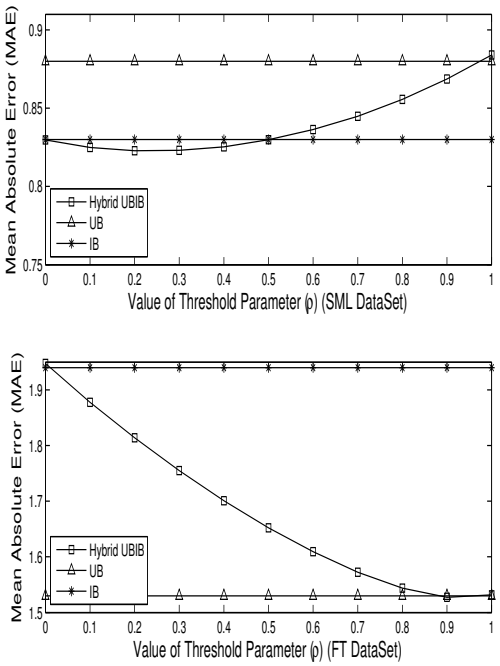

Figure A.8: Learning the optimal value of threshold parameter ρ, over the validation set, for the imbalanced datasets (refer to Section 6.4)

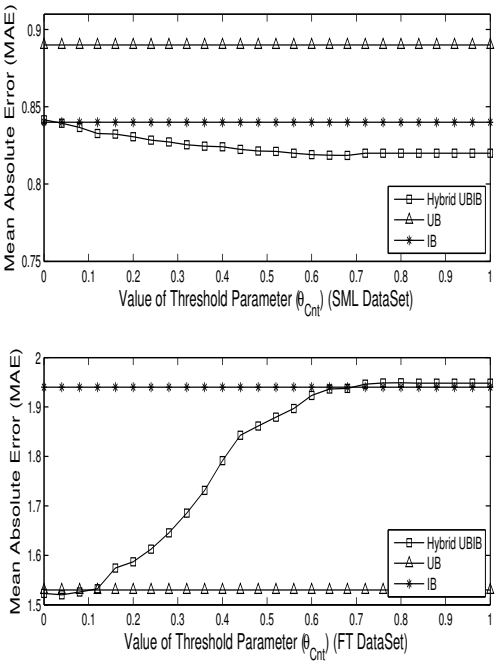

Figure A.9: Learning the optimal value of threshold parameter, θ_{Cnt} over the validation set, for the imbalanced datasets (refer to Section 6.4)

Appendix A.1. Parameters β_{rat}, β_{feat}, and β_{demo}

Parameters β_{rat}, β_{feat}, and $\beta_{demo} = 1 - \beta_{rat} - \beta_{feat}$ determine the relative weights of rating, feature, and demographic kernels in the final prediction. Note that we assume the three β-values are all positive and sum to one. 66 parameter sets were generated by producing all possible combination of parameters values, ranging from 0 to 1.0 with differences of 0.1. The parameter sets $\beta_{rat} = 1$ and $\beta_{feat} = 0$ gave the lowest *MAE* for all the datasets.

Appendix A.2. Parameter ρ

Parameters ρ and $(1 - \rho)$ determine the relative weights of user-based and item-based CF in the final prediction respectively. We changed the value of ρ from 0 to 1 with a difference of 0.1 and the resulting MAE has been shown in Figure A.8. Figure A.8 shows that for the SML dataset, the MAE is minimum at $\rho = 0.3$, after which it starts increasing again; whereas, for the FT dataset, the MAE keeps on decreasing, reaches its minimum at $\rho = 0.9$, an then increases again. We choose the optimal value of ρ to be 0.3 and 0.9 for SML and the FT dataset respectively. Similarly, the value of ρ was trained for other datasets. It is worth noting that the item-based version got more weight (except for the FT dataset) in the final prediction, for all datasets.

Appendix A.3. Parameter θ_{Cnt}

In the hybrid variant, KMR^{Cnt}_{Hybrid} the parameter θ_{Cnt} determines the switching point between using the item-based and user-based algorithms depending on the number of ratings of the item and the user. We determine the best value of θ_{Cnt} by varying it between 0 and 1 in steps of 0.04. Figure A.9 shows the parameter θ_{Cnt} learned (for Case 1, as discussed in Section 6.4) over the validation set. We observe that for the SML dataset, the MAE keeps on decreasing with the increase in the value of θ_{Cnt}, reaches its minimum at $\theta_{Cnt} \in [0.64, 0.68]$ and then either stays stable or starts increasing again. For the FT dataset, the MAE decreases initially, when the value of θ_{Cnt} changes from 0 to 0.04 and then starts increasing when the value of θ_{Cnt} increases beyond 0.04. For this reason, we choose the value θ_{Cnt} to be 0.68 and 0.04 for SML and the FT datasets respectively. Similarly, the value of θ_{Cnt} was trained for other datasets.

Appendix A.4. Parameter θ_{Var}

In the hybrid algorithm, KMR^{Var}_{Hybrid}, the parameter θ_{Var} controls the switching from the user-based prediction to the item-based prediction depending on the uncertainty in the predictions measured by the variance in the $\mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i)$. To learn this parameter we changed its value from 0 to 1 in steps of 0.04 and observed the corresponding MAE. Figure A.10 shows the parameter θ_{Var} learned (for Case 1 as discussed in Section 6.4) over the validation. We observe that for the SML dataset, the MAE keeps on decreasing with the increase in the value of θ_{Var}, reaches its peak at 0.24, and then starts increasing again. For the FT dataset, the decrease in the MAE is not very significant, when $\theta_{Var} < 0.44$; however, afterwards, a sharp decrease in the MAE is observed. The MAE keeps on decreasing, reaches its minimum at 0.64, and then either it stays stable or starts increasing again. We choose the optimal value θ_{Var} to be 0.24 and 0.64 for SML and the FT datasets respectively. Similarly, the value of θ_{Var} was trained for other datasets.

Table A.11: A comparison of the proposed algorithm with others in terms of ROC-Sensitivity metric. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Algorithm	**Best *ROC-Sensitivity***				
	SML	**ML**	**ML**10	**FT**	**NF**
User-based CF	0.714 ± 0.004	0.742 ± 0.001	0.721 ± 0.002	0.526 ± 0.008	0.651 ± 0.002
Item-based CF	0.654 ± 0.003	0.730 ± 0.002	0.731 ± 0.002	0.541 ± 0.006	0.642 ± 0.002
Hybrid CF	0.708 ± 0.003	0.755 ± 0.002	0.724 ± 0.001	0.535 ± 0.006	0.658 ± 0.002
SVD	0.607 ± 0.002	0.634 ± 0.002	0.652 ± 0.002	0.469 ± 0.012	0.624 ± 0.003
KMR_{ib}	0.708 ± 0.002	0.732 ± 0.001	0.732 ± 0.002	0.570 ± 0.008	0.654 ± 0.003
KMR_{ub}	0.716 ± 0.002	0.752 ± 0.001	0.718 ± 0.002	0.590 ± 0.006	0.661 ± 0.003
KMR^{Var}_{Hybrid}	**0.729 ± 0.002**	**0.758 ± 0.001**	**0.738 ± 0.002**	**0.592 ± 0.006**	**0.662 ± 0.002**

Table A.12: A comparison of the proposed algorithm with others in terms of F1 (measured over top-20 recommendations) metric. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Algorithm	**Best *F*1**				
	SML	**ML**	**ML**10	**FT**	**NF**
User-based CF	0.536 ± 0.003	0.549 ± 0.002	0.542 ± 0.041	0.516 ± 0.004	0.427 ± 0.002
Item-based CF	0.526 ± 0.002	0.542 ± 0.002	0.550 ± 0.040	0.521 ± 0.004	0.434 ± 0.003
Hybrid CF	0.528 ± 0.021	0.562 ± 0.002	0.549 ± 0.040	0.518 ± 0.004	0.436 ± 0.002
SVD	0.441 ± 0.002	0.407 ± 0.003	0.477 ± 0.012	0.445 ± 0.003	0.404 ± 0.003
KMR_{ib}	0.533 ± 0.003	0.596 ± 0.003	0.549 ± 0.018	0.523 ± 0.005	0.439 ± 0.003
KMR_{ub}	0.531 ± 0.003	0.587 ± 0.003	0.545 ± 0.013	0.540 ± 0.004	0.440 ± 0.003
KMR^{Var}_{Hybrid}	**0.539 ± 0.003**	**0.606 ± 0.003**	**0.551 ± 0.004**	**0.545 ± 0.004**	**0.442 ± 0.003**

Table A.13: Comparing the MAE observed in different approaches under **new item cold-start scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the *KMR* algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	1.782 ± 0.003	1.692 ± 0.004	1.421 ± 0.004	1.321 ± 0.004	1.221 ± 0.004
KMR_{ub}	3.253 ± 0.005	3.055 ± 0.004	2.811 ± 0.004	2.610 ± 0.004	2.453 ± 0.004
KMR_F	0.881 ± 0.005	**0.821 ± 0.002**	**0.792 ± 0.002**	**0.783 ± 0.003**	**0.776 ± 0.002**
KMR_D	0.924 ± 0.005	0.873 ± 0.002	0.813 ± 0.002	0.809 ± 0.004	0.809 ± 0.002
$KMR^{M^4}_{ib}$	0.953 ± 0.003	0.948 ± 0.004	0.928 ± 0.003	0.918 ± 0.003	0.887 ± 0.003
$KMR^{M^4}_{ub}$	**0.840 ± 0.002**	0.848 ± 0.004	0.847 ± 0.003	0.837 ± 0.003	0.832 ± 0.002
$KMR^{M^4}_{F}$	0.905 ± 0.003	0.904 ± 0.003	0.838 ± 0.003	0.790 ± 0.003	0.782 ± 0.003
$KMR^{M^4}_{D}$	0.916 ± 0.003	0.916 ± 0.003	0.863 ± 0.003	0.815 ± 0.003	0.796 ± 0.003
$KMR^{M^4}_{F+ib}$	0.849 ± 0.002	0.837 ± 0.002	0.807 ± 0.002	0.795 ± 0.002	0.786 ± 0.002

Table A.14: Comparing MAE observed in different approaches under **long tail scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the *KMR* algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Approach	**Best MAE**					
	MAE2	**MAE**4	**MAE**6	**MAE**8	**MAE**10	**MAE**15
KMR_{ib}	3.666 ± 0.005	3.652 ± 0.005	3.487 ± 0.005	3.432 ± 0.005	3.414 ± 0.004	3.371 ± 0.005
KMR_{ub}	3.481 ± 0.004	3.415 ± 0.004	3.336 ± 0.005	3.265 ± 0.004	3.239 ± 0.004	3.208 ± 0.004
KMR_F	3.022 ± 0.004	3.017 ± 0.004	2.964 ± 0.004	2.894 ± 0.005	2.822 ± 0.004	2.761 ± 0.004
KMR_D	2.963 ± 0.004	2.946 ± 0.004	2.872 ± 0.004	2.820 ± 0.005	2.683 ± 0.004	2.608 ± 0.004
$KMR^{M^4}_{ib}$	0.976 ± 0.005	0.966 ± 0.003	0.865 ± 0.003	0.840 ± 0.003	0.820 ± 0.004	0.817 ± 0.003
$KMR^{M^4}_{ub}$	**0.884 ± 0.005**	0.875 ± 0.005	0.843 ± 0.003	0.834 ± 0.003	0.828 ± 0.003	0.820 ± 0.003
$KMR^{M^4}_{F}$	0.988 ± 0.003	0.970 ± 0.003	0.869 ± 0.003	0.845 ± 0.003	0.818 ± 0.003	0.810 ± 0.003
$KMR^{M^4}_{D}$	0.966 ± 0.004	0.964 ± 0.004	0.867 ± 0.004	0.841 ± 0.004	0.819 ± 0.004	0.815 ± 0.004
$KMR^{M^4}_{ib+F}$	0.885 ± 0.003	**0.860 ± 0.003**	**0.835 ± 0.003**	**0.829 ± 0.003**	**0.809 ± 0.003**	**0.802 ± 0.002**

Table A.15: Comparing the performance of the algorithms under **imbalanced and sparse datasets**. The superfix M^4 represents the corresponding version of the *KMR* algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. The average with the respective standard deviation of results over 5-fold is shown. The best results are shown in bold font.

Approach	MAE			
	$\mathbf{x} \in \{\mathbf{50\%}, \mathbf{100\%}\}$		$\mathbf{x} \in \{\mathbf{75\%}, \mathbf{100\%}\}$	
	FT	**SML**	**FT**	**SML**
KMR_{ib}	1.790 ± 0.002	1.040 ± 0.002	1.930 ± 0.002	1.171 ± 0.002
KMR_{ub}	2.237 ± 0.002	1.091 ± 0.002	2.250 ± 0.002	1.182 ± 0.002
KMR_D	1.801 ± 0.002	1.052 ± 0.001	1.943 ± 0.002	1.174 ± 0.002
KMR_F	1.773 ± 0.002	1.030 ± 0.001	1.912 ± 0.002	1.162 ± 0.002
$KMR_{ib}^{M^4}$	1.752 ± 0.002	0.941 ± 0.001	1.771 ± 0.002	0.981 ± 0.001
$KMR_{ub}^{M^4}$	1.775 ± 0.001	0.945 ± 0.001	1.791 ± 0.002	0.983 ± 0.001
$KMR_D^{M^4}$	1.762 ± 0.002	0.931 ± 0.001	1.781 ± 0.003	0.955 ± 0.001
$KMR_F^{M^4}$	1.758 ± 0.002	0.938 ± 0.001	1.775 ± 0.002	0.951 ± 0.001
$KMR_{ib+F}^{M^4}$	**1.739 ± 0.001**	**0.921 ± 0.001**	**1.749 ± 0.001**	**0.931 ± 0.001**

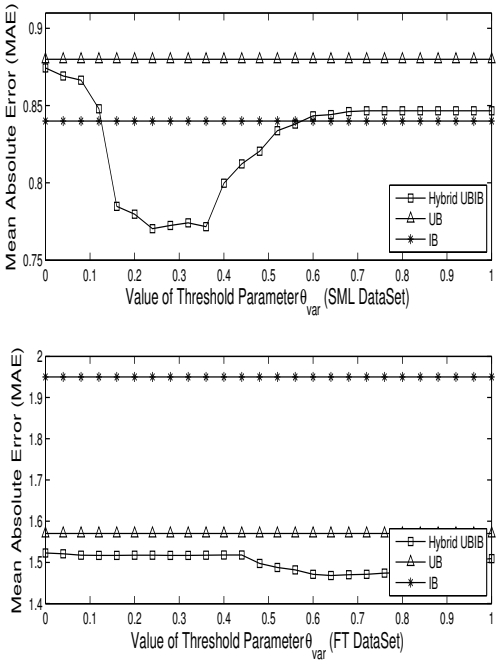

Figure A.10: Learning the optimal value of threshold parameter θ_{var}, over the validation set, for the imbalanced datasets (refer to Section 6.4)

Appendix B. Evaluation Metrics

Appendix B.1. Mean Absolute Error (MAE)

MAE measures the average absolute deviation between a recommender system's predicted rating and a true rating assigned by the user. It is computed as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|,$$

where p_i and a_i are the predicted and actual values of a rating respectively, and n is the total number of rating records in the test set. A rating record is a tuple consisting of a user ID, movie ID, and rating, $\langle uid, mid, r\rangle$, where r is the rating a recommender system has to predict. It has been used in [7, 46, 45, 44, 61, 58**?** , 12, 15**?**]. The aim of a recommender system is to minimize the MAE score.

Normalised Mean Absolute Error (NMAE) has been used in [28, 26], and is computed by normalising the MAE by a factor. The value of the factor depends on the range of the ratings; for example, for the MovieLens dataset, it is 1.6. The motivation behind this approach is that, the random guessing produces a score of 1. For further information, refer to [26].

A closely related measure to the MAE is the Root Mean Squared Error (RMSE), which is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}.$$

Both MAE and RMSE are quoted in the literature. RMSE will be slightly more sensitive to large outliers. The RMSE value will always be greater than or equal to the MAE value.

Appendix B.2. Receiver Operating Characteristic (ROC) Sensitivity

ROC measures the extent to which an information filtering system can distinguish between good and bad items. *ROC sensitivity* measures the probability with which a system accepts a good item. The ROC sensitivity ranges from 1 (perfect) to 0 (imperfect) with 0.5 for random. To use this metric for recommender systems, we must first determine which items are good (*signal*) and which are bad (*noise*). We followed the procedure describe in [13] while using this metric. It has been used in [15, 12, 13].

Table A.11 shows the ROC-sensitive for the same set of algorithms on the same datasets as shown in Table 5.

Appendix B.3. Precision, Recall, and F1

Precision, *recall*, and *F*1 evaluate the effectiveness of a recommender system by measuring the frequency with which it helps users selecting/recommending a good item. *Precision* gives us the probability that a selected item is relevant. *Recall* gives us the probability that a relevant item is selected.

Precision and recall should be reported together, as increasing the precision typically reduces the recall. The F1 Measure [18] combines the precision and recall into a single metric and has been used in many research projects, e.g. [46, 45, 49]. F1 is computed as follows:

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The first step in computing the precision and recall is to divide items into two classes; relevant and irrelevant, which is the same as in ROC-sensitivity. We calculated precision, recall, and F1 measures for each user, and reported the average results over all users.

Table A.12 shows the F1 measure for the same set of algorithms on the same datasets as shown in Table 5.

Appendix C. Sparse, skewed, and imbalanced datasets

In this appendix we present results for the new item cold-start scenario, the long tail scenario and for sparse datasets.

Appendix C.1. Performance evaluation under new item cold-start scenario

We tested the new item cold-start scenario in exactly the same way we did the new user cold-start scenario. That is, we selected 100 random items, and kept the number of users in the training set who have rated the these item to 2, 5, 10, 15, and 20. Table A.13 shows again that the standard predictor fails under this scenario, whereas including the mean, mode and median predictor gives very good performance. We note that for new items the feature-based and demographic-based recommenders work well for the cold-start scenario as these measures are not strongly influenced by a lack of rating information for an item.

Appendix C.2. Performance evaluation under long tail scenario

The long tail scenario [36] is an important scenario for practical recommender systems. In a large E-commerce system like Amazon, there are huge numbers of items that are rated by very few users and hence the recommendations generated for these items would be poor, which could weaken the customers' trust in the system.

To test the performance of the proposed algorithms under long tail scenario we created the artificial long tail scenario by randomly selecting the 80% of items in the tail. The number of ratings given in the tail part were varied between 2 to 15—this ensures that the item is new and has very few ratings. Table A.14 again shows the failure of the standard predictor in the long tail scenario and the improvement obtained by using the mean, mode and median predictor.

Appendix C.3. Performance evaluation under very sparse and imbalanced datasets

To check the performance of the proposed approaches under (very) sparse and imbalanced datasets, we created subsets of the datasets by withholding x% of the ratings from a rating profile of user/item, where $x \in [x_{min}, x_{max}]$. We show results for two scenarios: (1) $x_{min} = 50\%$, $x_{max} = 100\%$, (2) $x_{min} = 75\%$, $x_{max} = 100\%$. Changing the value of x_{min} creates different sparse subsets of the dataset, whereas keeping the value of x_{max} to 100% ensures that the imbalanced dataset is created for each scenario.

For the SML and FT datasets, the results are shown in Table A.15. Again this follows the same pattern as the long tail and cold-start scenarios.

Notes

[1] It might be due to the reasons that they used very simple kernels, such as correlation, identity, etc.; however, we used polynomial kernels, which are in turn are addition of correlation, identity, etc.

[2] We matched the movie titles, provided by the SML and FT dataset, against the titles in the IMDB (www.imdb.com).

[3] We used Google's stop word list `www.ranks.nl/resources/stopwords.html`.

[4] We used Porter Stemming algorithm for stemming.

[5] The authors did not provide any numeric value, only a graph is presented showing the minimum value approximately to 0.673.

References

[1] Hyung Jun Ahn, *A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem*, Information Sciences **178** (2008), 37–51.

[2] Katja Astikainen, Liisa Holm, Esa Pitkanen, Sandor Szedmak, and Juho Rousu, *Towards structured output prediction of enzyme function*, BMC Proceedings **2** (2008), no. Suppl 4, S2+.

[3] J. Basilico and T. Hofmann, *Unifying collaborative and content-based filtering*, Proceedings of the twenty-first international conference on Machine learning (New York, NY, USA), ACM Press, 2004, pp. 65–72.

[4] R.M. Bell, Y. Koren, and C. Volinsky, *The BellKor solution to the Netflix prize, in: AT&T Labs–Research: Technical report November*, 2007.

[5] Robert M. Bell and Yehuda Koren, *Lessons from the netflix prize challenge*, SIGKDD Explor. Newsl. **9** (2007), 75–79.

[6] Yolanda Blanco-Fernández, Martín López-Nores, Alberto Gil-Solla, Manuel Ramos-Cabrer, and José J. Pazos-Arias, *Exploring synergies between content-based filtering and Spreading Activation techniques in knowledge-based recommender systems*, Information Sciences **181** (2011), no. 21, 4823–4846.

[7] John S. Breese, David Heckerman, and Carl Kadie, *Empirical analysis of predictive algorithms for collaborative filtering*, Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (San Francisco, CA, USA), UAI'98, Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[8] Robin Burke, *Integrating knowledge-based and collaborative-filtering recommender systems*, In AAAI Workshop on AI in Electronic Commerce, AAAI, 1999, pp. 69–72.

[9] ———, *Hybrid recommender systems: Survey and experiments*, User Modeling and User-Adapted Interaction **12** (2002), no. 4, 331–370.

[10] Mark Claypool, Anuja Gokhale, Tim Mir, Pavel Murnikov, Dmitry Netes, and Matthew Sartin, *Combining content-based and collaborative filters in an online newspaper*, In Proceedings of ACM SIGIR Workshop on Recommender Systems (Berkeley, California), ACM, 1999.

[11] Dennis DeCoste, *Collaborative prediction using ensembles of maximum margin matrix factorizations*, Proceedings of the 23rd international conference on Machine learning (New York, NY, USA), ICML '06, ACM, 2006, pp. 249–256.

[12] Mustansar A. Ghazanfar and Adam Prügel-Bennett, *An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering*, Lecture Notes in Engineering and Computer Science: Proceedings of The International Multi Conference of Engineers and Computer Scientists 2010, IMECS 2010, 17–19 March, 2010, Hong Kong, 2010, pp. 493–502.

[13] ———, *Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative Filtering*, IAENG International Journal of Computer Science **37** (2010), no. 3, 272–287.

[14] ———, *Novel significance weighting schemes for collaborative filtering: Generating improved recommendations in sparse environments.*, DMIN, CSREA Press, 2010, pp. 334–342.

[15] ———, *A scalable, accurate hybrid recommender system*, Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining (Washington, DC, USA), WKDD '10, IEEE Computer Society, 2010, pp. 94–98.

[16] ———, *The advantage of careful imputation sources in sparse data-environment of recommender systems: Generating improved svd-based recommendations*, IADIS European Conference on Data Mining, July 2011.

[17] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry, *Using collaborative filtering to weave an information tapestry*, Commun. ACM **35** (1992), 61–70.

[18] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl, *Evaluating collaborative filtering recommender systems*, ACM Trans. Inf. Syst. **22** (2004), 5–53.

[19] Thorsten Joachims, *Training linear svms in linear time*, Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), KDD '06, ACM, 2006, pp. 217–226.

[20] Heung-Nam Kim, Abdulmotaleb El-Saddik, and Geun-Sik Jo, *Collaborative error-reflected models for cold-start recommender systems*, Decision Support Systems **51** (2011), no. 3, 519 – 531.

[21] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl, *Grouplens: applying collaborative filtering to usenet news*, Commun. ACM **40** (1997), 77–87.

[22] Yehuda Koren, *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), KDD '08, ACM, 2008, pp. 426–434.

[23] Yehuda Koren, *Factor in the neighbors: Scalable and accurate collaborative filtering*, ACM Transactions on Knowledge Discovery from Data (TKDD) **4** (2010), no. 1.

[24] M. Kurucz, A.A. Benczúr, and K. Csalogány, *Methods for large scale SVD with missing values*, Proceedings of KDD Cup and Workshop, Citeseer, 2007.

[25] Ken Lang, *NewsWeeder: learning to filter netnews*, Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995, pp. 331–339.

[26] Neil D. Lawrence and Raquel Urtasun, *Non-linear matrix factorization with gaussian processes*, Proceedings of the 26th Annual International Conference on Machine Learning (New York, NY, USA), ICML '09, ACM, 2009, pp. 601–608.

[27] Lester Mackey, David Weiss, and Michael I. Jordan, *Mixed membership matrix factorization*, Proceedings of the 27th International Conference on Machine Learning, June 2010.

[28] Benjamin Marlin, *Collaborative Filtering: A Machine Learning Perspective*, Master's thesis, University of Toronto, 2004.

[29] ———, *Modeling user rating profiles for collaborative filtering*, Advances in neural information processing systems **16** (2004), 627–634.

[30] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani, *Spectral regularization algorithms for learning large incomplete matrices*, J. Mach. Learn. Res. **99** (2010), 2287–2322.

[31] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan, *Content-boosted collaborative filtering for improved recommendations*, Eighteenth national conference on Artificial intelligence (Menlo Park, CA, USA), American Association for Artificial Intelligence, 2002, pp. 187–192.

[32] Stuart E. Middleton, *Capturing knowledge of user preferences with recommender systems*, Ph.D. thesis, UNIVERSITY OF SOUTHAMPTON, UK, September 2002.

[33] Stuart E. Middleton, Harith Alani, and David C. De Roure, *Exploiting Synergy Between Ontologies and Recommender Systems*, The Eleventh International World Wide Web Conference (WWW2002), 2002.

[34] Raymond J. Mooney and Loriene Roy, *Content-based book recommending using learning for text categorization*, Proceedings of the fifth ACM conference on Digital libraries (New York, NY, USA), DL '00, ACM, 2000, pp. 195–204.

[35] S.T. Park and D.M. Pennock, *Applying collaborative filtering techniques to movie search for better ranking and browsing*, Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2007, pp. 550–559.

[36] Yoon-Joo Park and Alexander Tuzhilin, *The long tail of recommender systems and how to leverage it*, Proceedings of the 2008 ACM conference on Recommender systems (New York, NY, USA), RecSys '08, ACM, 2008, pp. 11–18.

[37] Michael J. Pazzani, *A framework for collaborative, content-based and demographic filtering*, Artif. Intell. Rev. **13** (1999), 393–408.

[38] Michael J. Pazzani and Daniel Billsus, *The adaptive web*, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 325–341.

[39] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles, *Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach*, Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (San Francisco, CA, USA), UAI '00, Morgan Kaufmann Publishers Inc., 2000, pp. 473–480.

[40] C. Porcel, A. Tejeda-Lorente, M. A. Martínez, and E. Herrera-Viedma, *A hybrid recommender system for the selective dissemination of research resources in a technology transfer office*, Information Sciences **184** (2012), 1–19.

[41] Jasson D. M. Rennie and Nathan Srebro, *Fast maximum margin matrix factorization for collaborative prediction*, Proceedings of the 22nd international conference on Machine learning (New York, NY, USA), ICML '05, ACM, 2005, pp. 713–719.

[42] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl, *Grouplens: an open architecture for collaborative filtering of netnews*, Proceedings of the 1994 ACM conference on Computer supported cooperative work, CSCW '94, ACM, 1994, pp. 175–186.

[43] R. Salakhutdinov and A. Mnih, *Probabilistic matrix factorization*, Advances in Neural Information Processing Systems **20** (2008), 1257–1264.

[44] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl, *Item-based collaborative filtering recommendation algorithms*, Proceedings of the 10th international conference on World Wide Web (New York, NY, USA), WWW '01, ACM, 2001, pp. 285–295.

[45] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, *Analysis of recommendation algorithms for e-commerce*, Proceedings of the 2nd ACM conference on Electronic commerce (New York, NY, USA), EC '00, ACM, 2000, pp. 158–167.

[46] ———, *Application of dimensionality reduction in recommender system–a case study*, IN ACM WEBKDD WORKSHOP, Citeseer, 2000.

[47] ———, *Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering*, Proceedings of the Fifth International Conference on Computer and Information Technology, 2002.

[48] Vincent. Schickel-Zuber and Boi Faltings, *Using an Ontological A-priori Score to Infer User's Preferences*, Workshop on Recommender Systems-ECAI06, 2006, pp. 102–106.

[49] Jesus Serrano-Guerrero, Enrique Herrera-Viedma, Jose A. Olivas, Andres Cerezo, and Francisco P. Romero, *A google wave-based fuzzy recommender system to disseminate information in university digital libraries 2.0*, Information Sciences **181** (2011), 1503–1516.

[50] Upendra Shardanand and Pattie Maes, *Social information filtering: algorithms for automating word of mouth*, Proceedings of the SIGCHI conference on Human factors in computing systems (New York, NY, USA), CHI '95, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 210–217.

[51] Nathan Srebro, Jasson D. M Rennie, and T. Jaakkola, *Maximum-margin matrix factorization*, Advances in neural information processing systems **17** (2005), 1329–1336.

[52] David H. Stern, Ralf Herbrich, and Thore Graepel, *Matchbox: large scale online bayesian recommendations*, Proceedings of the 18th international conference on World wide web (New York, NY, USA), WWW '09, ACM, 2009, pp. 111–120.

[53] Sandor Szedmak, Ni Yizhao, and Gunn Steve R., *Maximum margin learning with incomplete data: Learning networks instead of tables.*, Journal of Machine Learning Research - Proceedings Track **11** (2010), 96–102.

[54] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk, *Investigation of various matrix factorization methods for large recommender systems*, Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition (New York, NY, USA), NETFLIX '08, ACM, 2008, pp. 6:1–6:8.

[55] ———, *Scalable collaborative filtering approaches for large recommender systems*, J. Mach. Learn. Res. **10** (2009), 623–656.

[56] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter, *Phoaks: a system for sharing recommendations*, Commun. ACM **40** (1997), 59–62.

[57] Robin van Meteren and Maarten van Someren, *Using content-based filtering for recommendation*, Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop, Citeseer, 2000.

[58] Manolis Vozalis and Konstantinos G. Margaritis, *Applying SVD on generalized item-based filtering*, International Journal of Computer Science and Applications **3** (2006), no. 3, 27–51.

[59] ———, *Using svd and demographic data for the enhancement of generalized collaborative filtering*, Information Sciences **177** (2007), 3017–3037.

[60] Mingru Wu, *Collaborative filtering via ensembles of matrix factorizations*, Proceedings of KDD Cup and Workshop, Citeseer, 2007.

[61] Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen, *Scalable collaborative filtering using cluster-based smoothing*, Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), SIGIR '05, ACM, 2005, pp. 114–121.

[62] Daoqiang Zhang, Zhi-Hua Zhou, and Songcan Chen, *Non-negative matrix factorization on kernels*, Proceedings of the 9th Pacific Rim international conference on Artificial intelligence (Berlin, Heidelberg), PRICAI'06, Springer-Verlag, 2006, pp. 404–412.