2
Collaborative recommendation

The main idea of collaborative recommendation approaches is to exploit information about the past behavior or the opinions of an existing user community for predicting which items the current user of the system will most probably like or be interested in. These types of systems are in widespread industrial use today, in particular as a tool in online retail sites to customize the content to the needs of a particular customer and to thereby promote additional items and increase sales.

From a research perspective, these types of systems have been explored for many years, and their advantages, their performance, and their limitations are nowadays well understood. Over the years, various algorithms and techniques have been proposed and successfully evaluated on real-world and artificial test data.

Pure collaborative approaches take a matrix of given user–item ratings as the only input and typically produce the following types of output: (a) a (numerical) prediction indicating to what degree the current user will like or dislike a certain item and (b) a list of *n* recommended items. Such a *top-N* list should, of course, not contain items that the current user has already bought.

2.1 User-based nearest neighbor recommendation

The first approach we discuss here is also one of the earliest methods, called *user-based nearest neighbor recommendation*. The main idea is simply as follows: given a ratings database and the ID of the current (active) user as an input, identify other users (sometimes referred to as *peer users* or *nearest neighbors*) that had similar preferences to those of the active user in the past. Then, for every product *p* that the active user has not yet seen, a prediction is computed based on the ratings for *p* made by the peer users. The underlying

Table 2.1. *Ratings database for collaborative recommendation.*

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

assumptions of such methods are that (a) if users had similar tastes in the past they will have similar tastes in the future and (b) user preferences remain stable and consistent over time.

2.1.1 First example

Let us examine a first example. Table 2.1 shows a database of ratings of the current user, Alice, and some other users. Alice has, for instance, rated "Item1" with a "5" on a 1-to-5 scale, which means that she strongly liked this item. The task of a recommender system in this simple example is to determine whether Alice will like or dislike "Item5", which Alice has not yet rated or seen. If we can predict that Alice will like this item very strongly, we should include it in Alice's recommendation list. To this purpose, we search for users whose taste is similar to Alice's and then take the ratings of this group for "Item5" to predict whether Alice will like this item.

Before we discuss the mathematical calculations required for these predictions in more detail, let us introduce the following conventions and symbols. We use $U = \{u_1, \ldots, u_n\}$ to denote the set of users, $P = \{p_1, \ldots, p_m\}$ for the set of products (items), and R as an $n \times m$ matrix of ratings $r_{i,j}$, with $i \in 1 \ldots n$, $j \in 1 \ldots m$. The possible rating values are defined on a numerical scale from 1 (strongly dislike) to 5 (strongly like). If a certain user i has not rated an item j, the corresponding matrix entry $r_{i,j}$ remains empty.

With respect to the determination of the set of similar users, one common measure used in recommender systems is Pearson's correlation coefficient. The similarity $sim(a, b)$ of users a and b, given the rating matrix R, is defined in Formula 2.1. The symbol $\overline{r_a}$ corresponds to the average rating of user a.

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}} \qquad (2.1)$$

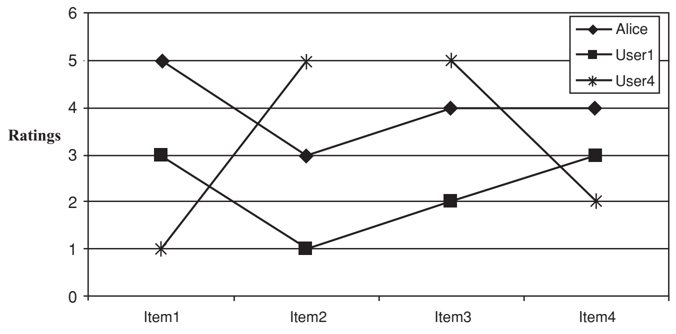


Figure 2.1. Comparing Alice with two other users.

The similarity of *Alice* to *User1* is thus as follows ($\overline{r_{Alice}} = \overline{r_a} = 4$, $\overline{r_{User1}} = \overline{r_b} = 2.4$):

$$\frac{(5-\overline{r_a})*(3-\overline{r_b})+(3-\overline{r_a})*(1-\overline{r_b})+\cdots+(4-\overline{r_a})*(3-\overline{r_b}))}{\sqrt{(5-\overline{r_a})^2+(3-\overline{r_a})^2+\cdots}\sqrt{(3-\overline{r_b})^2+(1-\overline{r_b})^2+\cdots}} = 0.85 \tag{2.2}$$

The Pearson correlation coefficient takes values from +1 (strong positive correlation) to −1 (strong negative correlation). The similarities to the other users, *User2* to *User4*, are 0.70, 0.00, and −0.79, respectively.

Based on these calculations, we observe that *User1* and *User2* were somehow similar to *Alice* in their rating behavior in the past. We also see that the Pearson measure considers the fact that users are different with respect to how they interpret the rating scale. Some users tend to give only high ratings, whereas others will never give a 5 to any item. The Pearson coefficient factors these averages out in the calculation to make users comparable – that is, although the absolute values of the ratings of *Alice* and *User1* are quite different, a rather clear linear correlation of the ratings and thus similarity of the users is detected.

This fact can also be seen in the visual representation in Figure 2.1, which both illustrates the similarity between *Alice* and *User1* and the differences in the ratings of *Alice* and *User4*.

To make a prediction for *Item5*, we now have to decide which of the neighbors' ratings we shall take into account and how strongly we shall value their opinions. In this example, an obvious choice would be to take *User1* and *User2* as peer users to predict Alice's rating. A possible formula for computing a prediction for the rating of user *a* for item *p* that also factors the relative *proximity* of the nearest neighbors N and a's average rating $\overline{r_a}$ is the following:

$$pred(a, p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a, b)} \tag{2.3}$$

In the example, the prediction for Alice's rating for *Item5* based on the ratings of near neighbors *User1* and *User2* will be

$$4 + 1/(0.85 + 0.7) * (0.85 * (3 - 2.4) + 0.70 * (5 - 3.8)) = 4.87 \tag{2.4}$$

Given these calculation schemes, we can now compute rating predictions for Alice for all items she has not yet seen and include the ones with the highest prediction values in the recommendation list. In the example, it will most probably be a good choice to include *Item5* in such a list.

The example rating database shown above is, of course, an idealization of the real world. In real-world applications, rating databases are much larger and can comprise thousands or even millions of users and items, which means that we must think about computational complexity. In addition, the rating matrix is typically very sparse, meaning that every user will rate only a very small subset of the available items. Finally, it is unclear what we can recommend to new users or how we deal with new items for which no ratings exist. We discuss these aspects in the following sections.

2.1.2 Better similarity and weighting metrics

In the example, we used Pearson's correlation coefficient to measure the similarity among users. In the literature, other metrics, such as *adjusted cosine similarity* (which will be discussed later in more detail), *Spearman*'s *rank correlation coefficient*, or the *mean squared difference* measure have also been proposed to determine the proximity between users. However, empirical analyses show that for user-based recommender systems – and at least for the best studied recommendation domains – the Pearson coefficient outperforms other measures of comparing users (Herlocker et al. 1999). For the later-described item-based recommendation techniques, however, it has been reported that the cosine similarity measure consistently outperforms the Pearson correlation metric.

Still, using the "pure" Pearson measure alone for finding neighbors and for weighting the ratings of those neighbors may not be the best choice. Consider, for instance, the fact that in most domains there will exist some items that are liked by everyone. A similarity measure such as Pearson will not take into account that an agreement by two users on a more controversial item has more "value" than an agreement on a generally liked item. As a resolution to this, Breese et al. (1998) proposed applying a transformation function to the item ratings, which reduces the relative importance of the agreement on universally liked items. In analogy to the original technique, which was developed in the information retrieval field, they called that factor the *inverse user frequency*. Herlocker et al. (1999) address the same problem through a *variance weighting factor* that increases the influence of items that have a high variance in the ratings – that is, items on which controversial opinions exist.

Our basic similarity measure used in the example also does not take into account whether two users have co-rated only a few items (on which they may agree by chance) or whether there are many items on which they agree. In fact, it has been shown that predictions based on the ratings of neighbors with which the active user has rated only a very few items in common are a bad choice and lead to poor predictions (Herlocker et al. 1999). Herlocker et al. (1999) therefore propose using another weighting factor, which they call *significance weighting*. Although the weighting scheme used in their experiments, reported by Herlocker et al. (1999, 2002), is a rather simple one, based on a linear reduction of the similarity weight when there are fewer than fifty co-rated items, the increases in prediction accuracy are significant. The question remains open, however, whether this weighting scheme and the heuristically determined thresholds are also helpful in real-world settings, in which the ratings database is smaller and we cannot expect to find many users who have co-rated fifty items.

Finally, another proposal for improving the accuracy of the recommendations by fine-tuning the prediction weights is termed *case amplification* (Breese et al. 1998). Case amplification refers to an adjustment of the weights of the neighbors in a way that values close to $+1$ and -1 are emphasized by multiplying the original weights by a constant factor ρ. Breese et al. (1998) used 2.5 for ρ in their experiments.

2.1.3 Neighborhood selection

In our example, we intuitively decided not to take all neighbors into account (*neighborhood selection*). For the calculation of the predictions, we included only those that had a positive correlation with the active user (and, of course, had rated the item for which we are looking for a prediction). If we included all users in the neighborhood, this would not only negatively influence the performance with respect to the required calculation time, but it would also have an effect on the accuracy of the recommendation, as the ratings of other users who are not really comparable would be taken into account.

The common techniques for reducing the size of the neighborhood are to define a specific minimum threshold of user similarity or to limit the size to a fixed number and to take only the k nearest neighbors into account. The potential problems of either technique are discussed by Anand and Mobasher (2005) and by Herlocker et al. (1999): if the similarity threshold is too high, the size of the neighborhood will be very small for many users, which in turn means that for many items no predictions can be made (*reduced coverage*). In contrast, when the threshold is too low, the neighborhood sizes are not significantly reduced.

The value chosen for k – the size of the neighborhood – does not influence coverage. However, the problem of finding a good value for k still exists: When the number of neighbors k taken into account is too high, too many neighbors with limited similarity bring additional "noise" into the predictions. When k is too small – for example, below 10 in the experiments from Herlocker et al. (1999) – the quality of the predictions may be negatively affected. An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002).

A detailed analysis of the effects of using different weighting and similarity schemes, as well as different neighborhood sizes, can be found in Herlocker et al. (2002).

2.2 Item-based nearest neighbor recommendation

Although user-based CF approaches have been applied successfully in different domains, some serious challenges remain when it comes to large e-commerce sites, on which we must handle millions of users and millions of catalog items. In particular, the need to scan a vast number of potential neighbors makes it impossible to compute predictions in real time. Large-scale e-commerce sites, therefore, often implement a different technique, *item-based recommendation*, which is more apt for offline preprocessing[1] and thus allows for the computation of recommendations in real time even for a very large rating matrix (Sarwar et al. 2001).

The main idea of item-based algorithms is to compute predictions using the similarity between items and not the similarity between users. Let us examine our ratings database again and make a prediction for Alice for *Item5*. We first compare the rating vectors of the other items and look for items that have ratings similar to *Item5*. In the example, we see that the ratings for *Item5* (3, 5, 4, 1) are similar to the ratings of *Item1* (3, 4, 3, 1) and there is also a partial similarity

[1] Details about data preprocessing for item-based filtering are given in Section 2.2.2.

with *Item4* (3, 3, 5, 2). The idea of item-based recommendation is now to simply look at Alice's ratings for these similar items. Alice gave a "5" to *Item1* and a "4" to *Item4*. An item-based algorithm computes a weighted average of these other ratings and will predict a rating for *Item5* somewhere between 4 and 5.

2.2.1 The cosine similarity measure

To find similar items, a similarity measure must be defined. In item-based recommendation approaches, *cosine similarity* is established as the standard metric, as it has been shown that it produces the most accurate results. The metric measures the similarity between two n-dimensional vectors based on the angle between them. This measure is also commonly used in the fields of information retrieval and text mining to compare two text documents, in which documents are represented as vectors of terms.

The similarity between two items a and b – viewed as the corresponding rating vectors $\vec{a}$ and $\vec{b}$ – is formally defined as follows:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\mid \vec{a} \mid * \mid \vec{b} \mid} \tag{2.5}$$

The $\cdot$ symbol is the dot product of vectors. $\mid \vec{a} \mid$ is the Euclidian length of the vector, which is defined as the square root of the dot product of the vector with itself.

The cosine similarity of *Item5* and *Item1* is therefore calculated as follows:

$$sim(I5, I1) = \frac{3 * 3 + 5 * 4 + 4 * 3 + 1 * 1}{\sqrt{3^2 + 5^2 + 4^2 + 1^2} * \sqrt{3^2 + 4^2 + 3^2 + 1^2}} = 0.99 \tag{2.6}$$

The possible similarity values are between 0 and 1, where values near to 1 indicate a strong similarity. The basic cosine measure does not take the differences in the average rating behavior of the users into account. This problem is solved by using the *adjusted cosine* measure, which subtracts the user average from the ratings. The values for the adjusted cosine measure correspondingly range from -1 to $+1$, as in the Pearson measure.

Let U be the set of users that rated both items a and b. The adjusted cosine measure is then calculated as follows:

$$sim(a, b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}} \tag{2.7}$$

We can therefore transform the original ratings database and replace the original rating values with their deviation from the average ratings as shown in Table 2.2.

Table 2.2. *Mean-adjusted ratings database.*

	Item1	Item2	Item3	Item4	Item5
Alice	1.00	−1.00	0.00	0.00	?
User1	0.60	−1.40	−0.40	0.60	0.60
User2	0.20	−0.80	0.20	−0.80	1.20
User3	−0.20	−0.20	−2.20	2.80	0.80
User4	−1.80	2.20	2.20	−0.80	−1.80

The adjusted cosine similarity value for *Item5* and *Item1* for the example is thus:

$$\frac{0.6 * 0.6 + 0.2 * 1.2 + (-0.2) * 0.80 + (-1.8) * (-1.8)}{\sqrt{(0.6^2 + 0.2^2 + (-0.2)^2 + (-1.8)^2} * \sqrt{0.6^2 + 1.2^2 + 0.8^2 + (-1.8)^2}} = 0.80 \tag{2.8}$$

After the similarities between the items are determined we can predict a rating for Alice for *Item5* by calculating a weighted sum of Alice's ratings for the items that are similar to *Item5*. Formally, we can predict the rating for user u for a product p as follows:

$$pred(u, p) = \frac{\sum_{i \in ratedItems(u)} sim(i, p) * r_{u,i}}{\sum_{i \in ratedItems(a)} sim(i, p)} \tag{2.9}$$

As in the user-based approach, the size of the considered neighborhood is typically also limited to a specific size – that is, not all neighbors are taken into account for the prediction.

2.2.2 Preprocessing data for item-based filtering

Item-to-item collaborative filtering is the technique used by Amazon.com to recommend books or CDs to their customers. Linden et al. (2003) report on how this technique was implemented for Amazon's online shop, which, in 2003, had 29 million users and millions of catalog items. The main problem with traditional user-based CF is that the algorithm does not scale well for such large numbers of users and catalog items. Given M customers and N catalog items, in the worst case, all M records containing up to N items must be evaluated. For realistic scenarios, Linden et al. (2003) argue that the actual complexity is much lower because most of the customers have rated or bought only a very small number of items. Still, when the number of customers M is around several million, the calculation of predictions in real time is still infeasible, given the short response times that must be obeyed in the online environment.

For making item-based recommendation algorithms applicable also for large scale e-commerce sites without sacrificing recommendation accuracy, an approach based on offline precomputation of the data is typically chosen. The idea is to construct in advance the *item similarity matrix* that describes the pairwise similarity of all catalog items. At run time, a prediction for a product p and user u is made by determining the items that are most similar to i and by building the weighted sum of u's ratings for these items in the neighborhood. The number of neighbors to be taken into account is limited to the number of items that the active user has rated. As the number of such items is typically rather small, the computation of the prediction can be easily accomplished within the short time frame allowed in interactive online applications.

With respect to memory requirements, a full item similarity matrix for N items can theoretically have up to N^2 entries. In practice, however, the number of entries is significantly lower, and further techniques can be applied to reduce the complexity. The options are, for instance, to consider only items that have a minimum number of co-ratings or to memorize only a limited neighborhood for each item; this, however, increases the danger that no prediction can be made for a given item (Sarwar et al. 2001).

In principle, such an offline precomputation of neighborhoods is also possible for user-based approaches. Still, in practical scenarios the number of overlapping ratings for two users is relatively small, which means that a few additional ratings may quickly influence the similarity value between users. Compared with these user similarities, the item similarities are much more stable, such that precomputation does not affect the preciseness of the predictions too much (Sarwar et al. 2001).

Besides different preprocessing techniques used in so-called model-based approaches, it is an option to exploit only a certain fraction of the rating matrix to reduce the computational complexity. Basic techniques include *subsampling*, which can be accomplished by randomly choosing a subset of the data or by ignoring customer records that have only a very small set of ratings or that only contain very popular items. A more advanced and information-theoretic technique for filtering out the most "relevant" customers was also proposed by Yu et al. (2003). In general, although some computational speedup can be achieved with such techniques, the capability of the system to generate accurate predictions might deteriorate, as these recommendations are based on less information.

Further model-based and preprocessing-based approaches for complexity and dimensionality reduction will be discussed in Section 2.4.

2.3 About ratings

Before we discuss further techniques for reducing the computational complexity and present additional algorithms that operate solely on the basis of a user–item ratings matrix, we present a few general remarks on ratings in collaborative recommendation approaches.

2.3.1 Implicit and explicit ratings

Among the existing alternatives for gathering users' opinions, asking for explicit item ratings is probably the most precise one. In most cases, five-point or seven-point Likert response scales ranging from "Strongly dislike" to "Strongly like" are used; they are then internally transformed to numeric values so the previously mentioned similarity measures can be applied. Some aspects of the usage of different rating scales, such as how the users' rating behavior changes when different scales must be used and how the quality of recommendation changes when the granularity is increased, are discussed by Cosley et al. (2003). What has been observed is that in the movie domain, a five-point rating scale may be too narrow for users to express their opinions, and a ten-point scale was better accepted. An even more fine-grained scale was chosen in the joke recommender discussed by Goldberg et al. (2001), where a continuous scale (from -10 to $+10$) and a graphical input bar were used. The main arguments for this approach are that there is no precision loss from the discretization, user preferences can be captured at a finer granularity, and, finally, end users actually "like" the graphical interaction method, which also lets them express their rating more as a "gut reaction" on a visual level.

The question of how the recommendation accuracy is influenced and what is the "optimal" number of levels in the scaling system is, however, still open, as the results reported by Cosley et al. (2003) were developed on only a small user basis and for a single domain.

The main problems with explicit ratings are that such ratings require additional efforts from the users of the recommender system and users might not be willing to provide such ratings as long as the value cannot be easily seen. Thus, the number of available ratings could be too small, which in turn results in poor recommendation quality.

Still, Shafer et al. (2006) argue that the problem of gathering explicit ratings is not as hard as one would expect because only a small group of "early adopters" who provide ratings for many items is required in the beginning to get the system working.

Besides that, one can observe that in the last few years in particular, with the emergence of what is called Web 2.0, the role of online communities has changed and users are more willing to contribute actively to their community's knowledge. Still, in light of these recent developments, more research focusing on the development of techniques and measures that can be used to persuade the online user to provide more ratings is required.

Implicit ratings are typically collected by the web shop or application in which the recommender system is embedded. When a customer buys an item, for instance, many recommender systems interpret this behavior as a positive rating. The system could also monitor the user's browsing behavior. If the user retrieves a page with detailed item information and remains at this page for a longer period of time, for example, a recommender could interpret this behavior as a positive orientation toward the item.

Although implicit ratings can be collected constantly and do not require additional efforts from the side of the user, one cannot be sure whether the user behavior is correctly interpreted. A user might not like all the books he or she has bought; the user also might have bought a book for someone else. Still, if a sufficient number of ratings is available, these particular cases will be factored out by the high number of cases in which the interpretation of the behavior was right. In fact, Shafer et al. (2006) report that in some domains (such as personalized online radio stations) collecting the implicit feedback can even result in more accurate user models than can be done with explicit ratings.

A further discussion of costs and benefits of implicit ratings can be found in Nichols (1998).

2.3.2 Data sparsity and the cold-start problem

In the rating matrices used in the previous examples, ratings existed for all but one user–item combination. In real-world applications, of course, the rating matrices tend to be very sparse, as customers typically provide ratings for (or have bought) only a small fraction of the catalog items.

In general, the challenge in that context is thus to compute good predictions when there are relatively few ratings available. One straightforward option for dealing with this problem is to exploit additional information about the users, such as gender, age, education, interests, or other available information that can help to classify the user. The set of similar users (neighbors) is thus based not only on the analysis of the explicit and implicit ratings, but also on information external to the ratings matrix. These systems – such as the hybrid one mentioned by Pazzani (1999b), which exploits demographic information – are,

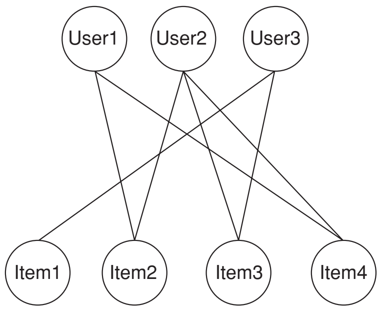


Figure 2.2. Graphical representation of user–item relationships.

however, no longer "purely" collaborative, and new questions of how to acquire the additional information and how to combine the different classifiers arise. Still, to reach the critical mass of users needed in a collaborative approach, such techniques might be helpful in the ramp-up phase of a newly installed recommendation service.

Over the years, several approaches to deal with the cold-start and data sparsity problems have been proposed. Here, we discuss one graph-based method proposed by Huang et al. (2004) as one example in more detail. The main idea of their approach is to exploit the supposed "transitivity" of customer tastes and thereby augment the matrix with additional information[2].

Consider the user-item relationship graph in Figure 2.2, which can be inferred from the binary ratings matrix in Table 2.3 (adapted from Huang et al. (2004)).

A 0 in this matrix should not be interpreted as an explicit (poor) rating, but rather as a missing rating. Assume that we are looking for a recommendation for *User1*. When using a standard CF approach, *User2* will be considered a peer for *User1* because they both bought *Item2* and *Item4*. Thus *Item3* will be recommended to *User1* because the nearest neighbor, *User2*, also bought or liked it. Huang et al. (2004) view the recommendation problem as a graph analysis problem, in which recommendations are determined by determining paths between users and items. In a standard user-based or item-based CF approach, paths of length 3 will be considered – that is, *Item3* is relevant for *User1* because there exists a three-step path (*User1–Item2–User2–Item3*) between them. Because the number of such paths of length 3 is small in sparse rating databases, the idea is to also consider longer paths (indirect associations) to compute recommendations. Using path length 5, for instance, would allow

[2] A similar idea of exploiting the neighborhood relationships in a recursive way was proposed by Zhang and Pu (2007).

Table 2.3. *Ratings database for spreading activation approach.*

	Item1	Item2	Item3	Item4
User1	0	1	0	1
User2	0	1	1	1
User3	1	0	1	0

for the recommendation also of *Item1*, as two five-step paths exist that connect *User1* and *Item1*.

Because the computation of these distant relationships is computationally expensive, Huang et al. (2004) propose transforming the rating matrix into a bipartite graph of users and items. Then, a specific graph-exploring approach called *spreading activation* is used to analyze the graph in an efficient manner. A comparison with the standard user-based and item-based algorithms shows that the quality of the recommendations can be significantly improved with the proposed technique based on indirect relationships, in particular when the ratings matrix is sparse. Also, for new users, the algorithm leads to measurable performance increases when compared with standard collaborative filtering techniques. When the rating matrix reaches a certain density, however, the quality of recommendations can also decrease when compared with standard algorithms. Still, the computation of distant relationships remains computationally expensive; it has not yet been shown how the approach can be applied to large ratings databases.

Default voting, as described by Breese et al. (1998), is another technique of dealing with sparse ratings databases. Remember that standard similarity measures take into account only items for which both the active user and the user to be compared will have submitted ratings. When this number is very small, coincidental rating commonalities and differences influence the similarity measure too much. The idea is therefore to assign default values to items that only one of the two users has rated (and possibly also to some additional items) to improve the prediction quality of sparse rating databases (Breese et al. 1998). These artificial default votes act as a sort of damping mechanism that reduces the effects of individual and coincidental rating similarities.

More recently, another approach to deal with the data sparsity problem was proposed by Wang et al. (2006). Based on the observation that most collaborative recommenders use only a certain part of the information – either user similarities or item similarities – in the ratings databases, they suggest combining the two different similarity types to improve the prediction accuracy.

In addition, a third type of information ("similar item ratings made by similar users"), which is not taken into account in previous approaches, is exploited in their prediction function. The "fusion" and smoothing of the different predictions from the different sources is accomplished in a probabilistic framework; first experiments show that the prediction accuracy increases, particularly when it comes to sparse rating databases.

The cold-start problem can be viewed as a special case of this sparsity problem (Huang et al. 2004). The questions here are (a) how to make recommendations to new users that have not rated any item yet and (b) how to deal with items that have not been rated or bought yet. Both problems can be addressed with the help of hybrid approaches – that is, with the help of additional, external information (Adomavicius and Tuzhilin 2005). For the new-users problem, other strategies are also possible. One option could be to ask the user for a minimum number of ratings before the service can be used. In such situations the system could intelligently ask for ratings for items that, from the viewpoint of information theory, carry the most information (Rashid et al. 2002). A similar strategy of asking the user for a *gauge set* of ratings is used for the Eigentaste algorithm presented by Goldberg et al. (2001).

2.4 Further model-based and preprocessing-based approaches

Collaborative recommendation techniques are often classified as being either *memory-based* or *model-based*. The traditional user-based technique is said to be memory-based because the original rating database is held in memory and used directly for generating the recommendations. In model-based approaches, on the other hand, the raw data are first processed offline, as described for item-based filtering or some dimensionality reduction techniques. At run time, only the precomputed or "learned" model is required to make predictions. Although memory-based approaches are theoretically more precise because full data are available for generating recommendations, such systems face problems of scalability if we think again of databases of tens of millions of users and millions of items.

In the next sections, we discuss some more model-based recommendation approaches before we conclude with a recent practical-oriented approach.

2.4.1 Matrix factorization/latent factor models

The Netflix Prize competition, which was completed in 2009, showed that advanced matrix factorization methods, which were employed by many participating teams, can be particularly helpful to improve the predictive accuracy of recommender systems[3].

Roughly speaking, matrix factorization methods can be used in recommender systems to derive a set of latent (hidden) factors from the rating patterns and characterize both users and items by such vectors of factors. In the movie domain, such automatically identified factors can correspond to obvious aspects of a movie such as the genre or the type (drama or action), but they can also be uninterpretable. A recommendation for an item i is made when the active user and the item i are similar with respect to these factors (Koren et al. 2009).

This general idea of exploiting latent "semantic" factors has been successfully applied in the context of information retrieval since the late 1980s. Specifically, Deerwester et al. (1990) proposed using *singular value decomposition* (SVD) as a method to discover the latent factors in documents; in information retrieval settings, this *latent semantic analysis* (LSA) technique is also referred to as *latent semantic indexing* (LSI).

In information retrieval scenarios, the problem usually consists of finding a set of documents, given a query by a user. As described in more detail in Chapter 3, both the existing documents and the user's query are encoded as a vector of terms. A basic retrieval method could simply measure the overlap of terms in the documents and the query. However, such a retrieval method does not work well when there are synonyms such as "car" and "automobile" and polysemous words such as "chip" or "model" in the documents or the query. With the help of SVD, the usually large matrix of document vectors can be collapsed into a smaller-rank approximation in which highly correlated and co-occurring terms are captured in a single factor. Thus, LSI-based retrieval makes it possible to retrieve relevant documents even if it does not contain (many) words of the user's query.

The idea of exploiting latent relationships in the data and using matrix factorization techniques such as SVD or principal component analysis was relatively soon transferred to the domain of recommender systems (Sarwar et al. 2000b; Goldberg et al. 2001; Canny 2002b). In the next section, we will show an example of how SVD can be used to generate recommendations; the example is adapted from the one given in the introduction to SVD-based recommendation by Grigorik (2007).

[3] The DVD rental company Netflix started this open competition in 2006. A $1 million prize was awarded for the development of a CF algorithm that is better than Netflix's own recommendation system by 10 percent; see http://www.netflixprize.com.

Table 2.4. *Ratings database for SVD-based recommendation.*

	User1	User2	User3	User4
Item1	3	4	3	1
Item2	1	3	2	6
Item3	2	4	1	5
Item4	3	3	5	2

Example for SVD-based recommendation. Consider again our rating matrix from Table 2.1, from which we remove Alice and that we transpose so we can show the different operations more clearly (see Table 2.4).

Informally, the SVD theorem (Golub and Kahan 1965) states that a given matrix M can be decomposed into a product of three matrices as follows, where U and V are called *left* and *right singular vectors* and the values of the diagonal of Σ are called the *singular values*.

$$M = U \Sigma V^T \tag{2.10}$$

Because the 4×4-matrix M in Table 2.4 is quadratic, U, Σ, and V are also quadratic 4×4 matrices. The main point of this decomposition is that we can approximate the full matrix by observing only the most important features – those with the largest singular values. In the example, we calculate U, V, and Σ (with the help of some linear algebra software) but retain only the two most important features by taking only the first two columns of U and V^T, see Table 2.5.

The projection of U and V^T in the two-dimensional space (U_2, V_2^T) is shown in Figure 2.3. Matrix V corresponds to the users and matrix U to the catalog items. Although in our particular example we cannot observe any clusters of users, we see that the items from U build two groups (above and below the x-axis). When looking at the original ratings, one can observe that *Item1* and

Table 2.5. *First two columns of decomposed matrix and singular values* Σ.

U_2		V_2		Σ_2	
−0.4312452	0.4931501	−0.3593326	0.36767659	12.2215	0
−0.5327375	−0.5305257	−0.5675075	0.08799758	0	4.9282
−0.5237456	−0.4052007	−0.4428526	0.56862492		
−0.5058743	0.5578152	−0.5938829	−0.73057242		

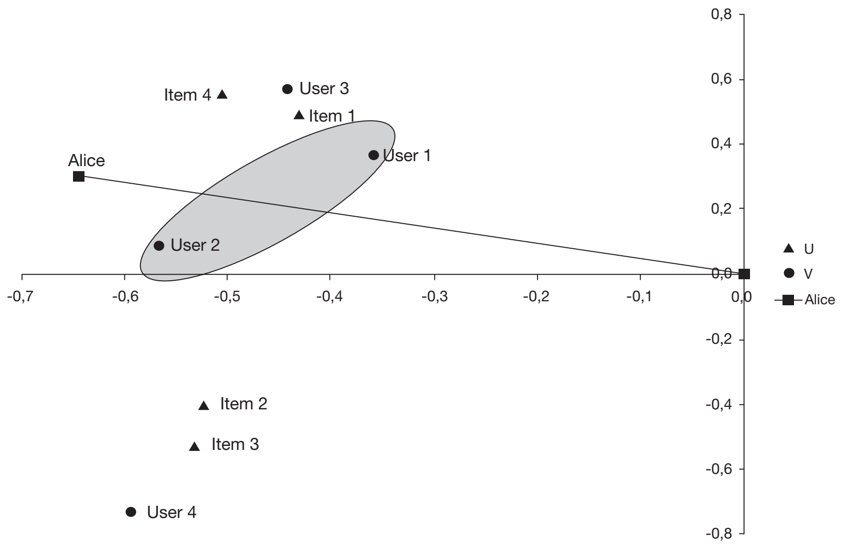


Figure 2.3. SVD-based projection in two-dimensional space.

Item4 received somewhat similar ratings. The same holds for *Item2* and *Item3*, which are depicted below the *x*-axis. With respect to the users, we can at least see that *User4* is a bit far from the others.

Because our goal is to make a prediction for Alice's ratings, we must first find out where Alice would be positioned in this two-dimensional space.

To find out Alice's datapoint, multiply Alice's rating vector [5, 3, 4, 4] by the two-column subset of U and the inverse of the two-column singular value matrix Σ.

$$Alice_{2D} = Alice \times U_2 \times \Sigma_2^{-1} = [-0.64, 0.30] \qquad (2.11)$$

Given Alice's datapoint, different strategies can be used to generate a recommendation for her. One option could be to look for neighbors in the compressed two-dimensional space and use their item ratings as predictors for Alice's rating. If we rely again on cosine similarity to determine user similarity, *User1* and *User2* will be the best predictors for Alice in the example. Again, different weighting schemes, similarity thresholds, and strategies for filling missing item ratings (e.g., based on product averages) can be used to fine-tune the prediction. Searching for neighbors in the compressed space is only one of the possible options to make a prediction for Alice. Alternatively, the interaction between user and items in the latent factor space (measured with the cosine similarity metric) can be used to approximate Alice's rating for an item (Koren et al. 2009).

Principal component analysis – Eigentaste. A different approach to dimensionality reduction was proposed by Goldberg et al. (2001) and initially applied to the implementation of a joke recommender. The idea is to preprocess the ratings database using *principal component analysis* (PCA) to filter out the "most important" aspects of the data that account for most of the variance. The authors call their method "Eigentaste," because PCA is a standard statistical analysis method based on the computation of the eigenvalue decomposition of a matrix. After the PCA step, the original rating data are projected along the most relevant of the principal eigenvectors. Then, based on this reduced dataset, users are grouped into clusters of neighbors, and the mean rating for the items is calculated. All these (computationally expensive) steps are done offline. At run time, new users are asked to rate a set of jokes (*gauge set*) on a numerical scale. These ratings are transformed based on the principal components, and the correct cluster is determined. The items with the highest ratings for this cluster are simply retrieved by a look-up in the preprocessed data. Thus, the computational complexity at run time is independent of the number of users, resulting in a "constant time" algorithm. The empirical evaluation and comparison with a basic nearest-neighborhood algorithm show that in some experiments, Eigentaste can provide comparable recommendation accuracy while the computation time can be significantly reduced. The need for a gauge set of, for example, ten ratings is one of the characteristics that may limit the practicality of the approach in some domains.

Discussion. Sarwar et al. (2000a) have analyzed how SVD-based dimensionality reduction affects the quality of the recommendations. Their experiments showed some interesting insights. In some cases, the prediction quality was worse when compared with memory-based prediction techniques, which can be interpreted as a consequence of not taking into account all the available information. On the other hand, in some settings, the recommendation accuracy was better, which can be accounted for by the fact that the dimensionality reduction technique also filtered out some "noise" in the data and, in addition, is capable of detecting nontrivial correlations in the data. To a great extent the quality of the recommendations seems to depend on the right choice of the amount of data reduction – that is, on the choice of the number of singular values to keep in an SVD approach. In many cases, these parameters can, however, be determined and fine-tuned only based on experiments in a certain domain. Koren et al. (2009) talk about 20 to 100 factors that are derived from the rating patterns.

As with all preprocessing approaches, the problem of data updates – how to integrate newly arriving ratings without recomputing the whole "model" again – also must be solved. Sarwar et al. (2002), for instance, proposed a technique that allows for the incremental update for SVD-based approaches. Similarly, George and Merugu (2005) proposed an approach based on co-clustering for building scalable CF recommenders that also support the dynamic update of the rating database.

Since the early experiments with matrix factorization techniques in recommender systems, more elaborate and specialized methods have been developed. For instance, Hofmann (2004; Hofmann and Puzicha 1999) proposed to apply *probabilistic LSA* (pLSA) a method to discover the (otherwise hidden) user communities and interest patterns in the ratings database and showed that good accuracy levels can be achieved based on that method. Hofmann's pLSA method is similar to LSA with respect to the goal of identifying hidden relationships; pLSA is, however, based not on linear algebra but rather on statistics and represents a "more principled approach which has a solid foundation in statistics" (Hofmann 1999).

An overview of recent and advanced topics in matrix factorization for recommender systems can be found in Koren et al. (2009). In this paper, Koren et al. focus particularly on the flexibility of the model and show, for instance, how additional information, such as demographic data, can be incorporated; how temporal aspects, such as changing user preferences, can be dealt with; or how existing rating bias can be taken into account. In addition, they also propose more elaborate methods to deal with missing rating data and report on some insights from applying these techniques in the Netflix prize competition.

2.4.2 Association rule mining

Association rule mining is a common technique used to identify rulelike relationship patterns in large-scale sales transactions. A typical application of this technique is the detection of pairs or groups of products in a supermarket that are often purchased together. A typical rule could be, "If a customer purchases baby food then he or she also buys diapers in 70 percent of the cases". When such relationships are known, this knowledge can, for instance, be exploited for promotional and cross-selling purposes or for design decisions regarding the layout of the shop.

This idea can be transferred to collaborative recommendation – in other words, the goal will be to automatically detect rules such as "If user *X* liked both *item1* and *item2*, then *X* will most probably also like *item5*." Recommendations for the active user can be made by evaluating which of the detected rules apply – in the example, checking whether the user liked *item1* and *item2* – and then generating a ranked list of proposed items based on statistics about the co-occurrence of items in the sales transactions.

We can describe the general problem more formally as follows, using the notation from Sarwar et al. (2000b). A (sales) *transaction* T is a subset of the set of available products $P = \{p_1, \ldots, p_m\}$ and describes a set of products that have been purchased together. Association rules are often written in the form $X \Rightarrow Y$, with X and Y being both subsets of P and $X \cap Y = \emptyset$. An association rule $X \Rightarrow Y$ (e.g., *baby-food* $\Rightarrow$ *diapers*) expresses that whenever the elements of X (the rule *body*) are contained in a transaction T, it is very likely that the elements in Y (the rule *head*) are elements of the same transaction.

The goal of rule-mining algorithms such as Apriori (Agrawal and Srikant 1994) is to automatically detect such rules and calculate a measure of quality for those rules. The standard measures for association rules are *support* and *confidence*. The *support* of a rule $X \Rightarrow Y$ is calculated as the percentage of transactions that contain all items of $X \cup Y$ with respect to the number of overall transactions (i.e., the probability of co-occurrence of X and Y in a transaction). *Confidence* is defined as the ratio of transactions that contain all items of $X \cup Y$ to the number of transactions that contain only X – in other words, confidence corresponds to the conditional probability of Y given X.

More formally,

$$support = \frac{\textit{number of transactions containing } X \cup Y}{\textit{number of transactions}} \tag{2.12}$$

$$confidence = \frac{\textit{number of transactions containing } X \cup Y}{\textit{number of transactions containing } X} \tag{2.13}$$

Let us consider again our small rating matrix from the previous section to show how recommendations can be made with a rule-mining approach. For demonstration purposes we will simplify the five-point ratings and use only a binary "like/dislike" scale. Table 2.6 shows the corresponding rating matrix; zeros correspond to "dislike" and ones to "like." The matrix was derived from Table 2.2 (showing the mean-adjusted ratings). It contains a 1 if a rating was above a user's average and a 0 otherwise.

Standard rule-mining algorithms can be used to analyze this database and calculate a list of association rules and their corresponding confidence and support values. To focus only on the relevant rules, minimum threshold values for support and confidence are typically defined, for instance, through experimentation.

Table 2.6. *Transformed ratings database for rule mining.*

	Item1	Item2	Item3	Item4	Item5
Alice	1	0	0	0	?
User1	1	0	0	1	1
User2	1	0	1	0	1
User3	0	0	0	1	1
User4	0	1	1	0	0

In the context of collaborative recommendation, a transaction could be viewed as the set of all previous (positive) ratings or purchases of a customer. A typical association that should be analyzed is the question of how likely it is that users who liked *Item1* will also like *Item5* (*Item1*$\Rightarrow$ *Item5*). In the example database, the support value for this rule (without taking Alice's ratings into account) is $2/4$; confidence is $2/2$.

The calculation of the set of interesting association rules with a sufficiently high value for confidence and support can be performed offline. At run time, recommendations for user Alice can be efficiently computed based on the following scheme described by Sarwar et al. (2000b):

(1) Determine the set of $X \Rightarrow Y$ association rules that are relevant for Alice – that is, where Alice has bought (or liked) all elements from X. Because Alice has bought *Item1*, the aforementioned rule is relevant for Alice.
(2) Compute the union of items appearing in the consequent Y of these association rules that have not been purchased by Alice.
(3) Sort the products according to the confidence of the rule that predicted them. If multiple rules suggested one product, take the rule with the highest confidence.
(4) Return the first N elements of this ordered list as a recommendation.

In the approach described by Sarwar et al. (2000b), only the actual purchases ("like" ratings) were taken into account – the system does not explicitly handle "dislike" statements. Consequently, no rules are inferred that express that, for example, whenever a user liked *Item2* he or she would *not* like *Item3*, which could be a plausible rule in our example.

Fortunately, association rule mining can be easily extended to also handle *categorical* attributes so both "like" and "dislike" rules can be derived from the data. Lin et al. (2002; Lin 2000), for instance, propose to transform the usual numerical item ratings into two categories, as shown in the example, and then to map ratings to "transactions" in the sense of standard association rule mining techniques. The detection of rules describing relationships between articles ("whenever *item2* is liked . . .") is only one of the options; the same mechanism can be used to detect like and dislike relationships between users, such as "90 percent of the articles liked by user A and user B are also liked by user C."

For the task of detecting the recommendation rules, Lin et al. (2002) propose a mining algorithm that takes the particularities of the domain into account and specifically searches only for rules that have a certain *target item* (user or article) in the rule head. Focusing the search for rules in that way not only improves the algorithm's efficiency but also allows for the detection of rules for infrequently bought items, which could be filtered out in a global search because of their limited support value. In addition, the algorithm can be parameterized with lower and upper bounds on the number of rules it should try to identify.

Depending on the mining scenario, different strategies for determining the set of recommended items can be used. Let us assume a scenario in which associations between customers instead of items are mined. An example of a detected rule would therefore be "If *User1* likes an item, and *User2* dislikes the item, Alice (the target user) will like the item."

To determine whether an item will be liked by Alice, we can check, for each item, whether the rule "fires" for the item – that is, if *User1* liked it and *User2* disliked it. Based on confidence and support values of these rules, an overall score can be computed for each item as follows (Lin et al. 2002):

$$score_{item_i} = \sum_{rules\ recommending\ item_i} (support_{rule} * confidence_{rule}) \tag{2.14}$$

If this overall item score surpasses a defined threshold value, the item will be recommended to the target user. The determination of a suitable threshold was done by Lin et al. (2002) based on experimentation.

When item (article) associations are used, an additional cutoff parameter can be determined in advance that describes some minimal support value. This cutoff not only reduces the computational complexity but also allows for the detection of rules for articles that have only a very few ratings.

In the experiments reported by Lin et al. (2002), a mixed strategy was implemented that, as a default, not only relies on the exploitation of user associations but also switches to article associations whenever the support values of the user association rules are below a defined threshold. The evaluation shows that user associations generally yield better results than article associations; article associations can, however, be computed more quickly. It can also be observed from the experiments that a limited number of rules is sufficient for generating good predictions and that increasing the number of rules does not contribute any more to the prediction accuracy. The first observation is interesting because it contrasts the observation made in nearest-neighbor approaches described earlier in which item-to-item correlation approaches have shown to lead to better results.

A comparative evaluation finally shows that in the popular movie domain, the rule-mining approach outperforms other algorithms, such as the one presented by Billsus and Pazzani (1998a), with respect to recommendation quality. In another domain – namely, the recommendation of interesting pages for web users, Fu et al. (2000), also report promising results for using association rules as a mechanism for predicting the relevance of individual pages. In contrast to many other approaches, they do not rely on explicit user ratings for web pages but rather aim to automatically store the navigation behavior of many users in a central repository and then to learn which users are similar with respect to their interests. More recent and elaborate works in that direction, such as those by Mobasher et al. (2001) or Shyu et al. (2005), also rely on web usage data and association rule mining as core mechanisms to predict the relevance of web pages in the context of adaptive user interfaces and web page recommendations.

2.4.3 Probabilistic recommendation approaches

Another way of making a prediction about how a given user will rate a certain item is to exploit existing formalisms of probability theory[4].

A first, and very simple, way to implement collaborative filtering with a probabilistic method is to view the prediction problem as a *classification problem*, which can generally be described as the task of "assigning an object to one of several predefined categories" (Tan et al. 2006). As an example of a classification problem, consider the task of classifying an incoming e-mail message as spam or non-spam. In order to automate this task, a function has to be developed that defines – based, for instance, on the words that occur in the message header or content – whether the message is classified as a spam e-mail or not. The classification task can therefore be seen as the problem of *learning* this mapping function from training examples. Such a function is also informally called the *classification model*.

One standard technique also used in the area of data mining is based on *Bayes classifiers*. We show, with a simplified example, how a basic probabilistic method can be used to calculate rating predictions. Consider a slightly different

[4] Although the selection of association rules, based on support and confidence values, as described in the previous section is also based on statistics, association rule mining is usually not classified as a probabilistic recommendation method.

Table 2.7. *Probabilistic models: the rating database.*

	Item1	Item2	Item3	Item4	Item5
Alice	1	3	3	2	?
User1	2	4	2	2	4
User2	1	3	3	5	1
User3	4	5	2	3	3
User4	1	1	5	2	1

ratings database (see Table 2.7). Again, a prediction for Alice's rating of *Item5* is what we are interested in.

In our setting, we formulate the prediction task as the problem of calculating the most probable rating value for *Item5*, given the set of Alice's other ratings and the ratings of the other users. In our method, we will calculate *conditional probabilities* for each possible rating value given Alice's other ratings, and then select the one with the highest probability as a prediction[5].

To predict the probability of rating value 1 for *Item5* we must calculate the conditional probability $P(\mathit{Item5} = 1|X)$, with X being Alice's other ratings: $X = (\mathit{Item1} = 1, \mathit{Item2} = 3, \mathit{Item3} = 3, \mathit{Item4} = 2)$.

For the calculation of this probability, the Bayes theorem is used, which allows us to compute this posterior probability $P(Y|X)$ through the *class-conditional* probability $P(X|Y)$, the probability of Y (i.e., the probability of a rating value 1 for *Item5* in the example), and the probability of X, more formally

$$P(Y|X) = \frac{P(X|Y) \times P(Y)}{P(X)} \tag{2.15}$$

Under the assumption that the attributes (i.e., the ratings users) are *conditionally independent*, we can compute the posterior probability for each value of Y with a *naive* Bayes classifier as follows, d being the number of attributes in each X:

$$P(Y|X) = \frac{\prod_{i=1}^{d} P(X_i|Y) \times P(Y)}{P(X)} \tag{2.16}$$

In many domains in which naive Bayes classifiers are applied, the assumption of conditional independence actually does not hold, but such classifiers perform well despite this fact.

[5] Again, a transformation of the ratings database into "like" and "dislike" statements is possible (Miyahara and Pazzani 2000).

As $P(X)$ is a constant value, we can omit it in our calculations. $P(Y)$ can be estimated for each rating value based on the ratings database: *P(Item5=1)* $=$ *2/4* (as two of four ratings for *Item5* had the value 1), *P(Item5=2)=0*, and so forth. What remains is the calculation of all class-conditional probabilities $P(X_i|Y)$:

$$\begin{aligned}
\text{P(X|Item5=1)} &= \text{P(Item1=1|Item5=1)} \times \text{P(Item2=3|Item5=1)} \\
&\quad \times \text{P(Item3=3|Item5=1)} \times \text{P(Item4=2|Item5=1)} \\
&= 2/2 \times 1/2 \times 1/2 \times 1/2 \\
&= 0.125
\end{aligned}$$

$$\begin{aligned}
\text{P(X|Item5=2)} &= \text{P(Item1=1|Item5=2)} \times \text{P(Item2=3|Item5=2)} \\
&\quad \times \text{P(Item3=3|Item5=2)} \times \text{P(Item4=2|Item5=2)} \\
&= 0/0 \times \cdots \times \cdots \times \cdots \\
&= 0
\end{aligned}$$

Based on these calculations, given that *P(Item5=1)* $= 2/4$ and omitting the constant factor $P(X)$ in the Bayes classifier, the posterior probability of a rating value 1 for *Item5* is $P(\mathit{Item5} = 1|X) = 2/4 \times 0.125 = 0.0625$. In the example ratings database, *P(Item5=1)* is higher than all other probabilities, which means that the probabilistic rating prediction for Alice will be 1 for *Item5*. One can see in the small example that when using this simple method the estimates of the posterior probabilities are 0 if one of the factors is 0 and that in the worst case, a rating vector cannot be classified. Techniques such as using the *m*-estimate or Laplace smoothing are therefore used to smooth conditional probabilities, in particular for sparse training sets. Of course, one could also – as in the association rule mining approach – use a preprocessed rating database and use "like" and "dislike" ratings and/or assign default ratings only to missing values.

The simple method that we developed for illustration purposes is computationally complex, does not work well with small or sparse rating databases, and will finally lead to probability values for each rating that differ only very slightly from each other. More advanced probabilistic techniques are thus required.

The most popular approaches that rely on a probabilistic model are based on the idea of grouping similar users (or items) into *clusters*, a technique that, in general, also promises to help with the problems of data sparsity and computational complexity.

The naïve Bayes model described by Breese et al. (1998) therefore includes an additional unobserved class variable C that can take a small number of discrete values that correspond to the clusters of the user base. When, again, conditional independence is assumed, the following formula expresses the probability model (mixture model):

$$P(C = c, v_1, \ldots, v_n) = P(C = c) \prod_{i=1}^{n} P(v_i | C = c) \qquad (2.17)$$

where $P(C = c, v_1, \ldots, v_n)$ denotes the probability of observing a full set of values for an individual of class c. What must be derived from the training data are the probabilities of class membership, $P(C = c)$, and the conditional probabilities of seeing a certain rating value when the class is known, $P(v_i | C = c)$. The problem that remains is to determine the parameters for a model and estimate a good number of clusters to use. This information is not directly contained in the ratings database but, fortunately, standard techniques in the field, such as the expectation maximization algorithm (Dempster et al. 1977), can be applied to determine the model parameters.

At run time, a prediction for a certain user u and an item i can be made based on the probability of user u falling into a certain cluster and the probability of liking item i when being in a certain cluster given the user's ratings; see Ungar and Foster (1998) for more details on model estimation for probabilistic clustering for collaborative filtering.

Other methods for clustering can be applied in the recommendation domain to for instance, reduce the complexity problem. Chee et al. (2001), for example, propose to use a modified k-means clustering algorithm to partition the set of users in homogeneous or cohesive groups (clusters) such that there is a high similarity between users with respect to some similarity measure in one cluster and the interpartition similarity is kept at a low level. When a rating prediction has to be made for a user at run time, the system determines the group of the user and then takes the ratings of only the small set of members of this group into account when computing a weighted rating value. The performance of such an algorithm depends, of course, on the number of groups and the respective group size. Smaller groups are better with respect to run-time performance; still, when groups are too small, the recommendation accuracy may degrade. Despite the run-time savings that can be achieved with this technique, such in-memory approaches do not scale for really large databases. A more recent approach that also relies on clustering users with the same tastes into groups with the help of k-means can be found in Xue et al. (2005).

Coming back to the probabilistic approaches, besides such naive Bayes approaches, Breese et al. (1998) also propose another form of implementing a Bayes classifier and modeling the class-conditional probabilities based on *Bayesian belief networks*. These networks allow us to make existing dependencies between variables explicit and encode these relationships in a directed acyclic graph. Model building thus first requires the system to learn the structure of the network (see Chickering et al. 1997) before the required conditional probabilities can be estimated in a second step.

The comparison of the probabilistic methods with other approaches, such as a user-based nearest-neighbor algorithm, shows that the technique based on Bayesian networks slightly outperforms the other algorithms in some test domains, although not in all. In a summary over all datasets and evaluation protocols, the Bayesian network method also exhibits the best overall performance (Breese et al. 1998). For some datasets, however – as in the popular movie domain – the Bayesian approach performs significantly worse than a user-based approach extended with default voting, inverse user frequency, and case amplification.

In general, Bayesian classification methods have the advantages that individual noise points in the data are averaged out and that irrelevant attributes have little or no impact on the calculated posterior probabilities (Tan et al. 2006). Bayesian networks have no strong trend of *overfitting* the model – that is, they can almost always learn appropriately generalized models, which leads to good predictive accuracy. In addition, they can also be used when the data are incomplete.

As a side issue, the run-time performance of the probabilistic approaches described herein is typically much better than that for memory-based approaches, as the model itself is learned offline and in advance. In parallel, Breese et al. (1998) argue that probabilistic approaches are also favorable with respect to memory requirements, partly owing to the fact that the resulting belief networks remain rather small.

An approach similar to the naive Bayes method of Breese et al. (1998) is described by Chen and George (1999), who also provide more details about the treatment of missing ratings and how users can be clustered based on the introduction of a hidden (latent) variable to model group membership. Miyahara and Pazzani (2000), propose a comparably straightforward but effective collaborative filtering technique based on a simple Bayes classifier and, in particular, also discuss the aspect of *feature selection*, a technique commonly used to leave out irrelevant items (features), improve accuracy, and reduce computation time.

A more recent statistical method that uses latent class variables to discover groups of similar users and items is that proposed by Hofmann (2004; Hofmann and Puzicha 1999), and it was shown that further quality improvements can be achieved when compared with the results of Breese et al. (1998). This method is also employed in Google's news recommender, which will be discussed in the next section. A recent comprehensive overview and comparison of different probabilistic approaches and mixture models can be found in Jin et al. (2006).

Further probabilistic approaches are described by Pennock et al. (2000) and Yu et al. (2004), both aiming to combine the ideas and advantages of model-based and memory-based recommendations in a probabilistic framework. Yu et al. (2004) develop what they call a "memory-based probabilistic framework for collaborative filtering". As a framework, it is particularly designed to accommodate extensions for particular challenges such as the new user problem or the problem of selecting a set of peer users from the ratings database; all these extensions are done in a principled, probabilistic way. The new user problem can be addressed in this framework through an *active learning approach* – that is, by asking a new user to rate a set of items as also proposed by Goldberg et al. (2001). The critical task of choosing the items that the new user will hopefully be able to rate is done on a decision-theoretic and probabilistic basis. Moreover, it is also shown how the process of generating and updating the *profile space*, which contains the most "informative" users in the user database and which is constructed to reduce the computational complexity, can be embedded in the same probabilistic framework. Although the main contribution, as the authors state it, is the provision of a framework that allows for extensions on a sound probabilistic basis, their experiments show that with the proposed techniques, comparable or superior prediction accuracy can be achieved when compared, for instance, with the results reported for probabilistic methods described by Breese et al. (1998).

2.5 Recent practical approaches and systems

Our discussion so far has shown the broad spectrum of different techniques that can, in principle, be used to generate predictions and recommendations based on the information from a user–item rating matrix. We can observe that these approaches differ not only with respect to their recommendation quality (which is the main goal of most research efforts) but also in the complexity of the algorithms themselves. Whereas the first memory-based algorithms are also rather straightforward with respect to implementation aspects, others are based on sophisticated (preprocessing and model-updating) techniques. Although mathematical software libraries are available for many methods, their usage requires in-depth mathematical expertise,[6] which may hamper the

[6] Such expertise is required, in particular, when the used approach is computationally complex and the algorithms must be applied in an optimized way.

Table 2.8. *Slope One prediction for* Alice *and* Item5 $= 2 + (2 - 1) = 3$.

	Item1	Item5
Alice	2	?
User1	1	2

practical usage of these approaches, in particular for small-sized businesses. In a recent paper, Lemire and Maclachlan (2005) therefore proposed a new and rather simple recommendation technique that, despite its simplicity, leads to reasonable recommendation quality. In addition to the goal of easy implementation for "an average engineer", their Slope One prediction schemes should also support on-the-fly data updates and efficient run-time queries. We discuss this method, which is in practical use on several web sites, in the next section.

In general, the number of publicly available reports on real-world commercial recommender systems (large scale or not) is still limited. In a recent paper, Das et al. (2007), report in some detail on the implementation of Google's news personalization engine that was designed to provide personalized recommendations in real time. A summary of this approach concludes the section and sheds light on practical aspects of implementing a large-scale recommender system that has an item catalog consisting of several million news articles and is used by millions of online users.

2.5.1 Slope One predictors

The original idea of "Slope One" predictors is simple and is based on what the authors call a "popularity differential" between items for users. Consider the following example (Table 2.8), which is based on a pairwise comparison of how items are liked by different users.

In the example, *User1* has rated *Item1* with 1 and *Item5* with 2. Alice has rated *Item1* with 2. The goal is to predict Alice's rating for *Item5*. A simple prediction would be to add to Alice's rating for *Item1* the relative difference of *User1*'s ratings for these two items: $p(Alice, Item5) = 2 + (2 - 1) = 3$. The ratings database, of course, consists of many such pairs, and one can take the average of these differences to make the prediction.

In general, the problem consists of finding functions of the form $f(x) = x + b$ (that is why the name is "Slope One") that predict, for a pair of items, the rating for one item from the rating of the other one.

Table 2.9. *Slope One prediction: a more detailed example.*

	Item1	Item2	Item3
Alice	2	5	?
User1	3	2	5
User2	4		3

Let us now look at the following slightly more complex example (Table 2.9) in which we search for a prediction for Alice for *Item3*.[7]

There are two co-ratings of *Item1* and *Item3*. One time *Item3* is rated two points higher ($5 - 3 = 2$), and the other time one point lower, than *Item1* ($3 - 4 = -1$). The average distance between these items is thus $(2 + (-1))/2 = 0.5$. There is only one co-rating of *Item3* and *Item2* and the distance is $(5 - 2) = 3$. The prediction for *Item3* based on *Item1* and Alice's rating of 2 would therefore be $2 + 0.5 = 2.5$. Based on *Item2*, the prediction is $5 + 3 = 8$. An overall prediction can now be made by taking the number of co-ratings into account to give more weight to deviations that are based on more data:

$$pred(Alice, Item3) = \frac{2 \times 2.5 + 1 \times 8}{2 + 1} = 4.33 \tag{2.18}$$

In detail, the approach can be described as follows, using a slightly different notation (Lemire and Maclachlan 2005) that makes the description of the calculations simpler. The whole ratings database shall be denoted R, as usual. The ratings of a certain user are contained in an incomplete array u, u_i being u's ratings for item i. Lemire and Maclachlan (2005) call such an array an *evaluation*, and it corresponds to a line in the matrix R. Given two items j and i, let $S_{j,i}(R)$ denote the set of evaluations that contain both ratings for i and j – that is, the lines that contain the co-ratings. The average deviation *dev* of two items i and j is then calculated as follows:

$$dev_{j,i} = \sum_{(u_j,u_i)\in S_{j,i}(R)} \frac{u_j - u_i}{|S_{j,i}(R)|} \tag{2.19}$$

As shown in the example, from every co-rated item i we can make a prediction for item j and user u as $dev_{j,i} + u_i$. A simple combination of these

[7] Adapted from Wikipedia (2008).

individual predictions would be to compute the average over all co-rated items:

$$pred(u, j) = \frac{\sum_{i \in Relevant(u,j)} (dev_{j,i} + u_i)}{|Relevant(u, j)|} \tag{2.20}$$

The function $Relevant(u, j)$ denotes the set of relevant items – those that have at least one co-rating with j by user u. In other words, $Relevant(u, j) = \{i | i \in S(u), i \neq j, |S_{j,i}(R)| > 0\}$, where $S(u)$ denotes the set of entries in u that contain a rating. This formula can be simplified in realistic scenarios and sufficiently dense datasets to $Relevant(u, j) = S(u) - \{j\}$ when $j \in S(u)$.

The intuitive problem of that basic scheme is that it does not take the number of co-rated items into account, although it is obvious that a predictor will be better if there is a high number of co-rated items. Thus, the scheme is extended in such a way that it weights the individual deviations based on the number of co-ratings as follows:

$$pred(u, j) = \frac{\sum_{i \in S(u)-\{j\}} (dev_{j,i} + u_i) * |S_{j,i}(R)|}{\sum_{i \in S(u)-\{j\}} *|S_{j,i}(R)|} \tag{2.21}$$

Another way of enhancing the basic prediction scheme is to weight the deviations based on the "like and dislike" patterns in the rating database (bipolar scheme). To that purpose, when making a prediction for user j, the relevant item ratings (and deviations) are divided into two groups, one group containing the items that were liked by both users and one group containing items both users disliked. A prediction is made by combining these deviations. The overall effect is that the scheme takes only those ratings into account in which the users agree on a positive or negative rating. Although this might seem problematic with respect to already sparse ratings databases, the desired effect is that the prediction scheme "predicts nothing from the fact that user A likes item K and user B dislikes the same item K" (Lemire and Maclachlan 2005).

When splitting the ratings into like and dislike groups, one should also take the particularities of real-world rating databases into account. In fact, when given a five-point scale (1–5), it can be observed that in typical datasets around 70 percent of the ratings are above the theoretical average of 3. This indicates that, in general, users either (a) tend to provide ratings for items that they like or (b) simply have a tendency to give rather high ratings and interpret a value of 3 to be a rather poor rating value. In the bipolar prediction scheme discussed here, the threshold was thus set to the average rating value of a user instead of using an overall threshold.

An evaluation of the Slope One predictors on popular test databases revealed that the quality of recommendations (measured by the usual metrics; see

Section 7.4.2) is comparable with the performance of existing approaches, such as collaborative filtering based on Pearson correlation and case amplification. The extensions of the basic scheme (weighted predictors, bipolar scheme) also result in a performance improvement, although these improvements remain rather small (1% to 2%) and are thus hardly significant.

Overall, despite its simplicity, the proposed item-based and ratings-based algorithm shows a reasonable performance on popular rating databases. In addition, the technique supports both dynamic updates of the predictions when new ratings arrive and efficient querying at run time (in exchange for increased memory requirements, of course). In the broader context, such rather simple techniques and the availability of small, open-source libraries in different popular programming languages can help to increase the number of real-world implementations of recommender systems.

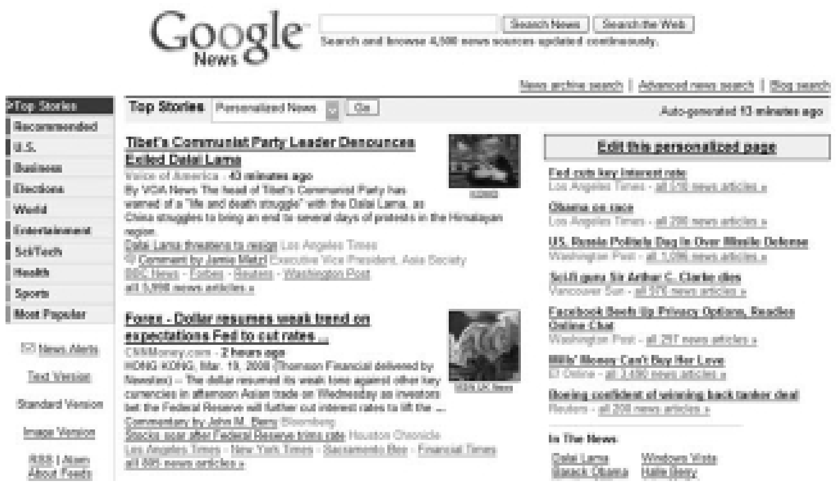
From an scientific perspective, however, a better understanding and more evaluations on different datasets are required to really understand the particular characteristics of the proposed Slope One algorithms in different applications and settings.

2.5.2 The Google News personalization engine

Google News is an online information portal that aggregates news articles from several thousand sources and displays them (after grouping similar articles) to signed-in users in a personalized way; see Figure 2.4. The recommendation approach is a collaborative one and is based on the click history of the active user and the history of the larger community – that is, a click on a story is interpreted as a positive rating. More elaborate rating acquisition and interpretation techniques are possible, of course; see, for instance, the work of Joachims et al. (2005).

On the news portal, the recommender system is used to fill one specific section with a personalized list of news articles. The main challenges are that (a) the goal is to generate the list in real time, allowing at most one second for generating the content and (b) there are very frequent changes in the "item catalog", as there is a constant stream of new items, whereas at the same time other articles may quickly become out of date. In addition, one of the goals is to immediately react to user interaction and take the latest article reads into account.

Because of the vast number of articles and users and the given response-time requirements, a pure memory-based approach is not applicable and a combination of model-based and memory-based techniques is used. The model-based part is based on two clustering techniques, *probabilistic latent semantic*


Google News

Search News | Search the Web

Search and browse 4,500 news sources updated continuously.

News archive search | Advanced news search | Blog search

Top Stories
Recommended
U.S.
Business
Elections
World
Entertainment
Sci/Tech
Health
Sports
Most Popular

News Alerts
Text Version
Standard Version
Image Version
RSS | Atom
About Feeds

Top Stories Personalized News Go

Auto-generated 13 minutes ago

Tibet's Communist Party Leader Denounces Exiled Dalai Lama
Voice of America - 43 minutes ago
By VOA News The head of Tibet's Communist Party has warned of a "life and death struggle" with the Dalai Lama, as China struggles to bring an end to several days of protests in the Himalayan region.
Dalai Lama threatens to resign Los Angeles Times
Comment by Jamie Metzl Executive Vice President, Asia Society
BBC News - Forbes - Reuters - Washington Post
all 5,998 news articles »

Forex - Dollar resumes weak trend on expectations Fed to cut rates ...
CNNMoney.com - 2 hours ago
HONG KONG, Mar. 19, 2008 (Thomson Financial delivered by Newstex) -- The dollar resumed its weak tone against other key currencies in afternoon Asian trade on Wednesday as investors bet the Federal Reserve will further cut interest rates to lift the ...
Commentary by John M. Berry Bloomberg
Stocks soar after Federal Reserve trims rate Houston Chronicle
Los Angeles Times - New York Times - Sacramento Bee - Financial Times
all 885 news articles »

Edit this personalized page

Fed cuts key interest rate
Los Angeles Times - all 570 news articles »

Obama on race
Los Angeles Times - all 206 news articles »

U.S. Russia Politely Dig In Over Missile Defense
Washington Post - all 1,086 news articles »

Sci-fi guru Sir Arthur C. Clarke dies
Vancouver Sun - all 376 news articles »

Facebook Beefs Up Privacy Options, Readies Online Chat
Washington Post - all 297 news articles »

Mills' Money Can't Buy Her Love
E! Online - all 3,490 news articles »

Boeing confident of winning back tanker deal
Reuters - all 208 news articles »

In The News
Dalai Lama | Windows Vista
Barack Obama | Halle Berry

Figure 2.4. Google News portal.

indexing (PLSI) as proposed by Hofmann (2004), and – as a new proposal – MinHash as a hashing method used for putting two users in the same cluster (hash bucket) based on the overlap of items that both users have viewed. To make this hashing process scalable, both a particular method for finding the neighbors and Google's own MapReduce technique for distributing the computation over several clusters of machines are employed.

The PLSI method can be seen as a "second generation" probabilistic technique for collaborative filtering that – similar to the idea of the probabilistic clustering technique of Breese et al. (1998) discussed earlier – aims to identify clusters of like-minded users and related items. In contrast to the work of Breese et al. (1998), in which every user belongs to exactly one cluster, in Hofmann's approach hidden variables with a finite set of states for every user–item pair are introduced. Thus, such models can also accommodate the fact that users may have interests in various topics in parallel. The parameters of the resulting mixture model are determined with the standard expectation maximization (EM) method (Dempster et al. 1977). As this process is computationally very expensive, with respect to both the number of operations and the amount of required main memory, an algorithm is proposed for parallelizing the EM computation via MapReduce over several machines. Although this parallelization can significantly speed up the process of learning the probability distributions, it is clearly not sufficient to retrain the network in real time when new users or items appear, because such modifications happen far too often in this domain. Therefore, for new stories, an approximate version of PLSI is applied that can be updated in real time. A recommendation score is computed based on cluster-membership probabilities and per-cluster statistics of the number of clicks for each story. The score is normalized in the interval [0 . . . 1].

For dealing with new users, the memory-based part of the recommender that analyzes story "co-visits" is important. A co-visit means that an article has been visited by the same user within a defined period of time. The rationale of exploiting such information directly corresponds to an item-based recommendation approach, as described in previous sections. The data, however, are not preprocessed offline, but a special data structure resembling the adjacency of clicks is constantly kept up to date. Predictions are made by iterating over the recent history of the active user and retrieving the neighboring articles from memory. For calculating the actual score, the weights stored in the adjacency matrix are taken into account, and the result is normalized on a 0-to-1 interval.

At run time, the overall recommendation score for each item in a defined set of candidate items is computed as a linear combination of all the scores obtained by the three methods (MinHash, PLSI, and co-visits). The preselection of an appropriate set of recommendation candidates can be done based on different pieces of information, such as language preferences, story freshness, or other user-specific personalization settings. Alternatively, the click history of other users in the same cluster could be used to limit the set of candidate items.

The evaluation of this algorithm on different datasets (movies and news articles) revealed that, when evaluated individually, PLSI performs best, followed by MinHash and the standard similarity-based recommendation. For live data, an experiment was made in which the new technique was compared with a nonpersonalized approach, in which articles were ranked according to their recent popularity. To compare the approaches, recommendation lists were generated that interleaved items from one algorithm with the other. The experiment then measured which items received more clicks by users. Not surprisingly, the personalized approach did significantly better (around 38%) except for the not-so-frequent situations in which there were extraordinarily popular stories. The interesting question of how to weight the scores of the individual algorithms, however, remains open to some extent.

In general, what can be learned from that report is that if we have a combination of massive datasets and frequent changes in the data, significant efforts (with respect to algorithms, engineering, and parallelization) are required such that existing techniques can be employed and real-time recommendations are possible. Pure memory-based approaches are not directly applicable and for model-based approaches, the problem of continuous model updates must be solved.

What is not answered in the study is the question of whether an approach that is not content-agnostic would yield better results. We will see in the next chapter that content-based recommendation techniques – algorithms that base their recommendations on the document content and explicit or learned user interests – are particularly suited for problems of that type.

2.6 Discussion and summary

Of all the different approaches to building recommender systems discussed in this book, CF is the best-researched technique – not only because the first recommender systems built in the mid-1990s were based on user communities that rated items, but also because most of today's most successful online recommenders rely on these techniques. Early systems were built using memory-based neighborhood and correlation-based algorithms. Later, more complex and model-based approaches were developed that, for example, apply techniques from various fields, such as machine learning, information retrieval, and data mining, and often rely on algebraic methods such as SVD.

In recent years, significant research efforts went into the development of more complex probabilistic models as discussed by Adomavicius and Tuzhilin (2005), in particular because the earliest reports of these methods (as in Breese et al. 1998) indicate that they lead to very accurate predictions.

The popularity of the collaborative filtering subfield of recommender systems has different reasons, most importantly the fact that real-world benchmark problems are available and that the data to be analyzed for generating recommendations have a very simple structure: a matrix of item ratings. Thus, the evaluation of whether a newly developed recommendation technique, or the application of existing methods to the recommendation problem, outperforms previous approaches is straightforward, in particular because the evaluation metrics are also more or less standardized. One can easily imagine that comparing different algorithms is not always as easy as with collaborative filtering, in particular if more knowledge is available than just the simple rating matrix. Think, for instance, of conversational recommender applications, in which the user is interactively asked about his or her preferences and in which additional domain knowledge is encoded.

However, the availability of test databases for CF in different domains favored the further development of various and more complex CF techniques. Still, this somehow also narrowed the range of domains on which CF techniques are actually applied. The most popular datasets are about movies and books, and many researchers aim to improve the accuracy of their algorithms only on these datasets. Whether a certain CF technique performs particularly well in one domain or another is unfortunately beyond the scope of many research efforts.

In fact, given the rich number of different proposals, the question of which recommendation algorithm to use under which circumstances is still open, even if we limit our considerations to purely collaborative approaches. Moreover, the accuracy results reported on the well-known test datasets do not convey a clear picture. Many researchers compare their measurements with the already rather old results from Breese et al. (1998) and report that they can achieve better results in one or another setting and experiment. A newer basis of comparison is required, given the dozens of different techniques that have been proposed over the past decade. Based on such a comparison, a new set of "baseline" algorithms could help to get a clearer picture.

Viewed from a practical perspective, one can see that item-based CF, as reported by Linden et al. (2003) and used by Amazon.com, is scalable enough to cope with very large rating databases and also produces recommendations of reasonable quality. The number of reports on other commercial implementations and accompanying technical details (let alone datasets) is unfortunately also limited, so an industry survey in that direction could help the research community validate whether and how new proposals make their way into industrial practice.

In addition, we will see in Chapter 5, which covers hybrid recommendation approaches, that recommendation algorithms that exploit additional information about items or users and combine different techniques can achieve significantly better recommendation results than purely collaborative approaches can. When we observe the trends and developments in the recent past, we can expect that in the next years more information, both about the catalog items and about the users, will be available at very low cost, thus favoring combined or hybrid approaches. The sources of such additional knowledge can be manifold: online users share more and more information about themselves in social networks and online communities; companies exchange item information in electronic form only and increasingly adopt exchange standards including defined product classification schemes. Finally, according to the promise of the "Semantic Web," such item and community information can easily be automatically extracted from existing web sources (see, e.g., Shchekotykhin et al. 2007 for an example of such an approach).

Overall, today's CF techniques are mature enough to be employed in practical applications, provided that moderate requirements are fulfilled. Collaborative recommenders can not be applied in every domain: think of a recommender system for cars, a domain in which no buying history exists or for which the system needs a more detailed picture of the users' preferences. In parallel, CF techniques require the existence of a user community of a certain size, meaning that even for the book or movie domains one cannot apply these techniques if there are not enough users or ratings available.

Alternative approaches to product recommendation that overcome these limitations in one or the other dimension at the price of, for instance, increased development and maintenance efforts will be discussed in the next chapters.

2.7 Bibliographical notes

The earliest reports on what we now call recommender systems were published in the early 1990s. The most cited ones might be the papers on the Tapestry (Goldberg et al. 1992) and the GroupLens (Resnick et al. 1994) systems, both first published in 1992. Tapestry was developed at Xerox Parc for mail filtering and was based on the then rather new idea of exploiting explicit feedback (ratings and annotations) of other users. One of the first uses of the term "collaborative filtering" can be found in this paper. The GroupLens[8] system was also developed for filtering text documents (i.e., news articles), but was designed for use in an open community and introduced the basic idea of automatically finding similar users in the database for making predictions. The Ringo system, presented by Shardanand and Maes (1995) describes a music recommender based on collaborative filtering using Pearson's correlation measure and the mean absolute error (MAE) evaluation metric.

As mentioned earlier, the evaluation of Breese et al. (1998) still serves as an important reference point, in particular as the paper also introduces some special techniques to the compared algorithms.

The first model-based version of the Jester joke recommender (Goldberg et al. 2001) that relied on principal component analysis and clustering was initially proposed around 1999 and is still being developed further (Nathanson et al. 2007). Hofmann and Puzicha published their influential approach based on latent class models for collaborative filtering in 1999. Dimensionality reduction based on SVD was proposed by Sarwar et al. (2000a).

Item-based filtering was analyzed by Sarwar et al. (2001); a short report about Amazon.com's patented implementation and experiences are described by Linden et al. (2003).

8 The homepage of the influential GroupLens research group can be found at http://www.grouplens.org.

Excellent overview papers on CF, which partially inspired the structure of this chapter and which can serve as a starting point for further readings, are those by Schafer et al. (2006) and Anand and Mobasher (2005). Another overview with an impressive list of references to recent techniques for collaborative filtering can be found in the article by Adomavicius and Tuzhilin (2005).

Because recommender systems have their roots in various fields, research papers on collaborative filtering techniques appear in different journals and conferences. Special issues on recommender systems appeared, for instance, in the *Communications of the ACM* (1999), *ACM Transactions on Information Systems* (2004), and more recently in the *Journal of Electronic Commerce* (2007), *IEEE Intelligent Systems* (2007), and *AI Communications* (2008). Many papers also appear first at dedicated workshops series, such as the Intelligent Techniques for Web Personalization Workshop and, more recently, at the ACM Recommender Systems conference.