Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems

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

ABSTRACT

The collaborative filtering approach to recommender systems predicts user preferences for products or services by learning past user-item relationships. In this work, we propose novel algorithms for predicting user ratings of items by integrating complementary models that focus on patterns at different scales. At a local scale, we use a neighborhood-based technique that infers ratings from observed ratings by similar users or of similar items. Unlike previous local approaches, our method is based on a formal model that accounts for interactions within the neighborhood, leading to improved estimation quality. At a higher, regional, scale, we use SVD-like matrix factorization for recovering the major structural patterns in the user-item rating matrix. Unlike previous approaches that require imputations in order to fill in the unknown matrix entries, our new iterative algorithm avoids imputation. Because the models involve estimation of millions, or even billions, of parameters, shrinkage of estimated values to account for sampling variability proves crucial to prevent overfitting. Both the local and the regional approaches, and in particular their combination through a unifying model, compare favorably with other approaches and deliver substantially better results than the commercial Netflix Cinematch recommender system on a large publicly available data set.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

collaborative filtering, recommender systems

1. INTRODUCTION

Recommender systems [1] are programs and algorithms that measure the user interest in given items or products to provide personalized recommendations for items that will suit the user's taste. More broadly, 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.

The growing popularity of e-commerce brings an increasing interest in recommender systems. While users browse a web site, well calculated recommendations expose them to interesting products or services that they may consume. The huge economic potential led some of the biggest e-commerce web, for example web merchant Amazon.com and the online movie rental company Netflix, make the recommender system a salient part of their web sites. High quality personalized recommendations deepen and add another dimension to the user experience. For example, Netflix users can base a significant portion of their movie selection on automatic recommendations tailored to their tastes.

Broadly speaking, recommender systems are based on two different strategies (or combinations thereof). The *content based approach* creates a profile for each user or product to characterize its nature. As an example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. User profiles might include demographic information or answers to a suitable questionnaire. The resulting profiles allow programs to associate users with matching products. However, content based strategies require gathering external information that might not be available or easy to collect.

An alternative strategy, our focus in this work, relies only on past user behavior without requiring 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 [4]. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. For example, some CF systems identify pairs of items that tend to be rated similarly or like-minded users with similar history of rating or purchasing to deduce unknown relationships between users and items. The only required information is the past behavior of users, which might be, e.g., their previous transactions or the way they rate products. A major appeal of CF is that it is domain free, yet it can address aspects of the data that are often elusive and very difficult to profile using content based techniques. This has led to many papers (e.g., [7]), research projects (e.g., [9]) and commercial systems (e.g., [11]) 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.

KDD'07, August 12–15, 2007, San Jose, California, USA.

In October 2006, the online movie renter, Netflix, announced a CF-related contest – *The Netflix Prize* [10]. Within this contest, Netflix published a comprehensive dataset including more than 100 million movie ratings, which were performed by about 480,000 real customers (with hidden identities) on 17,770 movies. Competitors in the challenge are required to estimate a few million ratings. To win the "grand prize," they need to deliver a 10% improvement in the prediction root mean squared error (RMSE) compared with the results of Cinematch, Netflix's proprietary recommender system. This real life dataset, which is orders of magnitudes larger than previous available datasets for in CF research, opens new possibilities and has the potential to reduce the gap between scientific research and the actual demands of commercial CF systems. The algorithms in this paper were tested and motivated by the Netflix data, but they are not limited to that dataset, or to user-movie ratings in general.

Our approach combines several components; each of which models the data at a different scale in order to estimate the unknown ratings. The *global component* utilizes the basic characteristics of users and movies, in order to reach a first-order estimate of the unknown ratings. At this stage the interaction between items and users is minimal. The more refined, *regional component*, defines –for each item and user– *factors* that strive to explain the way in which users interact with items and form ratings. Each such factor corresponds to a viewpoint on the given data along which we measure users and items simultaneously. High ratings correspond to user-item pairs with closely matching factor values. The *local component* models the relationships between similar items or users, and derive unknown ratings from values in user/item neighborhoods.

The main contributions detailed in this paper are the following:

- Derivation of a local, neighborhood-based approach where the interpolation weights solve a formal model. This model simultaneously accounts for relationships among all neighbors, rather than only for two items (or users) at a time. Furthermore, we extend the model to allow item-item weights to vary by user, and user-user weights to vary by item.
- A new iterative algorithm for an SVD-like factorization that avoids the need for imputation, thereby being highly scalable and adaptable to changes in the data. We extend this model to include an efficient integration of neighbors' information.
- Shrinkage of parameters towards common values to allow utilization of very rich models without overfitting.
- Derivation of confidence scores that facilitate combining scores across algorithms and indicate the degree of confidence in predicted ratings.

Both the local and regional components outperformed the published results by Netflix's proprietary Cinematch system. Additionally, combination of the components produced even better results, as we explain later.

The paper explains the three components, scaling up from the local, neighborhood-based approach (Section 3), into the regional, factorization-based approach (Section 4). Discussion of some related works is split between Sections 3 and 4, according to its relevancy. Experimental results, along with discussion of confidence scores is given in Section 5. We begin in Section 2 by describing the general data framework, the methodology for parameter shrinkage, and the approach for dealing with global effects.

2. GENERAL FRAMEWORK

2.1 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. Sometimes we refer to matrix R as a sparse matrix, although its sparsity pattern is driven by containing many unknown values rather than the common case of having many zeros. 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. The known entries – those (u, i) pairs for which r_{ui} is known – are stored in the set $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$.

2.2 Shrinking estimated parameters

The algorithms described in Sections 3 and 4 can each lead to estimating massive numbers of parameters (not including the individual ratings). For example, for the Netflix data, each algorithm estimates at least ten million parameters. Because many of the parameter estimates are based on small sample sizes, it is critical to take steps to avoid overfitting the observed data.

Although cross validation is a great tool for tuning a few parameters, it is inadequate for dealing with massive numbers of parameters. Cross validation works very well for determining whether the use of a given parameter improves predictions relative to other factors. However, that all-or-nothing choice does not scale up to selecting or fitting many parameters at once. Sometimes the amount of data available to fit different parameters will vary widely, perhaps by orders of magnitude. The idea behind "shrinkage" is to impose a penalty for those parameters that have less data associated with them. The penalty is to shrink the parameter estimate towards a null value, often zero. Shrinkage is a more continuous alternative to parameter selection, and as implemented in statistical methods like ridge and lasso regression [16] has been shown in many cases to improve predictive accuracy.

We can approach shrinkage from a Bayesian perspective. In the typical Bayesian paradigm, the best estimator of a parameter is the posterior mean, a linear combination of the prior mean of the parameter (null value) and an empirical estimator based fully on the data. The weights on the linear combination depend on the ratio of the prior variance to the variance of the empirical estimate. If the estimate is based on a small amount of data, the variance of that estimate is relatively large, and the data-based estimates are shrunk toward the null value. In this way shrinkage can simultaneously "correct" all estimates based on the amount of data that went into calculating them.

For example, we can calculate similarities between two items based on the correlation for users who have rated them. Because we expect pairwise correlations to be centered around zero, the empirical correlations are shrunk towards zero. If the empirical correlation of items i and j is s_{ij}, based on n_{ij} observations, then the shrunk correlation has the form $n_{ij} s_{ij}/(n_{ij} + \beta)$. For fixed β, the amount of shrinkage is inversely related to the number of users that have rated both items: the higher n_{ij} is, the more we "believe" the estimate, and the less we shrink towards zero. The amount of shrinkage is also influenced by β, which we generally tune using cross validation.

2.3 Removal of global effects

Before delving into more involved methods, a meaningful portion of the variation in ratings can be explained by using readily available variables that we refer to as global effects. The most obvious global effects correspond to item and user effects – i.e., the

tendency for ratings of some items or by some users to differ systematically from the average. These effects are removed from the data, and the subsequent algorithms work with the remaining residuals. The process is similar to double-centering the data, but each step must shrink the estimated parameters properly.

Other global effects that can be removed are dependencies between the rating data and some known attributes. While a pure CF framework does not utilize content associated with the data, when such content is given, there is no reason not to exploit it. An example of a universal external attribute is the dates of the ratings. Over time, items go out of fashion, people may change their tastes, their rating scales, or even their "identities" (e.g., a user may change from being primarily a parent to primarily a child from the same family). Therefore, it can be beneficial to regress against time the ratings by each user, and similarly, the ratings for each item. This way, time effects can explain additional variability that we recommend removing before turning to more involved methods.

3. NEIGHBORHOOD-BASED ESTIMATION

3.1 Previous work

The most common approach to CF is the local, or neighborhood-based approach. Its original form, which was shared by virtually all earlier CF systems, is the user-oriented approach; see [7] 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* $\mathrm{N}(u;i)$, which tend to rate similarly to u ("neighbors"), and 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:

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

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. Prediction quality can be improved further by correcting for user-specific means, which is related to the global effects removal discussed in Subsection 2.3. More advanced techniques account not only for mean translation, but also for scaling, leading to modest improvements; see, e.g., [15].

An analogous alternative to the user-oriented approach is the item-oriented approach [15, 11]. 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* $\mathrm{N}(i;u)$, which 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:

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

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. [15] recommend using an adjusted cosine similarity on ratings that had been translated by deducting user-means. They found that item-oriented approaches deliver better quality estimates than user-oriented approaches while allowing more efficient computations. This is because there are typically significantly fewer items than 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 our eyes, a main benefit is their ability to provide a concise and intuitive justification for the computed predictions, presenting the user a list of similar items that she has previously rated, as the basis for the estimated rating. This allows her to better assess its relevance (e.g., downgrade the estimated rating if it is based on an item that she no longer likes) and may encourage the user to alter outdated ratings.

However, neighborhood-based methods share some significant disadvantages. The most salient one is the heuristic nature of the similarity functions (s_{uv} or s_{ij}). Different rating algorithms use somewhat different similarity measures; all are trying to quantify the elusive notion of the level of user- or item-similarity. We could not find any fundamental justification for the chosen similarities.

Another problem is that previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item i and a neighbor $j \in \mathrm{N}(i;u)$, and consequently its weight in (2), 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 very close in their nature (e.g., sequels such as "Lord of the Rings 1–3"). An algorithm that ignores the similarity of the three movies when predicting the rating for another movie, may triple count the available information and disproportionally magnify the similarity aspect shared by these three very similar movies. A better algorithm would account for the fact that part of the prediction power of a movie such as "Lord of the Rings 3" may already be captured by "Lord of the Rings 1-2", and vice versa, and thus discount the similarity values accordingly.

3.2 Modeling neighborhood relations

We overcome the above problems of neighborhood-based approaches by relying on a suitable model. To our best knowledge, this is the first time that a neighborhood-based approach is derived as a solution to a model, and allows simultaneous, rather than isolated, computation of the similarities.

In the following discussion we derive an item-oriented approach, but a parallel idea was successfully applied in a user-oriented fashion. Also, we use the term "weights" (w_{ij}) rather than "similarities", to denote the coefficients in the weighted average of neighborhood ratings. This reflects the fact that for a fixed item i, we compute all related w_{ij}'s simultaneously, so that each w_{ij} may be influenced by other neighbors. In what follows, our target is always to estimate the unknown rating by user u of item i, that is r_{ui}.

The first phase of our method is neighbor selection. Among all items rated by u, we select the g most similar to i – $\mathrm{N}(i;u)$, by using a similarity function such as the correlation coefficient, properly shrunk as described in Section 2.2 . The choice of g reflects a tradeoff between accuracy and efficiency; typical values lie in the range of 20–50; see Section 5.

After identifying the set of neighbors, we define the cost function (the "model"), whose minimization determines the weights. We look for the set of interpolation weights $\{w_{ij} | j \in \mathrm{N}(i;u)\}$ that will enable the best prediction rule of the form

$$r_{ui} = \frac{\sum_{j\in \mathrm{N}(i;u)} w_{ij} r_{uj}}{\sum_{j\in \mathrm{N}(i;u)} w_{ij}} \tag{3}$$

We restrict all interpolation weights to be nonnegative, that is, $w_{ij} \geqslant 0$, which allows simpler formulation and, importantly, has proved beneficial in preventing overfitting.

We denote by $\mathrm{U}(i)$ the set of users who rated item i. Certainly, our target user, u, is not within this set. For each user $v \in \mathrm{U}(i)$, denote by $\mathrm{N}(i;u,v)$, the subset of $\mathrm{N}(i;u)$ that includes the items rated by v. In other words, $\mathrm{N}(i;u,v) = \{j \in \mathrm{N}(i;u) | (v,j) \in \mathcal{K}\}$. For each user $v \in \mathrm{U}(i)$, we seek weights that will perfectly interpolate the rating of i from the ratings of the given neighbors:

$$r_{vi} = \frac{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij} r_{vj}}{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij}} \tag{4}$$

Notice that the only unknowns here are the weights (w_{ij}). We can find many perfect interpolation weights for each particular user, which will reconstruct r_{vi} from the r_{vj}'s. After all, we have one equation and $|\mathrm{N}(i;u,v)|$ unknowns. The more interesting problem is to find weights that simultaneously work well for all users. This leads to a least squares problem:

$$\min_w \sum_{v \in \mathrm{U}(i)} \left(r_{vi} - \frac{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij} r_{vj}}{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij}} \right)^2 \tag{5}$$

Expression (5) is unappealing because it treats all squared deviations (or users) equally. First, we want to give more weight to users that rated many items of $\mathrm{N}(i;u)$. Second, we want to give more weight to users who rated items most similar to i. We account for these two considerations by weighting the term associated with user v by $\left(\sum_{j \in \mathrm{N}(i;u,v)} w_{ij}\right)^{\alpha}$, which signifies the relative importance of user v. To simplify subsequent derivation, we chose $\alpha = 2$, so the sum to be minimized is:

$$\min_w \sum_{v \in \mathrm{U}(i)} c_i \left(r_{vi} - \frac{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij} r_{vj}}{\sum_{j \in \mathrm{N}(i;u,v)} w_{ij}} \right)^2 / \sum_{v \in U(i)} c_i \tag{6}$$

where $c_i = \left(\sum_{j \in \mathrm{N}(i;u,v)} w_{ij}\right)^2$.

At this point, we switch to matrix notation. For notational convenience assume that the g items from which we interpolate the rating of i are indexed by $1, \ldots, g$, and arranged within $w \in \mathbb{R}^g$. Let us define two $g \times g$ matrices A and B, where the (j,k)-entry sums over all users in $\mathrm{U}(i)$ that rated both j and k, as follows:

$$A_{jk} = \sum_{v \in \mathrm{U}(i) \text{ s.t. } j,k \in \mathrm{N}(i;u,v)} (r_{vj} - r_{vi})(r_{vk} - r_{vi}) \tag{7}$$

$$B_{jk} = \sum_{v \in \mathrm{U}(i) \text{ s.t. } j,k \in \mathrm{N}(i;u,v)} 1 \tag{8}$$

Using these matrices, we can recast problem (6) in the equivalent form:

$$\min_w \frac{w^T A w}{w^T B w} \tag{9}$$

Notice that both A and B are symmetric positive semidefinite matrices as they correspond to squared sums (numerator and denominator of (6)). It can be shown analytically that the optimal solution of the problem is given by solving the generalized eigenvector equation $Aw = \lambda Bw$ and taking the weights as the eigenvector associated with the smallest eigenvalue. However, we requested that the interpolation weights are nonnegative, so we need to add a non-negativity constraint $w \geqslant 0$. In addition, since $\frac{w^T A w}{w^T B w}$ is invariant under scaling of w, we fix the scale of w obtaining the equivalent problem:

$$\min_w w^T A w \text{ s.t. } w^T B w = 1,\ w \geqslant 0 \tag{10}$$

It is no longer possible to find an analytic solution to the problem in the form of an eigenvector. To solve this problem, one can resort to solving a series of quadratic programs, by linearizing the quadratic constraint $w^T B w = 1$, using the substitution $x \leftarrow Bw$ and the constraint $x^T w = 1$. This typically leads to convergence with 3–4 iterations. In practice, we prefer a slightly modified method that does not require such an iterative process.

3.2.1 A revised model

A major challenge that every CF method faces is the sparseness and the non-uniformity of the rating data. Methods must account for the fact that almost all user-item ratings are unknown, and the known ratings are unevenly distributed so that some users/items are associated with many more ratings than others. The method described above avoids these issues, by relying directly on the known ratings and by weighting the importance of each user according to his support in the data. We now describe an alternative approach that initially treats the data as if it is dense, but accounts for the sparseness by averaging and shrinking.

If all ratings of users that rated i were known, the problem of interpolating the rating of i from ratings of other items – as in Eq. (3) – is related to multivariate regression, where we are looking for weights that best interpolate the vector associated with our item i, from the neighboring items $\mathrm{N}(i;u)$. In this case, problem (5) would form an adequate model, and the user-weighting that led to the subsequent refined formulations of the problem would no longer be necessary. To avoid overfitting, we still restrict the weights to be positive and also fix their scale. Using our previous matrix notation, this leads to the quadratic program:

$$\min_w w^T A w \text{ s.t. } \sum_i w_i = 1,\ w \geqslant 0 \tag{11}$$

Recall that the matrix A was defined in (7). In the hypothetical dense case, when all ratings by person v are known, the condition $j,k \in \mathrm{N}(i;u,v)$ is always true, and each A_{jk} entry is based on the full users set $\mathrm{U}(i)$. However, in the real, sparse case, each A_{jk} entry is based on a different set of users that rated both j and k. As a consequence, different A_{jk} entries might have very different scales depending on the size of their support. We account for this, by replacing the sum that constitutes A_{jk}, with a respective average whose magnitude is not sensitive to the support. In particular, since the support of A_{jk} is exactly B_{jk} as defined in (8), we replace the matrix A, with the matching $g \times g$ matrix A', defined as:

$$A'_{jk} = \frac{A_{jk}}{B_{jk}}$$

This is still not enough for overcoming the sparseness issue. Some A'_{jk} entries might rely on a very low support (low corresponding B_{jk}), so their values are less reliable and should be shrunk towards the average across (j,k)-pairs. Thus, we compute the average entry value of A', which is denoted as $avg = \sum_{j,k} A'_{jk}/(g^2)$, and define the corresponding $g \times g$ matrix $\hat{A}$:

$$\hat{A}_{jk} = \frac{B_{jk} \cdot A'_{jk} + \beta \cdot avg}{B_{jk} + \beta}$$

The non-negative parameter β controls the extent of the shrinkage. We typically use values between 10 and 100. A further improvement is achieved by separating the diagonal and non-diagonal entries, accounting for the fact that the diagonal entries are expected to have an inherently higher average because they sum only non-negative values.

The matrix $\hat{A}$ approximates the matrix A and thus replaces it in problem (11). One could use a standard quadratic programming solver to derive the weights. However, it is beneficial to exploit

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
  $\alpha \leftarrow \frac{r^T r}{r^T Ar}$ *% 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

the simple structure of the constraints. First, we inject the equality constraint into the cost function, so we actually optimize:

$$\min_w w^T \hat{A} w + \lambda(\sum_i w_i - 1)^2 \text{ s.t. } w \geqslant 0 \qquad (12)$$

We construct a $g \times g$ matrix $\hat{C}_{jk} = \hat{A}_{jk} + \lambda$ and a vector $\hat{b} \in \mathbb{R}^g = (\lambda, \lambda, \ldots, \lambda)^T$, so that our problem becomes $\min_w w^T \hat{C} w - 2\hat{b}^T w$ s.t. $w \geqslant 0$. Higher values of the parameter λ, impose stricter compliance to the $\sum_i w_i = 1$ constraint and tend to increase running time. We used $\lambda = 1$, where estimation quality was uniform across a wide range of λ values, and strict compliance to the $\sum_i w_i = 1$ constraint was not beneficial. Now, all constraints have a one sided fixed boundary (namely, 0), reaching a simplified quadratic program that we solve by calling the function NonNegativeQuadraticOpt($\hat{C}, \hat{b}$), which is given in Figure 1. The function is based on the principles of the Gradient Projection method; see, e.g., [12]. The running time of this function depends on g, which is a small constant independent of the magnitude of the data, so it is not the computational bottleneck in the process.

There is a performance price to pay for tailoring the interpolation weights to the specific neighbors set from which we interpolate the query rating. The main computational effort involves scanning the relevant ratings while building the matrices A and B. This makes our neighborhood-based method slower than previous methods, where all weights (or similarities) could have been precomputed. A related approach that vastly improves running time is described in a newer work [2].

3.3 Integrating user-user similarities

The fact that we tailor the computation of the interpolation weights to the given r_{ui} query opens another opportunity. We can weight all users by their similarity to u. That is, we insert the user-user similarities (s_{uv}) into (5), yielding the expression:

$$\min_w \sum_{v\in \mathrm{U}(i)} s_{uv} \left(r_{vi} - \frac{\sum_{j\in \mathrm{N}(i;u,v)} w_{ij} r_{vj}}{\sum_{j\in \mathrm{N}(i;u,v)} w_{ij}} \right)^2 \qquad (13)$$

And consequently, we redefine the A and B matrices as:

$$A_{jk} = \sum_{v\in \mathrm{U}(i) \text{ s.t. } j,k\in \mathrm{N}(i;u,v)} s_{uv}(r_{vj} - r_{vi})(r_{vk} - r_{vi})$$

$$B_{jk} = \sum_{v\in \mathrm{U}(i) \text{ s.t. } j,k\in \mathrm{N}(i;u,v)} s_{uv}$$

Recall that s_{uv} is a similarity measure between users u and v, properly shrunk as described in Section 2.2. Usually, this would be a squared correlation coefficient, or the inverse of the Euclidean distance. Incorporating these similarities into our equations means that the inferred relationship between items depends not only on the identity of the items, but also on the type of user. For example, some users put a lot of emphasis on the actors participating in a movie, while for other users the plot of the movie is much more important than the identity of actors. Certainly, interpolation weights must vary a lot between such different kinds of users. In practice, we found this enhancement of the method significant in improving the prediction accuracy.

To summarize, we build on the intuitive appeal of neighborhood-based approaches, which can easily explain their ratings to the user. Our method differs from previous approaches by being the solution of an optimization problem. Although this leads to more extensive computational effort, it introduces two important aspects that were not addressed in previous works: First, all interpolation weights used for a single prediction are interdependent, allowing consideration of interactions involving many items, not only pairs. Second, item-item interpolation weights depend also on the given user. That is, the relationships between items are not static, but depend on the characteristics of the user. Another benefit of relying on a cost function is the natural emergence of confidence scores for estimating prediction accuracy, as we discuss later in Subsection 5.4.

Our description assumed an item-oriented approach. However, we have used the same approach to produce user-user interpolation weights. This simply requires switching the roles of users and items throughout the algorithm. Despite the fact that the method is computationally intensive, we could apply it successfully to the full Netflix Data on a desktop PC. Our results, which exceeded those of Netflix's Cinematch, are reported in Section 5.

4. FACTORIZATION-BASED ESTIMATION

4.1 Background

Now we move up from the local, neighborhood-based approach, to a "regional" approach where we compute a limited set of features that characterize all users and items. These features allow us to link users with items and estimate the associated ratings. For example, consider user-movie ratings. In this case, regional features might be movie genres. One of the features could measure the fitting into the action genre, while another feature could measure fitting into the comedy genre. Our goal would be to place each movie and each user within these genre-oriented scales. Then, when given a certain user-movie pair, we estimate the rating by the closeness of the features representing the movie and the user.

Ranking users and items within prescribed features, such as movie genres, pertains to content-based methods, which requires additional external information on items and users beyond the past ratings. This might be a very complicated data gathering and cleaning task. However, our goal is to undercover latent features of the given data that explain the ratings, as a surrogate for the external information. This can be achieved by employing matrix factorization techniques such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA); in the context of information retrieval this is widely known as Latent Semantic Indexing [3].

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 portion of the data that might be mere noise.

It is important to understand that the CF context requires a unique application of SVD, due to the fact that most entries of R are unknown. Usually SVD, or PCA, are used for reducing the dimensionality of the data and lessen its representation complexity by providing a succinct description thereof. Interestingly, for CF we are utilizing SVD for an almost opposite purpose – to extend the given data by filling in the values of the unknown ratings. Each unknown rating, r_{ui} is estimated as R^f_{ui}, which is a dot product of the u-th row of P with the i-th row of Q. Consequently, we refer to P as the *user factors* and to Q as the *item factors*.

Beyond the conceptual difference in the way we use SVD, we also face a unique computational difficulty due to the sparsity issue. SVD computation can work only when 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 on adopting SVD- or PCA-based techniques for CF coped with these computational difficulties. Eigenstate [5] uses PCA factorization combined with recursive clustering in order to estimate ratings. Notably, they overcome the sparsity issue by profiling the taste of each user using a universal query that all users must answer. This way, they define a gauge set containing selected representative items for which the ratings of all users must be known. Restricting the rating matrix to the gauge set results in a dense matrix, so conventional PCA techniques can be used for its factorization. However, in practice, obtaining such a gauge set rated by each user, may not be feasible. Moreover, even when obtaining a gauge set is possible, it neglects almost all ratings (all those outside the gauge set), making it likely to overlook much of the given information, thereby degrading prediction accuracy.

An alternative approach is to rely on imputation to fill in missing ratings and make the rating matrix dense. For example, Sarwar et al. [14] filled the missing ratings with a normalized average rating of the related item. Later, Kim and Yum [8] suggested a more involved iterative method, where SVD is iteratively performed while improving the accuracy of imputed values based on results of prior iterations. In a typical CF application, where the imputed ratings significantly outnumber the original ratings, those methods relying on imputation risk distorting the data due to inaccurate imputation. Furthermore, from the computational viewpoint, imputation can be very expensive requiring tone o explicitly deal with each user-item combination, therefore significantly increasing the size of the data . This might be impractical for comprehensive datasets (such as the Netflix data), where the number of users may exceed a million with more than ten thousands items.

4.2 An EM approach to PCA

We propose a method that avoids the need for a gauge set or for imputation, by working directly on the sparse set of known ratings. Since SVD is not well defined on such sparse matrices, we resort to SVD-generalizations that can handle unknown values. In particular we were inspired by Roweis [13], who described an EM algorithm for PCA, to which we now turn.

The common way to compute the PCA of a matrix R is by working on its associated covariance matrix. However, Roweis suggested a completely different approach, which eventually leads to the same PCA factorization (up to orthogonalization). Recall that we try to compute rank-f matrices Q and P that will minimize $\|R - PQ^T\|_F$. Now, we can fix the matrix P as some matrix $\hat{P}$, such that minimization of $\|R-PQ^T\|_F$ would be equivalent to the least squares solution of $R = \hat{P}Q^T$. Analogously, we can fix Q as $\hat{Q}$, so our minimization problem becomes the least squares solution of $R = U\hat{Q}^T$. These least squares problems can be minimized by setting $Q^T = \left(\hat{P}^T\hat{P}\right)^{-1}\hat{P}^T R$ and $P = R\hat{Q}\left(\hat{Q}^T\hat{Q}\right)^{-1}$, leading to an iterative process that alternately recomputes the matrices P and Q, as follows:

$$Q^T \leftarrow \left(P^T P\right)^{-1} P^T R \tag{14}$$

$$P \leftarrow RQ\left(Q^T Q\right)^{-1} \tag{15}$$

It can be shown that the only possible minimum is the global one, so that P and Q must converge to the true SVD subspace [13].

One of the advantages of this iterative SVD computation is its ability to deal with missing values. Roweis proposed to treat the missing values in R as unknowns when obtaining the least squares solution of $R = \hat{P}Q^T$, which is still solvable by standard techniques. This approach actually uses imputation, by filling in the missing values of R as part of the iterative process. This would be infeasible for large datasets where the number of all possible user-item ratings is huge. Therefore, we modify Roweis' idea to enable it to deal with many missing values, while avoiding imputation.

4.3 Our approach

We would like to minimize the squared error between the factors-based estimates and known ratings, which is:

$$\mathrm{Err}(P, Q) \stackrel{\mathrm{def}}{=} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - p_u^T q_i)^2 \tag{16}$$

Here, p_u is the u-th row of P, which corresponds to user u. Likewise, q_i is the i-th row of Q, which corresponds to item i. Similar to Roweis' method, we could alternate between fixing Q and P, thereby obtaining a series of efficiently solvable least squares problems without requiring imputation. Each update of Q or P decreases $\mathrm{Err}(P, Q)$, so the process must converge. However, we recommend using a different process for reasons that will be clarified shortly.

A major question is what would be the optimal value of f, the rank of the matrices Q and P, which represents the number of latent factors we will compute. As f grows, we have more flexibility in minimizing the squared error $\mathrm{Err}(P, Q)$ (16), which must decrease as f increases. However, while $\mathrm{Err}(P, Q)$ measures our ability to recover the known ratings, the unknown ratings are the ones that really interest us. Achieving a low error $\mathrm{Err}(P, Q)$ might involve overfitting the given ratings, while lowering the estimation quality on the unknown ratings. Importantly, our decision to avoid imputation leaves us with fitting a relatively low number of known entries. Therefore, the problem does not allow many degrees of freedom, preventing the use of more than a very few factors. In fact, our experience with the process showed that using more than two factors ($f > 2$) degrades estimation quality. This is an undesirable situation, as we want to benefit by increasing the number of factors, thereby explaining more latent aspects of the data. However, we find that we can still treat only the known entries, but accompany the process with shrinkage to alleviate the overfitting problem.

The key to integrating shrinkage is computing the factors one by one, while shrinking the results after each additional factor is computed. This way, we increase the number of factors, while gradually limiting their strength. To be more specific, let us assume that we have already computed the first $f - 1$ factors, which are columns

$1, \ldots f-1$ of matrices P and Q. We provide below a detailed pseudo code for computing the next factor, which is the f-th column of matrices P and Q:

ComputeNextFactor(Known ratings: r_{ui},
User factors $Q_{m\times f}$, Item factors $Q_{n\times f}$)
% Compute f-th column of matrices P and Q to fit given ratings
% Columns $1, \ldots, f-1$ of P and Q were already computed

Constants: $\alpha = 25,\ \epsilon = 10^{-4}$
% Compute residuals–portion not explained by previous factors
for each given rating r_{ui} **do**
$\text{res}_{ui} \leftarrow r_{ui} - \sum_{l=1}^{f-1} P_{ul}Q_{il}$
$\text{res}_{ui} \leftarrow \frac{n_{ui}\text{res}_{ui}}{n_{ui}+\alpha f}$ *% shrinkage*

% Compute the f-th factor for each user and item by solving
% many least squares problems, each with a single unknown
while $\text{Err}(P^{\text{new}}, Q^{\text{new}})/\text{Err}(P^{\text{old}}, Q^{\text{old}}) < 1-\epsilon$
for each user $u = 1, \ldots, n$ **do**
$P_{uf} \leftarrow \frac{\sum_{i:(u,i)\in\mathcal{K}} \text{res}_{ui}Q_{if}}{\sum_{i:(u,i)\in\mathcal{K}} Q_{if}^2}$
for each item $i = 1, \ldots, m$ **do**
$Q_{if} \leftarrow \frac{\sum_{u:(u,i)\in\mathcal{K}} \text{res}_{ui}P_{uf}}{\sum_{u:(u,i)\in\mathcal{K}} P_{uf}^2}$
return P, Q

This way, we compute f factors by calling the function ComputeNextFactor f times, with increasing values of f. A well tuned shrinkage should insure that adding factors cannot worsen the estimation. The shrinkage we performed here, $\text{res}_{ui} \leftarrow \frac{n_{ui}\text{res}_{ui}}{n_{ui}+\alpha f}$, reduces the magnitude of the residual according to two elements. The first element is the number of already computed factors - f. As we compute more factors, we want them to explain lower variations of the data. The second element is the support behind the rating r_{ui}, which is denoted by n_{ui}. This support is the minimum between the number of ratings by user u and the number of users that rated item i. As the support grows, we have more information regarding the involved user and item, and hence we can exploit more factors for explaining them. By using shrinkage we observed improved estimation as factors were added. However, estimation improvement levels off beyond 30–50 factors and becomes insignificant; see Section 5. The second part of the function computes the f-th factor, by alternating between fixing item-values and user-values. We can conveniently deal with each user/item separately, thus the resulting least squares problem is trivial involving only one variable. The iterative process converges when no significant improvement of $\text{Err}(P, Q)$ is achieved. Typically, it happens after 3–5 iterations.

At the end of the process we obtain an approximation of all ratings in the form of a matrix product PQ^T. This way, each rating r_{ui} is estimated as the inner product of the f factors that we learned for u and i, that is $p_u^T q_i$. A major advantage of such a regional, factorization-based approach is its computational efficiency. The computational burden lies in an offline, preprocessing step where all factors are computed. The actual, online rating prediction is done instantaneously by taking the inner product of two length-f vectors. Moreover, since the factors are computed by an iterative algorithm, it is easy to adapt them to changes in the data such as addition of new ratings, users, or items. We can always train the relevant variables by running a few additional restricted iterations of the process updating only the relevant variables. While the factorization-based approach is significantly faster than our neighborhood-based approach, it is very competitive in terms of estimation quality, as we will show in Section 5. A further improvement in estimation quality is obtained by combining the local information provided by neighborhood-based approaches, with the regional information provided by the factorization-based approach. We turn now to one way to achieve this.

4.4 Neighborhood-aware factorization

The factorization-based approach describes a user u as a fixed linear combination of the f movie factors. That is, the profile of user u is captured by the vector $p_u \in \mathbb{R}^f$, such that his/her ratings are given by $p_u^T Q^T$. Interestingly, we can improve estimation quality by moving from a fixed linear combination (p_u) to a more adaptive linear combination that changes as a function of the item i to be rated by u. In other words, when estimating r_{ui} we first compute a vector $p_u^i \in \mathbb{R}^k$ – depending on both u and i – and then estimate r_{ui} as $\left(p_u^i\right)^T q_i$. The construction of p_u^i is described below.

The user vector p_u was previously computed such as to minimize, up to shrinkage, the squared error associated with u:

$$\sum_{j:(u,j)\in\mathcal{K}} (r_{uj} - p_u^T q_j)^2 \tag{17}$$

Now, if we know that the specific rating to be estimated is r_{ui}, we can tilt the squared error to overweight those items similar to i, obtaining the error function:

$$\sum_{j:(u,j)\in\mathcal{K}} s_{ij}(r_{uj} - p_u^T q_j)^2 \tag{18}$$

Recall that s_{ij} is a shrunk similarity measure between items i and j. We use here an inverse power of the Euclidean distance, but other similarity measures can be applied as well. The minimizer of error function (18) – up to shrinkage – would be p_u^i, which characterizes user u within i's neighborhood. It is still crucial to perform a factor-by-factor computation, while shrinking during the process. Therefore, we compute the f components of p_u^i one by one, as in the following algorithm:

NeighborhoodAdaptiveUserFactors(Known ratings: r_{ui}, user u,
item i, item factors $Q_{m\times f}$)
% Compute f factors associated with user u and adapted to item i
Const $\alpha = 25$
% Initialize residuals – portion not explained by previous factors
for each given rating r_{uj} **do**
$\text{res}_j \leftarrow r_{uj}$
% Factor-by-factor sweep:
for $l = 1, \ldots, f$ **do**
$p_u^i[l] \leftarrow \frac{\sum_{j:(u,j)\in\mathcal{K}} s_{ij}\text{res}_j Q_{jl}}{\sum_{j:(u,j)\in\mathcal{K}} s_{ij}Q_{jl}^2}$
for each given rating r_{uj} **do**
$\text{res}_j \leftarrow \text{res}_j - p_u^i[l]\cdot Q_{jl}$
$\text{res}_j \leftarrow \frac{n_{uj}\text{res}_j}{n_{uj}+\alpha l}$ *% shrinkage*
return $p_u^i = (p_u^i[1], p_u^i[2], \ldots, p_u^i[f])^T$

This introduction of neighborhood-awareness into the regional-oriented, factorization-based method significantly improves the quality of the results, compared to neighborhood-only, or regional-only approaches (see Section 5). Moreover, typically all item-item similarities (the s_{ij} values) are precomputed and stored for quick retrieval. This enables a very quick execution of the function NeighborhoodAdaptiveUserFactors, which contains no iterative component. Overall running time is only slightly more than for the original factorization-based approach.

A complementary step would be to recompute the item factors by making them neighborhood-aware. That is, replacing q_i with q_i^u, which can be computed analogously to p_u^i by accounting for similarities of other users to user u. Consequently, the rating r_{ui} is estimated by $\left(p_u^i\right)^T q_i^u$. This results in an additional improvement in estimation accuracy. Moreover, it naturally integrates item-item similarities and user-user similarities into a single estimate, by employing item-item similarities when computing the user- factors, and user-user similarities when computing the item-factors. However, making the item factors neighborhood-aware typically requires an extensive additional computational effort, when user-user similarities are not stored due to the large number of users.

5. EXPERIMENTAL STUDY

We evaluated our algorithms on the Netflix data [10]. As mentioned before, this dataset is based on more than 100 million ratings of movies performed by anonymous Netflix customers. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset. The Netflix data is currently the subject of substantial analysis, and thus is likely to become a standard benchmark for CF algorithms.

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 Probe set is a part of the training data and its true ratings are provided. We do not know the true ratings for the Quiz set, which are held by Netflix in order to evaluate entries. Importantly, these two test sets seem to be distributed equally across users and dates, so that the Probe data serves as a good "hold-out" sample on which to calibrate models. The test sets contain many more ratings by users that do not rate much and are harder to predict. In a way, these datasets 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.

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. In the experiments that follow we focus on the Probe set, since the ratings are known. At the end of the section, we discuss results for the Quiz set. When results are reported for the Quiz set, we subsume the Probe set into our training data, which improves prediction accuracy.

5.1 Probe Set Results for the Local Approach

We begin with evaluating the neighborhood-based estimation discussed in Section 3. We concentrate on item-item (or, movie-movie in this case) interpolation which we found to deliver better results than the alternative user-user interpolation, confirming the findings of Sarwar et al. [15]. Our method is compared with common item-item interpolation based on Pearson correlation coefficients. While the literature suggests many flavors of correlation-based interpolation, we found that two ingredients are very beneficial here. The first is working on residuals that remain after removing global effects (Subsection 2.3). The second is working with shrunk correlations instead of the original ones (Subsection 2.2). Therefore, to make a fair comparison, we applied these two techniques to the competing correlation-based method, significantly improving its performance. It is important to mention that our results use exactly the same neighborhoods as the correlation-based results. That is, for both methods we select as neighbors the available movies of highest shrunk Pearson correlation with the current movie. Our method differs only in calculating the interpolation weights for the neighboring ratings. In Figure 2 we compare the methods' performance against varying neighborhood sizes ranging from 10 to 50.

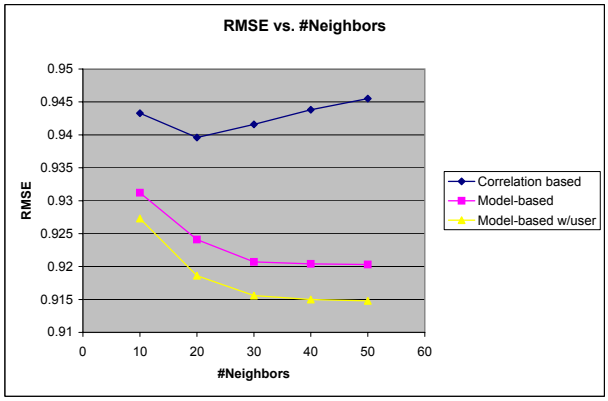

Figure 2: Comparison of three neighborhood-based approaches: a common method using shrunk Pearson correlations, our neighborhood-based approach by itself and optionally extended with user-user similarities. Performance is measured by RMSE, where lower RMSEs indicate better prediction accuracy. RMSE is shown as a function of varying neighborhood sizes on Probe set. All methods use the same neighborhood sets, differing only in the weights given to the neighbors.

Our model-based weights consistently deliver better results than the correlation-based weights. Performance of correlation-based weights peaks at around 20 neighbors, and then starts declining as neighborhood size grows. In contrast, our methods continue to improve (at a moderating pace) as neighborhoods grow. We attribute this difference to the fact that our weights are able to reflect interdependencies among neighbors, which pairwise correlation coefficients cannot utilize. As neighborhood size grows, the number of possible interdependencies among neighbors increases quadratically; hence, more information is overlooked by correlation-based weights. For example, correlation-based approaches, or any pairwise approach for the matter, would be unforgiving to using unneeded neighbors, whose correlation is still positive (high enough to get placed among closest neighbors), whereas our model can set unneeded weights to as low as zero, if their contribution is covered by other neighbors.

An important feature of our model is the ability to integrate user-user similarities within the computation of item-item weights (Subsection 3.3). We studied the added contribution of this feature by running our model twice, once being user-aware, accounting for user-user similarities, and once without this feature. As shown in Figure 2, accounting for user similarity typically lowers the RMSE by about 0.0060; e.g., using 50 neighbors, the user-aware model results in RMSE=0.9148, versus RMSE=0.9203 without user awareness.

As indicated above, all methods were run on residuals after global effects (shrunk user mean, movie mean and various time effects) were removed. This is an important part of the modelling, but it turns out much more so for the correlation-based approach. For our method, removing these global effects was beneficial, e.g., lowering the RMSE for 50 neighbors from 0.9271 to 0.9148. However, it was more crucial for the correlation-based methods, which delivered very weak results with RMSEs above 0.99 when applied

without removing global effects. Why might this be? Before removing global effects correlations between movies are significantly higher than after removing them. Hence, we speculate that correlation coefficients, which isolate movie pairs, fail to account for strong dependencies between movies coexisting in the neighbor set. Capturing such interdependencies among neighbors is one of the major motivations for our method. However, after removing the global effects, correlations among movies become lower and hence the correlation-based methods lose far less information by isolating movie pairs. Consequently, we strongly recommend applying correlation-based methods only after removing most global effects and decorrelating items.

In terms of running time, correlation based methods possess a significant advantage over our method. After precomputing all shrunk correlations, movie-movie interpolation could predict the full Probe set in less than 5 minutes on a Pentium 4 PC. In contrast, using our more involved weights took more than a day to process the Probe set. In a newer work [2] we explain how to eliminate this runing time gap.

5.2 Probe Set Results for Regional Approach

Now we move to the factorization approach, whose results are presented in Figure 3. The pure-regional model, which does not account for local, neighborhood-based relationships, achieved RMSEs that are slightly worse (i.e., higher) than the aforementioned movie-movie interpolation, and is actually on the same level that we could achieve from user-user interpolation. However, unlike our neighborhood-based models, the factorization-based model allows a very rapid computation, processing the full Probe set in about three minutes.

Accuracy is significantly improved by integrating neighborhood relationships into the factorization model, as explained in Subsection 4.4. This model, which integrates local and regional views on the data, outperforms all our models, achieving RMSE of 0.9090 on the Probe set. Importantly, this model is still very fast, and can complete the whole Probe set in about five minutes when using precomputed movie-movie similarities. A modest additional improvement of around 0.0015 (not shown in the figure), is achieved by integrating user-user similarities as well, but here computational effort rises significantly in the typical case where user-user similarities cannot be precomputed and stored. For these factorization-based methods the main computational effort lies in the preprocessing stage when factors are computed. This stage took about 3 hours on our Pentium 4 PC. Adding factors should improve explaining the data and thus reduce RMSE. This was our general experience, as indicated in the figure, except for a slow RMSE increase at the regional-only approach when using more than 40 factors. This probably indicates that our shrinkage mechanism was not tuned well.

5.3 Results on the Quiz Set

In Figure 4, we present our results for the Quiz set, the set held out by Netflix in order to evaluate entries of the Netflix Prize contest. Our final result, which yielded a RMSE of 0.8922 (6.22% improvement over Netflix Cinematch's 0.9514 result), was produced by combining the results of three different approaches: The first two are local ones, which are based on user-user and on movie-movie neighborhood interpolation. The third approach, which produces the lowest RMSE, was factorization-based enriched with local neighborhood information. Note that user-user interpolation is the weakest of the three, resulting in RMSE=0.918, which is still a 3.5% improvement over the commercial Cinematch system. The combination of the three results involved accounting for confidence scores, a topic which we briefly touch now.

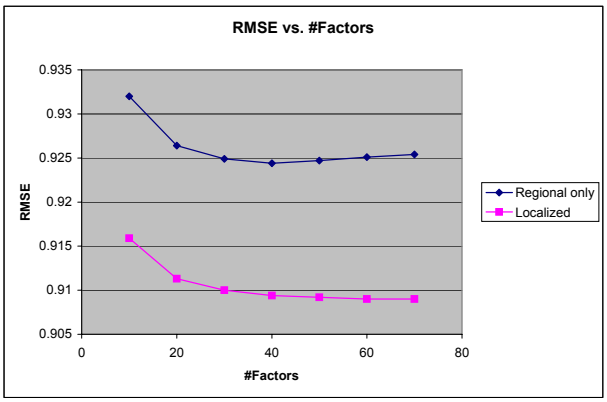

Figure 3: RMSEs of regional, factorization-based methods on Probe data, plotted against varying number of factors. The extended model, which includes also local item-item information, achieves a significant further decrease of the RMSE.

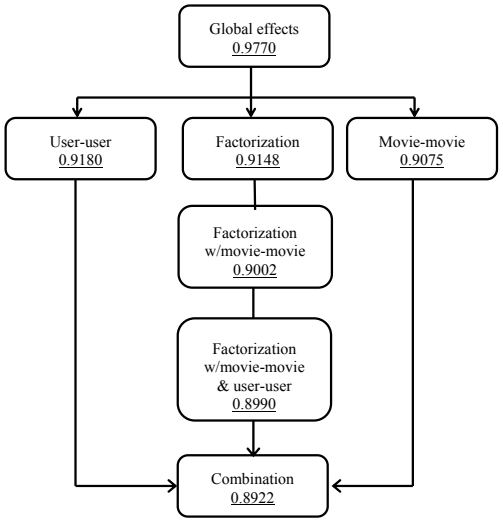

Figure 4: Our results (measured by RMSE) on the Quiz set. For comparison, the Netflix Cinematch reported result is RMSE=0.9514.

5.4 Confidence scores

The fact that our models are based on minimizing cost functions allows us to assess the quality of each prediction, associating it with a *confidence score*. More specifically, considering the neighborhood-based methods, each score is predicted using weights that minimize problem (9). Consequently, we use the attained value of (9), $w^T Aw/w^T Bw$, as a suitable confidence score. Low values correspond to weights that could fit the training data well. On the other hand, higher values mean that the given weights were not very successful on the training data, and thus they are likely to perform poorly also for unknown ratings. Turning to factorization-based approaches, we can assess the quality of a user factor by (17), or by (18) in the neighborhood-aware case. Analogously, we also compute confidence scores for each movie factor. This way, when predicting r_{ui} by factorization, we accompany the rating with two confidence scores – one related to the user u factors, and the other to item i factors – or by a combination thereof.

There are two important applications to confidence scores. First, when actually recommending products, we would like to choose not just the ones with high predicted ratings, but also to account for the quality of the predictions. This is because we have more faith in predictions that are associated with low (good) confidence scores.

This suggests a possible shrinkage of predicted ratings, to adjust for their quality. The second application of confidence scores is for joining the results of different algorithms. Different algorithms will assign varying confidence scores to each user-item pair. Therefore, when combining results, we would like to account for the confidence score associated with the results, overweighting ratings associated with better confidence scores.

Figure 5 shows how RMSEs of predicted ratings vary with confidence scores. We report results for the Probe set by using movie-movie interpolation (RMSE=0.9148). Ratings were split into deciles based on their associated confidence score. Clearly, RMSEs of deciles rise monotonically with confidence scores. Notice that top decile can be estimated quite accurately with an associated RMSE of around 0.6, while the bottom decile is much harder to estimate doubling the RMSE to above 1.2.

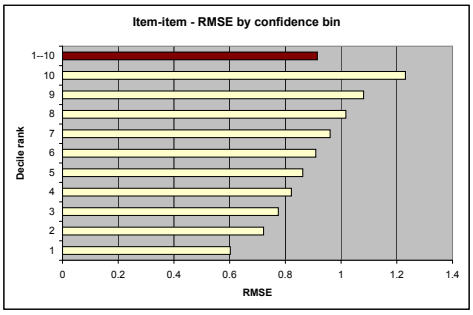

Figure 5: Dependency of actual prediction accuracy (measured by RMSE) on confidence scores is revealed by partitioning Probe results into ten deciles according to confidence scores

6. DISCUSSION

In this work, we designed new collaborative filtering methods based on models that strive to minimize quadratic errors, and demonstrated strong performance on a large, real-world dataset. The following components of our method were all crucial to its good performance: 1) removing "global" components up front and focusing our modelling on the residuals, 2) using shrinkage to prevent overfitting of the large number of parameters, 3) incorporating interactions among the users and movies by fitting a model that jointly optimizes parameter estimates, and 4) incorporating local, neighborhood based analysis into a regional, factorization model.

Formulating the methods around formal models allowed us to better understand their properties and to introduce controllable modifications to the models for evaluating possible extensions. Importantly, we have found that extending the models to combine multiple facets of the data, such as user-similarity with item-similarity or local-features with higher-scale features, is a key component in improving prediction accuracy. Another benefit of our error-based model is the natural definition of confidence scores that accompany the computed ratings and essentially "predict the quality of the predictions". In a future work, we would like to study the integration of these confidence scores with the predicted ratings in order to provide better recommendation to the user. Other future research ideas include non-linear extensions to our linear models, especially to the factorization model which might be extended by such non-linear functions.

We greatly benefited from the recently introduced Netflix data, which opens new opportunities to the design and evaluation of CF algorithms. None of our models use any information about the content (actors, genres, etc) and it would be interesting to devise models to incorporate such data into our CF-based model. Since our factorization attempts to indirectly model this external information, we'd like to see how much benefit that data would provide.

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

[2] R. Bell and Y. Koren, "Improved Neighborhood-based Collaborative Filtering", submitted, 2007.

[3] S. Deerwester, S. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the Society for Information Science* **41** (1990), 391–407.

[4] 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.

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

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.

[7] 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.

[8] D. Kim and B. Yum, "Collaborative Filtering Based on Iterative Principal Component Analysis", *Expert Systems with Applications* **28** (2005), 823–830.

[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] Netflix prize - `www.netflixprize.com`.

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

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

[13] S. Roweis, "EM Algorithms for PCA and SPCA", *Advances in Neural Information Processing Systems 10*, pp. 626–632, 1997.

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

[15] 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.

[16] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso", *Journal of the Royal Statistical Society B* **58** (1996).

[17] 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.