Kernel-Mapping Recommender: Scalable, Accurate, Hybrid Recommendation Algorithms

Mustansar Ali Ghazanfar and Adam Prügel-Bennett

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

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 we call the kernel-mapping recommender (KMR) which uses a novel structure learning technique. This paper makes contributions to the following: we show how (1) user-based and item-based versions of the KMR algorithms 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 it 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 recommender system problem that gives state-of-the-art performance. The main idea is to find a multilinear 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 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 Joachims (2006). This allows us to train the recommender in linear time. The method described here is a specialisation of a general structured learning framework developed by Szedmak and used in Szedmak et al. (2010) 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 set of ratings are treated as unseen while the other ratings are used for learning. The unseen data is then used to test the performance of the algorithm. To obtain accurate results, datasets are usually selected with users that have made a relatively high number of ratings. However, in real applications, 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, recommendations 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* (Gediminas Adomavicius, 2005) and the *long-tail problem* (Park & Tuzhilin, 2008). We find that the standard algorithm we developed performs poorly for these skewed data sets, 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 of a baseline algorithm by 10%. One lesson from this was that a highly effective way to achieve a very high recommendation performance on a static dataset is to combination of 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 applications 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. To the best of our knowledge the state-of-the-art algorithms are by Lawrence & Urtasun (2009) and Mackey et al. (2010). These achieve a considerable gap in performance advantage over the older algorithms. Our algorithm achieves similar performance to these approaches, although it is out-performed by Lawrence & Urtasun (2009) on a dataset with 1 000 000 ratings and by Mackey et al. (2010) on a dataset of 10 000 000 ratings. Our approach is however very different. The other two approaches are based on matrix factorisation, although Lawrence & Urtasun (2009) 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 our 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 data sets 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 (Goldberg et al., 1992; Shardanand & Maes, 1995; Terveen et al., 1997; Resnick et al., 1994; Konstan et al., 1997; Pennock et al., 2000; Ghazanfar & Prugel-Bennett, 2010c) 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 (Konstan et al., 1997), Ringo[1], etc. Collaborative filtering are classified into two sub-categories: memory-based CF and model-based CF. Memory-based approaches (Shardanand & Maes, 1995) make a prediction by taking into account the entire collection of previous rated items by a user, for example, the GroupLens recommender systems (Konstan et al., 1997). 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 recommendation for an *active user* (i.e. the user for whom the recommendations are computed) on a *target item* (the item a system wants to recommend). Examples include item-based CF (Sarwar et al., 2001), Singular Value Decomposition (SVD) based models (Sarwar et al., 2000b; Vozalis & Margaritis, 2007; Kurucz et al., 2007), Bayesian networks (Breese et al., 1998), and clustering methods (Park & Tuzhilin, 2008; Sarwar et al., 2002; Xue et al., 2005). Content-based filtering recommender systems (Pazzani & Billsus, 2007; Lang, 1995; Mooney & Roy, 2000; van Meteren & van Someren, 2000) 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 (Lang, 1995), a newsgroup filtering system uses the words of text as features. Other famous types of recommender systems include knowledge based systems (Burke, 1999), Ontology-based systems (Middleton et al., 2002), and hybrid systems (Burke, 2002).

Recommendations can be presented to an active user in the followings two different ways: 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.

A large number of approaches have been proposed to remedy the accuracy, sparsity, and scalability problems, ranging from supervised classification techniques (Billsus & Pazzani, 1998) to unsupervised clustering techniques (Park & Tuzhilin, 2008; Ghazanfar & Prugel-Bennett, 2011b), to dimensionality reduction techniques like singular value decomposition (SVD) or matrix factorization (Sarwar et al., 2000b; Lawrence & Urtasun, 2009). Classification techniques do not scale well with the dataset and furthermore they do not give accurate results. The clustering methods are effective at reducing the dimensionality of the datasets, however they do not give accurate results. Singular value decomposition or matrix factorization techniques give reasonably accurate results, however, they are expensive in terms of training and memory requirements, and often lead to the over-fitting. The proposed algorithms can be trained in linear time and provide accurate recommendations under all datasets.

Hybrid recommender systems have been proposed elsewhere (Melville et al., 2002; Burke, 2002; Pazzani, 1999; Claypool et al., 1999; Burke, 1999; Ghazanfar & Prugel-Bennett, 2010a,d,b), which combine individual recommender systems to avoid certain limitations of individual recommender systems. In out approach we can add more information (about items) in the forms of additional kernels, which can be thought of combining collaborative filtering with content-based filtering. A related approach has been proposed in Basilico & Hofmann (2004), 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[2].

In Szedmak et al. (2010), the authors proposed a structured learning algorithm for learning from incomplete dataset. The idea of the structure learning has been used in Astikainen et al. (2008), where the authors employed the structured output prediction for enzyme prediction. We show how the structure learning approach can be used to solve the recommender sys-

tem problem effectively.

3. Kernel-Mapping Recommender

Recommender system consists of two basic entities: users and items, where users provide their opinions (ratings) about items. We denote these users by $U = \{u_1, u_2, \cdots, u_M\}$, where the number of people using the system is $|U| = M$, and denote the set of items being recommended by $I = \{ i_1, i_2, \cdots, i_N \}$, with $|I| = N$. The users will have given ratings of some, but not all of the items. We denote these ratings by $(r_{iu}|(i,u) \in \mathcal{D})$, where $\mathcal{D} \subset I \times U$ 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 $|\mathcal{D}| = T \ll |I \times U| = N \times M$. 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 recommender 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 (Szedmak et al., 2010). 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

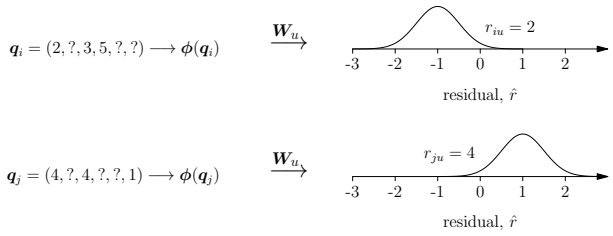

Figure 1: Schematic showing the aim of the algorithm. Information, $\boldsymbol{q}_i$ (in this case a rating vector) about an item i, is first mapped to a vector in an extended feature space $\boldsymbol{\phi}(\boldsymbol{q}_i)$. We then try to find the best linear mapping, W_u, for user u, to the 'vector', $\psi(\hat{r}_{iu})$, describing the residual. In this example we assume that $\bar{r}_i + \bar{r}_u + \bar{r} = \bar{r}_j + \bar{r}_u + \bar{r} = 3$.

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, such that the inner product satisfies the inequality

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

where $\zeta_i \geq 0$ is a slack variable. We then seek to minimise the Frobinius norm of $\mathbf{W}_u$ and the slack variables ζ_i. This is shown schematically in figure 1. We can describe the optimisation problem succinctly as

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

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 introduce 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

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

The Lagrangian is minimized 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 minimized 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 maximization 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

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

We are now in the position where we can apply the usual kernel trick. The kernel functions can be defined by

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

and 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 $f(\boldsymbol{\alpha})$

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

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

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

This guarantees that no step increases the objective function.

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

$$\sum_{i \in \mathcal{I}} \sum_{u \in \mathcal{D}_i} \alpha_{iu} \frac{\partial f(\boldsymbol{\alpha}_t)}{\partial \alpha_{iu}} + \text{const},$$

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 we can perform the linear programming independently for each

set of variables $\mathcal{A}_i$. Furthermore due to the simplicity of the constraint, it turns out that the linear programming problem can be solved in linear time (as opposed to cubic time for a general linear programming problem). The training is stopped after a fixed number of iterations which is a parameter of the training algorithm. The total complexity of this step for all users is $O(|D|)$. This leads to an algorithm with linear complexity in the number of available ranks. Note the nonzero elements of the matrix $\mathbf{M}$ is equal to $|\mathcal{D}_u|$ which tends to be constant, i.e. it does not increase with the number of users.

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

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

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. As we argued earlier $\mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i)$ 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

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

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

4. Extensions to the basic algorithm

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

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 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 relationship 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.

$$p_{iu}(\hat{r}) = \rho p^{ub}_{iu}(\hat{r}) + (1 - \rho) p^{ib}_{iu}(\hat{r}) \tag{5}$$

  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 rational behind this approach is the intuition that we if we have a large number of ratings for an item compared to the number 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 (the item with the most number of ratings). That is, we used

$$p_{iu}(\hat{r}) = \begin{cases} p^{ub}_{iu}(\hat{r}): & \text{if } \frac{|U_i|}{|U_{i_p}|} - \frac{|I_u|}{|I_{u_p}|} > \theta_{Cnt} \\ p^{ib}_{iu}(\hat{r}): & \text{otherwise.} \end{cases} \tag{6}$$

  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 item-based and user-based predictor. 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 we use according to

$$p_{iu}(\hat{r}) = \begin{cases} p^{ub}_{iu}(\hat{r}): & \text{if } \text{Var}_{ub} - \text{Var}_{ib} > \theta_{Var} \\ p^{ib}_{iu}(\hat{r}): & \text{otherwise.} \end{cases} \tag{7}$$

  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 (see next section 5.2 for details)

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

These kernels can be combine *linearly*

$$K = \beta_{rat}K_{rat} + \beta_{demo}K_{demo} + \beta_{feat}K_{feat}, \tag{8}$$

where the parameters β_{rat}, β_{demo} and $\beta_{feat} = 1 - \beta_{rat} - \beta_{demo}$ can be tunned 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*

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

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

$$\boldsymbol{\phi} = \boldsymbol{\phi}_{rat} \otimes \boldsymbol{\phi}_{demo} \otimes \boldsymbol{\phi}_{feat}. \tag{10}$$

5. Experimental Setup

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

5.1. Datasets

As is common in the field of recommender systems we used data from film recommendation sites to test our 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 the state-of-the-art. We used the following datasets:

- FilmTrust (denoted below by FT) obtained by crawling (on 10th of March 2009) the FilmTrust website[3]. Only users and movies having more than 5 ratings were used. This has been used before in Ghazanfar & Prugel-Bennett (2010a,d).
- MovieLens which we split into three groups
  - Small MovieLens (shown by SML) with 100 000
  - 1 million rating dataset (ML)
  - 10 million rating dataset (ML10)

  This has been widely used (Sarwar et al., 2001; Vozalis & Margaritis, 2007; Ghazanfar & Prugel-Bennett, 2010d,a; Lawrence & Urtasun, 2009).
- Random sub-sample of 20 000 user from the Netflix dataset (denoted below by NF). Netflix has been very widely (e.g see Koren (2008); Bell et al. (2007); Bell & Koren (2007)), 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 recommender 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 recommender algorithm using textual information we also obtain 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[4]. For the ML10 dataset, we used the tags and genre information that is provided with this dataset. After stop word[5] removal and stemming[6], 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). We used document frequency (DF) thresholding feature selection technique 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 Vozalis & Margaritis (2007); Ghazanfar & Prugel-Bennett (2010d) with the exception that we used the hierarchy of genre as shown in figure 2. To determine the weight of a genre in the genre vector, we used a simple weighting scheme as employed in QuickStep, a research paper Ontology based recommender system (Middleton, 2002). The main idea is that 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.

Table 1: 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	1.512	3.591

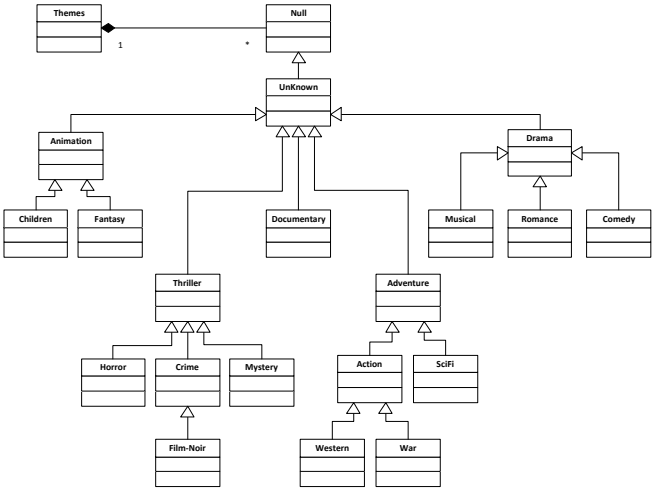

Figure 2: Hierarchy of genres based on Schickel-Zuber & Faltings (2006). All the super classes of a genre get a share when a genre receives some interest. For instance if a rated movie falls into "crime" genre, then the "crime" subject will get weight q, the immediate super class, "Thriller" will get weight of $q/2$; the next super class "Unknown" will get a weight of $q/4$.

5.3. Metrics

In the majority of the report, we have used the *Mean Absolute Error (MAE)* as our measure of performance as this is the most commonly use measure and *de facto* standard for benchmarking recommender systems. However, in practice 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 algorithms 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 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 our training set into a test and training set for measuring the parameters sensitivity. For learning the parameters, we conducted 2-fold cross validation on the training set.

5.5. Learning system parameters

There are 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. 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). Figure 3 and 4, 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.

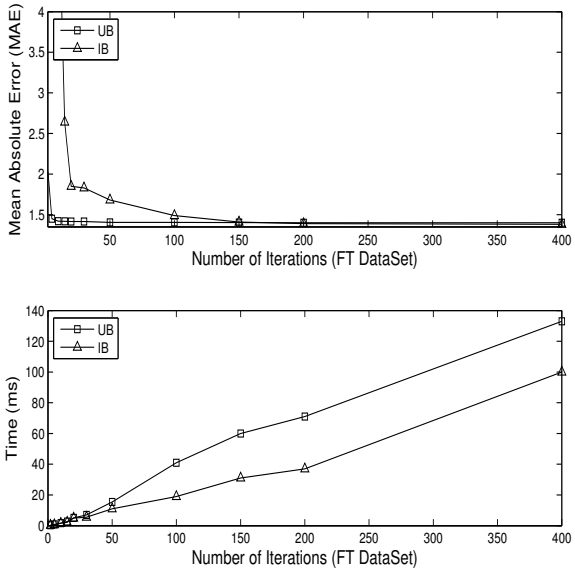

Figure 3: The number of iterations and time required to converge the proposed algorithms for the FT dataset.

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

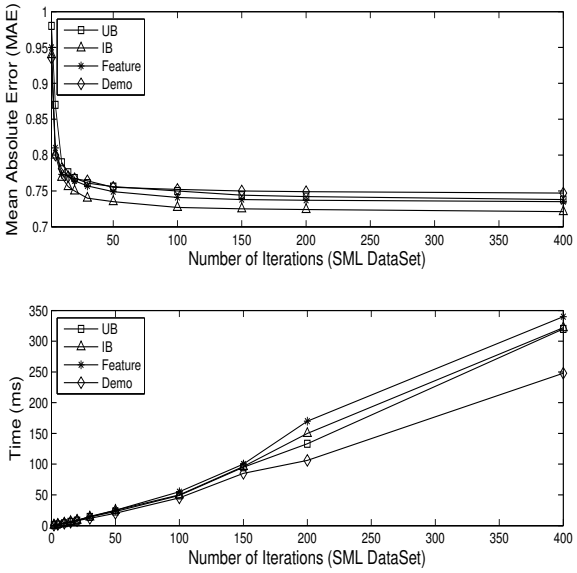

Figure 4: The number of iterations and time required to converge the proposed algorithms for the SML dataset.

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.

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 ML10 and NF dataset, 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

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^q}{\tau}\right), \tag{11}$$

where the best parameter (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 our 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 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 based on the number of citation 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 our algorithms with three different algorithms: user-based collaborative filtering (CF) with default Voting (DV)

Table 2: A comparison of the proposed algorithm with others in terms of MAE. The average with the respective standard deviation of results over 5 folds have been shown. The best results have been shown in bold.

Algorithm	**Best *MAE***				
	SML	**ML**	**ML**10	**FT**	**NF**
User-based CF	0.749 ± 0.003	0.715 ± 0.002	0.678 ± 0.041	1.433 ± 0.042	0.721 ± 0.001
Item-based CF	0.747 ± 0.002	0.711 ± 0.021	0.665 ± 0.040	1.436 ± 0.045	0.729 ± 0.001
Hybrid CF	0.742 ± 0.003	0.702 ± 0.002	0.664 ± 0.040	1.427 ± 0.045	0.715 ± 0.001
SVD	0.776 ± 0.003	0.735 ± 0.003	0.681 ± 0.012	1.483 ± 0.013	0.732 ± 0.040
KMR_{ib}	0.720 ± 0.004	0.668 ± 0.003	0.638 ± 0.019	1.385 ± 0.020	0.692 ± 0.042
KMR_{ub}	0.735 ± 0.002	0.687 ± 0.003	0.649 ± 0.014	1.403 ± 0.017	0.694 ± 0.040
KMR^{Var}_{Hybrid}	**0.717 ± 0.002**	**0.664 ± 0.003**	**0.633 ± 0.004**	**1.371 ± 0.015**	**0.687 ± 0.040**

proposed in Breese et al. (1998) (which provides a useful baseline for comparing algorihtms), item-based CF propose in Sarwar et al. (2001) (shown by *Item-Based CF*), and a SVD based approach proposed in Sarwar et al. (2000b) (shown by *SimpleSVD*). To provide as fair a comparison as possible, we tuned all parameters of the algorithms.

Table 2 shows that the KMR-based algorithms outperforms 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 3.3%, 2.1%, and 4.3% for the FT dataset; 3.9%, 1.9%, and 4.3% for the SML dataset; 6.6%, 3.9%, and 7.1% for the ML dataset; 5.9%, 4.2%, and 6.6% for the ML10 dataset; and 4.02%, 3.7%, and 4.7% for the NF dataset. The ROC-sensitivity and F1-measure on the same dataset are shown in tables B.7 and B.8, 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 generalization test procedures of Marlin (2004a) 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 3 random train-test splits as used in Lawrence & Urtasun (2009); Marlin (2004a); Rennie & Srebro (2005).

A comparison in terms of Normalized MAE (NMAE)—see Appendix B—of the algorithms is given in table 3. In table 3, URP represents the algorithm proposed in Marlin (2004b), Attitude represents the algorithm proposed in Marlin (2004a), MatchBox[7] is proposed in Stern et al. (2009), MMMF represents the maximum margin matrix factorization algorithm proposed in Rennie & Srebro (2005), ImputedSVD is proposed in Ghazanfar & Prugel-Bennett (2011a)[8], Item has been proposed in Park & Pennock (2007), E-MMF represents the ensemble maximum margin matrix factorization technique proposed in DeCoste (2006), and NLMF represents the non-linear matrix factorization technique (with linear and RBF versions) as proposed in Lawrence & Urtasun (2009).

Table 3 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.

Table 3: A comparison of different algorithms in terms of NMAE (Normalized MAE) for the ML dataset. The proposed algorithms outperforms URP (Marlin, 2004b), Attitude (Marlin, 2004a), MatchBox (Stern et al., 2009), MMMF (Rennie & Srebro, 2005), ImputedSVD (Ghazanfar & Prugel-Bennett, 2011a), and Item (Park & Pennock, 2007). They give the comparable results to E-MMF (DeCoste, 2006) and NLMF (Lawrence & Urtasun, 2009). Our results and the best results have been shown in bold.

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 ± 0.007
KMR_{ub}	0.425 ± 0.007
KMR^{Var}_{Hybrid}	**0.4065 ± 0.007**

To the best of our knowledge, the best results for the MovieLens $10M$ dataset that has been reported in the literature are those proposed in Lawrence & Urtasun (2009) and Mackey et al. (2010). They claimed their proposed algorithm gives

Table 4: A comparison of different algorithms in terms of RMSE for the ML10 dataset. NLMF represents the non-linear matrix factorization technique as proposed in Lawrence & Urtasun (2009) and M^3F-TIB represents the Mixed Membership Matrix Factorization model as proposed in Mackey et al. (2010). Our results and the best results have been shown in bold.

Algorithm	**RMSE**
NLMF	0.8740 ± 0.02
M^3F-TIB	**0.8447 ± 0.009**
KMR_{ib}	0.8721 ± 0.007
KMR_{ub}	0.8990 ± 0.007
KMR^{Var}_{Hybrid}	**0.8632 ± 0.007**

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 4. Table 4 shows that the proposed algorithms outperform Lawrence & Urtasun (2009)'s results. The percentage improvement is found to be less than 1% in the case of KMR_{ib} and KMR^{var}_{Hybrib}. The M^3F-TIB algorithm gave the best results outperforming our best algorithm KMR^{Var}_{Hybrid} with 2.1% decrease in error. M^3F-TIB is actually a combination of two matrix factorisation algorithms. It illustrates the power of carefully combining different algorithms.

Unfortunately, no NMAE (or MAE) was provided for M^3F-TIB technique (Mackey et al., 2010) over Movielens 1M dataset, which makes it harder to compare different algorithms results with M^3F-TIB. Considering these result, we conclude that our approach appears to be competitive with the current state-of-the-art.

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. Our framework allows these different sources to be exploited by combining different kernels built from different information vectors. In particular, we consider the rating information, feature information, and demographics information as described in section 5.2.

Table 5 shows the performance of the different combinations of kernels on 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 (see 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 5 is that multiplying kernels together seem to be more successful than adding different kernels.

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

6.4. Sparse 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 datasets where some of the users/items have relatively 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.

6.4.1. 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 6. 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 5 shows how the weights that have been learned change in the new user

Table 5: Comparing the performance found with different combinations of kernel for the SML dataset. The average with the respective standard deviation of results over 5 folds have been shown. The best results have been shown in bold.

Algorithm	**MAE**	**ROC**	**Precision**	**Recall**	**F**1
KMR_{ib}	0.720 ± 0.007	**0.697 ± 0.013**	**0.562 ± 0.002**	**0.546 ± 0.014**	**0.522 ± 0.008**
KMR_{D}	0.748 ± 0.006	0.682 ± 0.008	0.546 ± 0.009	0.532 ± 0.009	0.505 ± 0.010
KMR_{F}	0.730 ± 0.007	0.683 ± 0.010	0.552 ± 0.008	0.526 ± 0.010	0.506 ± 0.008
KMR_{ib+F+D}	0.733 ± 0.007	0.695 ± 0.010	0.550 ± 0.007	0.540 ± 0.007	0.517 ± 0.007
KMR_{ib+F}	0.721 ± 0.007	0.696 ± 0.012	0.562 ± 0.003	0.545 ± 0.010	0.522 ± 0.007
KMR_{ib+D}	0.732 ± 0.007	0.695 ± 0.010	0.556 ± 0.007	0.542 ± 0.012	0.517 ± 0.009
KMR_{F+D}	0.735 ± 0.006	0.689 ± 0.010	0.544 ± 0.005	0.516 ± 0.013	0.501 ± 0.006
$KMR_{ib\otimes F\otimes D}$	0.736 ± 0.007	0.687 ± 0.012	0.551 ± 0.007	0.542 ± 0.040	0.510 ± 0.008
$KMR_{ib\otimes F}$	**0.719 ± 0.008**	0.688 ± 0.008	0.555 ± 0.012	0.532 ± 0.012	0.510 ± 0.008
$KMR_{ib\otimes D}$	0.727 ± 0.007	0.695 ± 0.008	0.554 ± 0.005	0.542 ± 0.008	0.518 ± 0.007
$KMR_{F\otimes D}$	0.739 ± 0.007	0.685 ± 0.008	0.551 ± 0.005	0.540 ± 0.008	0.509 ± 0.007

Table 6: 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 have been shown in bold.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	3.841 ± 0.022	3.542 ± 0.022	2.872 ± 0.022	2.683 ± 0.027	2.504 ± 0.024
KMR_{ub}	2.102 ± 0.021	1.984 ± 0.021	1.672 ± 0.015	1.547 ± 0.021	1.374 ± 0.016
KMR_{D}	3.623 ± 0.022	3.321 ± 0.022	2.091 ± 0.022	1.955 ± 0.026	1.896 ± 0.024
KMR_{F}	3.652 ± 0.022	3.452 ± 0.022	1.944 ± 0.022	1.836 ± 0.026	1.757 ± 0.024
$KMR_{ib}^{M^4}$	0.858 ± 0.018	0.851 ± 0.018	0.809 ± 0.014	0.786 ± 0.014	0.779 ± 0.015
$KMR_{ub}^{M^4}$	**0.843 ± 0.017**	**0.841 ± 0.017**	**0.802 ± 0.015**	**0.781 ± 0.015**	**0.778 ± 0.017**
$KMR_{F}^{M^4}$	0.860 ± 0.019	0.856 ± 0.020	0.814 ± 0.020	0.802 ± 0.021	0.788 ± 0.021
$KMR_{D}^{M^4}$	0.866 ± 0.021	0.877 ± 0.021	0.810 ± 0.022	0.795 ± 0.022	0.786 ± 0.022
$KMR_{ib+F}^{M^4}$	0.866 ± 0.021	0.877 ± 0.021	0.810 ± 0.022	0.795 ± 0.022	0.786 ± 0.022

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

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 factorization 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 problems in one way or the other. Against this background, we propose a new class of kernel-based recommender 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 kernel-based methods is competitive with what we believe to be the recommender with the best performance proposed by Lawrence & Urtasun (2009) and Mackey et al. (2010). Interestingly both the proposed algorithm and Lawrence & Urtasun (2009) recommender 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 al-

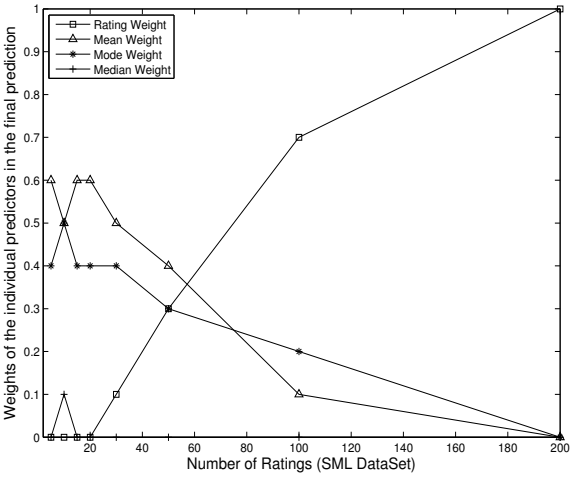

Figure 5: Weight learning over the validation set for the new user cold-start problem (SML dataset).

gorithm 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 our 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 is 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.

Acknowledgment

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.

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

Parameters $\beta_{rat}, \beta_{feat}$, and $\beta_{demo} = \beta_{rat} - \beta_{feat}$[9] determine the relative weights of rating, feature, and demographic kernels in the final prediction. The 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 parameters sets $\beta_{rat} = 1$ and $\beta_{feat} = 0$ gave the lowest *MAE* for all the datasets.

Appendix A.2. Parameter ρ

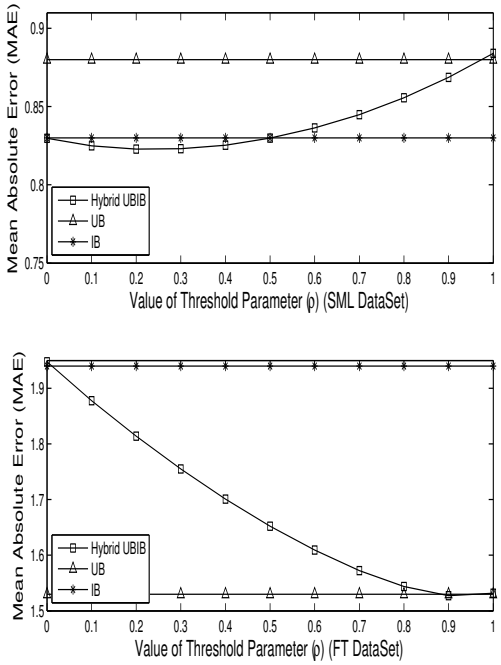

Figure A.6: Learning the optimal value of threshold parameter ρ, over the validation set.

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.6. Figure A.6 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 at its minimum at $\rho = 0.9$, an then increases again. For this reason, 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. Threshold θ_{Cnt}

In the hybrid variant, KMR^{Cnt}_{Hybrid} the threshold 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 user. We determine the best value of θ_{Cnt} by varying it between 0 and 1 in steps of 0.04. Figure A.7 shows the parameter θ_{Cnt} learned (for case 1, as discussed in section Appendix C.4) over the validation set. We observe that for

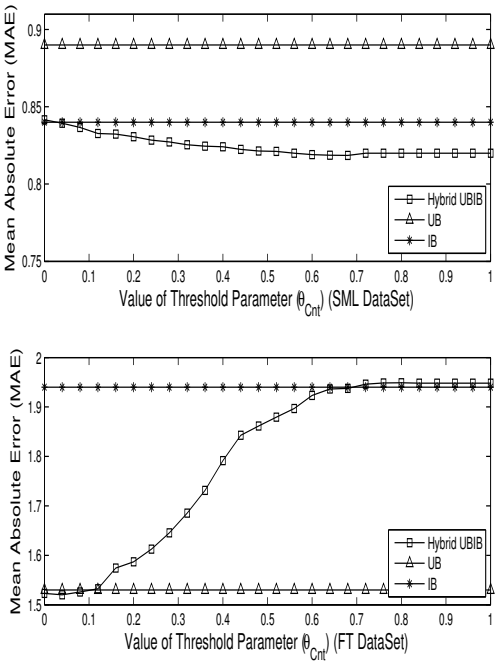

Figure A.7: Learning the optimal value of threshold parameter θ_{Cnt} over the validation set.

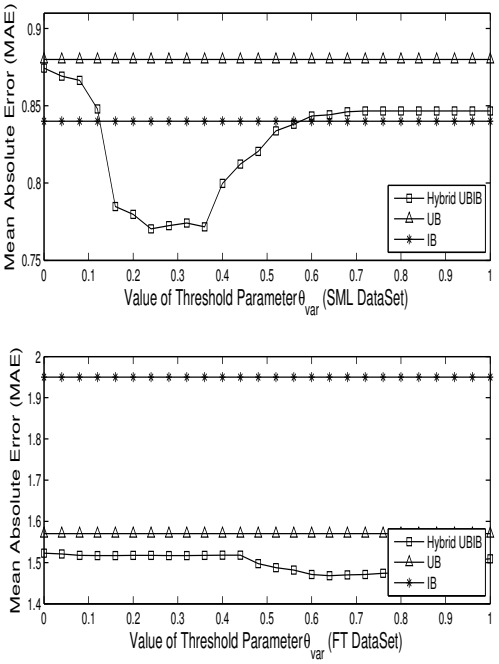

Figure A.8: Learning the optimal value of threshold parameter θ_{var}, over the validation set.

the SML dataset, the MAE keeps on decreasing with the increase in the value of θ_{Cnt}, reaches at its minimum between $\theta_{Cnt} \in [0.64 - 0.68]$ 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. Threshold 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 it value from 0 to 1 in steps of 0.04 and observed the corresponding MAE. Figure A.8 shows the parameter θ_{var} learned (for case 1 as discussed in section Appendix C.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 at the 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. For this reason 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} were trained for other datasets.

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

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

where p_i and a_i are the predicted and actual values of a rating respectively, and O 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, $< uid, mid, r >$, where r is the rating a recommender system has to predict. It has been used in Breese et al. (1998); Sarwar et al. (2001); Ghazanfar & Prugel-Bennett (2010d,a); Breese et al. (1998); Sarwar et al. (2000b); Vozalis & Margaritis (2006, 2007); Xue et al. (2005); Sarwar et al. (2000a). The aim of a recommender system is to minimize the MAE score.

Normalised mean absolute error (NMAE) has been used in Marlin (2004a); Lawrence & Urtasun (2009), 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 Lawrence & Urtasun (2009).

A closely related measure to the MAE is the root-mean-squared error RMSE, which is calculated as follows:

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

Both MAE and RMSE are quoted in the literature. RMSE will be slightly more sensitive to large outliers.

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 accept 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 took details from Ghazanfar & Prugel-Bennett (2010b) while using this metric. It has been used in Ghazanfar & Prugel-Bennett (2010d,a,b).

Table B.7 shows the ROC-sensitive for the same set of algorithms on the same databases as shown in table 2.

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 measure together, as it has been claimed that they are inversely proportional to each other, and furthermore, they depend on the size of the resultant vector returned to the user. *F1 Measure* (Jonathan L. Herlocker & Riedl, 2004) combines the *precision* and *accuracy* into a single metric and has been used in many research projects (Sarwar et al., 2000b,a). F1 is computed as follows:

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

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. Precision, recall, and F1 have been used in many research projects, such as Sarwar et al. (2000b,a). We calculated precision, recall, and F1 measures for each user, and reported the average results over all users.

Table B.8 shows the F1 measure for the same set of algorithms on the same databases as shown in table 2.

Appendix C. Sparse and skewed 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 C.9 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 is an important scenario for practical recommender systems. In a large E-commerce system like Amazon, there are huge number 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 (Park & Tuzhilin, 2008), 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 20—this ensure that the item is new and have very few ratings. Table C.10 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 dataset

To check the performance of the proposed approaches under (very) sparse and imbalanced dataset, we created subsets of the datasets by withholding x% 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) $x_{min} = 66\%$, $x_{max} = 100\%$. 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 dataset, the results are shown in table C.11. Again this follows the same pattern as the long-tail and cold start scenarios.

Appendix C.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 their x% ratings. We checked the performance for two cases: for case 1, the value of x lies 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 C.12 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

Table B.7: A comparison of the proposed algorithm with others in terms ROC-Sensitivity metric. The average with the respective standard deviation of results over 5 folds have been shown. The best results have been shown in bold.

Algorithm	**Best *ROC-Sensitivity***				
	(SML)	**(ML)**	**(ML**10)	**(FT)**	**(NF)**
User-based CF	0.714 ± 0.003	0.742 ± 0.002	0.651 ± 0.041	0.512 ± 0.042	0.644 ± 0.031
Item-based CF	0.674 ± 0.002	0.752 ± 0.021	0.732 ± 0.040	0.522 ± 0.045	**0.669 ± 0.036**
Hybrid CF	0.708 ± 0.021	0.755 ± 0.002	0.72 ± 0.040	0.525 ± 0.045	0.661 ± 0.031
SVD	0.607 ± 0.003	0.634 ± 0.003	0.652 ± 0.011	0.469 ± 0.012	0.665 ± 0.052
KMR_{ib}	0.695 ± 0.004	0.732 ± 0.003	0.732 ± 0.019	0.570 ± 0.019	0.659 ± 0.052
KMR_{ub}	0.703 ± 0.003	0.746 ± 0.003	0.718 ± 0.013	0.590 ± 0.016	0.661 ± 0.045
KMR^{Var}_{Hybrid}	**0.716 ± 0.002**	**0.758 ± 0.003**	**0.738 ± 0.004**	**0.592 ± 0.015**	0.667 ± 0.042

Table B.8: A comparison of the proposed algorithm with others in terms F1 (measured over top-20 recommendations) metric. The average with the respective standard deviation of results over 5 folds have been shown. The best results have been shown in bold.

Algorithm	**Best *F*1**				
	(SML)	**(ML)**	**(ML**10)	**(FT)**	**(NF)**
User-based CF	0.536 ± 0.003	0.479 ± 0.002	0.481 ± 0.041	0.480 ± 0.0423	0.397 ± 0.04
Item-based CF	0.493 ± 0.0024	0.522 ± 0.0211	0.530 ± 0.040	0.516 ± 0.045	**0.413 ± 0.03**
Hybrid CF	0.558 ± 0.021	0.499 ± 0.002	0.59 ± 0.040	0.508 ± 0.045	0.411 ± 0.04
SVD	0.441 ± 0.003	0.407 ± 0.003	1.700 ± 0.012	0.445 ± 0.012	0.384 ± 0.04
KMR_{ib}	0.522 ± 0.0036	0.481 ± 0.003	0.510 ± 0.018	0.5189 ± 0.019	0.399 ± 0.04
KMR_{ub}	0.522 ± 0.0034	0.478 ± 0.003	0.506 ± 0.013	0.5458 ± 0.016	0.401 ± 0.04
KMR^{Var}_{Hybrid}	**0.529 ± 0.0031**	**0.506 ± 0.0025**	**0.515 ± 0.004**	**0.554 ± 0.015**	0.411 ± 0.041

Table C.9: 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 folds have been shown. The best results have been shown in bold.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	1.782 ± 0.029	1.692 ± 0.039	1.421 ± 0.042	1.321 ± 0.041	1.221 ± 0.072
KMR_{ub}	3.253 ± 0.049	3.055 ± 0.044	2.811 ± 0.045	2.610 ± 0.043	2.453 ± 0.061
KMR_F	0.881 ± 0.049	**0.827 ± 0.023**	**0.792 ± 0.022**	**0.783 ± 0.043**	**0.776 ± 0.020**
KMR_D	0.924 ± 0.050	0.873 ± 0.024	0.813 ± 0.021	0.809 ± 0.043	0.809 ± 0.020
$KMR^{M^4}_{ib}$	0.953 ± 0.033	0.948 ± 0.035	0.928 ± 0.034	0.918 ± 0.034	0.887 ± 0.035
$KMR^{M^4}_{ub}$	**0.850 ± 0.023**	0.848 ± 0.043	0.847 ± 0.024	0.837 ± 0.027	0.832 ± 0.027
$KMR^{M^4}_{F}$	0.905 ± 0.029	0.904 ± 0.030	0.838 ± 0.030	0.790 ± 0.032	0.782 ± 0.043
$KMR^{M^4}_{D}$	0.916 ± 0.032	0.916 ± 0.031	0.863 ± 0.032	0.815 ± 0.035	0.796 ± 0.034
$KMR^{M^4}_{F+ib}$	0.866 ± 0.021	0.877 ± 0.021	0.810 ± 0.022	0.795 ± 0.022	0.786 ± 0.022

Table C.10: 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 folds have been shown. The best results have been shown in bold.

Approach	**Best MAE**					
	MAE2	**MAE**4	**MAE**6	**MAE**8	**MAE**10	**MAE**15
KMR_{ib}	3.666 ± 0.010	3.652 ± 0.010	3.487 ± 0.010	3.432 ± 0.010	3.414 ± 0.009	3.371 ± 0.010
KMR_{ub}	3.481 ± 0.009	3.415 ± 0.009	3.336 ± 0.010	3.265 ± 0.009	3.239 ± 0.009	3.208 ± 0.009
KMR_F	3.022 ± 0.009	3.017 ± 0.009	2.964 ± 0.009	2.894 ± 0.010	2.822 ± 0.009	2.761 ± 0.009
KMR_D	2.963 ± 0.009	2.946 ± 0.009	2.872 ± 0.009	2.820 ± 0.010	2.683 ± 0.009	2.608 ± 0.009
$KMR^{M^4}_{ib}$	0.976 ± 0.006	0.966 ± 0.006	0.865 ± 0.006	0.840 ± 0.006	0.820 ± 0.008	0.817 ± 0.007
$KMR^{M^4}_{ub}$	**0.894 ± 0.005**	0.875 ± 0.005	0.843 ± 0.006	0.834 ± 0.006	0.828 ± 0.006	0.820 ± 0.006
$KMR^{M^4}_{F}$	0.988 ± 0.006	0.970 ± 0.006	0.869 ± 0.006	0.845 ± 0.006	0.818 ± 0.006	0.810 ± 0.006
$KMR^{M^4}_{D}$	0.966 ± 0.008	0.964 ± 0.008	0.867 ± 0.008	0.841 ± 0.008	0.819 ± 0.008	0.815 ± 0.008
$KMR^{M^4}_{ib+F}$	0.895 ± 0.005	**0.872 ± 0.005**	**0.835 ± 0.005**	**0.833 ± 0.005**	**0.809 ± 0.005**	**0.804 ± 0.005**

Table C.11: 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 folds have been shown. The best results have been shown in bold.

Approach	**MAE**			
	$\mathbf{x} \in \{\mathbf{50\%}, \mathbf{100\%}\}$		$\mathbf{x} \in \{\mathbf{75\%}, \mathbf{100\%}\}$	
	FT	**SML**	**FT**	**SML**
KMR_{ib}	1.790 ± 0.052	1.040 ± 0.041	1.930 ± 0.055	1.171 ± 0.006
KMR_{ub}	2.237 ± 0.041	1.040 ± 0.006	2.250 ± 0.053	1.182 ± 0.007
KMR_{D}	1.801 ± 0.040	1.040 ± 0.006	1.932 ± 0.051	1.182 ± 0.007
KMR_{F}	1.773 ± 0.042	1.030 ± 0.006	1.912 ± 0.052	1.162 ± 0.007
$KMR_{ib}^{M^4}$	1.752 ± 0.031	0.941 ± 0.006	1.771 ± 0.042	0.981 ± 0.007
$KMR_{ub}^{M^4}$	1.775 ± 0.032	0.945 ± 0.006	1.791 ± 0.040	0.983 ± 0.007
$KMR_{D}^{M^4}$	1.762 ± 0.046	0.931 ± 0.006	1.781 ± 0.051	0.951 ± 0.007
$KMR_{F}^{M^4}$	1.758 ± 0.042	0.938 ± 0.006	1.775 ± 0.045	0.951 ± 0.007
$KMR_{ib+F}^{M^4}$	**1.739 ± 0.031**	**0.921 ± 0.006**	1.749 ± 0.034	**0.931 ± 0.007**

Table C.12: Combining the user-based and item-based versions under Imbalanced datasets. The best results have been shown in bold.

Approaches	**MAE**			
	Case1		**Case2**	
	FT	**SML**	**FT**	**SML**
KMR_{ib}	1.969 ± 0.020	0.831 ± 0.041	1.996 ± 0.020	0.941 ± 0.041
KMR_{ub}	1.525 ± 0.016	0.882 ± 0.041	1.751 ± 0.015	0.903 ± 0.041
$(KMR_{ib} + KMR_{ub}/2)$	1.675 ± 0.021	0.829 ± 0.050	1.763 ± 0.019	0.901 ± 0.041
KMR_{Hybrid}^{Linear}	1.524 ± 0.020	0.826 ± 0.050	1.715 ± 0.021	0.895 ± 0.041
KMR_{Hybrid}^{Cnt}	1.516 ± 0.020	0.825 ± 0.006	1.704 ± 0.020	0.903 ± 0.006
KMR_{Hybrid}^{Var}	**1.463 ± 0.019**	**0.765 ± 0.005**	**1.545 ± 0.018**	**0.802 ± 0.005**

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 C.12 is that user and item-based versions of the algorithm are complementary and can improve the performance, if combined in a systematical way, for the imbalanced dataset.

Notes

[1]www.ringo.com

[2]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.

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

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

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

[6]We used Porter Stemming algorithm for stemming.

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

[8]Algorithm using the combination of SVD and expectation maximization (EM) was chosen.

[9]We assume that $\beta_{rat} + \beta_{feat} + \beta_{demo} = 1$ without the loss of generalization.

References

Astikainen, K., Holm, L., Pitkänen, E., Szedmak, S., & Rousu, J. (2008). Towards structured output prediction of enzyme function. In *BMC proceedings*, vol. 2, (p. S2). BioMed Central Ltd.

Basilico, J., & Hofmann, T. (2004). Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*, (p. 9). ACM.

Bell, R., & Koren, Y. (2007). Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, *9*(2), 75–79.

Bell, R., Koren, Y., & Volinsky, C. (2007). The BellKor solution to the Netflix prize. *AT&T Labs–Research: Technical report November*.

Billsus, D., & Pazzani, M. (1998). Learning collaborative information filters. In *Proc. Intl Conf. Machine Learning*.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. (pp. 43–52). Morgan Kaufmann.

Burke, R. (1999). Integrating knowledge-based and collaborative-filtering recommender systems. In *Proceedings of the Workshop on AI and Electronic Commerce*.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, *12*(4), 331–370.

Claypool, M., Gokhale, A., Mir, T., Murnikov, P., Netes, D., & Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper. In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*.

DeCoste, D. (2006). Collaborative prediction using ensembles of maximum margin matrix factorizations. In *Proceedings of the 23rd international conference on Machine learning*, (pp. 249–256). ACM.

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

Ghazanfar, M., & Prugel-Bennett, A. (2010a). An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering. In *Lecture Notes in Engineering and Computer Science: Proceedings of The International Multi Conference of Engineers and Computer Scientists 2010*, (pp. 493–502). IMECS 2010, 17–19 March, 2010, Hong Kong. URL `http://eprints.ecs.soton.ac.uk/18483/`

Ghazanfar, M., & Prugel-Bennett, A. (2010b). Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative

Filtering. *IAENG International Journal of Computer Science*, *37*(3), 272–287.

Ghazanfar, M., & Prugel-Bennett, A. (2010c). Novel Significance Weighting Schemes for Collaborative Filtering: Generating Improved Recommendations in Sparse Environments. In *DMIN'10, the 2010 International Conference on Data Mining*. WORLDCOMP'10, 12–15 July, 2010, USA.
URL `http://eprints.ecs.soton.ac.uk/20846/`

Ghazanfar, M., & Prugel-Bennett, A. (2010d). A scalable, accurate hybrid recommender system. In *The 3rd International Conference on Knowledge Discovery and Data Mining (WKDD 2010)*. IEEE, 9–10 January, 2010, Thailand.
URL `http://eprints.ecs.soton.ac.uk/18430/`

Ghazanfar, M., & Prugel-Bennett, A. (2011a). The advantage of careful imputation sources in sparse data-environment of recommender systems: Generating improved svd-based recommendations. In *IADIS European Conference on Data Mining*.

Ghazanfar, M., & Prugel-Bennett, A. (2011b). Fulfilling the needs of gray-sheep users in recommender systems, a clustering solution. In *2011 International Conference on Information Systems and Computational Intelligence*.
URL `http://eprints.ecs.soton.ac.uk/21770/`

Goldberg, D., Nichols, D., Oki, B., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 70.

Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 217–226). ACM.

Jonathan L. Herlocker, L. G. T., Joseph A. Konstan, & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS) archive*, *22*, 734–749.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, *40*(3), 77–87.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 426–434). ACM.

Kurucz, M., Benczúr, A., & Csalogány, K. (2007). Methods for large scale SVD with missing values. In *Proceedings of KDD Cup and Workshop*. Citeseer.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *In Proceedings of the Twelfth International Conference on Machine Learning*.

Lawrence, N. D., & Urtasun, R. (2009). Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (pp. 601–608). New York, NY, USA: ACM.
URL `http://doi.acm.org/10.1145/1553374.1553452`

Mackey, L., Weiss, D., & Jordan, M. (2010). Mixed membership matrix factorization. In *Proc. ICML*. Citeseer.

Marlin, B. (2004a). Collaborative filtering: A machine learning perspective.

Marlin, B. (2004b). Modeling user rating profiles for collaborative filtering. *Advances in neural information processing systems*, *16*, 627–634.

Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *in Eighteenth National Conference on Artificial Intelligence*, (pp. 187–192).

Middleton, S. E. (2002). *Capturing knowledge of user preferences with recommender systems*. Ph.D. thesis, UNIVERSITY OF SOUTHAMPTON, UK.

Middleton, S. E., Alani, H., Shadbolt, N., & Roure, D. D. (2002). Exploiting synergy between ontologies and recommender systems. In *Semantic Web Workshop*.

Mooney, R. J., & Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, (pp. 195–204). San Antonio, US: ACM Press, New York, US.

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

Park, Y., & Tuzhilin, A. (2008). The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*, (pp. 11–18). ACM New York, NY, USA.

Pazzani, M., & Billsus, D. (2007). Content-based recommendation systems. (pp. 325–341).

Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, *13*(5 - 6), 393–408.

Pennock, D., Horvitz, E., Lawrence, S., & Giles, C. (2000). Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the 16th conference on uncertainty in artificial intelligence*, (pp. 473–480). Citeseer.

Rennie, J., & Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, (pp. 713–719). ACM.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, (pp. 175–186). ACM New York, NY, USA.

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, (pp. 285–295). ACM New York, NY, USA.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000a). Analysis of recommendation algorithms for e-commerce. (pp. 158–167). ACM Press.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000b). Application of dimensionality reduction in recommender system–a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*. Citeseer.

Sarwar, B. M., Karypis, G., Konstan, J., & Riedl, J. (2002). Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering.

Schickel-Zuber, V., & Faltings, B. (2006). Using an Ontological A-priori Score to Infer User's Preferences. In *Workshop on Recommender Systems-ECAI06*, (pp. 102–106).

Shardanand, U., & Maes, P. (1995). Social information filtering:algorithms for automating word of mouth. In *Proc. Conf. Human Factors in Computing Systems*, (pp. 210–217).

Stern, D., Herbrich, R., & Graepel, T. (2009). Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, (pp. 111–120). ACM.

Szedmak, S., Ni, Y., & Gunn, S. R. (2010). Maximum margin learning with incomplete data: Learning networks instead of tables. *Journal of Machine Learning Research, Proceedings*, *11, Workshop on Applications of Pattern Analysis*, 96–102. Http://jmlr.csail.mit.edu/proceedings/papers/v11/szedmak10a/szedmak10a.pdf.

Terveen, L., Hill, W., Amento, B., McDonald, D., & Creter, J. (1997). Phoaks: A system for sharing recommendations. In *Comm. ACM*, vol. 40, (pp. 59–62).

van Meteren, R., & van Someren, M. (2000). Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*. Citeseer.

Vozalis, M., & Margaritis, K. (2006). Applying SVD on generalized item-based filtering. *International Journal of Computer Science and Applications*, *3*(3), 27–51.

Vozalis, M., & Margaritis, K. (2007). Using SVD and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, *177*(15), 3017–3037.

Xue, G.-R., Lin, C., Yang, Q., Xi, W., Zeng, H.-J., Yu, Y., & Chen, Z. (2005). Scalable collaborative filtering using cluster-based smoothing. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 114–121). New York, NY, USA: ACM Press.