Chapter 3
Advances in Collaborative Filtering

Yehuda Koren and Robert Bell

3.1 Introduction

Collaborative filtering recommender system (CF) methods produce user specific recommendations of items based on patterns of ratings or usage (e.g., purchases) without need for exogenous information about either items or users. While well established methods work adequately for many purposes, we present several recent extensions available to analysts who are looking for the best possible recommendations.

The Netflix Prize competition that began in October 2006 has fueled much recent progress in the field of collaborative filtering. For the first time, the research

---

This article includes copyrighted materials, which were reproduced with permission of ACM and IEEE. The original articles are:

R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, pp. 43–52, © 2007 IEEE. Reprinted by permission.

Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model", *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, © 2008 ACM, Inc. Reprinted by permission. http://doi.acm.org/10.1145/1401890.1401944.

Y. Koren, "Collaborative Filtering with Temporal Dynamics." *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 447–456, © 2009 ACM, Inc. Reprinted by permission. http://doi.acm.org/10.1145/1557019.1557072.

Y. Koren (✉)
Google Research, Mountain View, CA, USA
e-mail: yehudako@gmail.com

R. Bell
AT&T Labs – Research, Middletown, NJ, USA
e-mail: rbell@research.att.com

© Springer Science+Business Media New York 2015
F. Ricci et al. (eds.), *Recommender Systems Handbook*,
DOI 10.1007/978-1-4899-7637-6_3

community gained access to a large-scale, industrial strength data set of 100 million movie ratings—attracting thousands of scientists, students, engineers and enthusiasts to the field. The nature of the competition has encouraged rapid development, where innovators built on each generation of techniques to improve prediction accuracy. Because all methods are judged by the same rigid yardstick on common data, the evolution of more powerful models has been especially efficient.

Recommender systems rely on various types of input. Most convenient is high quality *explicit feedback*, where users directly report on their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons.

Because explicit feedback is not always available, some recommenders infer user preferences from the more abundant *implicit feedback*, which indirectly reflects opinion through observing user behavior [20]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user who purchased many books by the same author probably likes that author. This chapter focuses on models suitable for explicit feedback. Nonetheless, we recognize the importance of implicit feedback, an especially valuable information source for users who do not provide much explicit feedback. Hence, we show how to address implicit feedback within the models as a secondary source of information.

In order to establish recommendations, CF systems need to relate two fundamentally different entities: items and users. There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: *the neighborhood approach* and *latent factor models*. Neighborhood methods focus on relationships between items or, alternatively, between users. An item-item approach models the preference of a user to an item based on ratings of similar items by the same user. Latent factor models, such as matrix factorization (aka, SVD), comprise an alternative approach by transforming both items and users to the same latent factor space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback.

Producing more accurate prediction methods requires deepening their foundations and reducing reliance on arbitrary decisions. In this chapter, we describe a variety of recent improvements to the primary CF modeling techniques. Yet, the quest for more accurate models goes beyond this. At least as important is the identification of all the signals, or features, available in the data. Conventional techniques address the sparse data of user-item ratings. Accuracy significantly improves by also utilising other sources of information. One prime example includes all kinds of temporal effects reflecting the dynamic, time-drifting nature of user-item interactions. No less important is listening to hidden feedback such as which items users chose to rate (regardless of rating values). Rated items are not selected at random, but rather reveal interesting aspects of user preferences, going beyond the numerical values of the ratings.

Section 3.3 surveys matrix factorization techniques, which combine implementation convenience with a relatively high accuracy. This has made them the preferred technique for addressing the largest publicly available dataset—the Netflix data.

This section describes the theory and practical details behind those techniques. In addition, much of the strength of matrix factorization models stems from their natural ability to handle additional features of the data, including implicit feedback and temporal information. This section describes in detail how to enhance matrix factorization models to address such features.

Section 3.4 turns attention to neighborhood methods. The basic methods in this family are well known, and to a large extent are based on heuristics. Some recently proposed techniques address shortcomings of neighborhood techniques by suggesting more rigorous formulations, thereby improving prediction accuracy. We continue at Sect. 3.5 with a more advanced method, which uses the insights of common neighborhood methods, with global optimization techniques typical of factorization models. This method allows lifting the limit on neighborhood size, and also addressing implicit feedback and temporal dynamics. The resulting accuracy is close to that of matrix factorization models, while offering some practical advantages.

Pushing the foundations of the models to their limits reveals surprising links among seemingly unrelated techniques. We elaborate on this in Sect. 3.6 to show that, at their limits, user-user and item-item neighborhood models may converge to a single model. Furthermore, at that point, both become equivalent to a simple matrix factorization model. The connections reduce the relevance of some previous distinctions such as the traditional broad categorization of matrix factorization as "model based" and neighborhood models as "memory based".

3.2 Preliminaries

We are given ratings for m users (aka customers) and n items (aka products). We reserve special indexing letters to distinguish users from items: for users u, v, and for items i, j, l. A rating r_{ui} indicates the preference by user u of item i, where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}.

The scalar t_{ui} denotes the time of rating r_{ui}. One can use different time units, based on what is appropriate for the application at hand. For example, when time is measured in days, then t_{ui} counts the number of days elapsed since some early time point. Usually the vast majority of ratings are unknown. For example, in the Netflix data 99 % of the possible ratings are missing because a user typically rates only a small portion of the movies. The (u, i) pairs for which r_{ui} is known are stored in the set $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$. Each user u is associated with a set of items denoted by $\mathrm{R}(u)$, which contains all the items for which ratings by u are available. Likewise, $\mathrm{R}(i)$ denotes the set of users who rated item i. Sometimes, we also use a set denoted by $\mathrm{N}(u)$, which contains all items for which u provided an implicit preference (items that he rented/purchased/watched, etc.).

Models for the rating data are learnt by fitting the previously observed ratings. However, our goal is to generalize those in a way that allows us to predict future, unknown ratings. Thus, caution should be exercised to avoid overfitting the observed data. We achieve this by regularizing the learnt parameters, whose magnitudes are penalized. Regularization is controlled by constants which are denoted as: $\lambda_1, \lambda_2, \ldots$ Exact values of these constants are determined by cross validation. As they grow, regularization becomes heavier.

3.2.1 Baseline Predictors

CF models try to capture the interactions between users and items that produce the different rating values. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. A principal example is that typical CF data exhibit large user and item biases—i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others.

We will encapsulate those effects, which do not involve user-item interaction, within the *baseline predictors* (also known as *biases*). Because these predictors tend to capture much of the observed signal, it is vital to model them accurately. Such modeling enables isolating the part of the signal that truly represents user-item interaction, and subjecting it to more appropriate user preference models.

Denote by μ the overall average rating. A baseline prediction for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \tag{3.1}$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline predictor for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline predictor for Titanic's rating by Joe would be 3.9 stars by calculating $3.7-0.3+0.5$. In order to estimate b_u and b_i one can solve the least squares problem

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1\Big(\sum_u b_u^2 + \sum_i b_i^2\Big).$$

Here, the first term $\sum_{(u,i)\in\mathcal{K}}(r_{ui} - \mu + b_u + b_i)^2$ strives to find b_u's and b_i's that fit the given ratings. The regularizing term—$\lambda_1(\sum_u b_u^2 + \sum_i b_i^2)$—avoids overfitting by penalizing the magnitudes of the parameters. This least square problem can be solved fairly efficiently by the method of stochastic gradient descent (described in Sect. 3.3.1).

For the Netflix data the mean rating (μ) is 3.6. As for the learned user biases (b_u), their average is 0.044 with standard deviation of 0.41. The average of their absolute values ($|b_u|$) is: 0.32. The learned item biases (b_i) average to -0.26 with a standard deviation of 0.48. The average of their absolute values ($|b_i|$) is 0.43.

An easier, yet somewhat less accurate way to estimate the parameters is by decoupling the calculation of the b_i's from the calculation of the b_u's. First, for each item i we set

$$b_i = \frac{\sum_{u \in \mathrm{R}(i)} (r_{ui} - \mu)}{\lambda_2 + |\mathrm{R}(i)|}.$$

Then, for each user u we set

$$b_u = \frac{\sum_{i \in \mathrm{R}(u)} (r_{ui} - \mu - b_i)}{\lambda_3 + |\mathrm{R}(u)|}.$$

Averages are shrunk towards zero by using the regularization parameters, λ_2, λ_3, which are determined by cross validation. Typical values on the Netflix dataset are: $\lambda_2 = 25, \lambda_3 = 10$.

In Sect. 3.3.3.1, we show how the baseline predictors can be improved by also considering temporal dynamics within the data.

3.2.2 *The Netflix Data*

In order to compare the relative accuracy of algorithms described in this chapter, we evaluated all of them on the Netflix data of more than 100 million date-stamped movie ratings performed by anonymous Netflix customers between November, 1999 and December 2005 [5]. Ratings are integers ranging between 1 and 5. The data spans 17,770 movies rated by over 480,000 users. Thus, on average, a movie receives 5600 ratings, while a user rates 208 movies, with substantial variation around each of these averages. To maintain compatibility with results published by others, we adopt some standards that were set by Netflix. First, quality of the results is usually measured by the root mean squared error (RMSE):

$$\sqrt{\sum_{(u,i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|}$$

a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. (Consider Chap. 8 for a comprehensive survey of alternative evaluation metrics of recommender systems.)

We report results on a test set provided by Netflix (also known as the Quiz set), which contains over 1.4 million recent ratings. Compared with the training data, the test set contains many more ratings by users that do not rate much and are therefore harder to predict. In a way, this represents real requirements for a CF system, which needs to predict new ratings from older ones, and to equally address all users, not just the heavy raters.

The Netflix data is part of the Netflix Prize competition, where the benchmark is Netflix's proprietary system, Cinematch, which achieved a RMSE of 0.9514 on the test set. The grand prize was awarded to a team that managed to drive this RMSE below 0.8563 (10 % improvement) after almost 3 years of extensive efforts. Achievable RMSE values on the test set lie in a quite compressed range, as evident by the difficulty to win the grand prize. Nonetheless, there is evidence that small improvements in RMSE terms can have a significant impact on the quality of the top few presented recommendations [16, 17].

3.2.3 Implicit Feedback

This chapter is centered on explicit user feedback. Nonetheless, when additional sources of implicit feedback are available, they can be exploited for better understanding user behavior. This helps to combat data sparseness and can be particularly helpful for users with few explicit ratings. We describe extensions for some of the models to address implicit feedback.

For a dataset such as the Netflix data, the most natural choice for implicit feedback would probably be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. For other datasets, browsing or purchase history could be used as implicit feedback. However, such data is not available to us for experimentation. Nonetheless, a less obvious kind of implicit data does exist within the Netflix dataset. The dataset does not only tell us the rating values, but also *which* movies users rate, regardless of *how* they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This creates a binary matrix, where "1" stands for "rated", and "0" for "not rated". While this binary data may not be as informative as other independent sources of implicit feedback, incorporating this kind of implicit data does significantly improves prediction accuracy. The benefit of using the binary data is closely related to the fact that ratings are not missing at random; users deliberately choose which items to rate (see Marlin et al. [19]).

3.3 Matrix Factorization Models

Latent factor models approach collaborative filtering with the holistic goal to uncover latent features that explain observed ratings; examples include pLSA [14], neural networks [22], Latent Dirichlet Allocation [7], and models that are induced by factorization of the user-item ratings matrix (also known as SVD-based models). Recently, matrix factorization models have gained popularity, thanks to their attractive accuracy and scalability.

In information retrieval, SVD is well established for identifying latent semantic factors [9]. However, applying SVD to explicit ratings in the CF domain raises difficulties due to the high portion of missing values. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works relied on imputation [15, 24], which fills in missing ratings and makes the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [4, 6, 10, 16, 21, 22, 26] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model.

In this section we describe several matrix factorization techniques, with increasing complexity and accuracy. We start with the basic model—"SVD". Then, we show how to integrate other sources of user feedback in order to increase prediction accuracy, through the "SVD++ model". Finally we deal with the fact that customer preferences for products may drift over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste. This leads to a factor model that addresses temporal dynamics for better tracking user behavior.

3.3.1 SVD

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or "quirkiness"; or completely uninterpretable dimensions.

Accordingly, each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with a vector $p_u \in \mathbb{R}^f$. For a given item i, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors (again, these may be positive or negative).

The resulting dot product,[1] $q_i^T p_u$, captures the interaction between user u and item i—i.e., the overall interest of the user in characteristics of the item. The final rating is created by also adding in the aforementioned baseline predictors that depend only on the user or item. Thus, a rating is predicted by the rule

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \,. \tag{3.2}$$

In order to learn the model parameters (b_u, b_i, p_u and q_i) we minimize the regularized squared error

$$\min_{b_*,q_*,p_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda_4(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \,.$$

The constant λ_4, which controls the extent of regularization, is usually determined by cross validation. Minimization is typically performed by either stochastic gradient descent or alternating least squares.

Alternating least squares techniques rotate between fixing the p_u's to solve for the q_i's and fixing the q_i's to solve for the p_u's. Notice that when one of these is taken as a constant, the optimization problem is quadratic and can be optimally solved; see [2, 4].

An easy stochastic gradient descent optimization was popularized by Funk [10] and successfully practiced by many others [16, 21, 22, 26]. The algorithm loops through all ratings in the training data. For each given rating r_{ui}, a prediction ($\hat{r}_{ui}$) is made, and the associated prediction error $e_{ui} \overset{\text{def}}{=} r_{ui} - \hat{r}_{ui}$ is computed. For a given training case r_{ui}, we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda_4 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_4 \cdot p_u)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters: $\gamma = 0.005, \lambda_4 = 0.02$. Henceforth, we dub this method "SVD".

A general remark is in place. One can expect better accuracy by dedicating separate learning rates (γ) and regularization (λ) to each type of learned parameter. Thus, for example, it is advised to employ distinct learning rates to user biases, item biases and the factors themselves. A good, intensive use of such a strategy is described in Takács et al. [27]. When producing exemplary results for this chapter, we did not use such a strategy consistently, and in particular many of the given constants are not fully tuned.

[1] Recall that the dot product between two vectors $x, y \in \mathbb{R}^f$ is defined as: $x^T y = \sum_{k=1}^{f} x_k \cdot y_k$.

3.3.2 SVD++

Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. This is especially helpful for those users that provided much more implicit feedback than explicit one. As explained earlier, even in cases where independent implicit feedback is absent, one can capture a significant signal by accounting for which items users rate, regardless of their rating value. This led to several methods [16, 21, 23] that modeled a user factor by the identity of the items he/she has rated. Here we focus on the SVD++ method [16], which was shown to offer accuracy superior to SVD.

To this end, a second set of item factors is added, relating each item i to a factor vector $y_i \in \mathbb{R}^f$. Those new item factors are used to characterize users based on the set of items that they rated. The exact model is as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right) \tag{3.3}$$

The set $\mathrm{R}(u)$ contains the items rated by user u.

Now, a user u is modeled as $p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j$. We use a free user-factors vector, p_u, much like in (3.2), which is learnt from the given explicit ratings. This vector is complemented by the sum $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j$, which represents the perspective of implicit feedback. Since the y_j's are centered around zero (by the regularization), the sum is normalized by $|\mathrm{R}(u)|^{-\frac{1}{2}}$, in order to stabilize its variance across the range of observed values of $|\mathrm{R}(u)|$.

Model parameters are determined by minimizing the associated regularized squared error function through stochastic gradient descent. We loop over all known ratings in $\mathcal{K}$, computing:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j) - \lambda_6 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$
- $\forall j \in \mathrm{R}(u) : y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |\mathrm{R}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_6 \cdot y_j)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters: $\gamma = 0.007$, $\lambda_5 = 0.005$, $\lambda_6 = 0.015$. It is beneficial to decrease step sizes (the γ's) by a factor of 0.9 after each iteration. The iterative process runs for around 30 iterations until convergence.

Several types of implicit feedback can be simultaneously introduced into the model by using extra sets of item factors. For example, if a user u has a certain kind of implicit preference to the items in $\mathrm{N}^1(u)$ (e.g., she rented them), and a different type of implicit feedback to the items in $\mathrm{N}^2(u)$ (e.g., she browsed them), we could use the model

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |\mathrm{N}^1(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^1(u)} y_j^{(1)} + |\mathrm{N}^2(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^2(u)} y_j^{(2)} \right). \tag{3.4}$$

The relative importance of each source of implicit feedback will be automatically learned by the algorithm by its setting of the respective values of model parameters.

3.3.3 Time-Aware Factor Model

The matrix-factorization approach lends itself well to modeling temporal effects, which can significantly improve its accuracy. Decomposing ratings into distinct terms allows us to treat different temporal aspects separately. Specifically, we identify the following effects that each vary over time: (1) user biases $b_u(t)$, (2) item biases $b_i(t)$, and (3) user preferences $p_u(t)$. On the other hand, we specify static item characteristics, q_i, because we do not expect significant temporal variation for items, which, unlike humans, are static in nature. We start with a detailed discussion of the temporal effects that are contained within the baseline predictors.

3.3.3.1 Time Changing Baseline Predictors

Much of the temporal variability is included within the baseline predictors, through two major temporal effects. The first addresses the fact that an item's popularity may change over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by treating the item bias b_i as a function of time. The second major temporal effect allows users to change their baseline ratings over time. For example, a user who tended to rate an average movie "4 stars", may now rate such a movie "3 stars". This may reflect several factors including a natural drift in a user's rating scale, the fact that ratings are given in relationship to other ratings that were given recently and also the fact that the identity of the rater within a household can change over time. Hence, in our models we take the parameter b_u as a function of time. This induces a template for a time sensitive baseline predictor for u's rating of i at day t_{ui}:

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) \tag{3.5}$$

Here, $b_u(\cdot)$ and $b_i(\cdot)$ are real valued functions that change over time. The exact way to build these functions should reflect a reasonable way to parameterize the involving temporal changes. Our choice in the context of the movie rating dataset demonstrates some typical considerations.

A major distinction is between temporal effects that span extended periods of time and more transient effects. In the movie rating case, we do not expect movie likability to fluctuate on a daily basis, but rather to change over more extended periods. On the other hand, we observe that user effects can change on a daily basis, reflecting inconsistencies natural to customer behavior. This requires finer time resolution when modeling user-biases compared with a lower resolution that suffices for capturing item-related time effects.

We start with our choice of time-changing item biases $b_i(t)$. We found it adequate to split the item biases into time-based bins, using a constant item bias for each time period. The decision of how to split the timeline into bins should balance the desire to achieve finer resolution (hence, smaller bins) with the need for enough ratings per bin (hence, larger bins). For the movie rating data, there is a wide variety of bin sizes that yield about the same accuracy. In our implementation, each bin corresponds to roughly ten consecutive weeks of data, leading to 30 bins spanning all days in the dataset. A day t is associated with an integer $\mathrm{Bin}(t)$ (a number between 1 and 30 in our data), such that the movie bias is split into a stationary part and a time changing part

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)} \,. \tag{3.6}$$

While binning the parameters works well on the items, it is more of a challenge on the users side. On the one hand, we would like a finer resolution for users to detect very short lived temporal effects. On the other hand, we do not expect enough ratings per user to produce reliable estimates for isolated bins. Different functional forms can be considered for parameterizing temporal user behavior, with varying complexity and accuracy.

One simple modeling choice uses a linear function to capture a possible gradual drift of user bias. For each user u, we denote the mean date of rating by t_u. Now, if u rated a movie on day t, then the associated time deviation of this rating is defined as

$$\mathrm{dev}_u(t) = \mathrm{sign}(t - t_u) \cdot |t - t_u|^\beta \,.$$

Here $|t - t_u|$ measures the number of days between dates t and t_u. We set the value of β by cross validation; in our implementation $\beta = 0.4$. We introduce a single new parameter for each user called α_u so that we get our first definition of a time-dependent user-bias

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) \,. \tag{3.7}$$

This simple linear model for approximating a drifting behavior requires learning two parameters per user: b_u and α_u.

A more flexible parameterization is offered by splines. Let u be a user associated with n_u ratings. We designate k_u time points—$\{t_1^u, \dots, t_{k_u}^u\}$—spaced uniformly across the dates of u's ratings as kernels that control the following function:

$$b_u^{(2)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|}} \tag{3.8}$$

The parameters $b_{t_l}^u$ are associated with the control points (or, kernels), and are automatically learned from the data. This way the user bias is formed as a time-weighted combination of those parameters. The number of control points, k_u, balances flexibility and computational efficiency. In our application we set $k_u = n_u^{0.25}$, letting it grow with the number of available ratings. The constant σ determines the smoothness of the spline; we set $\sigma = 0.3$ by cross validation.

So far we have discussed smooth functions for modeling the user bias, which mesh well with *gradual concept drift*. However, in many applications there are *sudden drifts* emerging as "spikes" associated with a single day or session. For example, in the movie rating dataset we have found that multiple ratings a user gives in a single day, tend to concentrate around a single value. Such an effect need not span more than a single day. The effect may reflect the mood of the user that day, the impact of ratings given in a single day on each other, or changes in the actual rater in multi-person accounts. To address such short lived effects, we assign a single parameter per user and day, absorbing the day-specific variability. This parameter is denoted by $b_{u,t}$. Notice that in some applications the basic primitive time unit to work with can be shorter or longer than a day.

In the Netflix movie rating data, a user rates on 40 different days on average. Thus, working with $b_{u,t}$ requires, on average, 40 parameters to describe each user bias. It is expected that $b_{u,t}$ is inadequate as a standalone for capturing the user bias, since it misses all sorts of signals that span more than a single day. Thus, it serves as an additive component within the previously described schemes. The time-linear model (3.7) becomes

$$b_u^{(3)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t}\,. \tag{3.9}$$

Similarly, the spline-based model becomes

$$b_u^{(4)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|}} + b_{u,t}\,. \tag{3.10}$$

A baseline predictor on its own cannot yield personalized recommendations, as it disregards all interactions between users and items. In a sense, it is capturing the portion of the data that is less relevant for establishing recommendations. Nonetheless, to better assess the relative merits of the various choices of time-dependent user-bias, we compare their accuracy as standalone predictors. In order to learn the involved parameters we minimize the associated regularized squared error by using stochastic gradient descent. For example, in our actual implementation we adopt rule (3.9) for modeling the drifting user bias, thus arriving at the baseline predictor

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\text{Bin}(t_{ui})}\,. \tag{3.11}$$

Table 3.1 Comparing baseline predictors capturing main movie and user effects

Model	Static	Mov	Linear	Spline	Linear+	Spline+
RMSE	0.9799	0.9771	0.9731	0.9714	0.9605	0.9603

As temporal modeling becomes more accurate, prediction accuracy improves (lowering RMSE)

To learn the involved parameters, $b_u, \alpha_u, b_{u,t}, b_i$ and $b_{i,\text{Bin}(t)}$, one should solve

$$\min \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - \alpha_u \text{dev}_u(t_{ui}) - b_{u,t_{ui}} - b_i - b_{i,\text{Bin}(t_{ui})})^2$$
$$+ \lambda_7(b_u^2 + \alpha_u^2 + b_{u,t_{ui}}^2 + b_i^2 + b_{i,\text{Bin}(t_{ui})}^2) \,.$$

Here, the first term strives to construct parameters that fit the given ratings. The regularization term, $\lambda_7(b_u^2 + \dots)$, avoids overfitting by penalizing the magnitudes of the parameters, assuming a neutral 0 prior. Learning is done by a stochastic gradient descent algorithm running 20–30 iterations, with $\lambda_7 = 0.01$.

Table 3.1 compares the ability of various suggested baseline predictors to explain signal in the data. As usual, the amount of captured signal is measured by the root mean squared error on the test set. As a reminder, test cases come later in time than the training cases for the same user, so predictions often involve extrapolation in terms of time. We code the predictors as follows:

- *Static*, no temporal effects: $b_{ui} = \mu + b_u + b_i$.
- *Mov*, accounting only for movie-related temporal effects: $b_{ui} = \mu + b_u + b_i + b_{i,\text{Bin}(t_{ui})}$.
- *Linear* , linear modeling of user biases: $b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_i + b_{i,\text{Bin}(t_{ui})}$.
- *Spline*, spline modeling of user biases: $b_{ui} = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-t_l^u|}} + b_i + b_{i,\text{Bin}(t_{ui})}$.
- *Linear+*, linear modeling of user biases and single day effect: $b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\text{Bin}(t_{ui})}$.
- *Spline+*, spline modeling of user biases and single day effect: $b_{ui} = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-d_l|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-t_l^u|}} + b_{u,t_{ui}} + b_i + b_{i,\text{Bin}(t_{ui})}$.

The table shows that while temporal movie effects reside in the data (lowering RMSE from 0.9799 to 0.9771), the drift in user biases is much more influential. The additional flexibility of splines at modeling user effects leads to better accuracy compared to a linear model. However, sudden changes in user biases, which are captured by the per-day parameters, are most significant. Indeed, when including those changes, the difference between linear modeling ("linear+") and spline modeling ("spline+") virtually vanishes.

Beyond the temporal effects described so far, one can use the same methodology to capture more effects. A primary example is capturing periodic effects. For example, some products may be more popular in specific seasons or near certain holidays. Similarly, different types of television or radio shows are popular throughout different segments of the day (known as "dayparting"). Periodic effects can be found also on the user side. As an example, a user may have different attitudes or buying patterns during the weekend compared to the working week. A way to model such periodic effects is to dedicate a parameter for the combinations of time periods with items or users. This way, the item bias of (3.6), becomes

$$b_i(t) = b_i + b_{i,\text{Bin}(t)} + b_{i,\text{period}(t)} \,.$$

For example, if we try to capture the change of item bias with the season of the year, then $\text{period}(t) \in \{\text{fall, winter, spring, summer}\}$. Similarly, recurring user effects may be modeled by modifying (3.9) to be

$$b_u(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t} + b_{u,\text{period}(t)} \,.$$

However, we have not found periodic effects with a significant predictive power within the movie-rating dataset, thus our reported results do not include those.

Another temporal effect within the scope of basic predictors is related to the changing scale of user ratings. While $b_i(t)$ is a user-independent measure for the merit of item i at time t, users tend to respond to such a measure differently. For example, different users employ different rating scales, and a single user can change his rating scale over time. Accordingly, the raw value of the movie bias is not completely user-independent. To address this, we add a time-dependent scaling feature to the baseline predictors, denoted by $c_u(t)$. Thus, the baseline predictor (3.11) becomes

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot c_u(t_{ui}) \,. \tag{3.12}$$

All discussed ways to implement $b_u(t)$ would be valid for implementing $c_u(t)$ as well. We chose to dedicate a separate parameter per day, resulting in: $c_u(t) = c_u + c_{u,t}$. As usual, c_u is the stable part of $c_u(t)$, whereas $c_{u,t}$ represents day-specific variability. Adding the multiplicative factor $c_u(t)$ to the baseline predictor lowers RMSE to 0.9555. Interestingly, this basic model, which captures just main effects disregarding user-item interactions, can explain almost as much of the data variability as the commercial Netflix Cinematch recommender system, whose published RMSE on the same test set is 0.9514 [5].

3.3.3.2 Time Changing Factor Model

In the previous section we discussed the way time affects baseline predictors. However, as hinted earlier, temporal dynamics go beyond this, they also affect user preferences and thereby the interaction between users and items. Users change their preferences over time. For example, a fan of the "psychological thrillers" genre may become a fan of "crime dramas" a year later. Similarly, humans change their perception on certain actors and directors. This type of evolution is modeled by taking the user factors (the vector p_u) as a function of time. Once again, we need to model those changes at the very fine level of a daily basis, while facing the built-in scarcity of user ratings. In fact, these temporal effects are the hardest to capture, because preferences are not as pronounced as main effects (user-biases), but are split over many factors.

We modeled each component of the user preferences $p_u(t)^T = (p_{u1}(t), \dots, p_{uf}(t))$ in the same way that we treated user biases. Within the movie-rating dataset, we have found modeling after (3.9) effective, leading to

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) + p_{uk,t} \quad k = 1, \dots, f\,. \tag{3.13}$$

Here p_{uk} captures the stationary portion of the factor, $\alpha_{uk} \cdot \mathrm{dev}_u(t)$ approximates a possible portion that changes linearly over time, and $p_{uk,t}$ absorbs the very local, day-specific variability.

At this point, we can tie all pieces together and extend the SVD++ factor model by incorporating the time changing parameters. The resulting model will be denoted as *timeSVD++*, where the prediction rule is as follows:

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + q_i^T \left(p_u(t_{ui}) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} y_j \right) \tag{3.14}$$

The exact definitions of the time drifting parameters $b_i(t), b_u(t)$ and $p_u(t)$ were given in (3.6), (3.9) and (3.13). Learning is performed by minimizing the associated squared error function on the training set using a regularized stochastic gradient descent algorithm. The procedure is analogous to the one involving the original SVD++ algorithm. Time complexity per iteration is still linear with the input size, while wall clock running time is approximately doubled compared to SVD++, due to the extra overhead required for updating the temporal parameters. Importantly, convergence rate was not affected by the temporal parameterization, and the process converges in around 30 iterations.

3.3.4 Comparison

In Table 3.2 we compare results of the three algorithms discussed in this section. First is SVD, the plain matrix factorization algorithm. Second, is the SVD++ method, which improves upon SVD by incorporating a kind of implicit feedback. Finally is timeSVD++, which accounts for temporal effects. The three methods are compared over a range of factorization dimensions (f). All benefit from a growing number of factor dimensions that enables them to better express complex movie-user interactions. Note that the number of parameters in SVD++ is comparable to their number in SVD. This is because SVD++ adds only item factors, while

Table 3.2 Comparison of three factor models: prediction accuracy is measured by RMSE (lower is better) for varying factor dimensionality (f)

Model	$f = 10$	$f = 20$	$f = 50$	$f = 100$	$f = 200$
SVD	0.9140	0.9074	0.9046	0.9025	0.9009
SVD++	0.9131	0.9032	0.8952	0.8924	0.8911
Timesvd++	0.8971	0.8891	0.8824	0.8805	0.8799

For all models, accuracy improves with growing number of dimensions. SVD++ improves accuracy by incorporating implicit feedback into the SVD model. Further accuracy gains are achieved by also addressing the temporal dynamics in the data through the timeSVD++ model

complexity of our dataset is dominated by the much larger set of users. On the other hand, timeSVD++ requires a significant increase in the number of parameters, because of its refined representation of each user factor. Addressing implicit feedback by the SVD++ model leads to accuracy gains within the movie rating dataset. Yet, the improvement delivered by timeSVD++ over SVD++ is consistently more significant. We are not aware of any single algorithm in the literature that could deliver such accuracy. Further evidence of the importance of capturing temporal dynamics is the fact that a timeSVD++ model of dimension 10 is already more accurate than an SVD model of dimension 200. Similarly, a timeSVD++ model of dimension 20 is enough to outperform an SVD++ model of dimension 200.

3.3.4.1 Predicting Future Days

Our models include day-specific parameters. An apparent question would be how these models can be used for predicting ratings in the future, on new dates for which we cannot train the day-specific parameters? The simple answer is that for those future (untrained) dates, the day-specific parameters should take their default value. In particular for (3.12), $c_u(t_{ui})$ is set to c_u, and $b_{u,t_{ui}}$ is set to zero. Yet, one wonders, if we cannot use the day-specific parameters for predicting the future, why are they good at all? After all, prediction is interesting only when it is about the future. To further sharpen the question, we should mention the fact that the Netflix test sets include many ratings on dates for which we have no other rating by the same user and hence day-specific parameters cannot be exploited.

To answer this, notice that our temporal modeling makes no attempt to capture future changes. All it is trying to do is to capture transient temporal effects, which had a significant influence on past user feedback. When such effects are identified they must be tuned down, so that we can model the more enduring signal. This allows our model to better capture the long-term characteristics of the data, while letting dedicated parameters absorb short term fluctuations. For example, if a user gave many higher than usual ratings on a particular single day, our models discount those by accounting for a possible day-specific good mood, which does not reflects the longer term behavior of this user. This way, the day-specific parameters accomplish a kind of data cleaning, which improves prediction of future dates.

3.3.5 Summary

In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from patterns of item ratings. High correspondence between item and user factors leads to recommendation of an item to a user. These methods deliver prediction accuracy superior to other published collaborative filtering techniques. At the same time, they offer a memory efficient compact model, which can be trained relatively easy. Those advantages, together with the implementation ease of gradient based matrix factorization model (SVD), made this the method of choice within the Netflix Prize competition.

What makes these techniques even more convenient is their ability to address several crucial aspects of the data. First, is the ability to integrate multiple forms of user feedback. One can better predict user ratings by also observing other related actions by the same user, such as purchase and browsing history. The proposed SVD++ model leverages multiple sorts of user feedback for improving user profiling.

Another important aspect is the temporal dynamics that make users' tastes evolve over time. Each user and product potentially goes through a distinct series of changes in their characteristics. A mere decay of older instances cannot adequately identify communal patterns of behavior in time changing data. The solution we adopted is to model the temporal dynamics along the whole time period, allowing us to intelligently separate transient factors from lasting ones. The inclusion of temporal dynamics proved very useful in improving quality of predictions, more than various algorithmic enhancements.

3.4 Neighborhood Models

The most common approach to CF is based on neighborhood models. Chapter 2 provides an extensive survey on this approach. Its original form, which was shared by virtually all earlier CF systems, is user-user based; see [13] for a good analysis. User-user methods estimate unknown ratings based on recorded ratings of like-minded users.

Later, an analogous item-item approach [18, 25] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-item approach more favorable in many cases [2, 25, 26]. In addition, item-item methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like-minded users. We focus mostly on item-item approaches, but the same techniques can be directly applied within a user-user approach; see also Sect. 3.5.2.2.

In general, latent factor models offer high expressive ability to describe various aspects of the data. Thus, they tend to provide more accurate results than neighborhood models. However, most literature and commercial systems (e.g., those of Amazon [18] and TiVo [1]) are based on the neighborhood models. The prevalence of neighborhood models is partly due to their relative simplicity. However, there are more important reasons for real life systems to stick with those models. First, they naturally provide intuitive explanations of the reasoning behind recommendations, which often enhance user experience beyond what improved accuracy may achieve. Second, they can provide immediate recommendations based on newly entered user feedback.

The structure of this section is as follows. First, we describe how to estimate the similarity between two items, which is a basic building block of most neighborhood techniques. Then, we move on to the widely used similarity-based neighborhood method, which constitutes a straightforward application of the similarity weights. We identify certain limitations of this similarity based approach. As a consequence, in Sect. 3.4.3 we suggest a way to solve these issues, thereby improving prediction accuracy at the cost of a slight increase in computation time.

3.4.1 Similarity Measures

Central to most item-item approaches is a similarity measure between items. Frequently, it is based on the Pearson correlation coefficient, ρ_{ij}, which measures the tendency of users to rate items i and j similarly. Since many ratings are unknown, some items may share only a handful of common observed raters. The empirical correlation coefficient, $\hat{\rho}_{ij}$, is based only on the common user support. It is advised to work with residuals from the baseline predictors (the b_{ui}'s; see Sect. 3.2.1) to compensate for user- and item-specific deviations. Thus the approximated correlation coefficient is given by

$$\hat{\rho}_{ij} = \frac{\sum_{u \in \mathrm{U}(i,j)} (r_{ui} - b_{ui})(r_{uj} - b_{uj})}{\sqrt{\sum_{u \in \mathrm{U}(i,j)} (r_{ui} - b_{ui})^2 \cdot \sum_{u \in \mathrm{U}(i,j)} (r_{uj} - b_{uj})^2}} . \tag{3.15}$$

The set $\mathrm{U}(i,j)$ contains the users who rated both items i and j.

Because estimated correlations based on a greater user support are more reliable, an appropriate similarity measure, denoted by s_{ij}, is a shrunk correlation coefficient of the form

$$s_{ij} \overset{\text{def}}{=} \frac{n_{ij} - 1}{n_{ij} - 1 + \lambda_8} \rho_{ij} . \tag{3.16}$$

The variable $n_{ij} = |\mathrm{U}(i,j)|$ denotes the number of users that rated both i and j. A typical value for λ_8 is 100.

Such shrinkage can be motivated from a Bayesian perspective; see Sect. 2.6 of Gelman et al. [11]. Suppose that the true ρ_{ij} are independent random variables drawn from a normal distribution,

$$\rho_{ij} \sim N(0, \tau^2)$$

for known τ^2. The mean of 0 is justified if the b_{ui} account for both user and item deviations from average. Meanwhile, suppose that

$$\hat{\rho}_{ij} | \rho_{ij} \sim N(\rho_{ij}, \sigma_{ij}^2)$$

for known σ_{ij}^2. We estimate ρ_{ij} by its posterior mean:

$$E(\rho_{ij} | \hat{\rho}_{ij}) = \frac{\tau^2 \hat{\rho}_{ij}}{\tau^2 + \sigma_{ij}^2}$$

the empirical estimator $\hat{\rho}_{ij}$ shrunk a fraction, $\sigma_{ij}^2/(\tau^2 + \sigma_{ij}^2)$, of the way toward zero.

Formula (3.16) follows from approximating the variance of a correlation by $\sigma_{ij}^2 = 1/(n_{ij} - 1)$, the value for ρ_{ij} near 0.

Notice that the literature suggests additional alternatives for a similarity measure [25, 26].

3.4.2 *Similarity-Based Interpolation*

Here we describe the most popular approach to neighborhood modeling, and apparently also to CF in general. Our goal is to predict r_{ui}—the unobserved rating by user u for item i. Using the similarity measure, we identify the k items rated by u that are most similar to i. This set of k neighbors is denoted by $\mathrm{S}^k(i; u)$. The predicted value of r_{ui} is taken as a weighted average of the ratings of neighboring items, while adjusting for user and item effects through the baseline predictors

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}}. \tag{3.17}$$

Note the dual use of the similarities for both identification of nearest neighbors and as the interpolation weights in Eq. (3.17).

Sometimes, instead of relying directly on the similarity weights as interpolation coefficients, one can achieve better results by transforming these weights. For example, we have found at several datasets that squaring the correlation-based similarities is helpful. This leads to a rule like: $\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}^2(r_{uj} - b_{uj})}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}^2}$. Toscher et al. [29] discuss more sophisticated transformations of these weights.

Similarity-based methods became very popular because they are intuitive and relatively simple to implement. They also offer the following two useful properties:

1. *Explainability.* The importance of explaining automated recommendations is widely recognized [12, 28]. Users expect a system to give a reason for its predictions, rather than presenting "black box" recommendations. Explanations not only enrich the user experience, but also encourage users to interact with the system, fix wrong impressions and improve long-term accuracy. The neighborhood framework allows identifying which of the past user actions are most influential on the computed prediction.
2. *New ratings.* Item-item neighborhood models can provide updated recommendations immediately after users enter new ratings. This includes handling new users as soon as they provide feedback to the system, without needing to re-train the model and estimate new parameters. This assumes that relationships between items (the s_{ij} values) are stable and barely change on a daily basis. Notice that for items new to the system we do have to learn new parameters. Interestingly, this asymmetry between users and items meshes well with common practices: systems need to provide immediate recommendations to new users (or new ratings by old users) who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system.

However, standard neighborhood-based methods raise some concerns:

1. The similarity function (s_{ij}), which directly defines the interpolation weights, is arbitrary. Various CF algorithms use somewhat different similarity measures, trying to quantify the elusive notion of user- or item-similarity. Suppose that a particular item is predicted perfectly by a subset of the neighbors. In that case, we would want the predictive subset to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.
2. Previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item i and a neighbor $j \in \mathrm{S}^k(i;u)$ is computed independently of the content of $\mathrm{S}^k(i;u)$ and the other similarities: s_{il} for $l \in \mathrm{S}^k(i;u) - \{j\}$. 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 more robust baseline predictors. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.
4. Neighborhood methods may not work well if variability of ratings differs substantially among neighbors.

Some of these issues can be fixed to a certain degree, while others are more difficult to solve within the basic framework. For example, the third item, dealing with the sum-to-one constraint, can be alleviated by using the following prediction rule:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}(r_{uj} - b_{uj})}{\lambda_9 + \sum_{j \in \mathrm{S}^k(i;u)} s_{ij}} \tag{3.18}$$

The constant λ_9 penalizes the neighborhood portion when there is not much neighborhood information, e.g., when $\sum_{j \in \mathrm{S}^k(i;u)} s_{ij} \ll \lambda_9$. Indeed, we have found that setting an appropriate value of λ_9 leads to accuracy improvements over (3.17). Nonetheless, the whole framework here is not justified by a formal model. Thus, we strive for better results with a more fundamental approach, as we describe in the following.

3.4.3 Jointly Derived Interpolation Weights

In this section we describe a more accurate neighborhood model that overcomes the difficulties discussed above, while retaining known merits of item-item models. As above, we use the similarity measure to define neighbors for each prediction. However, we search for optimum interpolation weights without regard to values of the similarity measure.

Given a set of neighbors $\mathrm{S}^k(i;u)$ we need to compute *interpolation weights* $\{\theta^u_{ij} | j \in \mathrm{S}^k(i;u)\}$ that enable the best prediction rule of the form

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{S}^k(i;u)} \theta^u_{ij}(r_{uj} - b_{uj}) \,. \tag{3.19}$$

Typical values of k (number of neighbors) lie in the range of 20–50; see [2]. During this subsection we assume that baseline predictors have already been removed. Hence, we introduce a notation for the residual ratings: $z_{ui} \overset{\text{def}}{=} r_{ui} - b_{ui}$. For notational convenience assume that the items in $\mathrm{S}^k(i;u)$ are indexed by $1, \ldots, k$.

We seek a formal computation of the interpolation weights that stems directly from their usage within prediction rule (3.19). 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.

3.4.3.1 Formal Model

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

$$\min_{\theta^u} \sum_{v \neq u} \left(z_{vi} - \sum_{j \in \mathrm{S}^k(i;u)} \theta_{ij}^u z_{vj} \right)^2 . \tag{3.20}$$

Notice that the only unknowns here are the θ_{ij}^u's. The optimal solution to the least squares problem (3.20) 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 z_{vi} on the z_{vj} for $j \in \mathrm{S}^k(i;u)$. Specifically, the optimal weights are given by

$$Aw = b . \tag{3.21}$$

Here, $w \in \mathbb{R}^k$ is an unknown vector such that w_j stands for the sought coefficient θ_{ij}^u. A is a $k \times k$ matrix defined as

$$A_{jl} = \sum_{v \neq u} z_{vj} z_{vl} . \tag{3.22}$$

Similarly the vector $b \in \mathbb{R}^k$ is given by

$$b_j = \sum_{v \neq u} z_{vj} z_{vi} . \tag{3.23}$$

For a sparse ratings matrix there are likely to be very few users who rated i and all its neighbors $\mathrm{S}^k(i;u)$. Accordingly, it would be unwise to base A and b as given in (3.22)–(3.23) 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 pairwise support, leading to the following reformulation:

$$\bar{A}_{jl} = \frac{\sum_{v \in \mathrm{U}(j,l)} z_{vj} z_{vl}}{|\mathrm{U}(j,l)|} \tag{3.24}$$

$$\bar{b}_j = \frac{\sum_{v \in \mathrm{U}(i,j)} z_{vj} z_{vi}}{|\mathrm{U}(i,j)|} \tag{3.25}$$

As a reminder, $\mathrm{U}(j,l)$ is the set of users who rated both j and l.

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

$$\hat{A}_{jl} = \frac{|\mathrm{U}(j,l)| \cdot \bar{A}_{jl} + \beta \cdot avg}{|\mathrm{U}(j,l)| + \beta} \tag{3.26}$$

$$\hat{b}_j = \frac{|\mathrm{U}(i,j)| \cdot \bar{b}_j + \beta \cdot avg}{|\mathrm{U}(i,j)| + \beta} \tag{3.27}$$

The parameter β controls the extent of the shrinkage. A typical value would be $\beta = 500$.

Our best estimate for A and b are $\hat{A}$ and $\hat{b}$, respectively. Therefore, we modify (3.21) so that the interpolation weights are defined as the solution of the linear system

$$\hat{A}w = \hat{b}\,. \tag{3.28}$$

The resulting interpolation weights are used within (3.19) in order to predict r_{ui}.

This method addresses all four concerns raised in Sect. 3.4.2. 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.

3.4.3.2 Computational Issues

Efficient computation of an item-item neighborhood method requires pre-computing certain values associated with each item-item pair for rapid retrieval. First, we need a quick access to all item-item similarities, by pre-computing all s_{ij} values, as explained in Sect. 3.4.1.

Second, we pre-compute all possible entries of $\hat{A}$ and $\hat{b}$. To this end, for each two items i and j, we compute

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

Then, the aforementioned baseline value avg, which is used in (3.26)–(3.27), is taken as the average entry of the pre-computed $n \times n$ 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 generally-larger diagonal entries, which 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 (3.26), using the appropriate value of avg. 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 $i \geqslant j$. Our experience shows that it is enough to allocate one byte for each individual value, so the overall space required for n items is exactly $n(n+1)$ bytes.

Pre-computing all possible entries of matrix $\hat{A}$ saves the otherwise lengthy time needed to construct entries on the fly. After quickly retrieving the relevant entries of $\hat{A}$, we can compute the interpolation weights by solving a $k \times k$ system of Eq. (3.28) using a standard linear solver. However, a modest increase in prediction accuracy was achieved when constraining w to be nonnegative through a quadratic program [2]. Solving the system of equations is an overhead over the basic neighborhood method described in Sect. 3.4.2. For typical values of k (between 20 and 50), the extra time overhead is comparable to the time needed for computing the k nearest neighbors, which is common to neighborhood-based approaches. Hence, while the method relies on a much more detailed computation of the interpolation weights compared to previous methods, it does not significantly increase running time; see [2].

3.4.4 Summary

Collaborative filtering through neighborhood-based interpolation is probably the most popular way to create a recommender system. Three major components characterize the neighborhood approach: (1) data normalization, (2) neighbor selection, and (3) determination of interpolation weights.

Normalization is essential to collaborative filtering in general, and in particular to the more local neighborhood methods. Otherwise, even more sophisticated methods are bound to fail, as they mix incompatible ratings pertaining to different unnormalized users or items. We described a suitable approach to data normalization, based around baseline predictors.

Neighborhood selection is another important component. It is directly related to the employed similarity measure. Here, we emphasized the importance of shrinking unreliable similarities, in order to avoid detection of neighbors with a low rating support.

Finally, the success of neighborhood methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, most known methods lack a rigorous way to derive these weights. We showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role.

3.5 Enriching Neighborhood Models

Most neighborhood methods are local in their nature—concentrating on only a small subset of related ratings. This contrasts with matrix factorization, which casts a very wide net to try to characterize items and users. It appears that accuracy can be improved by employing this global viewpoint, which motivates the methods of this section. We suggest a new neighborhood model drawing on principles of both classical neighborhood methods and matrix factorization models. Like other neighborhood models, the building stones here are item-item relations (or, alternatively, user-user relations), which provide the system some practical advantages discussed earlier. At the same time, much like matrix factorization, the model is centered around a global optimization framework, which improves accuracy by considering the many weak signals existing in the data.

The main method, which is described in Sect. 3.5.1, allows us to enrich the model with implicit feedback data. In addition, it facilitates two new possibilities. First is a factorized neighborhood model, as described in Sect. 3.5.2, bringing great improvements in computational efficiency. Second is a treatment of temporal dynamics, leading to better prediction accuracy, as described in Sect. 3.5.3.

3.5.1 A Global Neighborhood Model

In this subsection, we introduce a neighborhood model based on global optimization. The model offers an improved prediction accuracy, by offering the aforementioned merits of the model described in Sect. 3.4.3, with additional advantages that are summarized as follows:

1. No reliance on arbitrary or heuristic item-item similarities. The new model is cast as the solution to a global optimization problem.
2. Inherent overfitting prevention or "risk control": the model reverts to robust baseline predictors, unless a user entered sufficiently many relevant ratings.
3. The model can capture the totality of weak signals encompassed in all of a user's ratings, not needing to concentrate only on the few ratings for most similar items.
4. The model naturally allows integrating different forms of user input, such as explicit and implicit feedback.
5. A highly scalable implementation (Sect. 3.5.2) allows linear time and space complexity, thus facilitating both item-item and user-user implementations to scale well to very large datasets.
6. Time drifting aspects of the data can be integrated into the model, thereby improving its accuracy; see Sect. 3.5.3.

3.5.1.1 Building the Model

We gradually construct the various components of the model, through an ongoing refinement of our formulations. Previous models were centered around *user-specific* interpolation weights—θ_{ij}^u in (3.19) or $s_{ij}/\sum_{j\in \mathrm{S}^k(i;u)} s_{ij}$ in (3.17)—relating item i to the items in a user-specific neighborhood $\mathrm{S}^k(i;u)$. In order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global item-item weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learned from the data through optimization. An initial sketch of the model describes each rating r_{ui} by the equation

$$\hat{r}_{ui} = b_{ui} + \sum_{j\in \mathrm{R}(u)} (r_{uj} - b_{uj})w_{ij}\,. \tag{3.29}$$

This rule starts with the crude, yet robust, baseline predictors (b_{ui}). Then, the estimate is adjusted by summing over *all* ratings by u.

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, we adopt a different viewpoint, that enables a more flexible usage of the weights. We no longer treat weights as interpolation coefficients. Instead, we take weights as part of adjustments, or *offsets*, added to the baseline predictors. This way, the weight w_{ij} is the extent by which we increase our baseline prediction of r_{ui} based on the observed value of r_{uj}. For two related items i and j, we expect w_{ij} to be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$ is high), we would like to increase our estimate for u's rating of i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline prediction. Likewise, our estimate will not deviate much from the baseline by an item j that u rated just as expected ($r_{uj} - b_{uj}$ is around zero), or by an item j that is not known to be predictive on i (w_{ij} is close to zero).

This viewpoint suggests several enhancements to (3.29). First, we can use the form of binary user input, which was found beneficial for factorization models. Namely, analyzing which items were rated regardless of rating value. To this end, we add another set of weights, and rewrite (3.29) as

$$\hat{r}_{ui} = b_{ui} + \sum_{j\in \mathrm{R}(u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]\,. \tag{3.30}$$

Similarly, one could employ here another set of implicit feedback, $\mathrm{N}(u)$—e.g., the set of items rented or purchased by the user—leading to the rule

$$\hat{r}_{ui} = b_{ui} + \sum_{j\in \mathrm{R}(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j\in \mathrm{N}(u)} c_{ij}\,. \tag{3.31}$$

Much like the w_{ij}'s, the c_{ij}'s are offsets added to the baseline predictor. For two items i and j, an implicit preference by u for j leads us to adjust our estimate of r_{ui} by c_{ij}, which is expected to be high if j is predictive on i.

Employing global weights, rather than user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, a user's opinion is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate "Shrek 3" high also gave high ratings to "Shrek 1–2". This will establish high weights from "Shrek 1–2" to "Shrek 3". Now, if a user did not rate "Shrek 1–2" at all, his predicted rating for "Shrek 3" will be penalized, as some necessary weights cannot be added to the sum.

For prior models (3.17) and (3.19) that interpolated $r_{ui} - b_{ui}$ from $\{r_{uj} - b_{uj} | j \in \mathrm{S}^k(i;u)\}$, it was necessary to maintain compatibility between the b_{ui} values and the b_{uj} values. However, here we do not use interpolation, so we can decouple the definitions of b_{ui} and b_{uj}. Accordingly, a more general prediction rule would be: $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}$. The constant $\tilde{b}_{ui}$ can represent predictions of r_{ui} by other methods such as a latent factor model. Here, we suggest the following rule that was found to work well:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in \mathrm{R}(u)} [(r_{uj} - b_{uj}) w_{ij} + c_{ij}] \tag{3.32}$$

Importantly, the b_{uj}'s remain constants, which are derived as explained in Sect. 3.2.1. However, the b_u's and b_i's become parameters that are optimized much like the w_{ij}'s and c_{ij}'s.

We have found that it is beneficial to normalize sums in the model leading to the form

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\alpha} \sum_{j \in \mathrm{R}(u)} [(r_{uj} - b_{uj}) w_{ij} + c_{ij}] \, . \tag{3.33}$$

The constant α controls the extent of normalization. A non-normalized rule ($\alpha = 0$), encourages greater deviations from baseline predictions for users that provided many ratings (high $|\mathrm{R}(u)|$). On the other hand, a fully normalized rule, eliminates the effect of number of ratings on deviations from baseline predictions. In many cases it would be a good practice for recommender systems to have greater deviation from baselines for users that rate a lot. This way, we take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. At the same time, we are less certain about the modeling of users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. Our experience with the Netflix dataset shows that best results are achieved with $\alpha = 0.5$, as in the prediction rule

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} [(r_{uj} - b_{uj}) w_{ij} + c_{ij}] \, . \tag{3.34}$$

As an optional refinement, complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations. Let us denote by $\mathrm{S}^k(i)$ the set of k items most similar to i, as determined by e.g., a similarity measure s_{ij} or a natural hierarchy associated with the item set. Additionally, we use $\mathrm{R}^k(i;u) \overset{\text{def}}{=} \mathrm{R}(u) \cap \mathrm{S}^k(i)$.[2] Now, when predicting r_{ui} according to (3.34), it is expected that the most influential weights will be associated with items similar to i. Hence, we replace (3.34) with

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]\,. \tag{3.35}$$

When $k = \infty$, rule (3.35) coincides with (3.34). However, for other values of k it offers the potential to significantly reduce the number of variables involved.

3.5.1.2 Parameter Estimation

Prediction rule (3.35) allows fast online prediction. More computational work is needed at a pre-processing stage where parameters are estimated. A major design goal of the new neighborhood model was facilitating an efficient global optimization procedure, which prior neighborhood models lacked. Thus, model parameters are learned by solving the regularized least squares problem associated with (3.35):

$$\min_{b_*,w_*,c_*} \sum_{(u,i)\in\mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}^k(i;u)} \big((r_{uj} - b_{uj})w_{ij} + c_{ij}\big) \right)^2 + \lambda_{10} \left(b_u^2 + b_i^2 + \sum_{j\in\mathrm{R}^k(i;u)} w_{ij}^2 + c_{ij}^2 \right) \tag{3.36}$$

An optimal solution of this convex problem can be obtained by least square solvers, which are part of standard linear algebra packages. However, we have found that the following simple stochastic gradient descent solver works much faster. Let us denote the prediction error, $r_{ui} - \hat{r}_{ui}$, by e_{ui}. We loop through all known ratings in $\mathcal{K}$. For a given training case r_{ui}, we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_{10} \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_{10} \cdot b_i)$
- $\forall j \in \mathrm{R}^k(i;u)$:

$$w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_{10} \cdot w_{ij} \right)$$
$$c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_{10} \cdot c_{ij} \right)$$

[2]Notational clarification: With other neighborhood models it was beneficial to use $\mathrm{S}^k(i;u)$, which denotes the k items most similar to i among those rated by u. Hence, if u rated at least k items, we will always have $|\mathrm{S}^k(i;u)| = k$, regardless of how similar those items are to i. However, $|\mathrm{R}^k(i;u)|$ is typically smaller than k, as some of those items most similar to i were not rated by u.

The meta-parameters γ (step size) and λ_{10} are determined by cross-validation. We used $\gamma = 0.005$ and $\lambda_{10} = 0.002$ for the Netflix data. Another important parameter is k, which controls the neighborhood size. Our experience shows that increasing k always benefits the accuracy of the results on the test set. Hence, the choice of k should reflect a tradeoff between prediction accuracy and computational cost. In Sect. 3.5.2 we will describe a factored version of the model that eliminates this tradeoff by allowing us to work with the most accurate $k = \infty$ while lowering running time.

A typical number of iterations throughout the training data is 15–20. As for time complexity per iteration, let us analyze the most accurate case where $k = \infty$, which is equivalent to using prediction rule (3.34). For each user u and item $i \in \mathrm{R}(u)$ we need to modify $\{w_{ij}, c_{ij} | j \in \mathrm{R}(u)\}$. Thus the overall time complexity of the training phase is $O(\sum_u |\mathrm{R}(u)|^2)$.

3.5.1.3 Comparison of Accuracy

Experimental results on the Netflix data with the globally optimized neighborhood model, henceforth dubbed GlobalNgbr, are presented in Fig. 3.1. We studied the model under different values of parameter k. The solid black curve with square symbols shows that accuracy monotonically improves with rising k values, as root mean squared error (RMSE) falls from 0.9139 for $k = 250$ to 0.9002 for $k = \infty$. (Notice that since the Netflix data contains 17,770 movies, $k = \infty$ is equivalent to k =17,769, where all item-item relations are explored.) We repeated the experiments without using the implicit feedback, that is, dropping the c_{ij} parameters from our model. The results depicted by the solid black curve with X's show a significant decline in estimation accuracy, which widens as k grows. This demonstrates the value of incorporating implicit feedback into the model.

For comparison we provide the results of the two previously described neighborhood models. First is a similarity-based neighborhood model (in Sect. 3.4.2), which is the most popular CF method in the literature. We denote this model as CorNgbr. Second is the more accurate model described in Sect. 3.4.3, which will be denoted as JointNgbr. For both these two models, we tried to pick optimal parameters and neighborhood sizes, which were 20 for CorNgbr, and 50 for JointNgbr. The results are depicted by the dotted and dashed lines, respectively. It is clear that the popular CorNgbr method is noticeably less accurate than the other neighborhood models. On the opposite side, GlobalNgbr delivers more accurate results even when compared with JointNgbr, as long as the value of k is at least 500. Notice that the k value (the x-axis) is irrelevant to the previous models, as their different notion of neighborhood makes neighborhood sizes incompatible. Yet, we observed that while the performance of GlobalNgbr keeps improving as more neighbors are added, this was not true with the two other models. For CorNgbr and JointNgbr, performance peaks with a relatively small number of neighbors and declines thereafter. This may

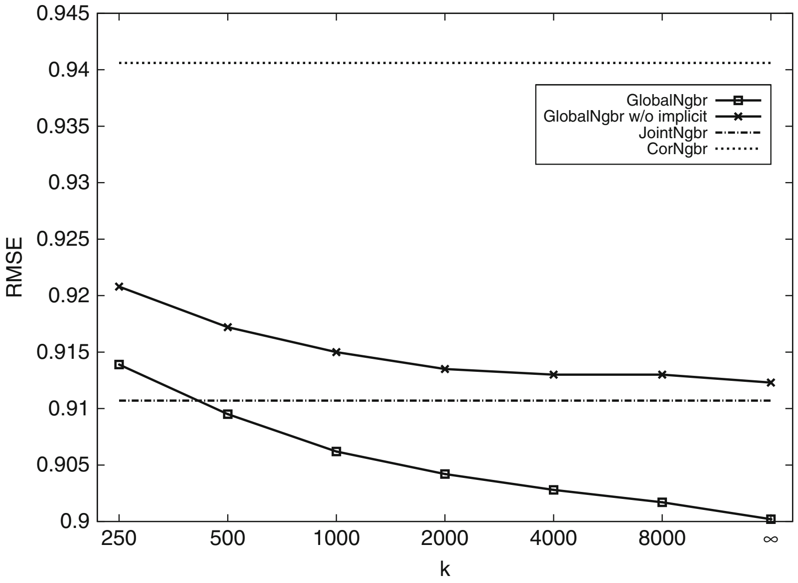

Fig. 3.1 Comparison of neighborhood-based models. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. We measure the accuracy of the globally optimized model (GlobalNgbr) with and without implicit feedback. RMSE is shown as a function of varying values of k, which dictates the neighborhood size. The accuracy of two other models is shown as *two horizontal lines*; for each we picked an optimal neighborhood size

be explained by the fact that in GlobalNgbr, parameters are directly learned from the data through a formal optimization procedure that facilitates using many more parameters effectively.

Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, JointNgbr [2] requires solving a small system of equations for each provided prediction. The new model does involve pre-processing where parameters are estimated. However, online prediction is immediate by following rule (3.35). Pre-processing time grows with the value of k. Figure 3.2 shows typical running times per iteration on the Netflix data, as measured on a single processor 3.4 GHz Pentium 4 PC.

3.5.2 A Factorized Neighborhood Model

In the previous section we presented a more accurate neighborhood model, which is based on prediction rule (3.34) with training time complexity $O(\sum_u |\mathrm{R}(u)|^2)$ and space complexity $O(m+n^2)$. (Recall that m is the number of users, and n is the number of items.) We could improve time and space complexity by sparsifying the model through pruning unlikely item-item relations. Sparsification was controlled by the parameter $k \leqslant n$, which reduced running time and allowed space complexity of $O(m+nk)$. However, as k gets lower, the accuracy of the model declines as well.

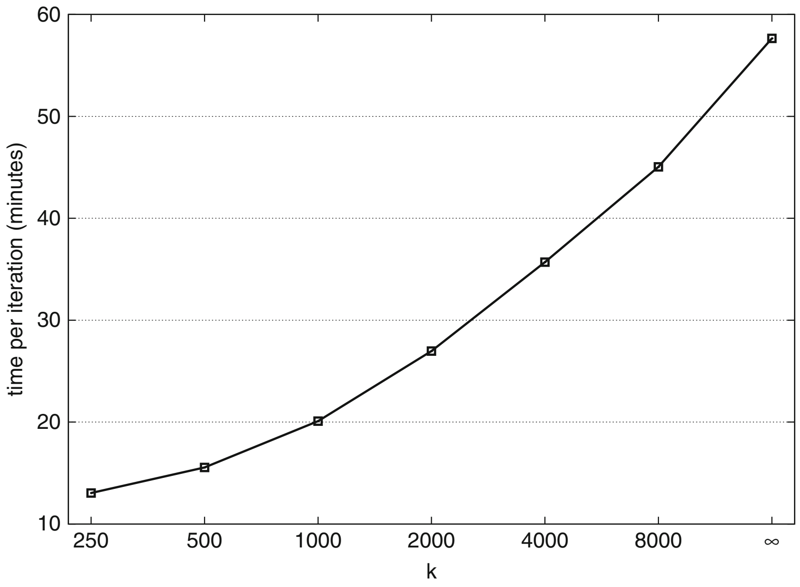

Fig. 3.2 Running time per iteration of the globally optimized neighborhood model, as a function of the parameter *k*

In addition, sparsification required relying on an external, less natural, similarity measure, which we would have liked to avoid. Thus, we will now show how to retain the accuracy of the full dense prediction rule (3.34), while significantly lowering time and space complexity.

3.5.2.1 Factoring Item-Item Relationships

We factor item-item relationships by associating each item i with three vectors: $q_i, x_i, y_i \in \mathbb{R}^f$. This way, we confine w_{ij} to be $q_i^T x_i$. Similarly, we impose the structure $c_{ij} = q_i^T y_j$. Essentially, these vectors strive to map items into an f-dimensional latent factor space where they are measured against various aspects that are revealed automatically by learning from the data. By substituting this into (3.34) we get the following prediction rule:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} [(r_{uj} - b_{uj}) q_i^T x_j + q_i^T y_j] \tag{3.37}$$

Computational gains become more obvious by using the equivalent rule

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + y_j \right). \tag{3.38}$$

Notice that the bulk of the rule $(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})x_j + y_j)$ depends only on u while being independent of i. This leads to an efficient way to learn the model parameters. As usual, we minimize the regularized squared error function associated with (3.38)

$$\min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})x_j + y_j \right) \right)^2 + \lambda_{11} \left(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathrm{R}(u)} \|x_j\|^2 + \|y_j\|^2 \right). \tag{3.39}$$

Optimization is done by a stochastic gradient descent scheme, which is described in the following pseudo code:

```
**LearnFactorizedNeighborhoodModel**(Known ratings: $r_{ui}$, rank: $f$)
% For each item i compute $q_i, x_i, y_i \in \mathbb{R}^f$
% which form a neighborhood model
Const #Iterations $= 20, \gamma = 0.002, \lambda = 0.04$
% Gradient descent sweeps:
for count = 1, ..., #Iterations do
    for $u = 1, \ldots, m$ do
        % Compute the component independent of i:
        $p_u \leftarrow |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})x_j + y_j$
        $sum \leftarrow 0$
        for all $i \in \mathrm{R}(u)$ do
            $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + q_i^T p_u$
            $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
            % Accumulate information for gradient steps on $x_i, y_i$:
            $sum \leftarrow sum + e_{ui} \cdot q_i$
            % Perform gradient step on $q_i, b_u, b_i$:
            $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
            $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
            $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$
        for all $i \in \mathrm{R}(u)$ do
            % Perform gradient step on $x_i$:
            $x_i \leftarrow x_i + \gamma \cdot (|\mathrm{R}(u)|^{-\frac{1}{2}} \cdot (r_{ui} - b_{ui}) \cdot sum - \lambda \cdot x_i)$
            % Perform gradient step on $y_i$:
            $y_i \leftarrow y_i + \gamma \cdot (|\mathrm{R}(u)|^{-\frac{1}{2}} \cdot sum - \lambda \cdot y_i)$
return $\{q_i, x_i, y_i | i = 1, \ldots, n\}$
```

The time complexity of this model is linear with the input size, $O(f \cdot \sum_u (|\mathrm{R}(u)|))$, which is significantly better than the non-factorized model that required time $O(\sum_u |\mathrm{R}(u)|^2)$. We measured the performance of the model on the Netflix data;

Table 3.3 Performance of the factorized item-item neighborhood model

#Factors	50	100	200	500
RMSE	0.9037	0.9013	0.9000	0.8998
Time/iteration (min)	4.5	8	14	34

The models with $\geqslant$ 200 factors slightly outperform the non-factorized model, while providing much shorter running time

see Table 3.3. Accuracy is improved as we use more factors (increasing f). However, going beyond 200 factors could barely improve performance, while slowing running time. Interestingly, we have found that with $f \geqslant 200$ accuracy negligibly exceeds the best non-factorized model (with $k = \infty$). In addition, the improved time complexity translates into a big difference in wall-clock measured running time. For example, the time-per-iteration for the non-factorized model (with $k = \infty$) was close to 58 min. On the other hand, a factorized model with $f = 200$ could complete an iteration in 14 min without degrading accuracy at all.

The most important benefit of the factorized model is the reduced space complexity, which is $O(m + nf)$—linear in the input size. Previous neighborhood models required storing all pairwise relations between items, leading to a quadratic space complexity of $O(m + n^2)$. For the Netflix dataset which contains 17,770 movies, such quadratic space can still fit within core memory. Some commercial recommenders process a much higher number of items. For example, an online movie rental service like Netflix is currently offering over 100,000 titles. Music download shops offer even more titles. Such more comprehensive systems with data on 100,000s items eventually need to resort to external storage in order to fit the entire set of pairwise relations. However, as the number of items is growing towards millions, as in the Amazon item-item recommender system, which accesses stored similarity information for several million catalog items [18], designers must keep a sparse version of the pairwise relations. To this end, only values relating an item to its top-k most similar neighbors are stored thereby reducing space complexity to $O(m + nk)$. However, a sparsification technique will inevitably degrade accuracy by missing important relations, as demonstrated in the previous section. In addition, identification of the top k most similar items in such a high dimensional space is a non-trivial task that can require considerable computational efforts. All these issues do not exist in our factorized neighborhood model, which offers a linear time and space complexity without trading off accuracy.

The factorized neighborhood model resembles some latent factor models. The important distinction is that here we factorize the item-item relationships, rather than the ratings themselves. The results reported in Table 3.3 are comparable to those of the widely used SVD model, but not as good as those of SVD++; see Sect. 3.3. Nonetheless, the factorized neighborhood model retains the practical advantages of traditional neighborhood models discussed earlier—the abilities to explain recommendations and to immediately reflect new ratings.

As a side remark, we would like to mention that the decision to use three separate sets of factors was intended to give us more flexibility. Indeed, on the Netflix data this allowed achieving most accurate results. However, another reasonable choice could be using a smaller set of vectors, e.g., by requiring: $q_i = x_i$ (implying symmetric weights: $w_{ij} = w_{ji}$).

3.5.2.2 A User-User Model

A user-user neighborhood model predicts ratings by considering how like-minded users rated the same items. Such models can be implemented by switching the roles of users and items throughout our derivation of the item-item model. Here, we would like to concentrate on a user-user model, which is dual to the item-item model of (3.34). The major difference is replacing the w_{ij} weights relating item pairs, with weights relating user pairs:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vi} - b_{vi}) w_{uv} \tag{3.40}$$

The set $\mathrm{R}(i)$ contains all the users who rated item i. Notice that here we decided to not account for implicit feedback. This is because adding such feedback was not very beneficial for the user-user model when working with the Netflix data.

User-user models can become useful in various situations. For example, some recommenders may deal with items that are rapidly replaced, thus making item-item relations very volatile. On the other hand, a stable user base enables establishment of long term relationships between users. An example of such a case is a recommender system for web articles or news items, which are rapidly changing by their nature; see, e.g., [8]. In such cases, systems centered around user-user relations are more appealing.

In addition, user-user approaches identify different kinds of relations that item-item approaches may fail to recognize, and thus can be useful on certain occasions. For example, suppose that we want to predict r_{ui}, but none of the items rated by user u is really relevant to i. In this case, an item-item approach will face obvious difficulties. However, when employing a user-user perspective, we may find a set of users similar to u, who rated i. The ratings of i by these users would allow us to improve prediction of r_{ui}.

The major disadvantage of user-user models is computational. Since typically there are many more users than items, pre-computing and storing all user-user relations, or even a reasonably sparsified version thereof, is overly expensive or completely impractical. In addition to the high $O(m^2)$ space complexity, the time complexity for optimizing model (3.40) is also much higher than its item-item counterpart, being $O(\sum_i |\mathrm{R}(i)|^2)$ (notice that $|\mathrm{R}(i)|$ is expected to be much higher than $|\mathrm{R}(u)|$). These issues have rendered user-user models as a less practical choice.

A Factorized Model All those computational differences disappear by factorizing the user-user model along the same lines as in the item-item model. Now, we associate each user u with two vectors $p_u, z_u \in \mathbb{R}^f$. We assume the user-user relations to be structured as: $w_{uv} = p_u^T z_v$. Let us substitute this into (3.40) to get

Table 3.4 Performance of the factorized user-user neighborhood model

#Factors	50	100	200	500
RMSE	0.9119	0.9110	0.9101	0.9093
Time/iteration (min)	3	5	8.5	18

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vi} - b_{vi}) p_u^T z_v \,. \tag{3.41}$$

Once again, an efficient computation is achieved by including the terms that depends on i but are independent of u in a separate sum, so the prediction rule is presented in the equivalent form

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vi} - b_{vi}) z_v \,. \tag{3.42}$$

In a parallel fashion to the item-item model, all parameters are learned in linear time $O(f \cdot \sum_i |\mathrm{R}(i)|)$. The space complexity is also linear with the input size being $O(n + mf)$. This significantly lowers the complexity of the user-user model compared to previously known results. In fact, running time measured on the Netflix data shows that now the user-user model is even faster than the item-item model; see Table 3.4. We should remark that unlike the item-item model, our implementation of the user-user model did not account for implicit feedback, which probably led to its shorter running time. Accuracy of the user-user model is significantly better than that of the widely-used correlation-based item-item model that achieves RMSE $= 0.9406$ as reported in Fig. 3.1. Furthermore, accuracy is slightly better than the variant of the item-item model, which also did not account for implicit feedback (yellow curve in Fig. 3.1). This is quite surprising given the common wisdom that item-item methods are more accurate than user-user ones. It appears that a well implemented user-user model can match speed and accuracy of an item-item model. However, our item-item model could significantly benefit by accounting for implicit feedback.

Fusing Item-Item and User-User Models Since item-item and user-user models address different aspects of the data, overall accuracy is expected to improve by combining predictions of both models. Such an approach was previously suggested and was shown to improve accuracy; see, e.g. [4, 30]. However, past efforts were based on blending the item-item and user-user predictions during a post-processing stage, after each individual model was trained independently of the other model. A more principled approach optimizes the two models simultaneously, letting them know of each other while parameters are being learned. Thus, throughout the entire training phase each model is aware of the capabilities of the other model and strives to complement it. Our approach, which states the neighborhood models as formal optimization problems, allows doing that naturally. We devise a model that sums the item-item model (3.37) and the user-user model (3.41), leading to

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} [(r_{uj} - b_{uj}) q_i^T x_j + q_i^T y_j] + |\mathrm{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathrm{R}(i)} (r_{vi} - b_{vi}) p_u^T z_v \,. \tag{3.43}$$

Model parameters are learned by stochastic gradient descent optimization of the associated squared error function. Our experiments with the Netflix data show that prediction accuracy is indeed better than that of each individual model. For example, with 100 factors the obtained RMSE is 0.8966, while with 200 factors the obtained RMSE is 0.8953.

Here we would like to comment that our approach allows integrating the neighborhood models also with completely different models in a similar way. For example, in [16] we showed an integrated model that combines the item-item model with a latent factor model (SVD++), thereby achieving improved prediction accuracy with RMSE below 0.887. Therefore, other possibilities with potentially better accuracy should be explored before considering the integration of item-item and user-user models.

3.5.3 Temporal Dynamics at Neighborhood Models

One of the advantages of the item-item model based on global optimization (Sect. 3.5.1), is that it enables us to capture temporal dynamics in a principled manner. As we commented earlier, user preferences are drifting over time, and hence it is important to introduce temporal aspects into CF models.

When adapting rule (3.34) to address temporal dynamics, two components should be considered separately. First component, $\mu + b_i + b_u$, corresponds to the baseline predictor portion. Typically, this component explains most variability in the observed signal. Second component, $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}$, captures the more informative signal, which deals with user-item interaction. As for the baseline part, nothing changes from the factor model, and we replace it with $\mu + b_i(t_{ui}) + b_u(t_{ui})$, according to (3.6) and (3.9). However, capturing temporal dynamics within the interaction part requires a different strategy.

Item-item weights (w_{ij} and c_{ij}) reflect inherent item characteristics and are not expected to drift over time. The learning process should capture unbiased long term values, without being too affected from drifting aspects. Indeed, the time changing nature of the data can mask much of the longer term item-item relationships if not treated adequately. For instance, a user rating both items i and j high within a short time period, is a good indicator for relating them, thereby pushing higher the value of w_{ij}. On the other hand, if those two ratings are given 4 years apart, while the user's taste (if not her identity) could considerably change, this provides less evidence of any relation between the items. On top of this, we would argue that those considerations are pretty much user-dependent; some users are more consistent than others and allow relating their longer term actions.

Our goal here is to distill accurate values for the item-item weights, despite the interfering temporal effects. First we need to parameterize the decaying relations between two items rated by user u. We adopt exponential decay formed by the function $e^{-\beta_u \cdot \Delta t}$, where $\beta_u > 0$ controls the user specific decay rate and should be learned from the data. We also experimented with other decay forms, like the more computationally-friendly $(1 + \beta_u \Delta t)^{-1}$, which resulted in about the same accuracy, with an improved running time.

This leads to the prediction rule

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}) . \qquad (3.44)$$

The involved parameters, $b_i(t_{ui}) = b_i + b_{i,\mathrm{Bin}(t_{ui})}$, $b_u(t_{ui}) = b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}}$, β_u, w_{ij} and c_{ij}, are learned by minimizing the associated regularized squared error

$$\sum_{(u,i) \in \mathcal{K}} \Bigg(r_{ui} - \mu - b_i - b_{i,\mathrm{Bin}(t_{ui})} - b_u - \alpha_u \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} \\ - |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - b_{uj}) w_{ij} + c_{ij}) \Bigg)^2 \\ + \lambda_{12} \left(b_i^2 + b_{i,\mathrm{Bin}(t_{ui})}^2 + b_u^2 + \alpha_u^2 + b_{u,t}^2 + w_{ij}^2 + c_{ij}^2 \right) . \qquad (3.45)$$

Minimization is performed by stochastic gradient descent. We run the process for 25 iterations, with $\lambda_{12} = 0.002$, and step size (learning rate) of 0.005. An exception is the update of the exponent β_u, where we are using a much smaller step size of 10^{-7}. Training time complexity is the same as the original algorithm, which is: $O(\sum_u |\mathrm{R}(u)|^2)$. One can tradeoff complexity with accuracy by sparsifying the set of item-item relations as explained in Sect. 3.5.1.

As in the factor case, properly considering temporal dynamics improves the accuracy of the neighborhood model within the movie ratings dataset. The RMSE decreases from 0.9002 [16] to 0.8885. To our best knowledge, this is significantly better than previously known results by neighborhood methods. To put this in some perspective, this result is even better than those reported by using hybrid approaches such as applying a neighborhood approach on residuals of other algorithms [2, 21, 29]. A lesson is that addressing temporal dynamics in the data can have a more significant impact on accuracy than designing more complex learning algorithms.

We would like to highlight an interesting point. Let u be a user whose preferences are quickly drifting (β_u is large). Hence, old ratings by u should not be very influential on his status at the current time t. One could be tempted to decay the weight of u's older ratings, leading to "instance weighting" through a cost function like

$$\sum_{(u,i)\in\mathcal{K}} e^{-\beta_u\cdot|t-t_{ui}|}\Bigg(r_{ui}-\mu-b_i-b_{i,\text{Bin}(t_{ui})}-b_u-\alpha_u\text{dev}_u(t_{ui})$$

$$-b_{u,t_{ui}}-|\text{R}(u)|^{-\frac{1}{2}}\sum_{j\in\text{R}(u)}((r_{uj}-b_{uj})w_{ij}+c_{ij})\Bigg)^2+\lambda_{12}(\cdots)\,.$$

Such a function is focused at the *current* state of the user (at time t), while de-emphasizing past actions. We would argue against this choice, and opt for equally weighting the prediction error at all past ratings as in (3.45), thereby modeling *all* past user behavior. Therefore, equal-weighting allows us to exploit the signal at each of the past ratings, a signal that is extracted as item-item weights. Learning those weights would equally benefit from all ratings by a user. In other words, we can deduce that two items are related if users rated them similarly within a short time frame, even if this happened long ago.

3.5.4 Summary

This section follows a less traditional neighborhood based model, which unlike previous neighborhood methods is based on formally optimizing a global cost function. The resulting model is no longer localized, considering relationships between a small set of strong neighbors, but rather considers all possible pairwise relations. This leads to improved prediction accuracy, while maintaining some merits of the neighborhood approach such as explainability of predictions and ability to handle new ratings (or new users) without re-training the model.

The formal optimization framework offers several new possibilities. First, is a factorized version of the neighborhood model, which improves its computational complexity while retaining prediction accuracy. In particular, it is free from the quadratic storage requirements that limited past neighborhood models.

Second addition is the incorporation of temporal dynamics into the model. In order to reveal accurate relations among items, a proposed model learns how influence between two items rated by a user decays over time. Much like in the matrix factorization case, accounting for temporal effects results in a significant improvement in predictive accuracy.

3.6 Between Neighborhood and Factorization

This chapter was structured around two different approaches to CF: factorization and neighborhood. Each approach evolved from different basic principles, which led to distinct prediction rules. We also argued that factorization can lead to somewhat more accurate results, while neighborhood models may have some practical advantages. In this section we will show that despite those differences, the two approaches share much in common. After all, they are both *linear models*.

Let us consider the SVD model of Sect. 3.3.1, based on

$$\hat{r}_{ui} = q_i^T p_u \,. \tag{3.46}$$

For simplicity, we ignore here the baseline predictors, but one can easily reintroduce them or just assume that they were subtracted from all ratings at an earlier stage.

We arrange all item-factors within the $n \times f$ matrix $Q = [q_1 q_2 \dots q_n]^T$. Similarly, we arrange all user-factors within the $m \times f$ matrix $P = [p_1 p_2 \dots p_m]^T$. We use the $n_u \times f$ matrix $Q[u]$ to denote the restriction of Q to the items rated by u, where $n_u = |\mathrm{R}(u)|$. Let the vector $r_u \in \mathbb{R}^{n_u}$ contain the ratings given by u ordered as in $Q[u]$. Now, by activating (3.46) on all ratings given by u, we can reformulate it in a matrix form

$$\hat{r}_u = Q[u] p_u \tag{3.47}$$

For $Q[u]$ fixed, $\|r_u - Q[u]p_u\|_2$ is minimized by

$$p_u = (Q[u]^T Q[u])^{-1} Q[u]^T r_u$$

In practice, we will regularize with $\lambda \geqslant 0$ to get

$$p_u = (Q[u]^T Q[u] + \lambda I)^{-1} Q[u]^T r_u \,.$$

By substituting p_u in (3.47) we get

$$\hat{r}_u = Q[u](Q[u]^T Q[u] + \lambda I)^{-1} Q[u]^T r_u \,. \tag{3.48}$$

This expression can be simplified by introducing some new notation. Let us denote the $f \times f$ matrix $(Q[u]^T Q[u] + \lambda I)^{-1}$ as W^u, which should be considered as a weighting matrix associated with user u. Accordingly, the weighted similarity between items i and j from u's viewpoint is denoted by $s_{ij}^u = q_i^T W^u q_j$. Using this new notation and (3.48) the predicted preference of u for item i by SVD is rewritten as

$$\hat{r}_{ui} = \sum_{j \in \mathrm{R}(u)} s_{ij}^u r_{uj} \,. \tag{3.49}$$

We reduced the SVD model into a linear model that predicts preferences as a linear function of past actions, weighted by item-item similarity. Each past action receives a separate term in forming the prediction $\hat{r}_{ui}$. This is equivalent to an item-item neighborhood model. Quite surprisingly, we transformed the matrix factorization model into an item-item model, which is characterized by:

- Interpolation is made from *all* past user ratings, not only from those associated with items most similar to the current one.
- The weight relating items i and j is factorized as a product of two vectors, one related to i and the other to j.
- Item-item weights are subject to a user-specific normalization, through the matrix W^u.

Those properties support our findings on how to best construct a neighborhood model. First, we showed in Sect. 3.5.1 that best results for neighborhood models are achieved when the neighborhood size (controlled by constant k) is maximal, such that all past user ratings are considered. Second, in Sect. 3.5.2 we touted the practice of factoring item-item weights. As for the user-specific normalization, we used a simpler normalizer: $n_u^{-0.5}$. It is likely that SVD suggests a more fundamental normalization by the matrix W^u, which would work better. However, computing W^u would be expensive in practice. Another difference between our suggested item-item model and the one implied by SVD is that we chose to work with asymmetric weights ($w_{ij} \neq w_{ji}$), whereas in the SVD-induced rule: $s_{ij}^u = s_{ji}^u$.

In the derivation above we showed how SVD induces an equivalent item-item technique. In a fully analogous way, it can induce an equivalent user-user technique, by expressing q_i as a function of the ratings and user factors. This brings us to three equivalent models: SVD, item-item and user-user. Beyond linking SVD with neighborhood models, this also shows that user-user and item-item approaches, once well designed, are equivalent.

This last relation (between user-user and item-item approaches) can also be approached intuitively. Neighborhood models try to relate users to new items by following chains of user-item adjacencies. Such adjacencies represent preference- or rating-relations between the respective users and items. Both user-user and item-item models act by following exactly the same chains. They only differ in which "shortcuts" are exploited to speed up calculations. For example, recommending itemB to user1 would follow the chain user1–itemA–user2–itemB (user1 rated itemA, which was also rated by user2, who rated itemB). A user-user model follows such a chain with pre-computed user-user similarities. This way, it creates a "shortcut" that bypasses the sub-chain user1–itemB–user2, replacing it with a similarity value between user1 and user2. Analogously, an item-item approach follows exactly the same chain, but creates an alternative "shortcut", replacing the sub-chain itemA–user2–itemB with an itemA–itemB similarity value.

Another lesson here is that the distinction that deems neighborhood models as "memory based", while taking matrix factorization and the likes as "model based" is not always appropriate, at least not when using accurate neighborhood models that are model-based as much as SVD. In fact, the other direction is also true. The better matrix factorization models, such as SVD++, are also following memory-based habits, as they sum over all memory stored ratings when doing the online prediction; see rule (3.3). Hence, the traditional separation between "memory based" and "model based" techniques is not appropriate for categorizing the techniques surveyed in this chapter.

So far, we concentrated on relations between neighborhood models and matrix factorization models. However, in practice it may be beneficial to break these relations, and to augment factorization models with sufficiently different neighborhood models that are able to complement them. Such a combination can lead to improved prediction accuracy [3, 16]. A key to achieve this is by using the more localized neighborhood models (those of Sect. 3.4, rather than those of Sect. 3.5), where the number of considered neighbors is limited. The limited number of neighbors might not be the best way to construct a standalone neighborhood model, but it makes the neighborhood model different enough from the factorization model in order to add a local perspective that the rather global factorization model misses.

References

1. Ali, K., and van Stam, W., "TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture", *Proc. 10th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, pp. 394–401, 2004.
2. Bell, R., and Koren, Y., "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, pp. 43–52, 2007.
3. Bell, R., and Koren, Y., "Lessons from the Netflix Prize Challenge", *SIGKDD Explorations* **9** (2007), 75–79.
4. Bell, R.M., Koren, Y., and Volinsky, C., "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.
5. Bennet, J., and Lanning, S., "The Netflix Prize", *KDD Cup and Workshop*, 2007. www.netflixprize.com.
6. Canny, J., "Collaborative Filtering with Privacy via Factor Analysis", *Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'02)*, pp. 238–245, 2002.
7. Blei, D., Ng, A., and Jordan, M., "Latent Dirichlet Allocation", *Journal of Machine Learning Research* **3** (2003), 993–1022.
8. Das, A., Datar, M., Garg, A., and Rajaram, S., "Google News Personalization: Scalable Online Collaborative Filtering", *WWW'07*, pp. 271–280, 2007.
9. Deerwester, S., Dumais, S., Furnas, G.W., Landauer, T.K. and Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the Society for Information Science* **41** (1990), 391–407.
10. Funk, S., "Netflix Update: Try This At Home", http://sifter.org/~simon/journal/20061211.html, 2006.
11. Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B., *Bayesian Data Analysis*, Chapman and Hall, 1995.
12. Herlocker, J.L., Konstan, J.A., and Riedl, J., "Explaining Collaborative Filtering Recommendations", *Proc. ACM Conference on Computer Supported Cooperative Work*, pp. 241–250, 2000.
13. Herlocker, J.L., Konstan, J.A., Borchers, A., and Riedl, J., "An Algorithmic Framework for Performing Collaborative Filtering", *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
14. Hofmann, T., "Latent Semantic Models for Collaborative Filtering", *ACM Transactions on Information Systems* **22** (2004), 89–115.
15. Kim, D., and Yum, B., "Collaborative Filtering Based on Iterative Principal Component Analysis", *Expert Systems with Applications* **28** (2005), 823–830.

16. Koren, Y., “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”, *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.
17. Koren, Y., “Factor in the Neighbors: Scalable and Accurate Collaborative Filtering ”, *ACM Transactions on Knowledge Discovery from Data (TKDD)*,4(2010):1–24.
18. Linden, G., Smith, B., and York, J., “Amazon.com Recommendations: Item-to-Item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
19. Marlin, B.M., Zemel, R.S., Roweis, S., and Slaney, M., “Collaborative Filtering and the Missing at Random Assumption", *Proc. 23rd Conference on Uncertainty in Artificial Intelligence*, 2007.
20. Oard, D.W.„ and Kim, J., “Implicit Feedback for Recommender Systems”, *Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36, 1998.
21. Paterek, A., “Improving Regularized Singular Value Decomposition for Collaborative Filtering”, *Proc. KDD Cup and Workshop*, 2007.
22. Salakhutdinov, R., Mnih, A., and Hinton, G., “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.
23. Salakhutdinov, R., and Mnih, A., “Probabilistic Matrix Factorization”, *Advances in Neural Information Processing Systems 20 (NIPS’07)*, pp. 1257–1264, 2008.
24. Sarwar, B.M., Karypis, G., Konstan, J.A., and Riedl, J., “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000.*
25. Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
26. Takács G., Pilászy I., Németh B. and Tikk, D., “Major Components of the Gravity Recommendation System”, *SIGKDD Explorations* **9** (2007), 80–84.
27. Takács G., Pilászy I., Németh B. and Tikk, D., “Matrix Factorization and Neighbor based Algorithms for the Netflix Prize Problem”, *Proc. 2nd ACM conference on Recommender Systems (RecSys’08)*, pp. 267–274, 2008.
28. Tintarev, N., and Masthoff, J., “A Survey of Explanations in Recommender Systems”, *ICDE’07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.
29. Toscher, A., Jahrer, M., and Legenstein, R., “Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems”, *KDD’08 Workshop on Large Scale Recommenders Systems and the Netflix Prize*, 2008.
30. Wang, J., de Vries, A.P., and Reinders, M.J.T, “Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.