# Improved Neighborhood-based Collaborative Filtering

Robert M. Bell and Yehuda Koren
AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932
{rbell,yehuda}@research.att.com

## ABSTRACT

Recommender systems based on collaborative filtering predict user preferences for products or services by learning past user-item relationships. A predominant approach to collaborative filtering is neighborhood based ("$k$-nearest neighbors"), where a user-item preference rating is interpolated from ratings of similar items and/or users. In this work, we enhance the neighborhood-based approach leading to a substantial improvement of prediction accuracy, without a meaningful increase in running time. First, we remove certain so-called "global effects" from the data to make the different ratings more comparable, thereby improving interpolation accuracy. Second, we show how to simultaneously derive interpolation weights for all nearest neighbors. Unlike previous approaches where each interpolation weight is computed separately, simultaneous interpolation accounts for the many interactions between neighbors by globally solving a suitable optimization problem, also leading to improved accuracy. Our method is very fast in practice, generating a prediction in about 0.2 milliseconds. Importantly, it does not require training many parameters or a lengthy preprocessing, making it very practical for large scale applications. The method was evaluated on the Netflix dataset. We could process the 2.8 million queries of the Qualifying set in 10 minutes yielding a RMSE of 0.9086. Moreover, when an extensive training is allowed, such as SVD-factorization at the preprocessing stage, our method can produce results with a RMSE of 0.8982.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

## General Terms

Algorithms

## Keywords

collaborative filtering, recommender systems, k nearest neighbors, matrix factorization


*KDDCup'07,* August 12, 2007, San Jose, California, USA.


## 1. INTRODUCTION

Recommender systems analyze patterns of user interest in items or products to provide personalized recommendations for items that will suit a user's taste. In essence, recommender systems attempt to profile user preferences and model the interaction between users and products. Increasingly, their excellent ability to characterize and recommend items within huge collections represent a computerized alternative to human recommendations. Because good personalized recommendations can add another dimension to the user experience, some of the largest e-commerce web sites have invested in high-quality recommender systems. Two known examples are the web merchant Amazon.com and the online movie rental company Netflix, which make the recommender system a salient part of their web sites. For a recent survey on recommender systems, refer to [1].

Broadly speaking, recommender systems use either of two strategies. The *content based approach* profiles each user or product allowing programs to associate users with matching products. For example, a movie profile might describe its genre, the participating actors, its box office popularity, etc. User profiles could include demographic information or answers to a suitable questionnaire. Of course, content based strategies require gathering external information that might not be available or easy to collect.

We focus on an alternative strategy that relies only on past user behavior–e.g., their previous transactions or product ratings–and does not require the creation of explicit profiles. This approach is known as *Collaborative Filtering* (CF), a term coined by the developers of the first recommender system - Tapestry [6]. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. Besides avoiding the need for extensive data collection about items or users, CF requires no domain knowledge. In addition, it offers the potential to uncover patterns that would be difficult or impossible to profile using content based techniques. This has led to many papers (e.g., [8]), research projects (e.g., [9]) and commercial systems (e.g., [10]) based on CF.

In a more abstract manner, the CF problem can be cast as missing value estimation: we are given a user-item matrix of scores with many missing values, and our goal is to estimate the missing values based on the given ones. The known user-item scores measure the amount of interest between respective users and items. They can be explicitly given by users that rate their interest in certain items or might be derived from historical purchases. We call these user-item scores *ratings*, and they constitute the input to our algorithm.

The most common form of CF is the neighborhood-based approach (also known as "$k$ Nearest Neighbors" or kNN, for short). These kNN methods identify pairs of items that tend to be rated similarly or like-minded users with similar history of rating or pur-

chasing, in order to deduce unknown relationships between users and items. Merits of the neighborhood-based approach that have made it popular include: its intuitiveness, sparing the need to train and tune many parameters, and the ability to easily explain to the user the reasoning behind a recommendation.

Three major components characterize the kNN approach: (1) data normalization, (2) neighbor selection, and (3) determination of interpolation weights. Our experience has shown little difference among various neighbor selection strategies (e.g., distance-based vs. correlation-based). However, the two other components, namely data normalization and interpolation weights, have proved vital to the success of the scheme. Accordingly, in this work we revisit these two components and suggest novel methods to accomplish them. We significantly improve the accuracy of kNN approaches without meaningfully affecting running time. Our two main contributions are:

1. It is customary to normalize the data before activating a kNN method. This brings different ratings to a closer level, which allows a better mixing thereof. Usually this is achieved by adjusting for the varying mean ratings across users and/or items. In Section 3 we offer a more comprehensive treatment of this normalization issue, which considers additional effects that are readily available in virtually all ratings datasets. This allows us to explain and eliminate much of the interfering variability from the data and to work with residuals that are more amenable to mutual interpolation.

2. Past kNN methods relate items (or users) by various heuristic variants of correlation coefficients, which allowed direct interpolation from neighbors' scores. In Section 4 we offer a rigorous alternative to these interpolation weights based on global optimization of a cost function pertaining to all weights simultaneously. This results in another improvement of estimation quality with a minor increase in running time.

An encouraging evaluation of the results on the Netflix Prize user-movie dataset [3] is provided in Section 5.

# 2. RELATED WORKS

## 2.1 General Framework

Before discussing previous works, we define our notational conventions. We are given ratings about $m$ users and $n$ items, arranged in an $m \times n$ matrix $R = \{r_{ui}\}_{1 \leqslant u \leqslant m, 1 \leqslant i \leqslant n}$. We anticipate three characteristics of the data that may complicate prediction. First, the numbers of users and items may be very large, as in the Netflix data, with the former likely much larger than the latter. Second, an overwhelming portion of the user-item matrix (e.g., 99%) may be unknown. Third, the pattern of observed data may be very nonrandom, i.e., the amount of observed data may vary by several orders of magnitude among users or among items.

We reserve special indexing letters for distinguishing users from items: for users $u, v$, and for items $i, j, k$.

## 2.2 Neighborhood-based collaborative filtering

The most common approach to CF is the neighborhood-based approach. Its original form, which was shared by virtually all earlier CF systems, is the user-oriented approach; see [8] for a good analysis. Such user-oriented methods estimate unknown ratings based on recorded ratings of like minded users. More formally, in order to estimate the unknown rating $r_{ui}$, we resort to a set of *users* $\mathrm{N}(u;i)$ that tend to rate similarly to $u$ ("neighbors"), and that actually rated item $i$ (i.e., $r_{vi}$ is known for each $v \in \mathrm{N}(u;i)$).

Then, the estimated value of $r_{ui}$ is taken as a weighted average of the neighbors' ratings:

$$r_{ui} \leftarrow \frac{\sum_{v \in \mathrm{N}(u;i)} s_{uv} r_{vi}}{\sum_{v \in \mathrm{N}(u;i)} s_{uv}} \tag{1}$$

The similarities – denoted by $s_{uv}$ – play a central role here as they are used both for selecting the members of $\mathrm{N}(u;i)$ and for weighting the above average. Common choices are the Pearson correlation coefficient and the closely related cosine similarity. Various methods differ by their how they normalize or center data prior to activating interpolation rule (1). For example, prediction quality can be improved by correcting for user-specific means. We present a more comprehensive treatment to data normalization in Section 3.

An analogous alternative to the user-oriented approach is the item-oriented approach [10, 14]. In those methods, a rating is estimated using known ratings made by the same user on similar items. Now, to estimate the unknown $r_{ui}$, we identify a set of neighboring *items* $\mathrm{N}(i;u)$ that other users tend to rate similarly to their rating of $i$. Analogous to above, all items in $\mathrm{N}(i;u)$ must have been rated by $u$. Then, in parallel to (1), the estimated value of $r_{ui}$ is taken as a weighted average of the ratings of neighboring items:

$$r_{ui} \leftarrow \frac{\sum_{j \in \mathrm{N}(i;u)} s_{ij} r_{uj}}{\sum_{j \in \mathrm{N}(i;u)} s_{ij}} \tag{2}$$

As with the user-user similarities, the item-item similarities (denoted by $s_{ij}$) are typically taken as either correlation coefficients or cosine similarities. Sarwar et al. [14] recommend using an adjusted cosine similarity, where ratings are translated by deducting user-means before computing the cosine similarity. They also found that item-oriented approaches deliver better quality estimates than user-oriented approaches while allowing more efficient computations. This is because, typically, the number of items is significantly lower than the number of users, which allows pre-computing all item-item similarities for retrieval as needed.

Neighborhood-based methods became very popular because they are intuitive and relatively simple to implement. In particular, they do not require tuning many parameters or an extensive training stage. They also provide a concise and intuitive justification for the computed predictions. This enables presenting the user a list of similar items that he or she has previously rated, as the basis for the estimated rating. This way, the user actually understands the reasoning behind the recommendation, allowing him/her to better assess its relevance (e.g., downgrade the estimated rating if it is based on an item that he or she no longer likes), or even encourage the user to alter outdated ratings.

However, neighborhood-based methods raise some concerns:

1. The similarity function ($s_{uv}$ or $s_{ij}$), which directly defines the interpolation weights, is arbitrary. Different CF algorithms use somewhat different similarity measures; all are trying to quantify the elusive notion of user- or item-similarity. We could not find any fundamental justification for the chosen similarities. Consider an extreme example, where a particular item is predicted perfectly by some other item. In that case, we would want the latter item to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.

2. Another problem is that previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item $i$ and a neighbor $j \in \mathrm{N}(i;u)$ is computed independently of the content of $\mathrm{N}(i;u)$ and the

other similarities: $s_{ik}$ for $k \in \mathrm{N}(i;u) - \{j\}$. For example, suppose that our items are movies, and the neighbors set contains three movies that are highly correlated with each other (e.g., sequels such as "Lord of the Rings 1–3"). An algorithm that ignores the similarity of the three movies when determining their interpolation weights, may end up essentially triple counting the information provided by the group.

3. By definition, the interpolation weights sum to one, which may cause overfitting. Suppose that an item has no useful neighbors rated by a particular user. In that case, it would be best to ignore the neighborhood information, staying with the current data normalization. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.

4. Neighborhood methods may not work well if variability differs substantially among neighboring items (users)

In a recent work [2], we overcame most of these shortcomings, but the result was a rather slow algorithm, whose running time was several orders of magnitude slower than previous neighborhood-methods, thereby limiting its practical applicability. In this work, we show how to solve the aforementioned issues of neighborhood based approaches, without compromising running time efficiency.

## 2.3 Factorization-based collaborative filtering

Matrix factorization is an alternative approach to CF. It is related to the so-called model-based approaches, which train a compact model that explains the full ratings matrix. It is not directly related to the methods discussed in this paper, but we survey it here because it can be integrated with our methods. In a way, factorization complements the more localized neighborhood-based methods by providing a higher level perspective to the ratings data. Our experiments show that such a combination can improve upon the application of either method alone; see Section 5.

The goal of a factorization-based CF is to uncover latent features of the given data that explain the ratings. This can be achieved by employing matrix factorization techniques such as Singular Value Decomposition (SVD) or Principal Components Analysis (PCA). Given an $m \times n$ matrix $R$, SVD computes the best rank-$f$ approximation $R^f$, which is defined as the product of two rank-$f$ matrices $P_{m\times f}$ and $Q_{n\times f}$, where $f \leqslant m, n$. That is, $R^f = PQ^T$ minimizes the Frobenius norm $\|R - R^f\|_F$ among all rank-$f$ matrices. In this sense, the matrix $R^f$ captures the $f$ most prominent features of the data, leaving out less significant features of the data that might be mere noise. Consequently, each unknown rating, $r_{ui}$ is estimated as $R^f_{ui}$.

Applying an SVD-based technique to CF raises unique difficulties due to the sparsity issue. The conventional SVD computation requires that all entries of $R$ are known. In fact, the goal of SVD is not properly defined when some entries of $R$ are missing. Previous works have adapted matrix factorizations techniques to handle missing data in a variety of ways [2, 5, 7, 12, 13]. In all cases, a critical issue is to avoid overfitting for items and users with relatively sparse data.

Factorization-based approaches tend to produce competitive results in terms of accuracy. However, their major drawback is a lengthy training phase that is required for building the model. Whenever these factorization methods are applicable, they can be used in conjunction with the neighborhood-based methods discussed in this paper, as will be explained later in Section 5.

# 3. NORMALIZING BY REMOVING GLOBAL EFFECTS

Although neighborhood and factorization based CF are both powerful prediction methods, there are several reasons to precede either technique with simple models that estimate what we call "global effects." First, there may be large user and item effects–i.e., systematic tendencies for some users to give higher ratings than other users and for some items to receive higher ratings than others. The basic kNN interpolation method detailed in equations (1)-(2) requires ratings where user and item effects have been taken out in order to, e.g., avoid predicting too high for low-rated items that happen to have a lot of neighbors with high average ratings, and vice versa. Or, to adjust for the fact that some items were mostly rated by users that tend to rate higher, while some other items were rated by users that tend to rate lower.

Second, one may have access to information about either the items or users that can benefit the model. Although factorization offers the potential to detect such structure through estimation of latent variables, directly incorporating variables such as movie genre and user demographics may be more effective and does not require training a factorization model. While such content based analysis is beyond the scope of this paper, we do consider two types of easily derived variables–the number of ratings of an item or by a user and the average rating of an item or by a user. Variables like these allow us, for example, to distinguish users who like the most commonly rated movies best from those who prefer more specialized fare (after controlling for both movie and user effects).

Third, there may be characteristics of specific ratings, such as the date of the rating, that explain some of the variation in scores. For example, a particular user's ratings may slowly, or suddenly, rise over time, above and beyond any change explained by the inherent quality of the items being rating. Similarly, ratings for some movies may fall with time after their initial release dates, while others stand the test of time quite well. To the extent that patterns like these occur, neither factorization nor kNN could be expected to detect the patterns.

## 3.1 Methods

Our strategy is to estimate one "effect" at a time, in sequence (i.e., the main effect for items, the main effect for users, a user-time interaction, etc.). At each step, we use residuals from the previous step as the dependent variable for the current step. Consequently, after the first step, the $r_{ui}$ refer to residuals, rather than raw ratings.

For each of the effects mentioned above, our goal is to estimate either one parameter for each item or one parameter for each user. For the rest of this subsection, we describe our methods for estimating user specific parameters; the method for items is perfectly analogous.

We denote by $x_{ui}$ the explanatory variable of interest corresponding to user $u$ and item $i$. For user main effects, the $x_{ui}$'s are identically 1. For other global effects, we center $x_{ui}$ for each user by subtracting the mean of $x_{ui}$ for that user. In each case, our model is:

$$r_{ui} = \theta_u x_{ui} + error \tag{3}$$

With sufficient ratings for user $u$, we might use the unbiased estimator:

$$\hat{\theta}_u = \frac{\sum_i r_{ui} x_{ui}}{\sum_i {x_{ui}}^2}$$

where each summation is over all items rated by $u$. However, for sparse data, some values of $\hat{\theta}_u$ may be based on very few observations, thereby resulting in unreliable estimates.

To avoid overfitting, we shrink individual values of $\hat{\theta}_u$ towards a common value. Shrinkage can be motivated from a Bayesian perspective. Suppose that the true $\theta_u$ are independent random variables drawn from a normal distribution,

$$\theta_u \sim N(\mu, \tau^2)$$

for known $\mu$ and $\tau^2$, while

$$\hat{\theta}_u|\theta_u \sim N(\theta_u, \sigma_u^2)$$

for known $\sigma_u^2$. Then, the best estimator for $\theta_u$ is its posterior mean:

$$E(\theta_u|\hat{\theta}_u) = \frac{\tau^2\hat{\theta}_u + \sigma_u^2\mu}{\tau^2 + \sigma_u^2}$$

a linear combination of the empirical estimator $\hat{\theta}_u$ and the common mean $\mu$. The parameter $\sigma_u^2$ can be estimated from the formula for the variance of a weighted mean, while $\mu$ can be estimated by the mean of the $\theta_u$'s (optionally weighted by $n_u$). Empirical Bayes [4] suggests that the maximum likelihood estimate of $\tau^2$ can be found as the solution to:

$$\tau^2 = \frac{\sum_u[(\hat{\theta}_u - \mu)^2 - \sigma_u^2]/(\tau^2 + \sigma_u^2)^2}{\sum_u(\tau^2 + \sigma_u^2)^{-2}}$$

In practice, we used a slightly simpler estimator for $\theta_u$ by assuming $\mu = 0$ and $\sigma_u^2$ is proportional to $1/n_u$, which yields:

$$\frac{n_u\hat{\theta}_u}{n_u + \alpha}$$

where $n_u$ is the number of ratings by user $u$ and $\alpha$ is a constant. We determined $\alpha$ by cross validation.

## 3.2 Example

We illustrate the impact of estimating successive global effects on the Netflix data [3]. The dataset is based on more than 100 million ratings of movies performed by anonymous Netflix customers and is described in details in Section 5. Here, we report results on the *Probe set*, which is a test set containing about 1.4 million ratings compiled by Netflix. Ratings are predicted by accumulating a series of global effects. Accuracy of predictions is measured by their root mean squared error (RMSE). The effects that we describe here should be readily available in every user-item ratings system, and are not particular to the Netflix data.

Table 1 shows how the RMSE for the probe set declines with the inclusion of each successive global effect. Not surprisingly, by far the largest improvements in RMSE are associated with the two sets of main effects–the movie effect and the user effect. These two main effects are comparable to double centering enhanced with shrinkage. They reduce the RMSE from 1.1296 based on use of the mean rating in the training data, down to 0.9841.

Of more interest are various interactions. The first interaction term, User×Time(user)$^{1/2}$, refers to allowing each user's rating to change linearly with the square root of the number of days since the user's first rating. Exploratory analysis indicated that this transformation improved the fit relative to the untransformed number of days. This term, which allows for the type of drift hypothesized above, reduces the RMSE by 0.0032. Technically, in (3) we set each $x_{ui}$ to the square root of number of days elapsed since the first rating by user $u$ till the time $u$ rated item $i$. Then, for each fixed user $u$, we center all computed values of $x_{ui}$, and calculate $\hat{\theta}_u$ in order to regress $r_{ui}$ on $x_{ui}$.

Similarly, User×Time(movie)$^{1/2}$, allows each user's rating to change linearly with the square root of the number of days since the movie's first rating by anyone. Somewhat surprisingly, the latter interaction contributes almost as much as the first. Two additional time interactions concentrate on the complementary movie viewpoint. That is, for each fixed movie they model how its ratings change against time. Once again, the time variables are either square root of days since first rating of the movie, or square root of days since first rating by the user.

| Effect | RMSE | Improvement |
|---|---|---|
| Overall mean | 1.1296 | NA |
| Movie effect | 1.0527 | .0769 |
| User effect | 0.9841 | .0686 |
| User×Time(user)$^{1/2}$ | 0.9809 | .0032 |
| User×Time(movie)$^{1/2}$ | 0.9786 | .0023 |
| Movie×Time(movie)$^{1/2}$ | 0.9767 | .0019 |
| Movie×Time(user)$^{1/2}$ | 0.9759 | .0008 |
| User×Movie average | 0.9719 | .0040 |
| User×Movie support | 0.9690 | .0029 |
| Movie×User average | 0.9670 | .0020 |
| Movie×User support | 0.9657 | .0013 |

**Table 1: RMSE for Netflix probe data after adding a series of global effects to the model**

The next two effects interact users with movie characteristics. We measure the popularity of a movie in two different ways: (1) its average rating; (2) its support, which is the number of ratings associated with the movie. These effects—User×Movie average and User×Movie support—measure how users change their ratings based on the popularity of the movies. We try to estimate here how closely a user follows the public opinion, or perhaps the opposite, how contrarian is the user. The most effective interaction is between users and movie average–indicating that users varied in terms of how much they agree with the consensus.

Finally, we interact movies with user characteristics, regressing the ratings of each movie on the mean or support of the users. Although these interactions are more difficult to motivate, each contributes an additional modest decrease of the RMSE. Overall, the eight interactions combine to reduce the RMSE by 0.0184 to 0.9657.

# 4. A NEIGHBORHOOD RELATIONSHIPS MODEL

In this section we assume that global effects have already been removed, so whenever we refer to a rating we actually mean the residual remaining after removing global effects from the original rating data. However, our discussion here is completely general, so the following method applies whether global effects are removed or not.

We need to estimate the unknown rating by user $u$ of item $i$, that is $r_{ui}$. As with all neighborhood-based methods, our first step is neighbor selection. We focus on an item-oriented method. Among all items rated by $u$, we select the $K$ most similar to $i$, $\mathrm{N}(i;u)$, by using a similarity function such as the correlation coefficient, as described later in this section. Typical values of $K$ lie in the range of 20–50; see Section 5.

Given a set of neighbors $\mathrm{N}(i;u)$, we need to compute *interpolation weights* $\{w_{ij}|j \in \mathrm{N}(i;u)\}$ that will enable the best prediction rule of the form:

$$r_{ui} \leftarrow \sum_{j \in \mathrm{N}(i;u)} w_{ij} r_{uj} \quad (4)$$

For notational convenience assume that the $K$ neighbors in $\mathrm{N}(i;u)$

are indexed by $1, \ldots, K$ , and the corresponding interpolation weights are arranged within $w \in \mathbb{R}^K$.

We seek a formal computation of the interpolation weights, that stems directly from their usage within prediction rule (4). As explained earlier, it is important to derive all interpolation weights simultaneously to account for interdependencies among the neighbors. We achieve these goals by defining a suitable optimization problem.

To start, we consider a hypothetical dense case, where all users but $u$ rated both $i$ and *all* its neighbors in $\mathrm{N}(i;u)$. In that case, we could learn the interpolation weights by modeling the relationships between item $i$ and its neighbors through a least squares problem:

$$\min_{w} \sum_{v \neq u} \left( r_{vi} - \sum_{j \in \mathrm{N}(i;u)} w_{ij} r_{vj} \right)^2 \tag{5}$$

Notice that the only unknowns here are the $w_{ij}$'s. The optimal solution to the least squares problem (5) is found by differentiation as a solution of a linear system of equations. From a statistics viewpoint, it is equivalent to the result of a linear regression (without intercept) of $r_{vi}$ on the $r_{vj}$ for $j \in \mathrm{N}(i;u)$. Specifically, the optimal weights are given by:

$$Aw = b \tag{6}$$

Here, $A$ is a $K \times K$ matrix defined as:

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk} \tag{7}$$

Similarly the vector $b \in \mathbb{R}^K$ satisfies:

$$b_j = \sum_{v \neq u} r_{vj} r_{vi} \tag{8}$$

For a sparse ratings matrix there are likely to be very few users who rated $i$ and all its neighbors $\mathrm{N}(i;u)$. Accordingly, it would be unwise to base $A$ and $b$ as given in (7)–(8) only on users with complete data. Even if there are enough users with complete data for $A$ to be nonsingular, that estimate would ignore a large proportion of the information about pairwise relationships among ratings by the same user. However, we can still estimate $A$ and $b$, up to the same constant, by averaging over the given support, leading to the following reformulation:

$$\bar{A}_{jk} = \frac{\sum_{v \in \mathrm{U}(j,k)} r_{vj} r_{vk}}{|\mathrm{U}(j,k)|} \tag{9}$$

$$\bar{b}_j = \frac{\sum_{v \in \mathrm{U}(i,j)} r_{vj} r_{vi}}{|\mathrm{U}(i,j)|} \tag{10}$$

where $\mathrm{U}(j,k)$ is the set of users who rated both items $j$ and $k$.

This is still not enough for overcoming the sparseness issue. The averages represented by $\bar{A}_{jk}$ or $\bar{b}_j$ may differ by orders of magnitude in terms of the number of users included in the average. As discussed in the previous section, averages based on relatively low support (small values of $|\mathrm{U}(j,k)|$) can generally be improved by shrinkage towards a common mean. Thus, we compute a baseline value which is defined by taking the average of all possible $\bar{A}_{jk}$ values. Let us denote this baseline value by $avg$; its precise computation is described in the next subsection. Accordingly, we define the corresponding $K \times K$ matrix $\hat{A}$ and the vector $\hat{b} \in \mathbb{R}^K$:

$$\hat{A}_{jk} = \frac{|\mathrm{U}(j,k)| \cdot \bar{A}_{jk} + \beta \cdot avg}{|\mathrm{U}(j,k)| + \beta} \tag{11}$$

$$\hat{b}_j = \frac{|\mathrm{U}(i,j)| \cdot \bar{b}_j + \beta \cdot avg}{|\mathrm{U}(i,j)| + \beta} \tag{12}$$

The parameter $\beta$ controls the extent of the shrinkage. A typical value when working with residuals of full global effects is $\beta = 500$.

Our best estimate for $A$ and $b$ are $\hat{A}$ and $\hat{b}$, respectively. Therefore, we modify (6) so that the interpolation weights are defined as the solution of the linear system:

$$\hat{A}w = \hat{b} \tag{13}$$

The resulting interpolation weights are used within (4) in order to predict $r_{ui}$. When working with residuals of global effects, they should be added back to the predicted $r_{ui}$, which completes the computation.

Notice that this method addresses all four concerns raised in Section 2.1. First, interpolation weights are derived directly from the ratings, not based on any similarity measure. Second, the interpolation weights formula explicitly accounts for relationships among the neighbors. Third, the sum of the weights is not constrained to equal one. If an item (or user) has only weak neighbors, the estimated weights may all be very small. Fourth, the method automatically adjusts for variations among items in their means or variances.

## 4.1 Preprocessing

An efficient computation of an item-item neighborhood-based method depends on precomputing certain values associated with each movie-movie pair to enable their rapid retrieval.

First, we need a quick access to all item-item similarities, the $s_{ij}$ values. These similarities are required for identifying the $K$ neighbors that constitute $\mathrm{N}(i;u)$. Here, we usually take $s_{ij}$ as the Pearson correlation coefficient between $i$ and $j$ calculated on their common raters. It is important to shrink $s_{ij}$ based on their support, e.g., multiplying the correlation by: $|\mathrm{U}(i,j)|/(|\mathrm{U}(i,j)| + \alpha)$ for some small $\alpha$. An alternative, which works similarly well, is based on the mean squared error between items:

$$s_{ij} = \frac{|\mathrm{U}(i,j)|}{\sum_{u \in \mathrm{U}(i,j)} (r_{ui} - r_{uj})^2 + \alpha}$$

The second set of values that should be precomputed is all possible entries of $\hat{A}$ and $\hat{b}$. To this end, for each two items $i$ and $j$, we compute:

$$\bar{A}_{ij} = \frac{\sum_{v \in \mathrm{U}(i,j)} r_{vi} r_{vj}}{|\mathrm{U}(i,j)|}$$

Then, the aforementioned baseline value $avg$, which is used in (11)-(12), is taken as the average entry of the precomputed $n \times n$ matrix $\bar{A}$. In fact, we recommend using two different baseline values, one by averaging the non-diagonal entries of $\bar{A}$ and another one by averaging the diagonal entries. This accounts for the fact that the diagonal entries are expected to have an inherently higher average because they sum only non-negative values. Finally, we derive a full $n \times n$ matrix $\hat{A}$ from $\bar{A}$ by (11). Here, the non-diagonal average is used when deriving the non-diagonal entries of $\hat{A}$, whereas the diagonal average is used when deriving the diagonal entries of $\hat{A}$.

Because of symmetry, it is sufficient to store the values of $s_{ij}$ and $\hat{A}_{ij}$ only for $i \geqslant j$. We allocated one byte for each individual

**NonNegativeQuadraticOpt** ($A \in \mathbb{R}^{K \times K}, b \in \mathbb{R}^K$)
*% Minimize $x^T Ax - 2b^T x$ s.t. $x \geqslant 0$*

**do**
  $r \leftarrow Ax - b$ *% the residual, or "steepest gradient"*
  *% find active variables - those that are pinned because of*
  *% nonnegativity constraint, and set respective $r_i$'s to zero*
  **for** $i = 1, \ldots, k$ **do**
    **if** $x_i = 0$ and $r_i < 0$ **then**
      $r_i \leftarrow 0$
    **end if**
  **end for**
  $\alpha \leftarrow \frac{r^T r}{r^T A r}$ *% max step size*
  *% adjust step size to prevent negative values:*
  **for** $i = 1, \ldots, k$ **do**
    **if** $r_i < 0$ **then**
      $\alpha \leftarrow \min(\alpha, -x_i / r_i)$
    **end if**
  **end for**
  $x \leftarrow x + \alpha r$
**while** $\|r\| < \epsilon$ *% stop when residual is close to 0*
**return** $x$

**Figure 1: Minimizing a quadratic function with non-negativity constraints**

value, so overall space complexity for $n$ items is exactly $n(n+1)$ bytes. E.g., for the Netflix dataset which contains 17770 movies, overall memory requirements are 300MB, easily fitting within core memory. A more comprehensive system, which includes data on 100,000 items would require about 10GB of space, which can fit within core memory of current 64bit servers. For even larger systems, disk resident storage of item-item values is still practical, as evident by the Amazon item-item recommender system, which is accessing stored similarity information for several million catalog items [10]. Preprocessing time is quadratic in $n$ and linear in the number of ratings. The time required for computing all $s_{ij}$'s and $\hat{A}_{ij}$'s on the Netflix data (which contains 100 million ratings) was about 15 minutes on a Pentium 4 PC. Notice that preprocessing can be easily parallelized.

The fact that all possible entries of matrix $\hat{A}$ are precomputed, saves the otherwise lengthy time needed to construct it. Thus, after quickly retrieving the relevant entries of $\hat{A}$, we can compute the interpolation weights by solving a $K \times K$ system of equations. For typical values of $K$ (between 20 and 50), this time is comparable to the time needed for computing the $K$ nearest neighbors, which is common to all neighborhood-based approaches. Hence, while our method relies on a much more detailed computation of the interpolation weights compared to previous methods, it does not significantly increase running time; see Section 5.

## 4.2 Calculation of the weights

The interpolation weights can be computed by solving (13) using standard linear equations solvers. However, we experienced a modest increase in accuracy when $w$ is constrained to be non-negative, which avoids certain redundant overfitting. This leads to a quadratic program which can be solved by calling "NonNegativeQuadraticOpt( $\hat{A}$, $\hat{b}$)", as described in Figure 1. The function is based on the principles of the Gradient Projection method; see, e.g., [11].

## 4.3 User-oriented implementation

In the above discussion we derived an item-oriented approach, but a parallel idea was successfully applied in a user-oriented fashion, by switching the roles of users and items throughout the above discussion. However, as with previous neighborhood-based approaches, an item-oriented approach enables a much faster implementation by precomputing and storing in main memory a full item-item matrix containing all $s_{ij}$ and $\hat{A}_{ij}$ values for future retrieval. Typically, the larger number of users will complicate such a precomputation in terms of both time and space, and out-of-core storage will be required. Moreover, our experience with the Netflix data, under various settings, showed that an item-oriented approach consistently delivered more accurate predictions than a user-oriented one. This advantage of item-oriented methods strengthens a similar observation by Sarwar et al. [14]. However, when a user-oriented approach is computationally feasible, it can be used for improving prediction accuracy by mixing its results with those of the item-oriented approach as in [15].

# 5. EXPERIMENTAL STUDY

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings [3]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset.

To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is measured by their root mean squared error (RMSE), a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. In addition, we report results on two test sets compiled by Netflix, one is known as *the Probe set* and the other is known as *the Quiz set*. These two test sets were constructed similarly, with both containing about 1.4 million of the most recent movie ratings (by date) performed by the users. The true ratings of the Probe set are provided. However, we do not know the true ratings for the Quiz set, which are held by Netflix being the subject of the Netflix Prize contest. Netflix provides RMSE values to competitors that submit their predictions for the Quiz set. The benchmark is Netflix's proprietary CF system, Cinematch, which achieved a RMSE of 0.9514 on this Quiz set. The two test sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

## 5.1 Experiments with the Probe set

Table 2 compares our method to a kNN method that takes interpolation weights as Pearson correlation coefficients (shrunk to improve results), which represent the common approach to the problem. All methods use item neighborhoods. Reported results are RMSE values computed on the Probe set. We apply the methods after varying stages of preprocessing the data. First, we applied the methods to the raw data, where scores were not normalized at all. Then, we removed the first two effects – the user effect and the item effect, which is similar to double-centering. When no neighborhood-interpolation is performed, such a normalization alone delivers a RMSE of 0.9841, as can be seen in the second column of the table. A more substantial normalization is achieved by accounting for all the global effects, as described in Section 3. Global effects on their own lower the Probe RMSE to 0.9657, before applying any neighborhood interpolation.

Finally, our method can be combined with the factorization approach described in Subsection 2.3. Notice that unlike the previ-

| Data normalization | No interpolation ($k = 0$) | Correlation-based interpolation $k = 20$ | $k = 35$ | $k = 50$ | Jointly derived interpolation $k = 20$ | $k = 35$ | $k = 50$ |
|---|---|---|---|---|---|---|---|
| none (raw scores) | NA | 0.9947 | 1.002 | 1.0085 | 0.9536 | 0.9596 | 0.9644 |
| double centering | 0.9841 | 0.9431 | 0.9470 | 0.9502 | 0.9216 | 0.9198 | 0.9197 |
| global effects | 0.9657 | 0.9364 | 0.9390 | 0.9413 | 0.9194 | 0.9179 | 0.9174 |
| factorization | 0.9167 | 0.9156 | 0.9142 | 0.9142 | 0.9071 | 0.9071 | 0.9071 |

**Table 2: Comparing our interpolation scheme against conventional correlation-based interpolation, by reporting RMSEs on the Probe set. Various levels of data normalization (preprocessing) are shown, and different sizes of neighborhoods ($K$) are considered.**

ously mentioned normalizations, factorization requires an extensive training in order to learn the factors. Our implementation delivered a RMSE of 0.9167 on the Probe set, before considering any neighborhood information. A kNN method can be used to improve factorization results, by applying it to the residuals remaining after subtracting the estimates generated by factorization. Effectively, error is smoothed by considering local information that the rather regional factorization approaches tend to miss.

The remaining columns of Table 2 contain results using the two methods for determining interpolation weights. We provide results representing the spectrum of viable choices for neighborhood size – $K$. For each value of $K$, the same neighborhoods are used for the two methods, the only difference lies in the determination of interpolation weights. The main findings are as follows.

- The jointly derived interpolation weights proposed in this paper uniformly outperformed standard correlation based weights. The improvement was 0.0071 when applied after factorization, but was much larger with less extensive data normalizations.

- Use of the neighborhood model with our proposed interpolation weights substantially improved the predictions across all the data normalizations. The neighborhood approach reduced RMSE by 0.0096 even after the much more time consuming factorization method. Notably, the correlation-based neighborhood model produced a much more modest improvement of 0.0025 relative to factorization.

- The best neighborhood size $K$ varied depending on the extent of prior data normalization. Interestingly, we found no difference in RMSE over the range 20 to 50 neighbors for the best combination–jointly derived interpolation weights after factorization.

- Even with incorporation of neighborhood information, more complete data normalization improved predictions. However, there were diminishing returns. For example, while global effects reduced RMSE by 0.0184 before the neighborhood model, only 0.0023 survived after incorporation of the neighborhood model.

As for running time, the correlation based method required about 200 seconds to process the whole Probe set, regardless of the value of $K$. Our method, with $K = 20$, required a slight increase in time to about 235 seconds for the whole Probe set, where the added time represents the effort needed to solve the $K \times K$ least squares problem for each query. Not surprisingly, increasing $K$ from 20 to 35 or 50 also raises running time to around 355 seconds ($K = 35$) or 470 seconds ($K = 50$), due to the growing effort needed for solving the larger least squares problems. Even so, the slower mode of our method ($K = 50$) processed a single rating query in less than 0.4 millisecond.

## 5.2 Experiments with the Quiz set

When computing predictions for the Quiz set, the test set held by Netflix, we subsume the Probe set into our training data. This makes Quiz results better than the Probe results. Accordingly, our method produced a RMSE of 0.9086 (4.5% improvement over Netflix system) for the quiz set when applied to residuals of global effects. When applied to residuals of factorization the RMSE dropped to 0.8982 (5.6% improvement over Netflix system), at the cost of requiring the training of a factorization model.

These results are comparable to recently reported results for the Netflix data by methods that require an extensive training phase [2, 12]. Moreover, when mixed with other methods, results can be substantially improved. For example, an alternative method to introducing neighborhood-awareness into factorization-based results was described in [2]. While the accuracy of the two methods is similar, they differ considerably in their nature, and thus produce different ratings. Thus, they can be mixed effectively to produce a more accurate solution. Their mixtures achieve solutions with RMSEs around 0.8900 (6.45% improvement over Netflix system), which would currently rank high on the Netflix Prize Leaderboard[1], even before mixing with other known approaches.

# 6. SUMMARY

Collaborative filtering through neighborhood-based interpolation ("kNN") is probably the most popular way to create a recommender system. The success of these methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, the literature lacks a rigorous way to derive these weights. In this work we showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role. Comparison to past kNN methods was made on the Netflix data, with encouraging results suggesting a significant improvement of prediction accuracy without a meaningful increase in running time.

Our second, complementary contribution is related to data normalization. Normalization is essential to kNN methods, as otherwise mixing ratings pertaining to different unnormalized users or items will lead to inferior results. This work offered a comprehensive approach to data normalization, which is related to removing 10 effects that can be readily observed in user-item rating data. Those effects cause much of the data variability and mask the fundamental relationships between ratings. Hence, their removal brings the ratings closer and facilitates improved estimation accuracy.

# 7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin, "Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", *IEEE*

[1] `www.netflixprize.com/leaderboard`

*Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.

[2] R. M. Bell, Y. Koren and C. Volinsky, “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems", *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[3] J. Bennet and S. Lanning, “The Netflix Prize", `www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf`.

[4] B. Efron and C. Morris, “Data analysis using Stein’s estimator and its generalization", *Journal American Statistical Association* **70** (1975), 311–319.

[5] S. Funk, “Netflix Update: Try This At Home", `sifter.org/~simon/journal/20061211.html`, 2006.

[6] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM* **35** (1992), 61–70.

[7] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm", *Information Retrieval* **4** (2001), 133–151.

[8] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, “An Algorithmic Framework for Performing Collaborative Filtering", *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.

[9] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, “GroupLens: Applying Collaborative Filtering to Usenet News", *Communications of the ACM* **40** (1997), 77–87, `www.grouplens.org`.

[10] G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering", *IEEE Internet Computing* **7** (2003), 76–80.

[11] J. Nocedal and S. Wright, *Numerical Optimization*, Springer (1999).

[12] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering", *Proc. 24th Annual International Conference on Machine Learning*, 2007.

[13] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Application of Dimensionality Reduction in Recommender System – A Case Study", *WEBKDD’2000*.

[14] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms", *Proc. 10th International Conference on the World Wide Web*, pp. 285-295, 2001.

[15] J. Wang, A. P. de Vries and M. J. T. Reinders,“Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion", *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.