DOI 10.1007/s10115-003-0123-8
Springer-Verlag London Ltd. © 2004
Knowledge and Information Systems (2005) 7: 1–22

Knowledge and Information Systems

Collaborative Filtering Using a Regression-Based Approach

Slobodan Vucetic, Zoran Obradovic

Center for Information Science and Technology, Temple University, Philadelphia, PA, USA

Abstract. The task of collaborative filtering is to predict the preferences of an active user for unseen items given preferences of other users. These preferences are typically expressed as numerical ratings. In this paper, we propose a novel regression-based approach that first learns a number of experts describing relationships in ratings between pairs of items. Based on ratings provided by an active user for some of the items, the experts are combined by using statistical methods to predict the user's preferences for the remaining items. The approach was designed to efficiently address the problem of data sparsity and prediction latency that characterise collaborative filtering. Extensive experiments on Eachmovie and Jester benchmark collaborative filtering data show that the proposed regression-based approach achieves improved accuracy and is orders of magnitude faster than the popular neighbour-based alternative. The difference in accuracy was more evident when the number of ratings provided by an active user was small, as is common for real-life recommendation systems. Additional benefits were observed in predicting items with large rating variability. To provide a more detailed characterisation of the proposed algorithm, additional experiments were performed on synthetic data with second-order statistics similar to that of the Eachmovie data. Strong experimental evidence was obtained that the proposed approach can be applied to data over a large range of sparsity scenarios and is superior to non-personalised predictors even when ratings data are very sparse.

Keywords: Web mining; Recommendation systems; Collaborative filtering; Regression; Expert integration

1. Introduction

In today's society there is an increasing need for automated systems providing personalised recommendations to a user faced with a large number of choices. For example, an increasing choice of available products is caused by companies shifting towards developing customised products that meet specific needs of different

Received 28 July 2001
Revised 24 September 2002
Accepted 2 June 2003
Published online 6 February 2004

groups of customers (Pine 1993). The product customisation trend, coupled with E-commerce, with which customers are not provided with an option to examine the products off-the-shelf in a traditional sense, makes the problem of providing accurate personalised recommendations very important. Increasing the quality of personalised recommendations would increase customer satisfaction and loyalty, and at the same time reduce the costs caused by product return. Another example where personalised recommendations are extremely useful is an information overload situation with an amount of data available through the Internet and other media greatly exceeding the ability of a person to process it. Here, automated methods are needed to provide a large number of users with the ability to efficiently locate and retrieve information according to their preferences.

Personalised recommendation systems can be classified into two main categories: content-based and collaborative filtering, although some work has also been done in merging these two approaches to improve the quality of information filtering (Claypool et al. 1999; Delgado et al. 1998). In content-based filtering, mostly used for retrieving relevant textual documents (Maes 1994; Salton and Buckley 1998), a search is performed for items with content most similar to the user's interests. The task of collaborative filtering is to predict the preferences of an active user given a database of preferences of other users, in which the preferences are typically expressed as numerical evaluation scores. Scores can be obtained explicitly by recording votes from each user on a subset of available items, or implicitly, by inferring from a user's behaviour or reactions regarding a given set of items.

Memory-based collaborative filtering algorithms maintain a database of previous users' preferences and perform certain calculations on a database each time a new prediction is needed (Breese et al. 1998). The most common representatives are neighbour-based algorithms by which a subset of users most similar to an active user is chosen and a weighted average of their scores is used to estimate the preferences of an active user for other items (Aggrawal et al. 1999; Greening 1997; Konstan et al. 1997; Shardanand and Maes 1995). In contrast, model-based algorithms first develop a description model from a database and use it to make predictions for an active user. Published systems of this type include Bayesian clustering (Breese et al. 1998), Bayesian networks (Pennock et al. 2000), classification-based algorithms (Billsus and Pazzani 1998; Nakamura and Abe 1998), and algorithms for combining rankings (Freund et al. 1998). Another interesting line of research that can be useful for both memory- and model-based collaborative filtering is extracting preference patterns from data by using latent class models (Hoffman and Puzicha 1999) or by performing an appropriate clustering of items and users (Ungar and Foster 1998). Yet another type of collaborative filtering algorithms is based on extracting association rules from ratings data and using them for recommendation (Lin et al. 2002). However, it seems that association rules cannot yet provide a good trade-off between computational time, accuracy, and coverage needed for most practical recommendation systems.

Neighbour-based collaborative filtering algorithms have been shown to be superior to model-based in terms of accuracy (Breese et al. 1998). However, their high latency in giving predictions for active users can be a serious drawback in systems with a large number of requests that should be processed in real-time. Often, to reduce computational effort, some form of clustering is used to group users with similar preferences (Aggrawal et al. 1999; Goldberg et al. 2001; Ungar and Foster 1998). However, this effort can result in a loss of recommendation quality. Additionally, previous results (Herlocker et al. 1999) have shown that as the number of items evaluated by an active user decreases, the prediction accuracy of neighbourhood-based algorithms deteriorates dramatically. (Demanding extensive user effort for successful profiling can discourage a user from using the recommendation system.) Finally, it is unlikely that two users will have exactly the same taste over all possible items, while it is more probable that the similarity will be larger over certain subsets of items (e.g., two users of a movie recommendation system could share opinions on dramas while disagreeing over science fiction).

In this paper we propose a regression-based approach to collaborative filtering on numerical ratings data that searches for similarities between items, builds a collection of experts in the form of simple linear models, and combines them efficiently to provide preference predictions for an active user. Our approach is similar to previously proposed item-based recommendation algorithms (Karypis 2001; Linden et al. 2001) in the way it uses correlation between item ratings to predict ratings on unseen items. The Top-N algorithm (Karypis 2001) was designed for binary ratings data; it uses several suitable heuristics to calculate item similarities, and sums the similarities to provide the list of the top N items of interest to an active user. The item-to-item similarity mappings approach (Linden et al. 2001) was designed for numerical ratings data; it calculates item similarities based on an overlap of previous users, and uses them as weights to predict ratings on unseen items from available ratings of an active user. The proposed heuristic strongly weights ratings from popular items and from recently purchased items. Unlike these algorithms that use different heuristics to utilise item-to-item correlations in recommendation, in our regression-based approach, we propose statistically based algorithms that are robust to different levels of data sparsity in historical and active user ratings, and are able to provide rapid on-line recommendations.

In Sect. 2 we provide a formal description of a recommendation problem, followed by a description of several recommendation algorithms used for comparison with our approach. There, we propose three simple recommendation algorithms: the first two are not personalised, while the third uses active user ratings in a very simple manner. We also outline the neighbour-based algorithm proposed by GroupLens (Herlocker et al. 1999) as a representative of a large class of similar algorithms such as FireFly (Shardanand and Maes 1995) or LikeMinds (Greening 1997). In Sect. 3 we describe and provide the motivation for the regression-based approach. We explain several algorithms with different computational requirements, achievable accuracy, and robustness to sparsity. We also propose several adjustments to the proposed algorithms for dealing with the specifics of user ratings in realistic recommendation scenarios. In Sect. 4 we characterise the proposed algorithms on real-life Eachmovie and Jester data sets, as well as on synthetic data with second-order characteristics similar to Eachmovie data. We examine their properties with respect to accuracy, coverage, and off-line and on-line efficiency in a number of data sparsity scenarios.

2. Preliminaries

2.1. Description of the Personalised Recommendation Task

Assuming a database of I items partially evaluated by U users, we are given a $\mathrm{U} \times \mathrm{I}$ matrix $\mathbf{D}$ with element r_{ui} representing the evaluation score of item i by user u. In realistic systems the matrix $\mathbf{D}$ is usually very sparse since users are likely to vote just for a small subset of available items. By r_{u*}, r_{*i}, and r_{**} we denote the average score for each user, the average score for each item, and the overall average score, respectively. Since each user does not vote for each item, by $\mathbf{I}_u$ we denote the subset of items rated by user u. Similarly, $\mathbf{U}_i$ denotes the subset of users that evaluated item i. Given the evaluation scores of an active user a on items $\mathbf{I}_a$ the recommendation system task is to estimate scores of user a on the remaining items $\mathbf{I}\backslash\mathbf{I}_a$.

2.2. Simple Recommendation Algorithms

We propose three simple recommendation algorithms to establish accuracy lower bounds on more complex recommendation systems. The first, MEAN, uses the overall average score r_{**} as a prediction of a preference of any user on any item. The second, GENERIC, uses the average scores r_{*i} of each item i as predictions of a new user's preferences. This algorithm is valid under the assumption that all users have similar preferences. The MEAN and GENERIC algorithms do not attempt to use information contained in scores provided by an active user.

If users have provided numerical ratings explicitly, it is likely that the subjective nature of ratings will distort overall ratings matrix $\mathbf{D}$. The most obvious consequence of subjective ratings would be the difference in average scores between otherwise correlated users. Therefore, the predictions of ADJUSTED_GENERIC for active user a are obtained as predictions from GENERIC adjusted by the difference $\Delta\mathrm{r}_a$ between the average score of user a and the average generic score over the same set of items, $\mathbf{I}_a$, defined as

$$\Delta\mathrm{r}_a = \frac{1}{|\mathbf{I}_a|}\sum_{i\in\mathbf{I}_a}(\mathrm{r}_{ai} - \mathrm{r}_{*i}), \tag{1}$$

where $|\mathbf{I}_a|$ is the cardinality of set $\mathbf{I}_a$. Hence the prediction of ADJUSTED_GENERIC for an active user preference on item i is calculated as $\mathrm{p}_{ai} = \mathrm{r}_{*i} + \Delta\mathrm{r}_a$.

2.3. Nearest-Neighbour-Based Algorithms

Nearest-neighbour algorithms are used in a number of recommendation systems. For benchmark comparison to our proposed alternatives, we have chosen a GropLens algorithm (Herlocker et al. 1999) as a representative of the class of nearest-neighbour algorithms. In the algorithm, Pearson correlation is used to measure similarity between user u and an active user a (Herlocker et al. 1999). Here,

$$\mathrm{w}_{a,u} = \frac{\sum_{i\in\mathbf{I}_a\cap\mathbf{I}_u}(\mathrm{r}_{ai} - \mathrm{r}_{a*})(\mathrm{r}_{ui} - \mathrm{r}_{u*})}{\sigma_a\sigma_u}, \tag{2}$$

where σ_a and σ_u are standard deviations of scores calculated over $\mathbf{I}_a \cap \mathbf{I}_u$. The score p_{ai} of user a for item i is predicted as a weighted sum of the votes of other users as

$$\mathrm{p}_{ai} = \mathrm{r}_{a*} + \frac{\sum_{u\in\mathbf{U}_i}\mathrm{w}_{a,u}(\mathrm{r}_{ui} - \mathrm{r}_{u*})}{\sum_{u\in\mathbf{U}_i}\mathrm{w}_{a,u}}. \tag{3}$$

If two users have a small number of co-rated items, it is probable that their Pearson correlation estimates will be high simply by chance. Existence of such false neighbours can significantly deteriorate the accuracy. Significance weighting (Herlocker et al. 1999) was proposed to reduce the weights if the number of co-rated items, n, is smaller than some predetermined number N. If this is the case, the obtained weight is multiplied by n/N. Neighbourhood selection was introduced to retain only a small subset of the most similar users for prediction (Herlocker et al. 1999). Predicting an item i is done effectively by retaining K of the most similar users from $\mathbf{U}_i$. The reported benefits were twofold: the computational time needed for each prediction was reduced and slight improvements in accuracy were observed. We implemented both modifications in a neighbour-based algorithm used for comparison with the proposed regression-based algorithms. We denote the neighbour-based algorithm with these modifications as NEIGHBOUR(N, K), where N and K are adjustable significance and selection parameters of the algorithm.

To make a prediction for a new user, Pearson correlations over U existing users should be calculated first using (2), which scales as O(UI), where I is the total number of items. Reducing to the K most similar users for each item to be predicted and applying (3) still scales as O(UI).

Also, the whole database $\mathbf{D}$ of size O(UI) should be maintained in memory during the on-line prediction process.

2.4. Performance Measures

We use the coverage, Mean Absolute Error (MAE), and ROC sensitivity to compare examined recommendation algorithms. The coverage is calculated as the percentage of items from a test set for which a given algorithm has been able to predict. The MAE of predicted scores is a statistical accuracy measure used for collaborative filtering algorithms, while the ROC sensitivity is a measure of the diagnostic power of prediction algorithms (Herlocker et al. 1999).

3. Regression-Based Recommendation Algorithms

Assuming a database $\mathbf{D}$ consisting of a large number of users voting for all items and given unlimited computational and storage resources, a recommendation task could be solved by standard classification (binary ratings) or regression (numerical ratings) methods. In particular, given scores $\mathbf{r}_a$ of an active user on items from $\mathbf{I}_a$, the problem of predicting its score on item i could be solved by learning a nonlinear mapping $g_{\mathbf{I}a,i}$: $\mathsf{R}^{|\mathbf{I}_a|} \to \mathsf{R}$ from a database $\mathbf{D}$, where $g_{\mathbf{I}a,i}$ is a function that predicts the score on item i given scores on items from $\mathbf{I}_a$, and then using $g_{\mathbf{I}a,i}(\mathbf{r}_a)$ for prediction. An active user can provide ratings for any of 2^{I} possible subsets of items (including scenarios with $|\mathbf{I}_a| = 0$, $|\mathbf{I}_a| = \mathrm{I}$). Therefore, to be able to provide recommendations for any $\mathbf{I}_a$, it is sufficient to learn $\sum_{k=0}^{I} \binom{I}{k} (I - k)$ functions $g_{\mathbf{I}a,i}$, where $\mathbf{I}_a \in \mathbf{I}$ and $i \notin \mathbf{I}_a$. Note that if $i \in \mathbf{I}_a$, there is no need for prediction since $\mathrm{p}_{ai} = \mathrm{r}_{ai}$. We denote such a recommendation algorithm as NAIVE.

Real-life recommendation problems pose challenges that cannot be easily addressed by a NAIVE algorithm. First, learning $\sum_{k=0}^{I} \binom{I}{k} (I - k)$ functions can be computationally unfeasible even for small item sets. Second, assuming the fraction p of users rated any of items, the training set for learning the function $g_{\mathbf{I}a,i}$ would contain on the order of $\mathrm{U} \cdot \mathrm{p}^{|\mathbf{I}_a|+1}$ examples, where U is the number of users in the data set. Even for a large pool of 10^6 users and for a very dense ratings matrix with

$p = 0.1$, training sets with $|\mathbf{I}_a| > 5$ for learning $g_{\mathbf{I}a,i}$ would practically be empty. As a result, practical recommendation algorithms should approximate NAIVE by providing accurate predictions even for sparse rating matrices and should scale well with their size.

To address the sparseness problem, our regression-based approach to collaborative filtering uses a first-order approximation of nonlinear mappings $g_{\mathbf{I}a,i}$. The first-order approximation to predicting the score p_{ai} of active user a on item i based on its scores on $\mathbf{I}_a$ can be expressed as

$$p_{ai} = \sum_{j \in \mathbf{I}_a} w_{j,i} \cdot f_{j,i}(r_{aj}) \,, \tag{4}$$

where $f_{j,i}$ is a function that predicts the score on item i based on a vote on item j, and $w_{j,i}$ is the corresponding weight. So, assuming a method for choosing the weights $w_{j,i}$ is known, learning $I(I-1)$ one-dimensional predictors $f_{j,i}$, i, $j = 1, \ldots$ I, $i \neq j$, is sufficient for solving the approximated recommendation problem.

From a different standpoint, the function $f_{j,i}$ can be considered an expert for predicting the score on item i, given a score on item j. If an active user voted for items from $\mathbf{I}_a$, there are $|\mathbf{I}_a|$ available experts for predicting the ratings of each item from $\mathbf{I} \backslash \mathbf{I}_a$. The recommendation problem can now be approached as the identification of an optimal combination of experts. To make the solution computationally efficient, we model the experts $f_{j,i}$ as linear functions:

$$f_{j,i}(x) = x\alpha_{j,i} + \beta_{j,i} \,, \tag{5}$$

where $\alpha_{j,i}$ and $\beta_{j,i}$ are the only two parameters to be estimated for each expert. The two parameters could be estimated by using ordinary least squares as

$$\alpha_{j,i} = \frac{\sum\limits_{u \in \mathbf{U}_i \cap \mathbf{U}_j} (r_{ui} - r_{*i})(r_{uj} - r_{*j})}{\sum\limits_{u \in \mathbf{U}_i \cap \mathbf{U}_j} (r_{ui} - r_{*i})^2} \,, \qquad \beta_{j,i} = r_{*i} - \alpha_{j,i} r_{*j} \,. \tag{6}$$

Note that if the ratings satisfy a multivariate Gaussian distribution, (4) leads to the minimisation of the mean squared error when using the linear predictors $f_{j,i}$ obtained from (6) and when properly choosing the weights $w_{j,i}$ (Newbold and Granger 1974). (These weights are obtained from (9), as described below.)

3.1. Experts Integration

Once I(I−1) linear experts are learned, they should be combined to give recommendations. An integration method based on simple averaging and two statistically based integration methods are proposed in this section.

Averaging

In simple averaging, all $|\mathbf{I}_a|$ experts in (4) are given the same weight, $w_{j,i} = 1/\,|\mathbf{I}_a|$. Since some of the experts can be slightly better than the generic predictor, they can only deteriorate prediction by decreasing the contribution of good experts. Therefore, we propose simple averaging with thresholding where only good experts are retained.

In the AVERAGING(θ) algorithm, we reject all experts whose R-squared value defined as

$$R^2_{j,i} = 1 - \mathrm{MSE}_{j,i}/\sigma_i^2\,, \tag{7}$$

is below θ, where

$$\mathrm{MSE}_{j,i} = E_{\mathbf{U}_i \cap \mathbf{U}_j}\left\{\left(r_{uj} - f_{j,i}(r_{uj})\right)^2\right\} = \frac{1}{\left|\mathbf{U}_i \cap \mathbf{U}_j\right|} \sum_{u \in \mathbf{U}_i \cap \mathbf{U}_j} \left(r_{uj} - f_{j,i}(r_{uj})\right)^2 \tag{8}$$

is the mean squared error (MSE) of expert $f_{j,i}$, and where $\sigma_i^2 = \mathrm{Var}_{\mathbf{U}_i \cap \mathbf{U}_j}\{\mathrm{r}_{ui}\}$ is the variance of the scores for item i. Note that for large θ all $|\mathbf{I}_a|$ experts could be rejected, which would result in a low coverage of recommendations. In practice, when such a case occurs, a generic prediction could be generated to preserve the complete coverage. Since AVERAGING($\theta = 0$) averages the predictions of all available experts, it is comparable to the prediction approach of the Top-N item-based algorithm (Karypis 2001). Thus, it will be useful for making comparisons with the remaining algorithms proposed in this section.

Determining Optimal Weights

Back in 1969, Bates and Granger (Bates and Granger 1969) suggested that a linear combination of individual predictions can produce results that are superior to any of the individual predictions. For the problem of combining linear experts, the optimal solution can be derived by minimising the error variance of the linear combination of experts from (4). To predict the score on item i, $i \in \mathbf{I}\backslash\mathbf{I}_a$, the optimal weights $\mathrm{w}_{j,i}$ (the ones that minimise the MSE) can be found by using the $\mathrm{I} \times \mathrm{I}$ covariance matrix $\mathbf{C}_i$ of prediction errors with elements $\{\mathbf{C}_i\}_{j,k}$, j, $k = 1, \ldots \mathrm{I}$, defined as $\{\mathbf{C}_i\}_{j,k} = \mathrm{E}_{\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k}[\mathrm{e}_{j,i} \cdot \mathrm{e}_{k,i}]$, where $\mathrm{e}_{j,i}$ is the error of expert $f_{j,i}$, $\mathrm{e}_{j,i} = \{\mathrm{r}_{ui} - f_{j,i}(\mathrm{r}_{uj})\}$. Note that $\{\mathbf{C}_i\}_{j,j} = \mathrm{MSE}_{j,i}$.

To calculate the optimal weights that minimise the MSE of prediction on item i using $|\mathbf{I}_a|$ experts $f_{j,i}$, $j \in |\mathbf{I}_a|$, one should first select the $|\mathbf{I}_a| \times |\mathbf{I}_a|$ matrix $\mathbf{C}_i(\mathbf{I}_a)$ from elements of the covariance matrix $\mathbf{C}_i$ as $\{\mathbf{C}_i(\mathbf{I}_a)\}_{j,k} = \{\mathbf{C}_i\}_{\mathbf{I}_a(j),\mathbf{I}_a(k)}$, j, $k = 1, \ldots, |\mathbf{I}_a|$, where $\mathbf{I}_a(j)$ is the j-th element of the item set $\mathbf{I}_a$. Then, the optimal weights, such that $\sum_{j \in \mathbf{I}_a} w_{j,i} = 1$, can be calculated as (Newbold and Granger 1974)

$$\mathrm{w}_{\mathbf{I}_a(j),i} = \frac{\sum_{k=1}^{|\mathbf{I}_a|}\left\{\mathbf{C}_i(\mathbf{I}_a)^{-1}\right\}_{j,k}}{\sum_{j,k=1}^{|\mathbf{I}_a|}\left\{\mathbf{C}_i(\mathbf{I}_a)^{-1}\right\}_{j,k}}\,. \tag{9}$$

We call this optimal weighting procedure OPTIMAL_AVERAGING. Problems associated with applying (9) include estimating $\mathbf{C}_i$, calculating the inverse of $\mathbf{C}_i(\mathbf{I}_a)$ when experts are highly correlated, and computing this sufficiently fast. To estimate $\{\mathbf{C}_i\}_{j,k}$, a subset of users that voted for items i, j, and k should be found and the errors of $f_{j,i}$ and $f_{k,i}$ should be calculated on this subset. Since a training database could be very sparse, the number of such triples could be too small to properly estimate $\{\mathbf{C}_i\}_{j,k}$, even for seemingly large databases. As a consequence, an estimated matrix $\mathbf{C}_i$ could be ill-conditioned or even non-invertible.

The second problem, discussed in Merz and Pazzani (1999), is that an inverse of $\mathbf{C}_i(\mathbf{I}_a)$ could be unstable for highly correlated experts. Since individual experts could

Table 1. Influence on the optimal weights assignment of the error covariance $\{C_i\}_{1,2}$ between experts with error variances $\{C_i\}_{1,1} = 1$ and $\{C_i\}_{2,2} = 1.1$.

$\{C_i\}_{1,2}$	0	0.5	0.9	0.95	0.99	1
$w_{1,i}/w_{2,i}$	1.1	1.2	2.0	3.0	11.0	∞

often be slightly to moderately better than the generic predictor, they could be highly correlated. The third problem is computational cost, since the optimal approach requires the calculation of an inverse of $\mathbf{C}_i(\mathbf{I}_a)$, which generally takes $O(|\mathbf{I}_a|^3)$ time. Since $|\mathbf{I}_a| = O(I)$ and $|\mathbf{I}\backslash\mathbf{I}_a| = O(I)$, producing predictions for $|\mathbf{I}\backslash\mathbf{I}_a|$ items by an active user with OPTIMAL_AVERAGING would take $O(I^4)$ time. In the following, we propose a computationally efficient approximation of this approach that is robust to sparse data.

Determining Suboptimal Weights Efficiently

One of the main consequences of highly correlated experts is that, when applying (9), two almost identical experts could receive very different weights, the larger being assigned to the slightly more accurate one. To illustrate this, in Table 1 we show the ratio of weights obtained from (9) for two experts with error variances $\{C_i\}_{1,1} = 1$ and $\{C_i\}_{2,2} = 1.1$, when their error covariance $\{C_i\}_{1,2}$ is varied from 0 to 1. As seen, increasing $\{C_i\}_{1,2}$ results in assigning a much higher weight to a slightly better expert with error variance $\{C_i\}_{1,1}$. Similar effect could be obtained using (9) by decreasing both $\{C_i\}_{1,1}$ and $\{C_i\}_{2,2}$ by a constant $K < \{C_i\}_{1,1}$ and setting $\{C_i\}_{1,2} = 0$. It can be shown that for the scenario with only two experts, using a constant $K = \{C_i\}_{1,2}$ results in the optimal weights. We use this idea for obtaining suboptimal weights computationally fast.

The proposed weighting method is based on an estimate of the average error correlation between linear experts over the whole ratings matrix. If we assume that $f_{j,i}$ and $f_{k,i}$ are unbiased predictors for item i, the correlation $\rho^i_{j,k}$ between these predictors can be estimated from

$$\begin{aligned} &E_{\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k}\left[(f_{j,i}(r_{uj}) - f_{k,i}(r_{uk}))^2\right] = \\ &E_{\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k}\left[((f_{j,i}(r_{uj}) - r_{ui}) - (f_{k,i}(r_{uk}) - r_{ui}))^2\right] = \\ &\{\mathbf{C}_i\}_{j,j} - 2\{\mathbf{C}_i\}_{j,k} + \{\mathbf{C}_i\}_{k,k} \end{aligned} \tag{10}$$

as

$$\rho^i_{j,k} = \frac{\left\{\{\mathbf{C}_i\}_{j,j} + \{\mathbf{C}_i\}_{k,k} - E_{\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k}\left[(f_{j,i} - f_{k,i})^2\right]\right\}}{2\sqrt{\{\mathbf{C}_i\}_{j,j}\{\mathbf{C}_i\}_{k,k}}}. \tag{11}$$

Therefore, to calculate $\rho^i_{j,k}$, only users voting for items i, j, and k are considered. Due to the sparsity of the ratings matrix, the number of such users could be very small. To address this problem, we approximate (11) by using $E_{\mathbf{U}_j \cap \mathbf{U}_k}\left[(f_{j,i} - f_{k,i})^2\right]$ instead of $E_{\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k}\left[(f_{j,i} - f_{k,i})^2\right]$. It should be noted that such an approximation is valid only under the assumption that the data distribution is identical over $\mathbf{U}_i \cap \mathbf{U}_j \cap \mathbf{U}_k$ and $\mathbf{U}_j \cap \mathbf{U}_k$, which may not be true.

For an efficient expert weighting procedure, instead of estimating all I^3 covariances $\{\mathbf{C}_i\}_{j,k}$, i, j, $k = 1, \ldots I$, required by OPTIMAL_AVERAGING, we estimate

Table 2. Pseudocode for the ROBUST_AVERAGING algorithm.

```
TRAINING PHASE
Given ratings database D
  1. Construct I(I−1) experts $f_{j,i}$, $i, j = 1, \ldots I$, $i \neq j$, using (5) and (6) and calculate
     the corresponding $\mathrm{MSE}_{j,i}$ using (8)
  2. Estimate values $\rho^i_{j,k}$ for several randomly chosen triples $(i, j, k)$ using (11) and
     average them to obtain $\rho_{AV}$.
PREDICTION PHASE
For each active user with provided ratings $r_{aj}$, $j \in \mathbf{I}_a$:
  For each item $i \in \mathbf{I} \backslash \mathbf{I}_a$:
    1. Calculate suboptimal weights $w_{j,i}$, $j \in \mathbf{I}_a$, using (14)
    2. Predict rating $p_{ai}$ on item $i$ using (4)
```

values $\rho^i_{j,k}$ only for several randomly chosen triples (i, j, k) using (11) and average them to obtain ρ_{AV}. We then calculate a diagonal covariance matrix $\mathbf{C}^*_i$ defined as

$$\mathbf{C}^*_i = diag(\mathbf{C}_i) - \rho_{\mathrm{AV}} \cdot \min_j(\{\mathbf{C}_i\}_{j,j}) \cdot \mathbf{ID}, \tag{12}$$

where $\mathbf{ID}$ is an identity matrix and $diag(\mathbf{C}_i)$ is a diagonal matrix with diagonal elements $\{\mathbf{C}_i\}_{j,j}$, $j = 1, \ldots$ I. Therefore, the diagonal elements of $\mathbf{C}^*_i$ can be calculated as

$$\{\mathbf{C}^*_i\}_{j,j} = \mathrm{MSE}_{j,i} - \rho_{\mathrm{AV}} \cdot \min_k(\mathrm{MSE}_{k,i}), \tag{13}$$

where $\mathrm{MSE}_{j,i}$ is calculated from (8). The constructed diagonal matrix $\mathbf{C}^*_i$ is used instead of $\mathbf{C}_i$ to derive suboptimal expert weights from (9). Since $\mathbf{C}^*_i$ is a diagonal matrix, the corresponding weights are easily calculated as

$$w_{j,i} = \frac{1/\{\mathbf{C}^*_i\}_{j,j}}{\sum_{k \in \mathbf{I}_a} 1/\{\mathbf{C}^*_i\}_{k,k}}, \quad \text{for } j \in \mathbf{I}_a;\ w_{j,i} = 0, \text{ for } j \notin \mathbf{I}_a. \tag{14}$$

We denote the proposed algorithm as ROBUST_AVERAGING and provide its pseudocode in Table 2.

Note that the weights obtained by using (14) are guaranteed to be positive. Observe also that computing the weights $w_{j,i}$, $j \in \mathbf{I}_a$, requires just $O(|\mathbf{I}_a|) = O(I)$ time, and so giving predictions for all unseen items by an active user would require only $O(I^2)$ time.

More About the Complexity of Regression-Based Algorithms

In Table 3 we summarise the performance of different regression-based algorithms with respect to their speed (off-line and on-line) and robustness to data sparsity (effective data size). The off-line speed of NAIVE is clearly impractical in any realistic scenario, since it is exponential with the number of items: for each of $O(I2^I)$ functions $g_{\mathbf{I}_{a,i}}$, O(UI) time is required for learning. Moreover, the effective number of examples for learning $g_{\mathbf{I}_{a,i}}$ decreases exponentially with $|I_a|$. OPTIMAL_AVERAGING is inferior to AVERAGING and ROBUST_AVERAGING, both in terms of speed and

Table 3. Performance summary for different recommendation algorithms. The off-line speed relates to the model-building phase, the on-line speed corresponds to predicting the rating of an active user on a single item, and the effective data size measures the expected number of examples available during the model-building phase if a fraction p of users votes for each item.

Algorithm	Off-line speed	Effective data size*	On-line speed
NAIVE	$O(UI^2 \cdot 2^I)$	$U \cdot p^{\lvert \mathbf{I}_a \rvert + 1}$	$O(\lvert I_a \rvert)$
OPTIMAL_AVERAGING	$O(UI^3)$	$U \cdot p^3$	$O(\lvert I_a \rvert^3)$
ROBUST_AVERAGING AVERAGING	$O(UI^2)$	$U \cdot p^2$	$O(\lvert I_a \rvert)$
GENERIC	$O(UI)$	$U \cdot p$	$O(1)$

* For NAIVE it corresponds to the effective data size available for learning function $g_{\mathbf{I}_{a,i}}$

sparsity robustness. This is due to the need to examine each triple of items when estimating the error covariance matrices $\{\mathbf{C}_i\}$, $i = 1, \dots$ I, in the training phase, and to invert the corresponding covariance matrix in the prediction phase. The advantage of ROBUST_AVERAGING comes from the approximation made in (12). Clearly, the simple GENERIC algorithm is the fastest and most robust to rating sparsity. From these results one should be able to properly decide on the most appropriate recommendation algorithm with respect to given data and available computational resources.

The algorithms AVERAGING and ROBUST_AVERAGING require $O(|I_a|I)$ time for predicting all non-rated items of an active user, which is an advantage over the O(UI) time needed by the NEIGHBOUR algorithm, since $|I_a|$ is usually significantly smaller than the number of users. While the on-line speed of AVERAGING and ROBUST_AVERAGING is clearly superior, they require learning $I(I-1)$ linear models, where O(U) time is required for each. However, $O(UI^2)$ time could be acceptable since it is done off-line. Additionally, ROBUST_AVERAGING requires saving two linear parameters and an estimate of the error variance for each expert, which requires storing $3I(I-1)$ variables. For most applications this memory requirement compares favourably to memory requirement of neighbour-based algorithms which scales as O(UI).

3.2. Regression-Based Algorithm With Adjustment

One of the problems with subjective ratings is that two users with similar preferences can have significantly different average scores for the same subset of items. This notion has been used in ADJUSTED_GENERIC and NEIGHBOUR algorithms to directly adjust the predictions according to each active user's mean score. Here, to improve the prediction accuracy of the ROBUST_AVERAGING algorithm, we propose an adjustment similar to the one used in ADJUSTED_GENERIC. Let us examine two extreme cases to show that the difference Δr_a from (1) cannot be applied directly to ROBUST_AVERAGING. The first is the case of an extremely accurate expert $f_{j,i}$ that can predict the score for item i with $MSE_{j,i}$ close to 0 (in such a case, $\alpha_{j,i}$ is close to ± 1). This prediction does not need to be adjusted by Δr_a, since such a linear expert already includes this adjustment. The second is the case of poor experts that are generic predictors with $MSE_{j,i}$ close to σ_i^2 (in such a case, $\alpha_{j,i}$ is close to 0). In this case the adjustment is needed, and Δr_a should be added to the prediction. Any realistic combination of experts will be between these two extremes with $0 < \mathrm{MSE}_{j,i} < \sigma_i^2$. Therefore, some value smaller than $\Delta \mathrm{r}_a$ should be used for adjustment depending on the quality of individual experts.

In accordance with the previous discussion, we propose to predict a score of an active user for item i (last line of Table 2) as

$$\mathrm{p}_{a,i} = \Delta \mathrm{r}_a + \sum_{j \in \mathbf{I}_a} \mathrm{w}_{j,i} \cdot f_{j,i}(\mathrm{r}_{aj} - \Delta \mathrm{r}_a)\,. \tag{15}$$

Therefore, $\Delta \mathrm{r}_a$ is first subtracted from each active user's vote, such modified ratings are used for prediction and, finally, predictions are adjusted by adding $\Delta \mathrm{r}_a$. Since $f_{j,i}$ is a linear function, note that the effective level of adjustment in (15) is $\Delta \mathrm{r}_a(1 - \sum \mathrm{w}_{j,i}\alpha_{j,i})$. This expression is non-negative since $\sum \mathrm{w}_{j,i} = 1$, $\alpha_{j,i} < 1$. We denote this algorithm as ADJ_ROBUST_AVERAGING.

3.3. Pessimistic Adjustment for Active Users with Small Item Sets

The practical difficulty when applying ADJUSTED_GENERIC and ADJ_ROBUST_AVERAGING occurs for small $|\mathbf{I}_a|$. Namely, if an active user votes just for a few items, the statistical significance of an estimate $\Delta \mathrm{r}_a$ will be low. In this case it would be advisable to use a pessimistic estimate $\delta \mathrm{r}_a$ instead of $\Delta \mathrm{r}_a$ from (1) so that this value will tend to zero if the number of scores is low while it will tend to r_a if the number of scores is high.

Given the ratings r_{ai} of an active user on $i \in \mathbf{I}_a$ and the corresponding generic ratings r_{*i}, the problem of estimating $\Delta \mathrm{r}_a$ can be approached as a regression on a data-generating process:

$$\mathrm{r}_{ai} = \Delta \mathrm{r}_a + \mathrm{r}_{*i} + \varepsilon_i, \qquad \boldsymbol{\varepsilon} \sim \mathrm{N}(0, \Sigma), \tag{16}$$

where $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2, \ldots \varepsilon_{|\mathbf{I}_a|}]^{\mathrm{T}}$ is a noise term with a Gaussian distribution having a covariance matrix Σ of size $|\mathbf{I}_a| \times |\mathbf{I}_a|$ with elements $\mathrm{Cov}_{\mathrm{U}_i \cap \mathrm{U}_j}(\mathrm{r}_{ui}, \mathrm{r}_{uj})$. Note that calculating Σ involves ratings, while calculating $\mathbf{C}_i$ from (9) involves prediction errors. This data-generating process is validated by the assumption that the score of an active user on item i could be approximated by a generic prediction incremented by an adjustment term and an error term that represents the variance of scores of previous users on item i. The covariance matrix Σ represents the fact that scores on different items can have different levels of variance and that scores on different items can be correlated. For example, if an active user rates highly correlated items, the confidence in adjustment $\Delta \mathrm{r}_a$ is smaller than if it rates uncorrelated items. Also, if an active user votes for two items with different rating variances, a higher weight in calculation of $\Delta \mathrm{r}_a$ should be given to a rating with a lower variance.

Adjustment $\Delta \mathrm{r}_a$ from (16) and its variance can be estimated by the generalised least squares algorithm (Griffiths et al. 1993) as

$$\Delta \mathrm{r}_a = \frac{\mathbf{1}'\Sigma^{-1}(\mathbf{r}_a - \mathbf{r}_*)}{\mathbf{1}'\Sigma^{-1}\mathbf{1}}\,, \qquad Var(\Delta \mathrm{r}_a) = \frac{1}{\mathbf{1}'\Sigma^{-1}\mathbf{1}}\,, \tag{17}$$

where $\mathbf{1}$ is a vector of ones, $\mathbf{r}_a$ is a vector with elements $\{\mathrm{r}_{ai}, i \in \mathbf{I}_a\}$, and $\mathbf{r}_*$ is a vector with elements $\{\mathrm{r}_{*i}, i \in \mathbf{I}_a\}$. From (17), a pessimistic adjustment $\delta \mathrm{r}_a$ is calculated as

$$\delta \mathrm{r}_a = \Delta \mathrm{r}_a - sign(\Delta \mathrm{r}_a) \cdot t_{\gamma, |\mathbf{I}_a|-1} \cdot \sqrt{Var(\Delta \mathrm{r}_a)}\,, \qquad \gamma \in [0.5, 1]\,, \tag{18}$$

where $t_{\gamma,\mathrm{n}}$ is a t-statistic with confidence γ and with n degrees of freedom. It should be noted that the confidence parameter γ determines the difference between $\delta \mathrm{r}_a$ and $\Delta \mathrm{r}_a$. At the extremes, if $\gamma = 0.5$ then $\delta \mathrm{r}_a = \Delta \mathrm{r}_a$, while if $\gamma = 1$ then $\delta \mathrm{r}_a = 0$. Note also that if covariance matrix Σ is diagonal with constant elements σ, $\mathrm{Var}(\Delta \mathrm{r}_a) = \sigma^2 / |\mathbf{I}_a|$.

Since a calculation of $\mathrm{Var}(\Delta \mathrm{r}_a)$ involves inverting the matrix Σ with $O(|\mathbf{I}_a|^3)$ time complexity, in the performed experiments we simplified the calculation of $\mathrm{Var}(\Delta \mathrm{r}_a)$ by using a matrix Σ with non-diagonal elements set to zero. We denote the algorithms that use the pessimistic adjustment $\delta \mathrm{r}_a$ with confidence γ instead of r_a as ADJUSTED_GENERIC (γ) and ADJ_ROBUST_AVERAGING (γ).

4. Experimental Evaluation

4.1. Data Sets

Eachmovie Data

The EachMovie data set (McJones 1997) is a publicly available collaborative filtering benchmark database collected during an 18-month period between 1996–1997. It contains ratings from 72 916 users on 1 628 movies with a total of 2 456 676 ratings. Therefore, the matrix is very sparse with only 2.07% rated elements. User ratings were collected on a numeric 6-point scale between 0 and 1, but to make the results more comprehensible we rescaled ratings to integers {0, 1, ..., 5}.

Jester Data

The Jester data set (Goldberg et al. 2001) is a collection of ratings from an online joke recommendation system (http://shadow.ieor.berkeley.edu/humor). The data contain anonymous ratings from 21 800 users on a subset of 100 jokes organised in a matrix of dimensions $21\,800 \times 100$, with the ratings ranging from -10 to $+10$. We scaled all ratings to a range [0, 1] in the experiments. The data set is very dense, with an average of 46 ratings per user, where all users rated 10 jokes from the so-called gauge set $\mathbf{I}_{\mathrm{gauge}} = \{5, 7, 8, 13, 15, 16, 17, 18, 19, 20\}$.

Synthetic Data

We also generated two synthetic data sets to characterise proposed recommendation algorithms in a controlled setting. Both data sets were generated as random samples drawn from a multivariate Gaussian distribution with zero mean, variance one, and a covariance matrix $\mathbf{P}$ resembling that of the Eachmovie data. The matrix $\mathbf{P}$ was generated using the fact that $\mathbf{P} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$ is positive definite (and therefore a valid covariance matrix) for an arbitrarily chosen $\mathbf{A}$. By properly choosing $\mathbf{A}$, the matrix $\mathbf{P}$ was generated to resemble the desired second-order statistics.

4.2. Evaluation Protocols

In the first set of experiments on the Eachmovie and Jester data, we randomly selected five scores from each active user for testing, while the rest were used as the set of training scores $\mathbf{I}_a$. We call this protocol *AllBut5*. In accordance with Breese et al. (1998) we performed another set of experiments allowing a smaller number of active user scores. Here, we randomly selected 2, 5, or 10 votes from each active user as the observed scores, and then predicted the remaining scores. The three protocols were named *Given2*, *Given5*, and *Given10*. Such a small number of scores given by an active user is more likely to be an environment for realistic recommendation systems. Finally, Jester and Synthetic data have the feature that all users rated a constant subset of items. By *GaugeN* we denote a protocol in which a given subset of N items is reserved for training, while the remaining items are reserved for prediction.

4.3. Experiments on Eachmovie Data

To perform a number of experiments, and to allow for a fair comparison between different algorithms, we used only a subset of the whole database. In our experiments the first 10 000 users represented the training set, and the following 10 000 users represented a set of active users. From both sets all users with less than 20 scores were filtered out as well as all the movies receiving less than 50 scores in the training set. This resulted in 3422 users in the training set and 4790 active users with 503 retained movies. The reduction of the number of users allowed a fairly fast evaluation of the benchmark neighbour-based system, while a lower number of movies allowed relatively fast learning of 503×502 linear experts needed for the regression-based algorithms (trained in 8 hours on a 700 MHz NT-based computer with 256 MB memory). Finally, to decrease the influence of unreliable experts, we replaced each expert $f_{j,i}$ trained using less than 30 examples (with $|\mathbf{U}_i \cap \mathbf{U}_j| < 30$) with a generic predictor for item i.

The algorithms considered in evaluations included MEAN, GENERIC, ADJ_GENERIC, NEIGHBOUR, AVERAGING, ROBUST_AVERAGING, and ROBUST_AVERAGING. Due to data sparsity and based on Table 3, NAIVE and OPTIMAL_AVERAGING were not considered. Out of several considered choices of parameters K and N for NEIGHBOUR, we report on the best performing neighbour-based algorithm achieved with $\mathrm{K} = 80$ and $\mathrm{N} = 50$.

In Table 4 we report the results for different recommendation algorithms with the *AllBut5* protocol. As can be seen, NEIGHBOUR had almost complete coverage, but with an accuracy just slightly better than the ADJUSTED_GENERIC algorithm. AVERAGING algorithms showed large sensitivity to the threshold, with $\theta = 0.06$ offering the best compromise between accuracy and coverage.

The accuracy for $\theta = 0.10$ was highest, but with a coverage of only 84.5 %, while it was lowest for $\theta = 0$, which corresponds to the Top-N item-based algorithms (Karypis 2001).

For the ROBUST_AVERAGING algorithm, it was determined that the average correlation between experts was slightly larger than 0.95. This value was used as the default value for (12). For illustration, we also report the results for $\rho = 0.9$ and $\rho = 0.99$. It can be seen that the values $\rho = 0.95$ and 0.99 gave similar results, while $\rho = 0.9$ was slightly inferior. Since the coverage of ROBUST_AVERAGING was 100 %, it can be concluded that, overall, it was slightly more successful than the ADJUSTED_GENERIC, NEIGHBOUR, and AVERAGING algorithms. ADJ_ROBUST_AVERAGING was clearly superior to the other algorithms, indicating that the introduced adjustment can significantly boost the prediction accuracy. It should be noted

Table 4. Performance of different algorithms with the *AllBut5* protocol on the Eachmovie data.

Algorithm	MAE	ROC(4,3.5) accuracy	Coverage [%]	Elapsed time [s]
MEAN	1.054	–	–	–
GENERIC	0.925	0.652	100	1.1
ADJUSTED_GENERIC	0.853	0.697	100	2.9
NEIGHBOUR(K = 80, N = 50)	0.848	0.698	99.9	38,056
AVERAGING($\theta = 0$)	0.883	0.681	100	
AVERAGING($\theta = 0.06$)	0.850	0.702	98.1	68
AVERAGING($\theta = 0.1$)	0.842	0.706	84.5	
ROBUST_AVERAGING($\rho = 0.9$)	0.843	0.705	100	
ROBUST_AVERAGING($\rho = 0.95$)	0.841	0.707	100	70
ROBUST_AVERAGING($\rho = 0.99$)	0.841	0.707	100	
ADJ_ROBUST_AVERAGING($\rho = 0.9$)	0.814	0.718	100	
ADJ_ROBUST_AVERAGING($\rho = 0.95$)	0.812	0.720	100	73
ADJ_ROBUST_AVERAGING($\rho = 0.99$)	0.812	0.720	100	

that a similar adjustment can be incorporated to the AVERAGING algorithm and that similar accuracy improvements can be expected.

In Table 4 we also used ROC($\theta_1 = 4$, $\theta_2 = 3.5$) to report the classification accuracy. There, each item with a score of 4 or 5 was regarded as a "good" item, and each prediction above 3.5 was regarded as a recommendation. ROC($\theta_1 = 4$, $\theta_2 = 3.5$) is therefore the accuracy of correctly predicting a "good" movie based on the given threshold of 3.5. From Table 4 it can be concluded that the MAE and ROC(4, 3.5) performance measures are almost identical with respect to comparing different algorithms. Therefore, for the remaining analysis we report only the MAE accuracy.

As explained in the previous two sections, regression-based algorithms have superior on-line speed to neighbour-based algorithms. Our implementation of these algorithms in Matlab on a 700 MHz NT-based computer with 256 MB memory showed that when performing 5×4790 predictions, ADJ_ROBUST_AVERAGING was 520 times faster than NEIGHBOUR, and ADJ_ROBUST_AVERAGING was 25 times slower than ADJUSTED_GENERIC (Table 4). Although we do not claim our implementation is optimal, these results validate the analysis of on-line speed of neighbour-based and the proposed regression-based algorithms from Sect. 3.4. Similar performance results were obtained in the remaining experiments on the Eachmovie and Jester data.

In Table 5 we show the performance of the MAE of the MEAN, GENERIC, ADJUSTED_GENERIC, NEIGHBOUR, and ADJ_ROBUST_AVERAGING algorithms for four categories of users depending on the number of provided ratings $|\mathbf{I}_a|$ for the *AllBut5* protocol. As can be seen, while NEIGHBOUR was very sensitive to $|\mathbf{I}_a|$, having the lowest accuracy for the active users giving less than 25 votes, ADJ_ROBUST_AVERAGING was just moderately sensitive, indicating that it can be successfully used even for users providing a relatively small number of votes.

To examine further the robustness of different algorithms to the realistic scenario of small $|\mathbf{I}_a|$, a comparison of four different protocols is reported in Table 6, where the *AllBut5* results are taken from Table 4. As can be seen, the regression-based approach was very robust to the number of scores given by an active user, comparing favourably to the NEIGHBOUR algorithm, whose accuracy decreased significantly

Table 5. MAE as a function of the number of votes of an active user for the *AllBut5* protocol on the Eachmovie data.

Algorithm	**Number of Votes**			
	15–25	**25–50**	**50–100**	**>100**
MEAN	1.056	1.048	1.043	1.089
GENERIC	0.922	0.910	0.929	0.954
ADJUSTED_GENERIC	0.866	0.848	0.851	0.853
NEIGHBOUR(K = 80, N = 50)	0.881	0.844	0.842	0.856
ROBUST_AVERAGING($\rho = 0.95$)	0.852	0.831	0.841	0.849
ADJ_ROBUST_AVERAGING($\rho = 0.95$)	**0.827**	**0.807**	**0.810**	**0.807**

Table 6. MAE of different protocols on the Eachmovie data. The parameter γ^* represents the choice that, for a given protocol, resulted in the highest recommendation accuracy.

Algorithm	**AllBut5**	**Given10**	**Given5**	**Given2**
MEAN	1.054	1.066	1.066	1.067
GENERIC	0.925	0.935	0.935	0.937
ADJUSTED_GENERIC	0.853	0.883	0.925	1.023
ADJUSTED_GENERIC(γ^*)	0.853	0.877	0.895	**0.928**
γ^*	0.6	0.7	0.8	0.9
NEIGHBOUR(K = 80, N = 50)	0.848	0.912	0.991	1.195
ROBUST_AVERAGING($\rho = 0.95$)	0.842	0.882	0.896	**0.930**
ADJ_ROBUST_AVERAGING($\rho = 0.95$)	**0.812**	0.868	0.917	1.043
ADJ_ROBUST_AVERAGING($\rho = 0.95$, γ^*)	**0.812**	**0.860**	**0.883**	**0.930**
γ^*	0.6	0.7	0.8	0.9

when the number of votes by an active user was less than 10. However, it is evident that the performances of ADJUSTED_GENERIC and ADJ_ROBUST_AVERAGING without the pessimistic adjustment (Sect. 3.3) also deteriorated for the *Given5* and *Given2* protocols. This was to be expected since estimates of adjustment Δr_a using only two or five scores have extremely low confidence. Applying the pessimistic adjustment allowed a significant accuracy improvement for the *Given5* and *Given2* protocols. For each protocol, we show the accuracy achieved with the best choice of γ confidence levels within an interval [0.5, 1]. As can be seen, the γ coefficient could be optimised with respect to the number of available votes; as the number of ratings given by an active user increased, the optimal value of γ coefficient decreased.

To characterise further the ADJ_ROBUST_AVERAGING algorithm, we compared its MAE with the MAE of the GENERIC algorithm for each of the 503 examined movies with the *AllBut5* protocol. The MAE of GENERIC is a measure of users' disagreement in rating a given movie. In the Eachmovie data it ranges from 0.6 to 1.4. By using linear regression, we obtained the relationship

$$\text{MAE}(\text{ADJ\_ROBUST\_AVERAGING}) = 0.10 + 0.76 \cdot \text{MAE}(\text{GENERIC}), \qquad (19)$$

indicating the benefits of ADJ_ROBUST_AVERAGING for the recommendation of movies for which there is a large disagreement among users. For example, from (19) it can be seen that for MAE(GENERIC) = 0.6, ADJ_ROBUST_AVERAGING is about 7 % more accurate, while for controversial movies with MAE(GENERIC) = 1.4,

Table 7. MAE for the *Gauge10*, *AllBut5*, *Given10*, *Given5*, and *Given2* protocols in the Jester data.

Algorithm	**Gauge10**	**AllBut5**	**Given10**	**Given5**	**Given2**
MEAN	0.217	0.222	0.219	0.219	0.219
GENERIC	0.203	0.208	0.205	0.205	0.205
ADJUSTED_GENERIC($\gamma = 0.7$)	**0.189**	0.179	0.180	0.185	0.195
NEIGHBOUR(K = 80, N = 50)	**0.188**	0.179	0.186	0.201	0.218
ROBUST_AVERAGING($\rho = 0.95$)	0.194	0.195	0.191	0.192	**0.194**
ADJ_ROBUST_AVERAGING($\rho = 0.95$, $\gamma = 0.7$)	**0.188**	**0.178**	**0.177**	**0.182**	0.195
OLS	**0.186**	–	–	–	–
Neural Networks	**0.186**	–	–	–	–

ADJ_ROBUST_AVERAGING is about 17 % more accurate then the generic recommendation.

4.4. Experiments on Jester Data

Similar to experiments on the Eachmovie data, we reserved 5000 randomly chosen users for the training set and another 5000 users for the set of active users. There was no need to prune either jokes or users. The fact that all users voted for all gauge jokes $\mathbf{I}_{\text{gauge}}$ allowed direct use of regression algorithms to learn 90 predictors of scores for non-gauge jokes. Therefore, for the *Gauge10* protocol, in addition to recommendation algorithms used in the Eachmovie experiments, we were able to use Ordinary Least Squares (OLS) and Neural Networks (NN) as predictors, and thus estimate the upper bound on the recommendation accuracy achievable with the Jester data. 90 neural networks used in the experiments had 10 inputs, 5 hidden nodes and 1 output, and were trained using resilient backpropagation (Riedmiller and Braun 1993). For ADJUSTED_GENERIC and ADJ_ROBUST_AVERAGING, we used pessimistic adjustment with a confidence level of $\gamma = 0.7$, since this value appeared to be the best overall choice in the Eachmovie experiments.

In Table 7 we first show accuracy results of different algorithms with the *Gauge10* protocol. The first observation is that the margin between the MEAN predictor and the best available predictor was similar to the difference observed in the Eachmovie experiments. However, the difference between a powerful neural network predictor and ADJUSTED_GENERIC was almost negligible. ADJ_ROBUST_AVERAGING, the superior recommendation algorithm in the Eachmovie experiments, and NEIGHBOUR also had comparable performances with MAE = 0.188. Such surprising results could be explained by the fact that the projection of gauge ratings on their two largest principal directions does not exhibit clustering structure and that the correlation between gauge ratings is extremely small. This property indicates the lack of grouping of user tastes into more distinguishing groups based on the gauge set of jokes that is an underlying assumption for collaborative filtering.

We performed additional experiments with the *Given2*, *Given5*, *Given10*, and *AllBut5* protocols, and we also present the results in Table 7. For all protocols except *Given2*, ADJ_ROBUST_AVERAGING achieved results better than ROBUST_AVERAGING, NEIGHBOUR, and ADJUSTED_GENERIC. An interesting result is that the MAE was significantly lower for the *Given10* than for the *Gauge10* protocol. This indicates that a better choice for gauge jokes could be possible in order to improve the recommendation accuracy of the proposed regression-based algorithms on the Jester data.

Table 8. Recommendation errors of different algorithms with the *Gauge2*, *Gauge5*, *Gauge10* protocols on the D_1 and D_2 data. The first number in each cell is the MSE, while the number in parentheses is the MAE.

Algorithm	**D_1**			**D_2**		
	Gauge2	*Gauge5*	*Gauge10*	*Gauge2*	*Gauge5*	*Gauge10*
GENERIC	1.00 (0.80)	1.00 (0.80)	1.00 (0.80)	1.00 (0.80)	1.00 (0.80)	1.00 (0.80)
ADJUSTED_GENERIC(γ = 0.7)	1.12 (0.84)	1.08 (0.83)	1.04 (0.81)	1.00 (0.80)	0.94 (0.77)	0.89 (0.76)
AVERAGING(θ = 0)	0.93 (0.77)	0.92 (0.77)	0.91 (0.76)	0.93 (0.77)	0.91 (0.76)	0.90 (0.76)
ROBUST_AVERAGING(ρ = 0.95)	0.92 (0.76)	0.86 (0.74)	0.83 (0.72)	0.92 (0.76)	0.87 (0.74)	0.85 (0.73)
OPTIMAL_AVERAGING	0.92 (0.76)	0.78 (0.71)	0.57 (0.58)	0.92 (0.76)	0.83 (0.72)	0.70 (0.67)
OLS	0.90 (0.76)	0.75 (0.69)	0.52 (0.55)	0.91 (0.76)	0.81 (0.71)	0.67 (0.65)

4.5. Characterisation of Regression-Based Algorithms on Synthetic Data

While experiments on the real-life Eachmovie and Jester data provided very useful insight into the properties of different recommendation algorithms, we performed additional experiments on synthetic data to provide a more detailed characterisation in a controlled setting.

From the Eachmovie data, it was determined that $E[\rho_E] = 0.17$ and $Std[\rho_E] = 0.16$, where ρ_E denotes a correlation between movie ratings in the Eachmovie data. For the first synthetic data set D_1, we generated 10 000 training (TR1) and 10 000 test (TS1) examples from a 20-dimensional Gaussian distribution with a mean of zero, variance of one, and covariance matrix $\mathbf{P}_1$ such that $E[\rho_{D1}] = 0.01$ and $Std[\rho_{D1}] = 0.21$. Matrix $\mathbf{P}_1$ was generated as $\mathbf{P}_1 = \mathbf{A}_1{}^T\mathbf{A}_1$, where $\mathbf{A}_1$ was a 20×20 matrix with elements randomly taken from a Gaussian distribution with a mean of zero and variance of one.

Therefore, D_1 contained approximately the same number of positively and negatively correlated variables, while the variability of correlations was slightly larger than in the Eachmovie data. For the second synthetic data set D_2, we generated 10 000 training (TR2) and 10 000 test (TS2) examples from a 40-dimensional Gaussian distribution with a mean of zero, variance of one, and covariance matrix $\mathbf{P}_2$ such that $E[\rho_{D2}] = 0.15$ and $Std[\rho_{D2}] = 0.16$. Matrix $\mathbf{P}_2$ was generated as $\mathbf{P}_2 = \mathbf{A}_2{}^T\mathbf{A}_2$, where $\mathbf{A}_2$ was a 40×40 matrix with elements randomly taken from a Gaussian distribution with a mean of 0.45 and variance of one. Therefore, D_2 highly resembled the Eachmovie data with respect to their second-order statistics.

Repeating the above procedure 100 times, we randomly generated 100 different data sets D_1 and D_2. We experimented with *GaugeN* protocols, with $N = 2$, 5, and 10, such that from each generated set we reserved the first N columns as gauge items while the column $N + 1$ was to be predicted.

Experiments on Data Without Missing Values

In the first set of experiments (Table 8), we examined the GENERIC, ADJUSTED_ GENERIC(γ = 0.7), AVERAGING($\theta = 0$), ROBUST_AVERAGING(ρ = 0.95), OPTIMAL_AVERAGING, and Ordinary Least Squares (OLS) algorithms with *Gauge2*, *Gauge5*, and *Gauge10* protocols on 100 different D_1 and D_2 data sets. We report on the MAE and MSE that were averaged over the 100 experiments. Note that the training data in both D_1 and D_2 were without missing values. This allowed the use of OLS predictors to obtain the upper bound on the accuracy, since OLS is the best unbiased predictor on the generated multidimensional Gaussian data. From Table 8, in the following we summarise the main findings.

The accuracy of ROBUST_AVERAGING increased with the number of ratings $|\mathbf{I}_a|$ for both D_1 and D_2 data. While its accuracy was very close to OLS and OPTIMAL_AVERAGING for $|\mathbf{I}_a| = 2$, the improvement in accuracy with the increase in $|\mathbf{I}_a|$ ($|\mathbf{I}_a| = 5$, 10) was significantly smaller than for OLS and OPTIMAL_AVERAGING. Therefore, if ratings data are sufficiently dense and computational resources allow, OLS or OPTIMAL_AVERAGING should be preferred over ROBUST_AVERAGING.

The accuracy of OPTIMAL_AVERAGING was comparable to that of OLS in all experiments. Since using OLS for recommendation assumes building $O(2^I)$ predictors, OPTIMAL_AVERAGING should be preferred over OLS. The accuracy of AVERAGING($\theta = 0$) was poorer than that of ROBUST_AVERAGING in all experiments. Therefore, while using the same set of experts, the robust approximation in (12) is superior to simple expert averaging. This is an indication that the Top-N algorithm (Karypis 2001) could be improved by introducing similar weighted averaging.

In our experiments, ADJUSTED_GENERIC was more accurate than GENERIC on D_2, and was comparable in accuracy to ROBUST_AVERAGING. This can be attributed to the fact that most correlations between ratings in D_2 are positive. Consequently, the accuracy of ADJUSTED_GENERIC on D_1 was worse than that of GENERIC. Therefore, if ratings are mostly positively correlated, ADJUSTED_GENERIC could be considered as a rapid and moderately accurate recommendation algorithm.

The MSE measure indicates much larger difference in accuracy between different algorithms than the MAE measure. This result could be another explanation for the seemingly small difference in accuracy between different algorithms obtained on the Eachmovie and Jester data. We adopted the MAE accuracy since it is more commonly used in the literature for the evaluation of collaborative filtering algorithms.

Experiments on Data With Missing Values

The main property of data in collaborative filtering is a large proportion of missing ratings, and any evaluation of algorithms needs to consider the problem of sparsity robustness. We examined GENERIC, ROBUST_AVERAGING($\rho = 0.95$), OPTIMAL_AVERAGING, and OLS for the *Gauge5* protocol with respect to their robustness to missing data. To perform the experiments, the D_1 data was used to produce ratings data with missing values, where only a fraction p of randomly chosen ratings was retained for each item from D_1.

In Table 9 we present the dependence of the algorithm accuracy on the fraction p of available ratings in the training set. The results were consistent with the performance analysis provided with Table 3. As can be seen, depending on the level of rating sparsity, different algorithms should be preferred; as the sparsity increases the most appropriate algorithms follow the sequence OLS, OPTIMAL_AVERAGING, ROBUST_AVERAGING($\rho = 0.95$), and, finally, GENERIC. The proposed regression-based algorithms, therefore, can be considered appropriate for a relatively large range of rating sparsity scenarios.

Table 9. Recommendation accuracy of different algorithms with the *Gauge5* protocol on the D_1 data with missing ratings. Only the MAE measure is reported. Empty entries correspond to the inability to provide predictions caused by data sparsity (see Table 2).

Algorithm	**Fraction of ratings in training data, p**					
	1	0.5	0.2	0.1	0.05	0.01
GENERIC	0.80	0.80	0.80	0.80	0.80	**0.80**
ROBUST_AVERAGING($\rho = 0.95$)	0.74	0.74	0.74	**0.74**	**0.75**	–
OPTIMAL_AVERAGING	0.71	**0.71**	**0.72**	1.00	–	–
OLS	**0.69**	**0.70**	–	–	–	–

While OLS is the most accurate for dense data with $p = \{0.5, 1\}$, the fact that the corresponding training data can be chosen only from users that voted for all gauge items and the target item makes it unusable for a higher level of sparsity. Moreover, even if OLS predictors were available, they could be used only with active users that voted for all gauge items. This happens quite infrequently in realistic scenarios with large item sets. OPTIMAL_AVERAGING showed a somewhat smaller sensitivity to sparseness (able to work at $p = 0.2$) while achieving comparable accuracy to OLS. Since ROBUST_AVERAGING is trained using only pairs of items, it is very robust to sparseness (worked even for $p = 0.05$) with significant advantages in accuracy over the GENERIC predictor. Finally, if data sparsity is extremely high ($p = 0.01$), only generic predictions can be provided. It is worth noting that the results in Table 9 were obtained on data sets with 10 000 users, and that Table 3 should be used as a guide in choosing the appropriate recommendation algorithm for a given application.

5. Discussion

Automated collaborative filtering is a key technique for providing customisation of E-commerce sites. Various neighbour-based recommendation methods are popular choices for collaborative filtering. However, their latency can be a serious drawback for scaling up to a large number of requests that must be processed in real-time. In this paper we propose a regression-based approach to collaborative filtering that searches for similarities between items, builds a collection of experts, and combines them in an appropriate way to give predictions for a new user. We examined a number of procedures varying in speed, accuracy, and sparsity robustness.

Experiments on two benchmark databases (Eachmovie and Jester data) and on synthetic data suggest that ADJ_ROBUST_AVERAGING provides the maximal coverage and comparable to significantly better accuracy, while it is several orders of magnitude faster than the neighbour-based alternative. Furthermore, while the accuracy of neighbour-based algorithms is very sensitive to the number of votes given by an active user, our approach is more accurate than the generic predictor even when this number is very small (two to five votes). However, it is worth noting that a fairly simple ADJUSTED_GENERIC algorithm appeared to be very robust and difficult to outperform by both neighbour and regression-based algorithms despite its simplicity.

The choice of recommendation algorithm should depend on the size and sparsity of the data, on the number of votes by an active user, and on the speed and memory requirements. We showed that the proposed regression-based algorithms seem to be an appropriate choice for a large range of recommendation scenarios satisfying $Up^2 > 1$, where U is the number of previous users, and p is the fraction of ratings available for a given item.

To further improve the on-line speed and memory requirements of the regression-based algorithms, deleting experts with poor predicting capabilities can be an acceptable alternative. If there are no available experts to predict on a given item, the maximum coverage could be saved by using simpler models such as ADJUSTED_GENERIC. With such an approach, the accuracy would not significantly deteriorate, since deleted experts are not much better than the mean predictor. The error variance of each expert estimated as part of the regression-based algorithms could be used for guided on-line recommendation systems in which, based on the previous votes, an active user is asked to vote on the items that would maximally decrease the overall prediction error. Deriving an optimal procedure for guided voting is the topic of our future research.

In an attempt to characterise and evaluate the regression-based algorithms, we compared them with the state-of-the-art neighbour-based recommendation algorithm. In some cases we were also able to make comparisons with standard machine-learning algorithms for regression – ordinary least squares and neural networks – which served for establishing upper bounds on the achievable recommendation accuracy. In addition, we proposed three simple algorithms that served as benchmarks for establishing a lower bound on the recommendation accuracy. A comprehensive evaluation of the regression-based approach with an even larger number of proposed recommendation systems would, without doubt, be highly desirable. However, in practice, an objective comparison is difficult to achieve due to proprietary limitations of collaborative filtering software and the large diversity of specific problems that were addressed. A serious effort toward a comparative evaluation of a larger number of existing recommendation systems would be extremely important for further advances in the area of collaborative filtering.

Acknowledgements. We thank Steve Glassman from Compaq SRC for providing us with access to the Eachmovie data set. We also thank Ken Goldberg from UC Berkeley for sharing with us his Jester data set. Suggestions from anonymous reviewers that resulted in significantly improved presentation are greatly appreciated.

References

Aggrawal CC, Wolf JL, Wu K, Yu PS (1999) Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In: Proceedings, ACM Knowledge Discovery in Databases Conference, pp 201–212

Bates JM, Granger CWJ (1969) The combination of forecasts. Oper Res Q 20:451–468

Billsus D, Pazzani MJ (1998) Learning collaborative information filters. In: Proceedings, Fifteenth International Conference on Machine Learning, pp 46–54

Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pp 43–52

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

Delgado J, Ishii N, Ura T (1998) Content-based collaborative information filtering: actively learning to classify and recommend documents. In: Klush M, Weiss G (eds) Cooperative Agents II, Proceedings/CIA'98, LNAI Series Vol 1435. Springer-Verlag, Heidelberg, pp 206–215

Freund Y, Iyer R, Schapire R, Singer Y (1998) An efficient boosting algorithm for combining preferences. In: Shavlik J (ed) Proceedings of the Fifteenth International Conference in Machine Learning, pp 170–178

Goldberg K, Roeder T, Gupta D, Perkins, K (2001) Eigentaste: a constant-time collaborative filtering algorithm. Inf Retrieval 4(2):133–151

Greening D (1997) Building customer trust with accurate product recommendations. LikeMinds White Paper LMWSWP-210-6966

Griffiths WE, Hill RC, Judge GG (1993) Learning and Practicing Econometrics. John Wiley & Sons

Herlocker JL, Konstan JA, Borchers A, Riedl J (1999) An algorithmic framework for performing collaborative filtering. In: Proceedings, 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp 230–237

Hoffman T, Puzicha J (1999) Latent class models for collaborative filtering. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence

Karypis G (2001) Evaluation of item-based Top-N recommendation algorithms. In: Proceedings of the 10th Conference of Information and Knowledge Management

Konstan JA, Miller BN, Maltz D, Herlocker JL, Gordon LR, Riedl J (1997) GroupLens: applying collaborative filtering to Usenet news. Commun ACM 40(3):77–87

Lin WY, Alvarez SA, Ruiz C (2002) Efficient adaptive-support association rule mining for recommender systems. Data Min Knowl Discovery 6(1):83–105

Linden GD, Jacobi JA, Benson EA (2001) Collaborative recommendations using item-to-item similarity mappings. US Patent 6 266 649

Maes P (1994) Agents that reduce work and information overload. Commun ACM 37(7):30–40

McJones P (1997) EachMovie collaborative filtering data set. DEC Systems Research Center, http://www.research.digital.com/SRC/eachmovie/

Merz CJ, Pazzani MJ (1999) A principal components approach to combining regression estimates. Mach Learn 36(1–2):9–32

Nakamura A, Abe N (1998) Collaborative filtering using weighted majority prediction algorithms. In: Proceedings, 15th International Conference on Machine Learning, pp 395–403

Newbold P, Granger CWJ (1974) Experience with forecasting univariate time series and the combination of forecasts. J R Stat Soc Ser A 137:131–146

Pennock DM, Horvitz E, Lawrence S, Giles LC (2000) Collaborative filtering by personality diagnosis: a hybrid memory- and model-based approach. In: Proceedings, 16th Conference on Uncertainty in Artificial Intelligence, pp 473–480

Pine BJ (1993) Mass Customization. Harvard Business School Press, Boston, MA

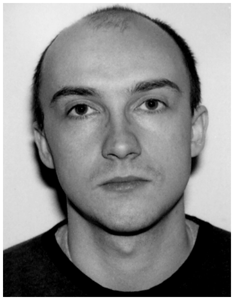
Riedmiller M, Braun H (1993) A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: Proceedings, IEEE International Conference on Neural Networks, pp 586–591

Salton G, Buckley C (1998) Term-weighting approaches in automatic text retrieval. Inf Process Manage 24(5):513–523

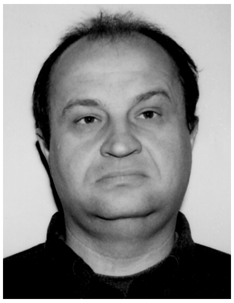
Shardanand U, Maes P (1995) Social information filtering: algorithms for automating “word of mouth”. In: Proceedings Computer Human Interaction Conference, pp 210–217

Ungar LH, Foster DP (1998) Clustering methods for collaborative filtering. In: Proceedings, Workshop on Recommendation Systems. AAAI Press, Menlo Park, CA

Author Biographies



Slobodan Vucetic is an Assistant Professor in the Department of Computer and Information Sciences at Temple University, Philadephia. He received his BS and MS in Electrical Engineering from the University of Novi Sad in 1994 and 1997, respectively, and his PhD in Electrical Engineering from Washington State University in 2001. His research interests are in data mining, machine learning, and bioinformatics.



Zoran Obradovic is the Director of the Center for Information Science and Technology and a Professor of Computer and Information Sciences at Temple University. His research interests focus on solving challenging scientific data-mining problems for efficient knowledge discovery in large databases. Funded by NSF, NIH, DOE, and industry, during the last decade he has contributed to about 150 refereed articles on these and related topics and to several academic and commercial software systems.

Correspondence and offprint requests to: Slobodan Vucetic, Center for Information Science and Technology, Temple University, 304 Wachman Hall, 1805 North Broad St., Philadelphia, PA 19122, USA. Email: vucetic@ist.temple.edu