Chapter 2

Neighborhood-Based Collaborative Filtering

"*When one neighbor helps another, we strengthen our communities.*" – Jennifer Pahlka

2.1 Introduction

Neighborhood-based collaborative filtering algorithms, also referred to as *memory-based algorithms*, were among the earliest algorithms developed for collaborative filtering. These algorithms are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings. There are two primary types of neighborhood-based algorithms:

1. *User-based collaborative filtering:* In this case, the ratings provided by similar users to a *target* user A are used to make recommendations for A. The predicted ratings of A are computed as the weighted average values of these "peer group" ratings for each item.

2. *Item-based collaborative filtering:* In order to make recommendations for *target* item B, the first step is to determine a set S of items, which are most similar to item B. Then, in order to predict the rating of any particular user A for item B, the ratings in set S, which are specified by A, are determined. The weighted average of these ratings is used to compute the predicted rating of user A for item B.

An important distinction between user-based collaborative filtering and item-based collaborative filtering algorithms is that the ratings in the former case are predicted using the ratings of neighboring *users*, whereas the ratings in the latter case are predicted using

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_2

the user's *own* ratings on neighboring (i.e., closely related) *items*. In the former case, neighborhoods are defined by similarities among users (rows of ratings matrix), whereas in the latter case, neighborhoods are defined by similarities among items (columns of ratings matrix). Thus, the two methods share a complementary relationship. Nevertheless, there are considerable differences in the types of recommendations that are achieved using these two methods.

For the purpose of subsequent discussion, we assume that the user-item ratings matrix is an incomplete $m \times n$ matrix $R = [r_{uj}]$ containing m users and n items. It is assumed that only a small subset of the ratings matrix is specified or observed. Like all other collaborative filtering algorithms, neighborhood-based collaborative filtering algorithms can be formulated in one of two ways:

1. *Predicting the rating value of a user-item combination:* This is the simplest and most primitive formulation of a recommender system. In this case, the missing rating r_{uj} of the user u for item j is predicted.

2. *Determining the top-k items or top-k users:* In most practical settings, the merchant is not necessarily looking for specific ratings values of user-item combinations. Rather, it is more interesting to learn the top-k most relevant items for a particular user, or the top-k most relevant users for a particular item. The problem of determining the top-k items is more common than that of finding the top-k users. This is because the former formulation is used to present lists of recommended items to users in Web-centric scenarios. In traditional recommender algorithms, the "top-k problem" almost always refers to the process of finding the top-k items, rather than the top-k users. However, the latter formulation is also useful to the merchant because it can be used to determine the best users to target with marketing efforts.

The two aforementioned problems are closely related. For example, in order to determine the top-k items for a particular user, one can predict the ratings of each item for that user. The top-k items can be selected on the basis of the predicted rating. In order to improve efficiency, neighborhood-based methods pre-compute some of the data needed for prediction in an offline phase. This pre-computed data can be used in order to perform the ranking in a more efficient way.

This chapter will discuss various neighborhood-based methods. We will study the impact of some properties of ratings matrices on collaborative filtering algorithms. In addition, we will study the impact of the ratings matrix on recommendation effectiveness and efficiency. We will discuss the use of clustering and graph-based representations for implementing neighborhood-based methods. We will also discuss the connections between neighborhood methods and regression modeling techniques. Regression methods provide an optimization framework for neighborhood-based methods. In particular, the neighborhood-based method can be shown to be a heuristic approximation of a least-squares regression model [72]. This approximate equivalence will be shown in section 2.6. Such an optimization framework also paves the way for the integration of neighborhood methods with other optimization models, such as latent factor models. The integrated approach is discussed in detail in section 3.7 of Chapter 3.

This chapter is organized as follows. Section 2.2 discusses a number of key properties of ratings matrices. Section 2.3 discusses the key algorithms for neighborhood-based collaborative filtering algorithms. Section 2.4 discusses how neighborhood-based algorithms can be made faster with the use of clustering methods. Section 2.5 discusses the use of dimensionality reduction methods for enhancing neighborhood-based collaborative filtering algorithms.

An optimization modeling view of neighborhood-based methods is discussed in section 2.6. A linear regression approach is used to simulate the neighborhood model within a learning and optimization framework. Section 2.7 discusses how graph-based representations can be used to alleviate the sparsity problem in neighborhood methods. The summary is provided in section 2.8.

2.2 Key Properties of Ratings Matrices

As discussed earlier, we assume that the ratings matrix is denoted by R, and it is an $m \times n$ matrix containing m users and n items. Therefore, the rating of user u for item j is denoted by r_{uj}. Only a small subset of the entries in the ratings matrix are typically specified. The specified entries of the matrix are referred to as the training data, whereas the unspecified entries of the matrix are referred to as the test data. This definition has a direct analog in classification, regression, and semisupervised learning algorithms [22]. In that case, all the unspecified entries belong to a special column, which is known as the class variable or dependent variable. Therefore, the recommendation problem can be viewed as a generalization of the problem of classification and regression.

Ratings can be defined in a variety of ways, depending on the application at hand:

1. *Continuous ratings:* The ratings are specified on a continuous scale, corresponding to the level of like or dislike of the item at hand. An example of such a system is the Jester joke recommendation engine [228, 689], in which the ratings can take on any value between -10 and 10. The drawback of this approach is that it creates a burden on the user of having to think of a real value from an infinite number of possibilities. Therefore, such an approach is relatively rare.

2. *Interval-based ratings:* In interval-based ratings, the ratings are often drawn from a 5-point or 7-point scale, although 10-point and 20-point scales are also possible. Examples of such ratings could be numerical integer values from 1 to 5, from -2 to 2, or from 1 to 7. An important assumption is that the numerical values explicitly define the distances between the ratings, and the rating values are typically equidistant.

3. *Ordinal ratings:* Ordinal ratings are much like interval-based ratings, except that ordered *categorical* values may be used. Examples of such ordered categorical values might be responses such as "Strongly Disagree," "Disagree," "Neutral," "Agree," and "Strongly Agree." A major difference from interval-based ratings is that it is not assumed that the difference between any pair of adjacent ratings values is the same. However, in practice, this difference is only theoretical, because these different ordered categorical values are often assigned to equally spaced utility values. For example, one might assign the "Strongly Disagree" response to a rating value of 1, and the "Strongly Agree" response to a rating value of 5. In such cases, ordinal ratings are almost equivalent to interval-based ratings. Generally, the numbers of positive and negative responses are equally balanced in order to avoid bias. In cases where an even number of responses are used, the "Neutral" option is not present. Such an approach is referred to as the *forced choice* method because the neutral option is not present.

4. *Binary ratings:* In the case of binary ratings, only two options are present, corresponding to positive or negative responses. Binary ratings can be considered a special case of both interval-based and ordinal ratings. For example, the Pandora Internet radio station provides users with the ability to either like or dislike a particular music track.

Binary ratings are an example of the case where forced choice is imposed on the user. In cases where the user is neutral, she will often not specify a rating at all.

5. *Unary ratings:* Such systems allow the user to specify a positive preference for an item, but there is no mechanism to specify a negative preference. This is often the case in many real-world settings, such as the use of a "*like*" button on Facebook. More often, unary ratings are derived from customer *actions*. For example, the act of a customer buying an item can be considered a positive vote for an item. On the other hand, if the customer has not bought the item, then it does not necessarily indicate a dislike for the item. Unary ratings are special because they simplify the development of specialized models in these settings.

It is noteworthy that the indirect derivation of unary ratings from customer actions is also referred to as *implicit feedback*, because the customer does not explicitly provide feedback. Rather, the feedback is inferred in an implicit way through the customer's actions. Such types of "ratings" are often easier to obtain because users are far more likely to interact with items on an online site than to explicitly rate them. The setting of implicit feedback (i.e., unary ratings) is inherently different, as it can be considered the matrix completion analog of the positive-unlabeled (PU) learning problem in classification and regression modeling.

The distribution of ratings among items often satisfies a property in real-world settings, which is referred to as the *long-tail* property. According to this property, only a small fraction of the items are rated frequently. Such items are referred to as *popular* items. The vast majority of items are rated rarely. This results in a highly skewed distribution of the underlying ratings. An example of a skewed rating distribution is illustrated in Figure 2.1. The X-axis shows the index of the item in order of decreasing frequency, and the Y-axis shows the frequency with which the item was rated. It is evident that most of the items are rated only a small number of times. Such a rating distribution has important implications for the recommendation process:

1. In many cases, the high-frequency items tend to be relatively competitive items with little profit for the merchant. On the other hand, the lower frequency items have larger profit margins. In such cases, it may be advantageous to the merchant to recommend lower frequency items. In fact, analysis suggests [49] that many companies, such as Amazon.com, make most of their profit by selling items in the long tail.

2. Because of the rarity of observed ratings in the long tail it is generally more difficult to provide robust rating predictions in the long tail. In fact, many recommendation algorithms have a tendency to suggest popular items rather than infrequent items [173]. This phenomenon also has a negative impact on diversity, and users may often become bored by receiving the same set of recommendations of popular items.

3. The long tailed distribution implies that the items, which are frequently rated by users, are fewer in number. This fact has important implications for neighborhood-based collaborative filtering algorithms because the neighborhoods are often defined on the basis of these frequently rated items. In many cases, the ratings of these high-frequency items are not representative of the low-frequency items because of the inherent differences in the rating patterns of the two classes of items. As a result, the prediction process may yield misleading results. As we will discuss in section 7.6 of Chapter 7, this phenomenon can also cause misleading *evaluations* of recommendation algorithms.

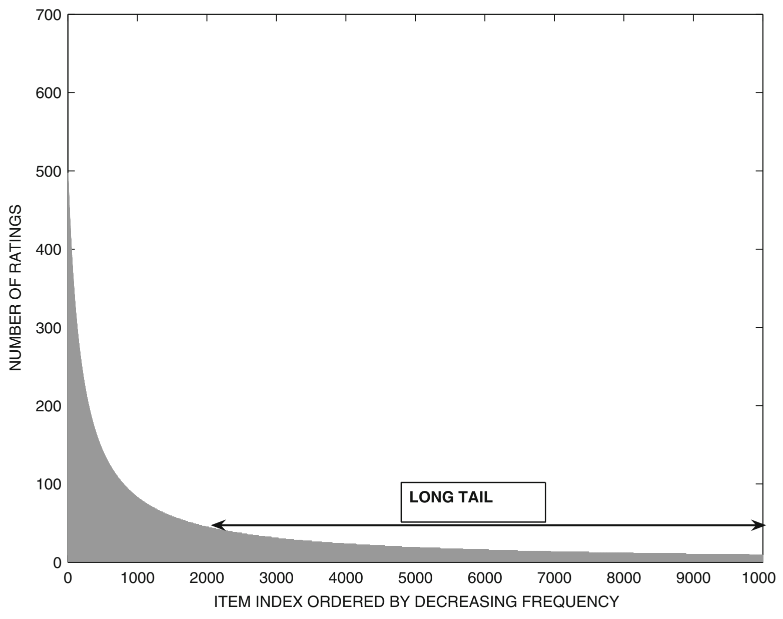


Figure 2.1: The long tail of rating frequencies

Important characteristics of ratings, such as sparsity and the long tail, need to be taken into account during the recommendation process. By adjusting the recommendation algorithms to take such real-world properties into account, it is possible to obtain more meaningful predictions [173, 463, 648].

2.3 Predicting Ratings with Neighborhood-Based Methods

The basic idea in neighborhood-based methods is to use either user-user similarity or item-item similarity to make recommendations from a ratings matrix. The concept of a *neighborhood* implies that we need to determine either similar users or similar items in order to make predictions. In the following, we will discuss how neighborhood-based methods can be used to predict the ratings of specific user-item combinations. There are two basic principles used in neighborhood-based models:

1. *User-based models:* Similar users have similar ratings on the same item. Therefore, if Alice and Bob have rated movies in a similar way in the past, then one can use Alice's observed ratings on the movie *Terminator* to predict Bob's unobserved ratings on this movie.

2. *Item-based models:* Similar items are rated in a similar way by the same user. Therefore, Bob's ratings on similar science fiction movies like *Alien* and *Predator* can be used to predict his rating on *Terminator*.

Since the collaborative filtering problem can be viewed as a generalization of the classification/regression modeling problem, neighborhood-based methods can be viewed as generalizations of nearest neighbor classifiers in the machine learning literature. Unlike

Table 2.1: User-user similarity computation between user 3 and other users

Item-Id ⇒ User-Id ⇓	1	2	3	4	5	6	Mean Rating	Cosine$(i, 3)$ (user-user)	Pearson$(i, 3)$ (user-user)
1	7	6	7	4	5	4	5.5	0.956	0.894
2	6	7	?	4	3	4	4.8	0.981	0.939
3	?	3	3	1	1	?	2	1.0	1.0
4	1	2	2	3	3	4	2.5	0.789	-1.0
5	1	?	1	2	3	3	2	0.645	-0.817

Table 2.2: Ratings matrix of Table 2.1 with mean-centering for adjusted cosine similarity computation among items. The adjusted cosine similarities of items 1 and 6 with other items are shown in the last two rows.

Item-Id ⇒ User-Id ⇓	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	-1	0	1	1
Cosine$(1, j)$ (item-item)	1	0.735	0.912	-0.848	-0.813	-0.990
Cosine$(6, j)$ (item-item)	-0.990	-0.622	-0.912	0.829	0.730	1

classification, where the nearest neighbors are always determined only on the basis of row similarity, it is possible to find the nearest neighbors in collaborative filtering on the basis of either rows or columns. This is because all missing entries are concentrated in a single column in classification, whereas the missing entries are spread out over the different rows and columns in collaborative filtering (cf. section 1.3.1.3 of Chapter 1). In the following discussion, we will discuss the details of both user-based and item-based neighborhood models, together with their natural variations.

2.3.1 User-Based Neighborhood Models

In this approach, user-based neighborhoods are defined in order to identify similar users to the *target* user for whom the rating predictions are being computed. In order to determine the neighborhood of the target user i, her similarity to all the other users is computed. Therefore, a similarity function needs to be defined between the ratings specified by users. Such a similarity computation is tricky because different users may have different scales of ratings. One user might be biased toward liking most items, whereas another user might be biased toward not liking most of the items. Furthermore, different users may have rated different items. Therefore, mechanisms need to be identified to address these issues.

For the $m \times n$ ratings matrix $R = [r_{uj}]$ with m users and n items, let I_u denote the set of item indices for which ratings have been specified by user (row) u. For example, if the ratings of the first, third, and fifth items (columns) of user (row) u are specified (observed) and the remaining are missing, then we have $I_u = \{1, 3, 5\}$. Therefore, the set of items rated by both users u and v is given by $I_u \cap I_v$. For example, if user v has rated the first four items, then $I_v = \{1, 2, 3, 4\}$, and $I_u \cap I_v = \{1, 3, 5\} \cap \{1, 2, 3, 4\} = \{1, 3\}$. It is possible (and quite common) for $I_u \cap I_v$ to be an empty set because ratings matrices are generally sparse. The set $I_u \cap I_v$ defines the mutually observed ratings, which are used to compute the similarity between the uth and vth users for neighborhood computation.

One measure that captures the similarity $\text{Sim}(u, v)$ between the rating vectors of two users u and v is the Pearson correlation coefficient. Because $I_u \cap I_v$ represents the set of item indices for which both user u and user v have specified ratings, the coefficient is computed only on this set of items. The first step is to compute the mean rating μ_u for each user u using her specified ratings:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \dots m\} \tag{2.1}$$

Then, the Pearson correlation coefficient between the rows (users) u and v is defined as follows:

$$\text{Sim}(u, v) = \text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.2}$$

Strictly speaking, the traditional definition of $\text{Pearson}(u, v)$ mandates that the values of μ_u and μ_v should be computed *only* over the items that are rated *both* by users u and v. Unlike Equation 2.1, such an approach will lead to a different value of μ_u, depending on the choice of the other user v to which the Pearson similarity is being computed. However, it is quite common (and computationally simpler) to compute each μ_u just once for each user u, according to Equation 2.1. It is hard to make an argument that one of these two ways of computing μ_u always provides strictly better recommendations than the other. In extreme cases, where the two users have only one mutually specified rating, it can be argued that using Equation 2.1 for computing μ_u will provide more informative results, because the Pearson coefficient will be indeterminate over a single common item in the traditional definition. Therefore, we will work with the simpler assumption of using Equation 2.1 in this chapter. Nevertheless, it is important for the reader to keep in mind that many implementations of user-based methods compute μ_u and μ_v in pairwise fashion during the Pearson computation.

The Pearson coefficient is computed between the target user and all the other users. One way of defining the peer group of the target user would be to use the set of k users with the highest Pearson coefficient with the target. However, since the number of observed ratings in the top-k peer group of a target user may vary significantly with the item at hand, the closest k users are found for the target user separately for each predicted item, such that each of these k users have specified ratings for that item. The weighted average of these ratings can be returned as the predicted rating for that item. Here, each rating is weighted with the Pearson correlation coefficient of its owner to the target user.

The main problem with this approach is that different users may provide ratings on different scales. One user might rate all items highly, whereas another user might rate all items negatively. The raw ratings, therefore, need to be mean-centered in row-wise fashion, before determining the (weighted) average rating of the peer group. The mean-centered rating s_{uj} of a user u for item j is defined by subtracting her mean rating from the raw rating r_{uj}.

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \dots m\} \tag{2.3}$$

As before, the weighted average of the mean-centered rating of an item in the top-k peer group of target user u is used to provide a *mean-centered* prediction. The mean rating of the target user is then added back to this prediction to provide a *raw* rating prediction $\hat{r}_{uj}$ of target user u for item j. The hat notation "^" on top of r_{uj} indicates a *predicted* rating, as opposed to one that was already observed in the original ratings matrix. Let $P_u(j)$ be the set[1] of k closest users to target user u, who have specified ratings for item j. Users with very low or negative correlations with target user u are sometimes filtered from $P_u(j)$ as a heuristic enhancement. Then, the overall neighborhood-based *prediction function* is as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} \tag{2.4}$$

This broader approach allows for a number of different variations in terms of how the similarity or prediction function is computed or in terms of which items are filtered out during the prediction process.

Example of User-Based Algorithm

Consider the example of Table 2.1. In this case, the ratings of five users $1 \ldots 5$ are indicated for six items denoted by $1 \ldots 6$. Each rating is drawn from the range $\{1 \ldots 7\}$. Consider the case where the target user index is 3, and we want to make item predictions on the basis of the ratings in Table 2.1. We need to compute the predictions $\hat{r}_{31}$ and $\hat{r}_{36}$ of user 3 for items 1 and 6 in order to determine the top recommended item.

The first step is to compute the similarity between user 3 and all the other users. We have shown two possible ways of computing similarity in the last two columns of the same table. The second-last column shows the similarity based on the raw cosine between the ratings and the last column shows the similarity based on the Pearson correlation coefficient. For example, the values of Cosine(1, 3) and Pearson(1, 3) are computed as follows:

$$\text{Cosine}(1,3) = \frac{6*3 + 7*3 + 4*1 + 5*1}{\sqrt{6^2 + 7^2 + 4^2 + 5^2} \cdot \sqrt{3^2 + 3^2 + 1^2 + 1^2}} = 0.956$$

$$\begin{aligned}
&\text{Pearson}(1,3) = \\
&= \frac{(6-5.5)*(3-2) + (7-5.5)*(3-2) + (4-5.5)*(1-2) + (5-5.5)*(1-2)}{\sqrt{1.5^2 + 1.5^2 + (-1.5)^2 + (-0.5)^2} \cdot \sqrt{1^2 + 1^2 + (-1)^2 + (-1)^2}} \\
&= 0.894
\end{aligned}$$

The Pearson and raw cosine similarities of user 3 with all other users are illustrated in the final two columns of Table 2.1. Note that the Pearson correlation coefficient is much more discriminative and the sign of the coefficient provides information about similarity and dissimilarity. The top-2 closest users to user 3 are users 1 and 2 according to both measures. By using the Pearson-weighted average of the *raw* ratings of users 1 and 2, the following predictions are obtained for user 3 with respect to her unrated items 1 and 6:

$$\hat{r}_{31} = \frac{7*0.894 + 6*0.939}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{4*0.894 + 4*0.939}{0.894 + 0.939} = 4$$

[1]In many cases, k valid peers of target user u with observed ratings for item j might not exist. This scenario is particularly common in sparse ratings matrices, such as the case where user u has less than k observed ratings. In such cases, the set $P_u(j)$ will have cardinality less than k.

Thus, item 1 should be prioritized over item 6 as a recommendation to user 3. Furthermore, the prediction suggests that user 3 is likely to be interested in *both* movies 1 and 6 to a greater degree than *any* of the movies she has already rated. This is, however, a result of the bias caused by the fact that the peer group $\{1, 2\}$ of user indices is a far more optimistic group with positive ratings, as compared to the target user 3. Let us now examine the impact of mean-centered ratings on the prediction. The mean-centered ratings are illustrated in Table 2.2. The corresponding predictions with mean-centered Equation 2.4 are as follows:

$$\hat{r}_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} \approx 3.35$$
$$\hat{r}_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} \approx 0.86$$

Thus, the mean-centered computation also provides the prediction that item 1 should be prioritized over item 6 as a recommendation to user 3. There is, however, one crucial difference from the previous recommendation. In this case, the predicted rating of item 6 is only 0.86, which is *less* than all the other items that user 3 has rated. This is a drastically different result than in the previous case, where the predicted rating for item 6 was greater than all the other items that user 3 had rated. Upon visually inspecting Table 2.1 (or Table 2.2), it is indeed evident that item 6 ought to be rated very low by user 3 (compared to her other items), because her closest peers (users 1 and 2) have also rated it lower than their other items. Thus, the mean-centering process enables a much better *relative* prediction with respect to the ratings that have already been observed. In many cases, it can also affect the relative order of the predicted items. The only weakness in this result is that the predicted rating of item 6 is 0.85, which is outside the range of allowed ratings. Such ratings can always be used for ranking, and the predicted value can be corrected to the closest value in the allowed range.

2.3.1.1 Similarity Function Variants

Several other variants of the similarity function are used in practice. One variant is to use the cosine function on the *raw* ratings rather than the mean-centered ratings:

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}} \qquad (2.5)$$

In some implementations of the raw cosine, the normalization factors in the denominator are based on all the specified items and not the mutually rated items.

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_v} r_{vk}^2}} \qquad (2.6)$$

In general, the Pearson correlation coefficient is preferable to the raw cosine because of the *bias adjustment* effect of mean-centering. This adjustment accounts for the fact that different users exhibit different levels of generosity in their global rating patterns.

The reliability of the similarity function $\text{Sim}(u, v)$ is often affected by the number of common ratings $|I_u \cap I_v|$ between users u and v. When the two users have only a small number of ratings in common, the similarity function should be reduced with a discount factor to de-emphasize the importance of that user pair. This method is referred to as *significance weighting*. The discount factor kicks in when the number of common ratings between the two users is less than a particular threshold β. The value of the discount factor is given by $\frac{\min\{|I_u \cap I_v|, \beta\}}{\beta}$, and it always lies in the range $[0, 1]$. Therefore, the discounted similarity DiscountedSim(u, v) is given by the following:

$$\text{DiscountedSim}(u, v) = \text{Sim}(u, v) \cdot \frac{\min\{|I_u \cap I_v|, \beta\}}{\beta} \tag{2.7}$$

The discounted similarity is used both for the process of determining the peer group and for computing the prediction according to Equation 2.4.

2.3.1.2 Variants of the Prediction Function

There are many variants of the prediction function used in Equation 2.4. For example, instead of mean-centering the raw rating r_{uj} to the centered value s_{uj}, one might use the Z-score z_{uj}, which further divides s_{uj} with the standard deviation σ_u of the observed ratings of user u. The standard deviation is defined as follows:

$$\sigma_u = \sqrt{\frac{\sum_{j \in I_u} (r_{uj} - \mu_u)^2}{|I_u| - 1}} \quad \forall u \in \{1 \ldots m\} \tag{2.8}$$

Then, the standardized rating is computed as follows:

$$z_{uj} = \frac{r_{uj} - \mu_u}{\sigma_u} = \frac{s_{uj}}{\sigma_u} \tag{2.9}$$

Let $P_u(j)$ denote the set of the top-k similar users of target user u, for which the ratings of item j have been observed. In this case, the predicted rating $\hat{r}_{uj}$ of target user u for item j is as follows:

$$\hat{r}_{uj} = \mu_u + \sigma_u \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot z_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.10}$$

Note that the weighted average needs to be *multiplied* with σ_u in this case. In general, if a function $g(\cdot)$ is applied during ratings normalization, then its inverse needs to be applied during the final prediction process. Although it is generally accepted that normalization improves the prediction, there seem to be conflicting conclusions in various studies on whether mean-centering or the Z-score provides higher-quality results [245, 258]. One problem with the Z-score is that the predicted ratings might frequently be outside the range of the permissible ratings. Nevertheless, even when the predicted values are outside the range of permissible ratings, they can be used to *rank* the items in order of desirability for a particular user.

A second issue in the prediction is that of the weighting of the various ratings in Equation 2.4. Each mean-centered rating s_{vj} of user v for item j is weighted with the similarity Sim(u, v) of user v to the target user u. While the value of Sim(u, v) was chosen to be the Pearson correlation coefficient, a commonly used practice is to *amplify* it by exponentiating it to the power of α. In other words, we have:

$$\text{Sim}(u, v) = \text{Pearson}(u, v)^{\alpha} \tag{2.11}$$

By choosing $\alpha > 1$, it is possible to amplify the importance of the similarity in the weighting of Equation 2.4.

As discussed earlier, neighborhood-based collaborative filtering methods are generalizations of nearest neighbor classification/regression methods. The aforementioned discussion is closer to nearest neighbor regression modeling, rather than nearest neighbor classification, because the predicted value is treated as a continuous variable throughout the prediction process. It is also possible to create a prediction function which is closer to a classification method by treating ratings as categorical values and ignoring the ordering among the ratings. Once the peer group of the target user u has been identified, the number of *votes* for each possible rating value (e.g., Agree, Neutral, Disagree) within the peer group is determined. The rating with the largest number of votes is predicted as the relevant one. This approach has the advantage of providing the most *likely* rating rather than the average rating. Such an approach is generally more effective in cases where the number of distinct ratings is small. It is also useful in the case of ordinal ratings, where the exact distances between pairs of rating values are not defined. In cases where the granularity of ratings is high, such an approach is less robust and loses a lot of ordering information among the ratings.

2.3.1.3 Variations in Filtering Peer Groups

The peer group for a target user may be defined and filtered in a wide variety of ways. The simplest approach is to use the top-k most similar users to the target user as her peer group. However, such an approach might include users that are weakly or negatively correlated with the target. Weakly correlated users might add to the error in the prediction. Furthermore, negatively correlated ratings often do not have as much predictive value in terms of potential inversion of the ratings. Although the prediction function technically allows the use of weak or negative ratings, their use is not consistent with the broader principle of neighborhood methods. Therefore, ratings with weak or negative correlations are often filtered out.

2.3.1.4 Impact of the Long Tail

As discussed in section 2.2, the distribution of ratings typically shows a long-tail distribution in many real scenarios. Some movies may be very popular and they may repeatedly occur as commonly rated items by different users. Such ratings can sometimes worsen the quality of the recommendations because they tend to be less discriminative across different users. The negative impact of these recommendations can be experienced both during the peer group computation and also during the prediction computation (cf. Equation 2.4). This notion is similar in principle to the deterioration in retrieval quality caused by popular and noninformative words (e.g., "a," "an," "the") in document retrieval applications. Therefore, the proposed solutions used in collaborative filtering are also similar to those used in the information retrieval literature. Just as the notion of *Inverse Document Frequency* (idf) exists in the information retrieval literature [400], one can use the notion of *Inverse User Frequency* in this case. If m_j is the number of ratings of item j, and m is the total number of users, then the weight w_j of the item j is set to the following:

$$w_j = \log\left(\frac{m}{m_j}\right) \quad \forall j \in \{1 \ldots n\} \tag{2.12}$$

Each item j is weighted by w_j both during the similarity computation and during the recommendation process. For example, the Pearson correlation coefficient can be modified to include the weights as follows:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{vk} - \mu_v)^2}} \tag{2.13}$$

Item weighting can also be incorporated in other collaborative filtering methods. For example, the final prediction step of item-based collaborative filtering algorithms can be modified to use weights, even though the adjusted cosine similarity between two items remains unchanged by the weights.

2.3.2 Item-Based Neighborhood Models

In item-based models, peer groups are constructed in terms of *items* rather than *users*. Therefore, similarities need to be computed between items (or columns in the ratings matrix). Before computing the similarities between the columns, each row of the ratings matrix is centered to a mean of zero. As in the case of user-based ratings, the average rating of each item in the ratings matrix is subtracted from each rating to create a mean-centered matrix. This process is identical to that discussed earlier (see Equation 2.3), which results in the computation of mean-centered ratings s_{uj}. Let U_i be the indices of the set of users who have specified ratings for item i. Therefore, if the first, third, and fourth users have specified ratings for item i, then we have $U_i = \{1, 3, 4\}$.

Then, the *adjusted* cosine similarity between the items (columns) i and j is defined as follows:

$$\text{AdjustedCosine}(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}} \tag{2.14}$$

This similarity is referred to as the adjusted cosine similarity because the ratings are mean-centered before computing the similarity value. Although the Pearson correlation can also be used on the columns in the case of the item-based method, the adjusted cosine generally provides superior results.

Consider the case in which the rating of target item t for user u needs to be determined. The first step is to determine the top-k most similar *items* to *item* t based on the aforementioned adjusted cosine similarity. Let the top-k matching items to item t, for which the user u has specified ratings, be denoted by $Q_t(u)$. The *weighted* average value of these (raw) ratings is reported as the predicted value. The weight of item j in this average is equal to the adjusted cosine similarity between item j and the target item t. Therefore, the predicted rating $\hat{r}_{ut}$ of user u for target item t is as follows:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j,t)|} \tag{2.15}$$

The basic idea is to leverage the user's *own* ratings on similar items in the final step of making the prediction. For example, in a movie recommendation system, the item peer group will typically be movies of a similar genre. The ratings history of the *same* user on such movies is a very reliable predictor of the interests of that user.

The previous section discussed a number of variants of the basic approach for user-based collaborative filtering. Because item-based algorithms are very similar to user-based algorithms, similar variants of the similarity function and the prediction function can be designed for item-based methods.

Example of Item-Based Algorithm

In order to illustrate the item-based algorithm, we will use the same example of Table 2.1, which was leveraged to demonstrate the user-based algorithm. The missing ratings of user

3 are predicted with the item-based algorithm. Because the ratings of items 1 and 6 are missing for user 3, the similarity of the columns for items 1 and 6 needs to be computed with respect to the other columns (items).

First, the similarity between items are computed after adjusting for mean-centering. The mean-centered ratings matrix is illustrated in Table 2.2. The corresponding adjusted cosine similarities of each item to 1 and 6, respectively, are indicated in the final two rows of the table. For example, the value of the adjusted cosine between items 1 and 3, denoted by AdjustedCosine(1, 3), is as follows:

$$\text{AdjustedCosine}(1,3) = \frac{1.5 * 1.5 + (-1.5) * (-0.5) + (-1) * (-1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \cdot \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}} = 0.912$$

Other item-item similarities are computed in an exactly analogous way, and are illustrated in the final two rows of Table 2.2. It is evident that items 2 and 3 are most similar to item 1, whereas items 4 and 5 are most similar to item 6. Therefore, the weighted average of the *raw* ratings of user 3 for items 2 and 3 is used to predict the rating $\hat{r}_{31}$ of item 1, whereas the weighted average of the raw ratings of user 3 for items 4 and 5 is used to predict the rating $\hat{r}_{36}$ of item 6:

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$
$$\hat{r}_{36} = \frac{1 * 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$

Thus, the item-based method also suggests that item 1 is more likely to be preferred by user 3 than item 6. However, in this case, because the ratings are predicted using the ratings of user 3 herself, the predicted ratings tend to be much more consistent with the other ratings of this user. As a specific example, it is noteworthy that the predicted rating of item 6 is no longer outside the range of allowed ratings, as in the case of the user-based method. The greater prediction accuracy of the item-based method is its main advantage. In some cases, the item-based method might provide a different set of top-k recommendations, even though the recommended lists will generally be roughly similar.

2.3.3 Efficient Implementation and Computational Complexity

Neighborhood-based methods are always used to determine the best item recommendations for a target user or the best user recommendations for a target item. The aforementioned discussion only shows how to predict the ratings for a particular user-item *combination*, but it does not discuss the actual ranking process. A straightforward approach is to compute all possible rating predictions for the relevant user-item pairs (e.g., all items for a particular user) and then rank them. While this is the basic approach used in current recommender systems, it is important to observe that the prediction process for many user-item combinations reuses many intermediate quantities. Therefore, it is advisable to have an offline phase to store these intermediate computations and then leverage them in the ranking process.

Neighborhood-based methods are always partitioned into an *offline* phase and an *online* phase. In the offline phase, the user-user (or item-item) similarity values and peer groups of the users (or items) are computed. For each user (or item), the relevant peer group is prestored on the basis of this computation. In the online phase, these similarity values and peer groups are leveraged to make predictions with the use of relationships such as Equation 2.4. Let $n' \ll n$ be the maximum number of specified ratings of a user (row), and

$m' \ll m$ be the maximum number of specified ratings of an item (column). Note that n' is the maximum running time for computing the similarity between a pair of users (rows), and m' is the maximum running time for computing the similarity between a pair of items (columns). In the case of user-based methods, the process of determining the peer group of a target user may require $O(m \cdot n')$ time. Therefore, the offline running time for computing the peer groups of all users is given by $O(m^2 \cdot n')$. For item-based methods, the corresponding offline running time is given by $O(n^2 \cdot m')$.

In order to be able to use the approach for varying values of k, one might end up having to store all pairs of nonzero similarities between pairs of users (or items). Therefore, the space requirements of user-based methods are $O(m^2)$, whereas the space requirements of item-based methods are $O(n^2)$. Because the number of users is typically greater than the number of items, the space requirements of user-based methods are generally greater than those of item-based methods.

The online computation of the predicted value according to Equation 2.4 requires $O(k)$ time for both user-based and item-based methods, where k is the size of the user/item neighborhood used for prediction. Furthermore, if this prediction needs to be executed over all items in order to rank them for a target *user*, then the running time is $O(k \cdot n)$ for both user-based and item-based methods. On the other hand, a merchant may occasionally wish to determine the top-r users to be targeted for a specific item. In this case, the prediction needs to be executed over all users in order to rank them for a target *item*, and the running time is $O(k \cdot m)$ for both user-based and item-based methods. It is noteworthy that the primary computational complexity of neighborhood-based methods resides in the offline phase, which needs to be executed occasionally. As a result, neighborhood-based methods tend to be efficient when they are used for online prediction. After all, one can afford to be generous in allocating significantly more computational time to the offline phase.

2.3.4 Comparing User-Based and Item-Based Methods

Item-based methods often provide more relevant recommendations because of the fact that a user's *own* ratings are used to perform the recommendation. In item-based methods, similar *items* are identified to a target item, and the user's own ratings on those items are used to extrapolate the ratings of the target. For example, similar items to a target historical movie might be a set of other historical movies. In such cases, the user's own recommendations for the similar set might be highly indicative of her preference for the target. This is not the case for user-based methods in which the ratings are extrapolated from other users, who might have overlapping but different interests. As a result, item-based methods often exhibit better accuracy.

Although item-based recommendations are often more likely to be accurate, the relative accuracy between item-based and user-based methods also depends on the data set at hand. As you will learn in Chapter 12, item-based methods are also more robust to *shilling attacks* in recommender systems. On the other hand, it is precisely these differences that can lead to greater diversity in the recommendation process for user-based methods over item-based methods. Diversity refers to the fact that the items in the ranked list tend to be somewhat different. If the items are not diverse, then if the user does not like the first item, she might not also like any of the other items in the list. Greater diversity also encourages serendipity, through which somewhat surprising and interesting items are discovered. Item-based methods might sometimes recommend obvious items, or items which are not *novel* from previous user experiences. The notions of novelty, diversity, and serendipity are discussed in detail in Chapter 7. Without sufficient novelty, diversity, and serendipity, users might become bored with very similar recommendations to what they have already watched.

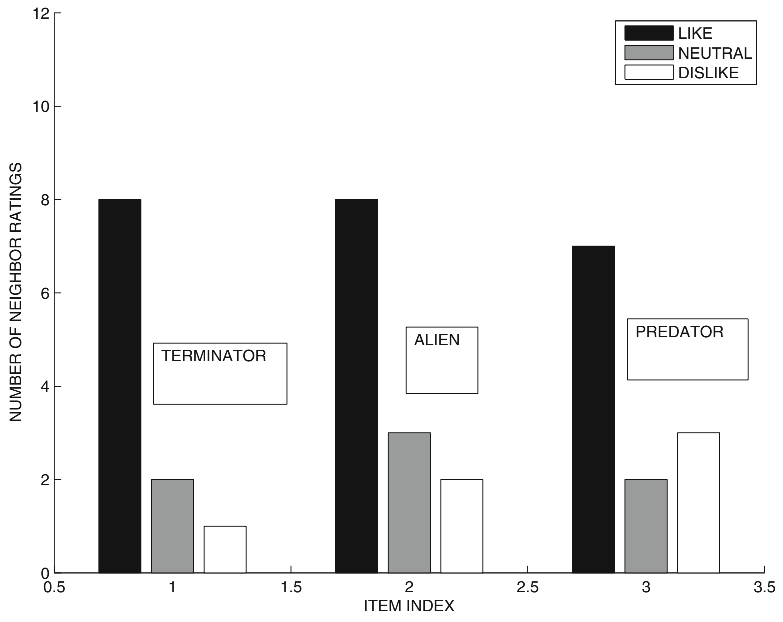


Figure 2.2: Explaining Alice's top recommendations with her neighbor rating histogram

Item-based methods can also provide a concrete reason for the recommendation. For example, Netflix often provides recommendations with statements such as the following:

Because you watched "Secrets of the Wings," [the recommendations are] $\langle List \rangle$.

Such explanations can be concretely addressed with item-based methods[2] by using the item neighborhoods. On the other hand, these explanations are harder to address withuser-based methods, because the peer group is simply a set of anonymous users and not directly usable in the recommendation process.

User-based methods provide different types of explanations. For example, consider a scenario where the movies *Terminator*, *Alien*, and *Predator*, are recommended to Alice. Then, a histogram of her neighbor's ratings for these movies can be shown to her. An example of such a histogram is shown in Figure 2.2. This histogram can be used by Alice to obtain an idea of how much she might like this movie. Nevertheless, the power of this type of explanation is somewhat limited because it does not give Alice an idea of how these movies relate to her *own* tastes or to those of friends she actually knows and trusts. Note that the identity of her neighbors is usually not available to Alice because of privacy concerns.

Finally, item-based methods are more stable with changes to the ratings. This is because of two reasons. First, the number of users is generally much larger than the number of items. In such cases, two users may have a very small number of mutually rated items, but two items are more likely to have a larger number of users who have co-rated them. In the case of user-based methods, the addition of a few ratings can change the similarity values drastically. This is not the case for item-based methods, which are more stable to changes in the values of the ratings. Second, new users are likely to be added more frequently in

[2]The precise method used by Netflix is proprietary and therefore not known. However, item-based methods do provide a viable methodology to achieve similar goals.

commercial systems than new items. In such cases, the computation of neighborhood items can be done only occasionally because item neighborhoods are unlikely to change drastically with the addition of new users. On the other hand, the computation of user neighborhoods needs to be performed more frequently with the addition of new users. In this context, incremental maintenance of the recommendation model is more challenging in the case of user-based methods.

2.3.5 Strengths and Weaknesses of Neighborhood-Based Methods

Neighborhood methods have several advantages related to their simplicity and intuitive approach. Because of the simple and intuitive approach of these methods, they are easy to implement and debug. It is often easy to justify why a specific item is recommended, and the interpretability of item-based methods is particularly notable. Such justifications are often not easily available in many of the model-based methods discussed in later chapters. Furthermore, the recommendations are relatively stable with the addition of new items and users. It is also possible to create incremental approximations of these methods.

The main disadvantage of these methods is that the offline phase can sometimes be impractical in large-scale settings. The offline phase of the user-based method requires at least $O(m^2)$ time and space. This might sometimes be too slow or space-intensive with desktop hardware, when m is of the order of tens of millions. Nevertheless, the online phase of neighborhood methods is always efficient. The other main disadvantage of these methods is their limited coverage because of sparsity. For example, if none of John's nearest neighbors have rated *Terminator*, it is not possible to provide a rating prediction of *Terminator* for John. On the other hand, we care only about the top-k items of John in most recommendation settings. If none of John's nearest neighbors have rated *Terminator*, then it might be evidence that this movie is not a good recommendation for John. Sparsity also creates challenges for robust similarity computation when the number of mutually rated items between two users is small.

2.3.6 A Unified View of User-Based and Item-Based Methods

The respective weaknesses of user-based and item-based methods arise out of the fact that the former ignores the similarity between the columns of the ratings matrix, whereas the latter ignores the similarity between the rows while determining the most similar entries. A natural question arises whether we can determine the most similar *entries* to a target entry by unifying the two methods. By doing so, one does not need to ignore the similarity along either rows or columns. Rather, one can *combine* the similarity information between rows and columns.

In order to achieve this goal, it is crucial to understand that the user-based and item-based methods are almost identical (with some minor differences), once the rows have been mean-centered. We can assume without loss of generality that the rows of the ratings matrix are mean-centered because the mean of each row can be added back to each entry after the prediction. It is also noteworthy that if the rows are mean-centered then the Pearson correlation coefficient between rows is identical[3] to the cosine coefficient. Based on this

[3]There can be some minor differences depending on how the mean is computed for each row within the Pearson coefficient. If the mean for each row is computed using all the observed entries of that row (rather than only the mutually specified entries), then the Pearson correlation coefficient is identical to the cosine coefficient for row-wise mean-centered matrices.

assumption, the user-based and item-based methods can be described in a unified way to predict the entry r_{uj} in the ratings matrix R:

1. For a target entry (u, j) determine the most similar rows/columns of the ratings matrix with the use of the cosine coefficient between rows/columns. For user-based methods rows are used, whereas for item-based methods, columns are used.

2. Predict the target entry (u, j) using a weighted combination of the ratings in the most similar rows/columns determined in the first step.

Note that the aforementioned description ignores *either* the rows or the columns in each step. One can, of course, propose a generalized description of the aforementioned steps in which the similarity and prediction information along rows and columns are *combined*:

1. For a target entry (u, j) determine the most similar *entries* of the ratings matrix with the use of a combination function of the similarity between rows and columns. For example, one can use the sum of the cosine similarity between rows and between columns to determine the most similar entries in the ratings matrix to (u, j).

2. Predict the target entry (u, j) using a weighted combination of the ratings in the most similar *entries* determined in the first step. The weights are based on the similarities computed in the first step.

We have highlighted the steps, which are different in the generalized method. This approach fuses the similarities along rows and columns with the use of a combination function. One can experiment with the use of various combination functions to obtain the most effective results. Detailed descriptions of such unified methods may be found in [613, 622]. This basic principle is also used in the multidimensional model of context-sensitive recommender systems, in which the similarities along users, items, and other contextual dimensions are unified into a single framework (cf. section 8.5.1 of Chapter 8).

2.4 Clustering and Neighborhood-Based Methods

The main problem with neighborhood-based methods is the complexity of the offline phase, which can be quite significant when the number of users or the number of items is very large. For example, when the number of users m is of the order of a few hundred million, the $O(m^2 \cdot n')$ running time of a user-based method will become impractical even for occasional offline computation. Consider the case where $m = 10^8$ and $n' = 100$. In such a case, $O(m^2 \cdot n') = O(10^{18})$ operations will be required. If we make the conservative assumption that each operation requires an elementary machine cycle, a 10GHz computer will require 10^8 seconds, which is approximately 115.74 days. Clearly, such an approach will not be very practical from a scalability point of view.

The main idea of clustering-based methods is to replace the offline nearest-neighbor computation phase with an offline clustering phase. Just as the offline nearest-neighbor phase creates a large number of peer groups, which are centered *at each possible target*, the clustering process creates a smaller number of peer groups which are not necessarily centered at each possible target. The process of clustering is much more efficient than the $O(m^2 \cdot n')$ time required for construction of the peer groups of every possible target. Once the clusters have been constructed, the process of predicting ratings is similar to the approach used in Equation 2.4. The main difference is that the top-k closest peers within the same cluster are used to perform the prediction. It is noteworthy that the pairwise similarity computation needs to be performed only within the same cluster and therefore, the approach can be significantly more efficient. This efficiency does result in some loss of accuracy because the set of closest neighbors to each target within a cluster is of lower quality than that over the entire data. Furthermore, the clustering granularity regulates the trade-off between accuracy and efficiency. When the clusters are fine-grained, the efficiency improves, but the accuracy is reduced. In many cases, very large gains in efficiency can be obtained for small reductions in accuracy. When the ratings matrices are very large, this approach provides a very practical alternative at a small cost.

One challenge with the use of this approach is the fact that the ratings matrix is incomplete. Therefore, clustering methods need to be adapted to work with massively incomplete data sets. In this context, k-means methods can be easily adapted to incomplete data. The basic idea of a k-means approach is to work with k central points (or "means"), which serve as the representatives of k different clusters. In k-means methods, the solution to a clustering can be fully represented by the specification of these k representatives. Given a set of k representatives $\overline{Y_1} \ldots \overline{Y_k}$, each data point is assigned to its closest representative with the use of a similarity or distance function. Therefore, the data partitioning can be uniquely defined by the set of representatives. For an $m \times n$ data set, each representative $\overline{Y_i}$ is an n-dimensional data point, which is a central point of the ith cluster. Ideally, we would like the central representative to be the mean of the cluster.

Therefore, the clusters are dependent on the representatives and vice versa. Such an interdependency is achieved with an iterative approach. We start with a set of representatives $\overline{Y_1} \ldots \overline{Y_k}$, which might be randomly chosen points generated in the range of the data space. We iteratively compute the cluster partitions using the representatives, and then recompute the representatives as the centroids of the resulting clusters. While computing the centroids, care must be taken to use only the observed values in each dimension. This two-step iterative approach is executed to convergence. The two-step approach is summarized as follows:

1. Determine the clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ by assigning each row in the $m \times n$ matrix to its closest representative from $\overline{Y_1} \ldots \overline{Y_k}$. Typically, the Euclidean distance or the Manhattan distance is used for similarity computation.

2. For each $i \in \{1 \ldots k\}$, reset $\overline{Y_i}$ to the centroid of the current set of points in $\mathcal{C}_i$.

The main problem with the use of this approach is that the $m \times n$ ratings matrix is incomplete. Therefore, the computation of the mean and the distance values becomes undefined. However, it is relatively easy to compute the means using only the observed values within a cluster. In some cases, the centroid itself might not be fully specified, when no rating is specified for one or more items in the cluster. The distance values are computed using only the subset of dimensions, which are specified both for the data point and cluster representative. The distance is also divided by the number of dimensions used in the computation. This is done in order to adjust for the fact that different numbers of dimensions are used for computing the distance of a data point to various centroids, when all the centroids are not fully specified. In this context, the Manhattan distance yields better adjustments than the Euclidean distance, and the normalized value can be interpreted more easily as an average distance along each observed value.

The aforementioned approach clusters the rows for user-based collaborative filtering. In item-based methods, it would be necessary to cluster the columns. The approach is exactly similar except that it is applied to the columns rather than the rows. A number of clustering methods for efficient collaborative filtering are discussed in [146, 167, 528, 643, 644, 647]. Some of these methods are user-based methods, whereas others are item-based methods. A number of co-clustering methods [643] can be used to cluster rows and columns simultaneously.

2.5 Dimensionality Reduction and Neighborhood Methods

Dimensionality reduction methods can be used to improve neighborhood-based methods both in terms of quality and in terms of efficiency. In particular, even though pairwise similarities are hard to robustly compute in sparse rating matrices, dimensionality reduction provides a dense low-dimensional representation in terms of latent factors. Therefore, such models are also referred to as *latent factor models*. Even when two users have very few items rated in common, a distance can be computed between their low-dimensional latent vectors. Furthermore, it is more efficient to determine the peer groups with low-dimensional latent vectors. Before discussing the details of dimensionality reduction methods, we make some comments about two distinct ways in which latent factor models are used in recommender systems:

1. A reduced representation of the data can be created in terms of *either* row-wise latent factors or in terms of column-wise latent factors. In other words, the reduced representation will either compress the item dimensionality or the user dimensionality into latent factors. This reduced representation can be used to alleviate the sparsity problem for neighborhood-based models. Depending on which dimension has been compressed into latent factors, the reduced representation can be used for either user-based neighborhood algorithms or item-based neighborhood algorithms.

2. The latent representations of *both* the row space and the column space are determined simultaneously. These latent representations are used to reconstruct the entire ratings matrix in one shot without the use of neighborhood-based methods.

Because the second class of methods is not directly related to neighborhood-based methods, it will not be discussed in this chapter. A detailed discussion of the second class of methods will be provided in Chapter 3. In this chapter, we will focus only on the first class of methods.

For ease of discussion, we will first describe only the user-based collaborative filtering method. In user-based collaborative filtering methods, the basic idea is to transform the $m \times n$ ratings matrix R into a lower-dimensional space by using principal component analysis. The resulting matrix R' is of size $m \times d$, where $d \ll n$. Thus, each of the (sparse) n-dimensional vector of ratings corresponding to a user is transformed into a reduced d-dimensional space. Furthermore, unlike the original rating vector, each of the d dimensions is fully specified. After this d-dimensional representation of each user is determined, the similarity is computed from the target user to each user using the reduced representation. The similarity computations in the reduced representation are more robust because the new low-dimensional vector is fully specified. Furthermore, the similarity computations are more efficient because of the low dimensionality of the latent representation. A simple cosine or dot product on the reduced vectors is sufficient to compute the similarity in this reduced space.

It remains to be described how the low-dimensional representation of each data point is computed. The low-dimensional representation can be computed using either SVD-like methods or PCA-like methods. In the following, we describe an SVD-like method.

Table 2.3: Example of bias in estimating covariances

User Index	*Godfather*	*Gladiator*	*Nero*
1	1	1	1
2	7	7	7
3	3	1	1
4	5	7	7
5	3	1	?
6	5	7	?
7	3	1	?
8	5	7	?
9	3	1	?
10	5	7	?
11	3	1	?
12	5	7	?

The first step is to augment the $m \times n$ incomplete ratings matrix R to fill in the missing entries. The missing entry is estimated to be equal to the mean of the corresponding row in the matrix (i.e., the mean rating of the corresponding user). An alternative approach is to estimate the missing entry as the mean of the corresponding column in the matrix (i.e., the mean rating of the corresponding item). Let the resulting matrix be denoted by R_f. Then, we compute the $n \times n$ similarity matrix between pairs of items, which is given by $S = R_f^T R_f$. This matrix is positive semi-definite. In order to determine the dominant basis vectors of R_f for SVD, we perform the diagonalization of the similarity matrix S as follows:

$$S = P \Delta P^T \tag{2.16}$$

Here, P is an $n \times n$ matrix, whose columns contain the orthonormal eigenvectors of S. Δ is a diagonal matrix containing the non-negative eigenvalues of S along its diagonal. Let P_d be the $n \times d$ matrix containing only the columns of P corresponding to the largest d eigenvectors. Then, the low-dimensional representation of R_f is given by the matrix product $R_f P_d$. Note that the dimensions of the reduced representation $R_f P_d$ are $m \times d$, because R_f is an $m \times n$ matrix and P_d is an $n \times d$ matrix. Therefore, each of the m users is now represented in a d-dimensional space. This representation is then used to determine the peer group of each user. Once the peers have been determined, the rating prediction can be easily performed with Equation 2.4. Such an approach can also be used for item-based collaborative filtering by applying the entire dimensionality reduction method to the transpose of R_f instead of R_f.

The aforementioned methodology can be viewed as a *singular value decomposition (SVD)* of the ratings matrix R_f. A number of other methods [24, 472] use *principal component analysis (PCA)* instead of SVD, but the overall result is very similar. In the PCA method, the covariance matrix of R_f is used instead of the similarity matrix $R_f^T R_f$. For data, which is mean-centered along columns, the two methods are identical. Therefore, one can subtract the mean of each column from its entries, and then apply the aforementioned approach to obtain a transformed representation of the data. This transformed representation is used to determine the peers of each user. Mean-centering has benefits in terms of reducing *bias* (see next section). An alternative approach is to first mean center along each row and then mean-center along each column. SVD can be applied to the transformed representation. This type of approach generally provides the most robust results.

2.5.1 Handling Problems with Bias

It is noteworthy that the matrix R_f is derived from the incomplete matrix R by filling in the unspecified entries with average values along either the rows or the columns. Such an approach is likely to cause considerable *bias*. To understand the nature of this bias, consider the example in Table 2.3 of ratings given by 12 users to the three movies *Godfather*, *Gladiator*, and *Nero*. Let us assume that PCA is used for dimensionality reduction, and therefore the covariance matrix needs to be estimated. Let us assume that missing values are replaced with the averages along the columns.

In this case, the ratings are drawn on a scale from 1 to 7 by a set of 4 users for 3 movies. It is visually evident that the correlations between the ratings of the movies *Gladiator* and *Nero* are extremely high because the ratings are very similar in the four cases in which they are specified. The correlation between *Godfather* and *Gladiator* seems to be less significant. However, many users have not specified their ratings for *Nero*. Because the mean rating of *Nero* is $(1 + 7 + 1 + 7)/4 = 4$, these unspecified ratings are replaced with the mean value of 4. The addition of these new entries significantly reduces the estimated covariance between *Gladiator* and *Nero*. However, the addition of the new entries has no impact on the covariance between *Godfather* and *Gladiator*. After filling in the missing ratings, the pairwise covariances between the three movies can be estimated as follows:

	Godfather	*Gladiator*	*Nero*
Godfather	2.55	4.36	2.18
Gladiator	4.36	9.82	3.27
Nero	2.18	3.27	3.27

According to the aforementioned estimation, the covariance between *Godfather* and *Gladiator* is larger than that between *Gladiator* and *Nero*. This does not seem to be correct because the ratings in Table 2.3 for *Gladiator* and *Nero* are identical for the case where both are specified. Therefore, the correlation between *Gladiator* and *Nero* ought to be higher. This error is a result of the bias caused by filling in the unspecified entries with the mean of that column. This kind of bias can be very significant in sparse matrices because most of the entries are unspecified. Therefore, methods need to be designed to reduce the bias caused by using the mean ratings in place of the unspecified entries. In the following, we explore two possible solutions to this problem.

2.5.1.1 Maximum Likelihood Estimation

The conceptual reconstruction method [24, 472] proposes the use of probabilistic techniques, such as the EM-algorithm, in order to estimate the covariance matrix. A generative model is assumed for the data and the specified entries are viewed as the outcomes of the generative model. The covariance matrix can be estimated as part of the process of estimating the parameters of this generative model. In the following, we provide a simplification of this approach. In this simplified approach, the maximum likelihood estimate of the covariance matrix is computed. The maximum likelihood estimate of the covariance between each pair of items is estimated as the covariance between only the specified entries. In other words, only the users that have specified ratings for a particular pair of items are used to estimate the covariance. In the event that there are no users in common between a pair of items, the covariance is estimated to be 0. By using this approach, the following covariance matrix is estimated for the data in Table 2.3.

	Godfather	*Gladiator*	*Nero*
Godfather	2.55	4.36	8
Gladiator	4.36	9.82	12
Nero	8	12	12

In this case, it becomes immediately evident that the covariance between *Gladiator* and *Nero* is almost three times that between *Godfather* and *Gladiator*. Furthermore, the movie *Nero* has more than three times as much variance than was originally estimated and has the largest variance in ratings among all movies. While the pairwise covariance between *Godfather* and *Gladiator* was the largest compared to all other pairwise covariances using the mean-filling technique, this same pair now shows the least of all pairwise covariances. This example suggests that the bias corrections can be very significant in some situations. The greater the proportion of unspecified entries in the matrix, the greater the bias of the mean-filling technique. Therefore, the modified technique of leveraging only the specified entries is used for computing the covariance matrix. While such a technique is not always effective, it is superior to the mean-filling technique. The reduced $n \times d$ basis matrix P_d is computed by selecting the top-d eigenvectors of the resulting covariance matrix.

In order to further reduce the bias in representation, the incomplete matrix R can be directly projected on the reduced matrix P_d, rather than projecting the filled matrix R_f on P_d. The idea is to compute the contribution of each observed rating to the projection on each latent vector of P_d, and then average the contribution over the number of such ratings. This averaged contribution is computed as follows. Let $\overline{e_i}$ be the ith column (eigenvector) of P_d, for which the jth entry is e_{ji}. Let r_{uj} be the observed rating of user u for item j in matrix R. Then, the contribution of user u to the projection on latent vector $\overline{e_i}$ is given by $r_{uj}e_{ji}$. Then, if the set I_u represents the indices of the specified item ratings of user u, the averaged contribution a_{ui} of user u on the ith latent vector is as follows:

$$a_{ui} = \frac{\sum_{j \in I_u} r_{uj} e_{ji}}{|I_u|} \tag{2.17}$$

This type of averaged normalization is particularly useful in cases where the different users have specified different numbers of ratings. The resulting $m \times d$ matrix $A = [a_{ui}]_{m \times d}$ is used as the reduced representation of the underlying ratings matrix. This reduced matrix is used to compute the neighborhood of the target user efficiently for user-based collaborative filtering. It is also possible to apply the approach to the transpose of the matrix R and reduce the dimensionality along the user dimension, rather than the item dimension. Such an approach is useful for computing the neighborhood of a target item in item-based collaborative filtering. This approach of using the reduced representation for missing value imputation is discussed in [24, 472].

2.5.1.2 Direct Matrix Factorization of Incomplete Data

Although the aforementioned methodology can correct for the bias in covariance estimation to some extent, it is not completely effective when the sparsity level of the ratings is high. This is because the covariance matrix estimation requires a sufficient number of observed ratings for each pair of items for robust estimation. When the matrix is sparse, the covariance estimates will be statistically unreliable.

A more direct approach is to use matrix factorization methods. Methods such as singular value decomposition are essentially matrix factorization methods. For a moment, assume that the $m \times n$ ratings matrix R is fully specified. It is a well-known fact of linear algebra [568] that any (fully specified) matrix R can be factorized as follows:

$$R = Q\Sigma P^T \tag{2.18}$$

Here, Q is an $m \times m$ matrix with columns containing the m orthonormal eigenvectors of RR^T. The matrix P is an $n \times n$ matrix with columns containing the n orthonormal eigenvectors of R^TR. Σ is an $m \times n$ diagonal matrix in which only diagonal entries[4] are nonzero and they contain the square-root of the nonzero eigenvalues of R^TR (or equivalently, RR^T). It is noteworthy that the eigenvectors of R^TR and RR^T are not the same and will have different dimensionality when $m \neq n$. However, they will always have the same number of (nonzero) eigenvalues, which are identical in value. The values on the diagonal of Σ are also referred to as *singular values.*

Furthermore, one can *approximately* factorize the matrix by using *truncated* SVD, where only the eigenvectors corresponding to the $d \leq \min\{m, n\}$ largest singular values are used. Truncated SVD is computed as follows:

$$R \approx Q_d\Sigma_d P_d^T \tag{2.19}$$

Here, Q_d, Σ_d, and P_d are $m \times d$, $d \times d$, and $n \times d$ matrices, respectively. The matrices Q_d and P_d, respectively, contain the d largest eigenvectors of RR^T and R^TR, whereas the matrix Σ_d contains the square-roots of the d largest eigenvalues of either matrix along its diagonal. It is noteworthy that the matrix P_d contains the top eigenvectors of R^TR, which is the *reduced* basis representation required for dimensionality reduction. Furthermore, the matrix $Q_d\Sigma_d$ contains the transformed and reduced $m \times d$ representation of the original ratings matrix in the basis corresponding to P_d. It can be shown that such an approximate factorization has the least mean-squared error of the approximated entries as compared to any other rank-d factorization. Therefore, if we can approximately factorize the ratings matrix R in the form corresponding to Equation 2.19, it provides us with the reduced basis as well as the representation of the ratings in the reduced basis. The main problem of using such an approach is that the ratings matrix is not fully specified. As a result, this factorization is undefined. Nevertheless, it is possible to recast the formulation as an optimization problem, in which the squared error of factorization is optimized *only over the observed entries* of the ratings matrix. It is also possible to explicitly solve this modified formulation using nonlinear optimization techniques. This results in a robust and unbiased lower dimensional representation. Furthermore, such an approach can be used to directly estimate the ratings matrix by using Equation 2.19, once the reduced factor matrices have been determined. In other words, such methods have a direct utility beyond neighborhood-based methods. More details of these latent factor models and nonlinear optimization techniques will be discussed in section 3.6 of Chapter 3. The reader should consult this section to learn how the reduced representation may be computed by using modified optimization formulations.

2.6 A Regression Modeling View of Neighborhood Methods

An important observation about both user-based and item-based methods is that they predict ratings as *linear functions* of either the ratings of the same item by neighboring users, or of the same user on neighboring items. In order to understand this point, we

[4]Diagonal matrices are usually square. Although this matrix is not square, only entries with equal indices are nonzero. This is a generalized definition of a diagonal matrix.

replicate the prediction function of user-based neighborhood methods (cf. Equation 2.4) below:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.20}$$

Note that the predicted rating is a *weighted* linear combination of other ratings of the same item. The linear combination has been restricted only to the ratings of item j belonging to users with sufficiently similar tastes to target user u. This restriction is enabled with the use of the peer rating set $P_u(j)$. Recall from the discussion earlier in this chapter that $P_u(j)$ is the set of k nearest users to target user u, who have also rated item j. Note that if we allowed the set $P_u(j)$ to contain all ratings of item j (and not just specific peer users), then the prediction function becomes similar[5] to that of linear regression [22]. In linear regression, the ratings are also predicted as weighted combinations of other ratings, and the weights (coefficients) are determined with the use of an optimization model. In the neighborhood-based approach, the coefficients of the linear function are chosen in a heuristic way with the user-user similarities, rather than with the use of an optimization model.

A similar observation applies to the case of item-based neighborhood methods, where the prediction function (cf. Equation 2.15) is as follows:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j, t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j, t)|} \tag{2.21}$$

The set $Q_t(u)$ represents the set of the k closest items to target item t that have also been rated by user u. In this case, the rating of a user u for a target item t is expressed as a linear combination of her *own* ratings. As in the case of user-based methods, the coefficients of the linear combination are heuristically defined with similarity values. Therefore, a user-based model expresses a predicted rating as a linear combination of ratings in the same *column*, whereas an item-based model expresses a predicted rating as a linear combination of ratings in the same *row*. From this point of view, *neighborhood-based models are heuristic variants of linear regression models*, in which the regression coefficients are heuristically set to similarity values for related (neighboring) items/users and to 0 for unrelated items/users.

It is noteworthy that the use of similarity values as combination weights is rather heuristic and arbitrary. Furthermore, the coefficients do not account for interdependencies among items. For example, if a user has rated certain sets of correlated items in a very similar way, then the coefficients associated with these items will be interdependent as well. The use of similarities as heuristic weights does not account for such interdependencies.

A question arises as to whether one can do better by *learning* the weights with the use of an optimization formulation. It turns out that one can derive analogous regression-based models to the user-based and item-based models. Several different optimization formulations have been proposed in the literature, which can leverage user-based models, item-based models, or a combination of the two. These models can be viewed as theoretical generalizations of the heuristic nearest neighbor model. The advantage of such models is that they are mathematically better founded in the context of a crisp optimization formulation, and the weights for combining the ratings can be better justified because of their optimality from a *modeling* perspective. In the following, we discuss an optimization-based neighborhood model, which is a simplification of the work in [309]. This also sets the stage for combining the power of this model with other optimization models, such as matrix factorization, in section 3.7 of Chapter 3.

[5]A discussion of linear regression is provided in section 4.4.5 of Chapter 4, but in the context of content-based systems.

2.6.1 User-Based Nearest Neighbor Regression

Consider the user-based prediction of Equation 2.20. One can replace the (normalized) similarity coefficient with the unknown parameter w_{vu}^{user} to *model* the predicted rating $\hat{r}_{uj}$ of target user u for item j as follows:

$$\hat{r}_{uj} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.22}$$

As in the case of neighborhood models, one can use the Pearson correlation coefficient to define $P_u(j)$. There is, however, a subtle but important difference in terms of how $P_u(j)$ is defined in this case. In neighborhood-based models, $P_u(j)$ is the set of k closest users to target user u, who have specified ratings for item j. Therefore, the size of $P_u(j)$ is often exactly k, when at least k users have rated item j. In the case of regression methods, the set $P_u(j)$ is defined by first determining the k closest peers for each user, and then retaining only those for which ratings are observed. Therefore, the size of set $P_u(j)$ is often *significantly* less than k. Note that the parameter k needs to be set to much larger values in the regression framework as compared to that in neighborhood models because of its different interpretation.

Intuitively, the unknown coefficient w_{vu}^{user} controls the portion of the prediction of ratings given by user u, which comes from her similarity to user v, because this portion is given by $w_{vu}^{user} \cdot (r_{vj} - \mu_v)$. It is possible for w_{vu}^{user} to be different from w_{uv}^{user}. It is also noteworthy that w_{vu}^{user} is only defined for the k different values of v (user indices) that are closest to user u on the basis of the Pearson coefficient. The other values of w_{vu}^{user} are not needed by the prediction function of Equation 2.22, and they therefore do not need to be learned. This has the beneficial effect of reducing the number of regression coefficients.

One can use the aggregate squared difference between the predicted ratings $\hat{r}_{uj}$ (according to Equation 2.22) and the observed ratings r_{uj} to create an objective function that estimates the quality of a particular set of coefficients. Therefore, one can use the observed ratings in the matrix to set up a least-squares optimization problem over the unknown values of w_{vu}^{user} in order to minimize the overall error. The idea is to predict each (observed) rating of user u with her nearest k users in a formal regression model, and then measure the error of the prediction. The squared errors can be added over all items rated by user u to create a least-squares formulation. Therefore, the optimization problem is set up for each target user u. Let I_u be the set of items that have been rated by the target user u. The least-squares objective function for the uth user can be stated as the sum of the squares of the errors in predicting each item in I_u with the k nearest neighbors of the user in a formal regression model:

$$\begin{aligned}\text{Minimize } J_u &= \sum_{j \in I_u} (r_{uj} - \hat{r}_{uj})^2 \\ &= \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2\end{aligned}$$

The second relationship is obtained by substituting the expression in Equation 2.22 for $\hat{r}_{uj}$. Note that this optimization problem is formulated separately for each target user u. However, one can add up the objective function values J_u over different target users $u \in \{1 \dots m\}$ with no difference to the optimal solution. This is because the various values of J_u are expressed in terms of mutually disjoint sets of optimization variables w_{vu}^{user}. Therefore, the consolidated optimization problem is expressed as follows:

$$\text{Minimize } \sum_{u=1}^{m} J_u = \sum_{u=1}^{m} \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2 \tag{2.23}$$

One can solve each of the smaller optimization problems (i.e., objective function J_u) in their decomposed form more efficiently without affecting the overall solution. However, the consolidated formulation has the advantage that it can be combined with other optimization models such as matrix factorization methods (cf. section 3.7 of Chapter 3) in which such a decomposition is not possible. Nevertheless, if linear regression is to be used on a standalone basis, it makes sense to solve these problems in their decomposed form.

Both the consolidated and decomposed versions of the optimization models are least-squares optimization problems. These methods can be solved with the use of any off-the-shelf optimization solver. Refer to section 4.4.5 of Chapter 4 for a discussion of closed form solutions to linear regression problems. A desirable property of most of these solvers is that they usually have *regularization* built in them, and they can therefore avoid overfitting to some extent. The basic idea in regularization is to reduce model complexity by adding the term $\lambda \sum_{j \in I_u} \sum_{v \in P_u(j)} (w_{vu}^{user})^2$ to each (decomposed) objective function J_u, where $\lambda > 0$ is a user-defined parameter regulating the weight of the regularization term. The term $\lambda \sum_{j \in I_u} \sum_{v \in P_u(j)} (w_{vu}^{user})^2$ penalizes large coefficients, and it therefore shrinks the absolute values of the coefficients. Smaller coefficients result in simpler models and reduce overfitting. However, as discussed below, it is sometimes not sufficient to use regularization alone to reduce overfitting.

2.6.1.1 Sparsity and Bias Issues

One problem with this regression approach is that the size of the $P_u(j)$ can be vastly different for the same user u and varying item indices (denoted by j). This is because of the extraordinary level of sparsity inherent in ratings matrices. As a result, the regression coefficients become heavily dependent on the *number* of peer users that have rated a particular item j along with user u. For example, consider a scenario where the target user u has rated both *Gladiator* and *Nero*. Out of the k nearest neighbors of the target u, only one user might rate the movie *Gladiator*, whereas all k might have rated *Nero*. As a result, the regression coefficient w_{vu}^{user} of the peer user v who rated *Gladiator* will be heavily influenced by the fact that she is the only user who has rated *Gladiator*. This will result in overfitting because this (statistically unreliable) regression coefficient might add noise to the rating predictions of other movies.

The basic idea is to change the prediction function and assume that the regression for item j predicts only a fraction $\frac{|P_u(j)|}{k}$ of the rating of target user u for item j. The implicit assumption is that the regression coefficients are based on *all* the peers of the target user, and one must interpolate incomplete information as a fraction. Therefore, this approach changes the interpretation of the regression coefficients. In this case, the prediction function of Equation 2.22 is modified as follows:

$$\hat{r}_{uj} \cdot \frac{|P_u(j)|}{k} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.24}$$

A number of other heuristic adjustments are sometimes used. For example, along the lines of the ideas in [312], one can use a heuristic adjustment factor of $\sqrt{|P_u(j)|/k}$. This factor can often be simplified to $\sqrt{|P_u(j)|}$ because constant factors are absorbed by the optimization variables. A related enhancement is that the *constant* offset μ_v is replaced with a bias *variable* b_u, which is learned in the optimization process. The corresponding prediction model, including heuristic adjustment factors, is as follows:

$$\hat{r}_{uj} = b_u^{user} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - b_v^{user})}{\sqrt{|P_u(j)|}} \tag{2.25}$$

Note that this model is no longer linear because of the multiplicative term $w_{vu}^{user} \cdot b_v^{user}$ between two optimization variables. Nevertheless, it is relatively easy to use the same least-squares formulation, as in the previous case. In addition to user biases, one can also incorporate *item* biases. In such a case, the model becomes the following:

$$\hat{r}_{uj} = b_u^{user} + b_j^{item} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - b_v^{user} - b_j^{item})}{\sqrt{|P_u(j)|}} \tag{2.26}$$

Furthermore, it is recommended to center the entire ratings matrix around its global mean by subtracting the mean of all the observed entries from it. The global mean needs to be added back to the predictions. The main problem with this model is computational. One must pre-compute and store all user-user relations, which is computationally expensive and requires $O(m^2)$ space over m users. This problem is similar to that encountered in traditional neighborhood-based models. Such models are suitable in settings in which the item space changes rapidly, but the users are relatively stable over time [312]. An example is the case of news recommender systems.

2.6.2 Item-Based Nearest Neighbor Regression

The item-based approach is similar to the user-based approach, except that the regression learns and leverages item-item correlations rather than user-user correlations. Consider the item-based prediction of Equation 2.21. One can replace the (normalized) similarity coefficient AdjustedCosine(j, t) with the unknown parameter w_{jt}^{item} to model the rating prediction of user u for target item t:

$$\hat{r}_{ut} = \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj} \tag{2.27}$$

The nearest items in $Q_t(u)$ can be determined using the adjusted cosine, as in item-based neighborhood methods. The set $Q_t(u)$ represents the subset of the k nearest neighbors of the target item t, for which user u has provided ratings. This way of defining $Q_t(u)$ is subtly different from that of traditional neighborhood-based methods, because the size of set $Q_t(u)$ might be significantly less than k. In traditional neighborhood methods, one determines the closest k items to target item t, for which the user u has specified ratings, and therefore the size of the neighborhood set is often exactly k. This change is required to be able to effectively implement the regression-based method.

Intuitively, the unknown coefficient w_{jt}^{item} controls the portion of the rating of item t, which comes from its similarity to item j, because this portion is given by $w_{jt}^{item} \cdot r_{uj}$. The prediction error of Equation 2.27 should be minimized to ensure the most robust predictive model. One can use the known ratings in the matrix to set up a least-squares optimization problem over the unknown values of w_{jt}^{item} in order to minimize the overall error. The idea is to predict each (observed) rating of target item t with its nearest k items and then, create an expression for the least-squares error. The optimization problem is set up for each target item t. Let U_t be the set of users who have rated the target item t. The least-squares objective function for the tth item can be stated as the sum of the squares of the errors in predicting each specified rating in U_t:

$$\begin{aligned}\text{Minimize } J_t &= \sum_{u \in U_t} (r_{ut} - \hat{r}_{ut})^2 \\ &= \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2\end{aligned}$$

Note that this optimization problem is formulated separately for each target item t. However, one can add up the terms over various values of the target item t with no difference to the optimization solution, because the unknown coefficients w_{jt}^{item} in the various objective functions are non-overlapping over different values of the target item $t \in \{1 \ldots n\}$. Therefore, we have the following consolidated formulation:

$$\text{Minimize } \sum_{t=1}^{n} \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2 \tag{2.28}$$

This is a least-squares regression problem and it can be solved with the use of any off-the-shelf solver. Furthermore, one can also solve each of the smaller optimization problems (i.e., objective function J_t) in its decomposed form more efficiently without affecting the overall solution. However, the consolidated formulation has the advantage that it can be combined with other optimization models, such as matrix factorization methods (cf. section 3.7 of Chapter 3). As in the case of user-based methods, significant challenges are associated with the problem of overfitting. One can add the regularization term $\lambda \sum_{u \in U_t} \sum_{j \in Q_t(u)} (w_{jt}^{item})^2$ to the objective function J_t.

As discussed in section 2.6.1.1 for the case of the user-based model, one can incorporate adjustment factors and bias variables to improve performance. For example, the user-based prediction model of Equation 2.26 takes on the following form in the item-wise model:

$$\hat{r}_{ut} = b_u^{user} + b_t^{item} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - b_u^{user} - b_j^{item})}{\sqrt{|Q_t(u)|}} \tag{2.29}$$

Furthermore, it is assumed that the ratings are centered around the global mean of the entire ratings matrix. Therefore, the global mean is subtracted from each of the ratings before building the model. All predictions are performed on the centered ratings, and then the global mean is added back to each prediction. In some variations of the model, the bias terms $b_u^{user} + b_j^{item}$ within brackets are replaced with a consolidated *constant* term B_{uj}. This constant term is derived using a non-personalized approach described in section 3.7.1 of Chapter 3. The resulting prediction model is as follows:

$$\hat{r}_{ut} = b_u^{user} + b_t^{item} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - B_{uj})}{\sqrt{|Q_t(u)|}} \tag{2.30}$$

A least-squares optimization model is formulated, and a gradient descent approach is used to solve for the optimization parameters. This is precisely the model used in [309]. The resulting gradient-descent steps are discussed in section 3.7.2 of Chapter 3. The user-user model is known to perform slightly better than the item-item model [312]. However, the item-based model is far more computationally and space-efficient in settings where the number of items is much smaller than the number of users.

2.6.3 Combining User-Based and Item-Based Methods

It is natural to combine the user and item-based models in a unified regression framework [312]. Therefore, a rating is predicted based on its relationship with similar users as well as similar items. This is achieved by combining the ideas in Equations 2.26 and 2.30 as follows:

$$\hat{r}_{uj} = b_u^{user} + b_j^{item} + \frac{\sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - B_{vj})}{\sqrt{|P_u(j)|}} + \frac{\sum_{j \in Q_t(u)} w_{jt}^{item} \cdot (r_{uj} - B_{uj})}{\sqrt{|Q_t(u)|}} \quad (2.31)$$

As in previous cases, it is assumed that the ratings matrix is centered around its global mean. A similar least-squares optimization formulation can be used in which the squared error over all the observed entries is minimized. In this case, it is no longer possible to decompose the optimization problem into independent subproblems. Therefore, a single least-squares optimization model is constructed over all the observed entries in the ratings matrix. As in the previous cases, the gradient-descent approach can be used. It was reported in [312] that the fusion of the user-based and item-based models generally performs better than the individual models.

2.6.4 Joint Interpolation with Similarity Weighting

The method in [72] uses a different idea to set up the joint neighborhood-based model. The basic idea is to predict each rating of target user u with the user-based model of Equation 2.22. Then, instead of comparing it with the observed value of the *same* item, we compare it with the observed ratings of *other* items of that user.

Let S be the set of all pairs of user-item combinations in the ratings matrix, which have been observed:

$$S = \{(u,t) : r_{ut} \text{ is observed}\} \quad (2.32)$$

We set up an objective function which is penalized when the predicted rating $\hat{r}_{uj}$ of an item j is far away from the observed rating given to a similar item s by the same target user u. In other words, the objective function for target user u is defined as follows:

$$\begin{aligned}\text{Minimize } & \sum_{s:(u,s) \in S} \sum_{j: j \neq s} AdjustedCosine(j,s) \cdot (r_{us} - \hat{r}_{uj})^2 \\ &= \sum_{s:(u,s) \in S} \sum_{j: j \neq s} AdjustedCosine(j,s) \cdot \left(r_{us} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2\end{aligned}$$

Regularization can be added to the objective function to reduce overfitting. Here, $P_u(j)$ is defined as the k closest users to target user u, who have also rated item j. Therefore, the conventional definition of $P_u(j)$ as used in neighborhood-based models is leveraged in this case.

By using the adjusted cosine as a multiplicative factor of each individual term in the objective function, the approach forces the target user's ratings of similar items to be more similar as well. It is noteworthy that both user and item similarities are used in this approach, but in different ways:

1. The item-item similarities are used as multiplicative factors of the terms in the objective function to force predicted ratings to be more similar to observed ratings of similar items.

2. The user-user similarities are used for predicting the ratings by restricting the regression coefficients to the relevant peer group $P_u(j)$ of the target user u.

Although it is also possible, in principle, to switch the roles of users and items to set up a different model, it is stated in [72] that the resulting model is not as effective as the one discussed above. This model can be solved with any off-the-shelf least-squares solver. A number of methods are also discussed in [72] for handling sparsity.

2.6.5 Sparse Linear Models (SLIM)

An interesting method, based on the item-item regression in section 2.6.2, is proposed in [455]. This family of models is referred to as *sparse linear models* because they encourage sparsity in the regression coefficients with the use of regularization methods. Unlike the methods in [72, 309], these methods work with non-negative rating values. Therefore, unlike the techniques in the previous sections, it will not be assumed that the ratings matrix is mean-centered. This is because mean-centering will automatically create negative ratings, corresponding to dislikes. However, the approach is designed to work with non-negative ratings, in which there is no mechanism to specify dislikes. From a practical point of view, the approach is most appropriate[6] for implicit feedback matrices (e.g., click-through data or sales data), where only positive preferences are expressed through user actions. Furthermore, as is common in implicit feedback settings, missing values are treated as 0s for the purposes of training in the optimization formulation. However, the optimization model might eventually predict some of these values to be highly positive, and such user-item combinations are excellent candidates for recommendation. Therefore, the approach ranks items on the basis of prediction errors on the training entries that have been set to 0.

Unlike the technique in section 2.6.2, these methods do not restrict the regression coefficients to only the neighborhood of the target item t. Then, the prediction function in *SLIM* is expressed as follows:

$$\hat{r}_{ut} = \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj} \quad \forall u \in \{1 \ldots m\}, \ \forall t \in \{1 \ldots n\} \tag{2.33}$$

Note the relationship with Equation 2.27 in which only the neighborhood of the target item is used to construct the regression. It is important to exclude the target item itself on the right-hand side to prevent overfitting. This can be achieved by requiring the constraint that $w_{tt}^{item} = 0$. Let $\hat{R} = [\hat{r}_{uj}]$ represent the predicted ratings matrix and let $W^{item} = [w_{jt}^{item}]$ represent the item-item regression matrix. Therefore, if we assume that the diagonal elements of W^{item} are constrained to be 0, then we can stack up the instantiations of Equation 2.33 over different users and target items to create the following matrix-based prediction function:

$$\hat{R} = RW^{item}$$
$$\text{Diagonal}(W^{item}) = 0$$

Therefore, the main goal is to minimize the Frobenius norm $||R - RW^{item}||^2$ along with some regularization terms. This objective function is disjoint over different columns of W (i.e., target items in regression). Therefore, one can solve each optimization problem (for

[6]The approach can be adapted to arbitrary rating matrices. However, the main advantages of the approach are realized for non-negative ratings matrices.

a given value of the target item t) independently, while setting w_{tt}^{item} to 0. In order to create a more interpretable sum-of-parts regression, the weight vectors are constrained to be non-negative. Therefore, the objective function for target item t may be expressed as follows:

$$\begin{aligned}
\text{Minimize } J_t^s &= \sum_{u=1}^{m} (r_{ut} - \hat{r}_{ut})^2 + \lambda \cdot \sum_{j=1}^{n} (w_{jt}^{item})^2 + \lambda_1 \cdot \sum_{j=1}^{n} |w_{jt}^{item}| \\
&= \sum_{u=1}^{m} (r_{ut} - \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj})^2 + \lambda \cdot \sum_{j=1}^{n} (w_{jt}^{item})^2 + \lambda_1 \cdot \sum_{j=1}^{n} |w_{jt}^{item}| \\
&\text{subject to:} \\
&w_{jt}^{item} \geq 0 \quad \forall j \in \{1 \ldots n\} \\
&w_{tt}^{item} = 0
\end{aligned}$$

The last two terms in the objective function correspond to the *elastic-net regularizer*, which combines L_1- and L_2-regularization. It can be shown [242] that the L_1-regularization component leads to sparse solutions for the weights w_{jt}, which means that most of the coefficients w_{jt} have zero values. The sparsity ensures that each predicted rating can be expressed as a more interpretable linear combination of the ratings of a small number of other related items. Furthermore, since the weights are non-negative, the corresponding items are positively related in a highly interpretable way in terms of the specific level of impact of each rating in the regression. The optimization problem is solved using the coordinate descent method, although any off-the-shelf solver can be used in principle. A number of faster techniques are discussed in [347]. The technique can also be hybridized [456] with side-information (cf. section 6.8.1 of Chapter 6).

It is evident that this model is closely related to the neighborhood-based regression models discussed in the previous sections. The main differences of the *SLIM* model from the linear regression model in [309] are as follows:

1. The method in [309] restricts the nonzero coefficients for each target to at most the k most similar items. The *SLIM* method can use as many as $|U_t|$ nonzero coefficients. For example, if an item is rated by all users, then all coefficients will be used. However, the value of w_{tt}^{item} is set to 0 to avoid overfitting. Furthermore, the *SLIM* method forces sparsity by using the elastic-net regularizer, whereas the method in [309] preselects the weights on the basis of explicit neighborhood computation. In other words, the work in [309] uses a heuristic approach for feature selection, whereas the *SLIM* approach uses a learning (regularization) approach for feature selection.

2. The *SLIM* method is primarily designed for implicit feedback data sets (e.g., buying an item or customer clicks), rather than explicit ratings. In such cases, ratings are typically unary, in which customer actions are indications of positive preference, but the act of not buying or clicking on an item does not necessarily indicate a negative preference. The approach can also be used for cases in which the "ratings" are arbitrary values indicating only positive preferences (e.g., amount of product bought). Note that such scenarios are generally conducive to regression methods that impose non-negativity in the coefficients of the model. As you will learn in Chapter 3, this observation is also true for other models, such as matrix factorization. For example, non-negative matrix factorization is primarily useful for implicit feedback data sets, but it is not quite as useful for arbitrary ratings. This is, in part, because the non-negative, sum-of-parts decomposition loses its interpretability when a rating indicates either a like or a dislike. For example, two "dislike" ratings do not add up to a "like" rating.

3. The regression coefficients in [309] can be either positive or negative. On the other hand, the coefficients in *SLIM* are constrained to be non-negative. This is because the *SLIM* method is primarily designed for the implicit feedback setting. Non-negativity is often more intuitive in these settings and the results are more interpretable. In fact, in some cases, imposing non-negativity might improve[7] the accuracy. However, some limited experimental results have been presented [347], which suggest that removing non-negativity constraints provides superior performance.

4. Although the *SLIM* method also proposes a prediction model for the ratings (according to Equation 2.33), the final *use* of the predicted values is for *ranking* the items in order of the predicted value. Note that the approach is generally used for data sets with unary ratings and therefore, it makes sense to use the predicted values to rank the items, rather than predict ratings. An alternative way of interpreting the predicted values is that each of them can be viewed as the *error* of replacing a non-negative rating with 0 in the ratings matrix. The larger the error is, the greater the predicted value of the rating will be. Therefore, the items can be ranked in the order of the predicted value.

5. Unlike the work in [309], the *SLIM* method does not explicitly adjust for the varying number of specified ratings with heuristic adjustment factors. For example, the right-hand side of Equation 2.29 uses an adjustment factor of $\sqrt{|Q_t(u)|}$ in the denominator. On the other hand, no such adjustment factor is used in the *SLIM* method. The adjustment issue is less pressing for the case of unary data sets, in which the presence of an item is usually the only information available. In such cases, replacing missing values with 0s is a common practice, and the bias of doing so is much lower than in the case where ratings indicate varying levels of likes or dislikes.

Therefore, the models share a number of conceptual similarities, although there are some differences at the detailed level.

2.7 Graph Models for Neighborhood-Based Methods

The sparsity of observed ratings causes a major problem in the computation of similarity in neighborhood-based methods. A number of graph models are used in order to define similarity in neighborhood-based methods, with the use of either structural transitivity or ranking techniques. Graphs are a powerful abstraction that enable many algorithmic tools from the network domain. The graphs provide a structural representation of the relationships among various users and/or items. The graphs can be constructed on the users, on the items, or on both. These different types of graphs result in a wide variety of algorithms, which use

[7] It is noteworthy that imposing an additional constraint, such as non-negativity, always reduces the quality of the optimal solution on the *observed* entries. On the other hand, imposing constraints increases the model bias and reduces model variance, which might reduce overfitting on the *unobserved* entries. In fact, when two closely related models have contradicting relative performances on the observed and unobserved entries, respectively, it is almost always a result of differential levels of overfitting in the two cases. You will learn more about the bias-variance trade-off in Chapter 6. In general, it is more reliable to predict item ratings with positive item-item relationships rather than negative relationships. The non-negativity constraint is based on this observation. The incorporation of model biases in the form of such natural constraints is particularly useful for smaller data sets.

either random-walk or shortest-path methods for recommendation. In the following, we will describe the algorithms used for performing recommendations with various types of graph representations of ratings matrices.

2.7.1 User-Item Graphs

It is possible to use structural measures on the *user-item graph*, rather than the Pearson correlation coefficient, for defining neighborhoods. Such an approach is more effective for sparse ratings matrices because one can use structural transitivity of edges for the recommendation process.

The user-item graph is defined as an undirected and bipartite graph $G = (N_u \cup N_i, A)$, where N_u is the set of nodes representing users, and N_i is the set of nodes representing items. All edges in the graph exist only between users and items. An undirected edge exists in A between a user i and an item j, if and only if user i has rated item j. Therefore, the number of edges is equal to the number of observed entries in the utility matrix. For example, the user-item graph for the ratings matrix of Figure 2.3(a) is illustrated in Figure 2.3(b). The main advantage of graph-based methods is that two users do not need to have rated many of the same items to be considered neighbors as long as many short paths exist between the two users. Therefore, this definition allows the construction of neighborhoods with the notion of *indirect* connectivity between nodes. Of course, if two users have rated many common items, then such a definition will also consider them close neighbors. Therefore, the graph-based approach provides a different way of defining neighborhoods, which can be useful in sparse settings.

The notion of indirect connectivity is achieved with the use of path- or walk-based definitions. Some common methods for achieving this goal include the use of random-walk measures or the *Katz measure*, which is discussed in section 2.7.1.2. Both these measures are closely related to the problem of *link prediction* in social network analysis (cf. section 10.4 of Chapter 10), and they demonstrate the fact that graphical models of recommender systems connect the link-prediction problem to the vanilla recommendation problem. In the following, we discuss different ways of defining neighborhoods on the graph representation.

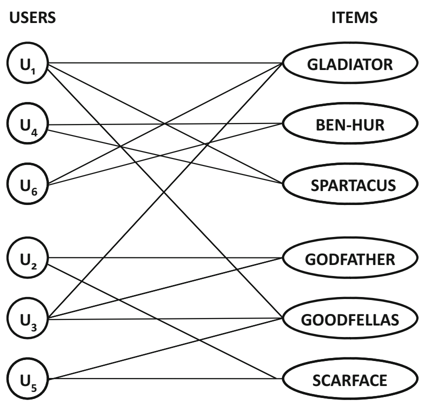
2.7.1.1 Defining Neighborhoods with Random Walks

The neighborhood of a user is defined by the set of users that are encountered frequently in a random walk starting at that user. How can the expected frequency of such random walks be measured? The answer to this problem is closely related to the random-walk methods, which are used frequently in Web-ranking applications. One can use either the personalized *PageRank* or the *SimRank* method (cf. Chapter 10) to determine the k most similar users to a given user for user-based collaborative filtering. Similarly, one can use this method to determine the k most similar items to a given item by starting the random walk at a given *item*. This approach is useful for item-based collaborative filtering. The other steps of user-based collaborative filtering and item-based collaborative filtering remain the same.

Why is this approach more effective for sparse matrices? In the case of the Pearson's correlation coefficient, two users need to be connected *directly* to a set of common items for the neighborhood to be defined meaningfully. In sparse user-item graphs, such direct connectivity may not exist for many nodes. On the other hand, a random-walk method also considers *indirect* connectivity, because a walk from one node to another may use any number of steps. Therefore, as long as large portions of the user-item graphs are connected,

	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS	SCARFACE	SPARTACUS
U_1	1			5		2
U_2		5			4	
U_3	5	3		1		
U_4			3			4
U_5				3	5	
U_6	5		4			

(a) Ratings matrix



(b) User-item graphs of specified ratings

Figure 2.3: A ratings matrix and corresponding user-item graph

it is always possible to meaningfully define neighborhoods. Such user-item graphs can also be used to directly predict ratings with the use of a variety of models. Such related methods will be discussed in section 10.2.3.3 of Chapter 10.

2.7.1.2 Defining Neighborhoods with the Katz Measure

Rather than using a probabilistic measure, such as random walks, it is possible to use the weighted number of walks between a pair of nodes in order to determine the affinity between them. The weight of each walk is a discount factor in $(0, 1)$, which is typically a decreasing function of its length. The weighted number of walks between a pair of nodes is referred to as the *Katz* measure. The weighted number of walks between a pair of nodes is often used as a link-prediction measure. The intuition is that if two users belong to the same neighborhood based on walk-based connectivity, then there is a propensity for a link to be formed between them in the user-item graph. The specific level of propensity is measured with the number of (discounted) walks between them.

Definition 2.7.1 (Katz Measure) *Let $n_{ij}^{(t)}$ be the number of walks of length t between nodes i and j. Then, for a user-defined parameter $\beta < 1$, the Katz measure between nodes i and j is defined as follows:*

$$Katz(i, j) = \sum_{t=1}^{\infty} \beta^t \cdot n_{ij}^{(t)} \tag{2.34}$$

The value of β is a discount factor that de-emphasizes walks of longer lengths. For small enough values of β, the infinite summation of Equation 2.34 will converge.

Let K be the $m \times m$ matrix of Katz coefficients between pairs of users. If A is the symmetric adjacency matrix of an undirected network, then the pairwise Katz coefficient matrix K can be computed as follows:

$$K = \sum_{i=1}^{\infty} (\beta A)^i = (I - \beta A)^{-1} - I \tag{2.35}$$

The value of β should always be selected to be smaller than the inverse of the largest eigenvalue of A to ensure convergence of the infinite summation. The Katz measure is closely rated to diffusion kernels in graphs. In fact, several collaborative recommendation methods directly use diffusion kernels to make recommendations [205].

A weighted version of the measure can be computed by replacing A with the weight matrix of the graph. This can be useful in cases where one wishes to weight the edges in the user-item graph with the corresponding rating. The top-k nodes with the largest Katz measures to the target node are isolated as its neighborhood. Once the neighborhood has been determined, it is used to perform the prediction according to Equation 2.4. Many variations of this basic principle are used to make recommendations:

1. It is possible to use a threshold on the maximum path length in Equation 2.34. This is because longer path lengths generally become noisy for the prediction process. Nevertheless, because of the use of the discount factor β, the impact of long paths on the measure is generally limited.

2. In the aforementioned discussion, the Katz measure is used only to determine the neighborhoods of users. Therefore, the Katz measure is used to compute the affinity between pairs of *users*. After the neighborhood of a user has been determined, it is used to make predictions in the same way as any other neighborhood-based method.

   However, a different way of *directly* performing the prediction, without using neighborhood methods, would be to measure the affinity between users and *items*. The Katz measure can be used to compute these affinities. In such cases, the links are weighted with ratings, and the problem is reduced to that of predicting links between users and items. These methods will be discussed in more detail in section 10.4.6 of Chapter 10.

The bibliographic notes contain a number of references to various path-based methods.

2.7.2 User-User Graphs

In user-item graphs, the user-user connectivity is defined by an even number of hops in the user-item graph. Instead of constructing user-item graphs, one might instead directly create user-user graphs based on 2-hop connectivity between users. The advantage of user-user graphs over user-item graphs is that the edges of the graph are more informative in the former. This is because the 2-hop connectivity can directly take the number and similarity of common items between the two users into account, while creating the edges. These notions, referred to as *horting* and *predictability*, will be discussed slightly later. The algorithm uses the notion of horting to quantify the number of mutually specified ratings between two users (nodes), whereas it uses the notion of predictability to quantify the level of similarity among these common ratings.

The user-user graph is constructed as follows. Each node u corresponds to one of the m users in the $m \times n$ user-item matrix. Let I_u be the set of items for which ratings have been specified by user u, and let I_v be the set of items for which ratings have been specified by user v. Edges are defined in this graph with the notion of *horting*. Horting is an asymmetric relationship between users, which is defined on the basis of their having rated similar items.

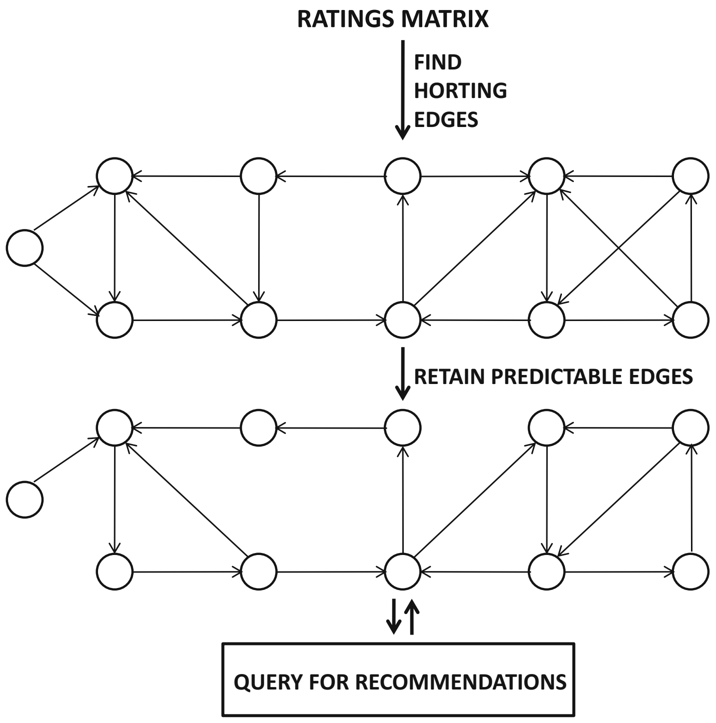


Figure 2.4: The user-user predictability approach

Definition 2.7.2 (Horting) *A user u is said to hort user v at level (F, G), if either of the following are true:*

$$|I_u \cap I_v| \geq F$$
$$|I_u \cap I_v|/|I_u| \geq G$$

Here, F and G are algorithm parameters. Note that it is sufficient for one of the two aforementioned conditions to hold for user u to hort user v. The notion of horting is used to further define predictability.

Definition 2.7.3 (Predictability) *The user v predicts user u, if u horts v and there exists a linear transformation function $f(\cdot)$ such that the following is true:*

$$\frac{\sum_{k \in I_u \cap I_v} |r_{uk} - f(r_{vk})|}{|I_u \cap I_v|} \leq U$$

Here, U is another algorithm parameter. It is noteworthy that the distance $\frac{\sum_{k \in I_u \cap I_v} |r_{uk} - f(r_{vk})|}{|I_u \cap I_v|}$ between the ratings of user u and the transformed ratings of user v is a variant of the Manhattan distance on their common specified ratings. The main difference from the Manhattan distance is that the distance is normalized by the number of mutually specified ratings between the two users. This distance is also referred to as the *Manhattan segmental distance.*

The directions of horting and predictability are opposite one another. In other words, for user v to predict user u, u must hort v. A directed graph G is defined, in which an edge exists from u to v, if v predicts u. This graph is referred to as the *user-user predictability graph*. Each edge in this graph corresponds to a linear transformation, as discussed in Definition 2.7.3. The linear transformation defines a prediction, where the rating at the head of the edge can be used to predict the rating at the tail of the edge. Furthermore, it is assumed that one can apply these linear transformations in a transitive way over a directed path in order to predict the rating of the source of the path from the rating at the destination of the path.

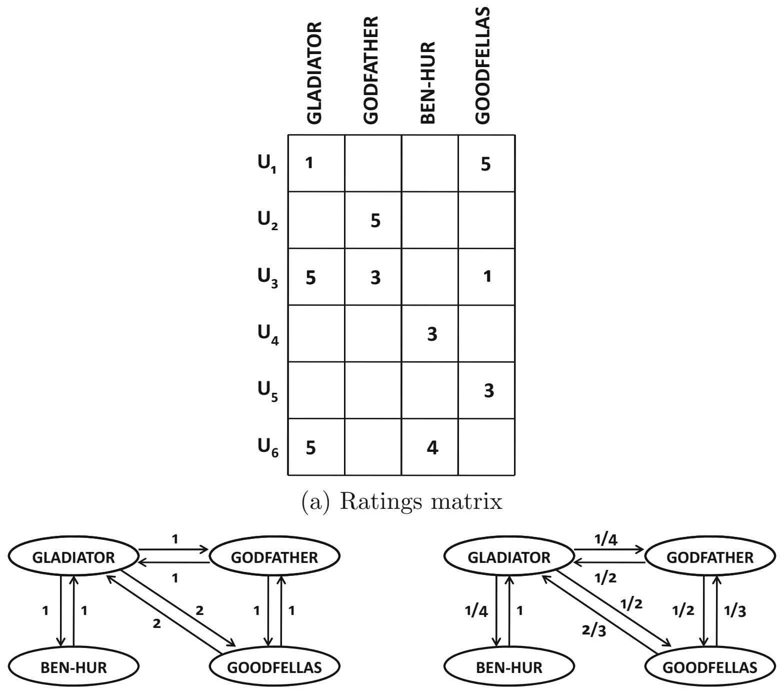
Then, the rating of a target user u for an item k is computed by determining all the directed shortest paths from user u to all other users who have rated item k. Consider a directed path of length r from user u to a user v who has rated item k. Let $f_1 \ldots f_r$ represent the sequence of linear transformations along the directed path starting from node u to this user v. Then, the rating prediction $\hat{r}_{uk}^{(v)}$ of the rating of target user u for item k (based only on user v) is given by applying the composition of the r linear mappings along this path from user u to v, to the rating r_{vk} of user v on item k:

$$\hat{r}_{uk}^{(v)} = (f_1 \circ f_2 \ldots \circ f_r)(r_{vk}) \tag{2.36}$$

The rating prediction $\hat{r}_{uk}^{(v)}$ contains the superscript v because it is based only on the rating of user v. Therefore, the final rating prediction $\hat{r}_{uk}$ is computing by averaging the value of $\hat{r}_{uk}^{(v)}$ over all users v that have rated item k, within a threshold distance D of the target user u.

Given a target user (node) u, one only needs to determine directed *paths* from this user to other users, who have rated the item at hand. The shortest path can be determined with the use of a breadth-first algorithm, which is quite efficient. Another important detail is that a threshold is imposed on the maximum path length that is usable for prediction. If no user, who has rated item k is found within a threshold length D of the target node u, then the algorithm terminates with failure. In other words, the rating of the target user u for item k simply cannot be determined robustly with the available ratings matrix. It is important to impose such thresholds to improve efficiency and also because the linear transformation along very long path lengths might lead to increasing distortion in the rating prediction. The overall approach is illustrated in Figure 2.4. Note that a directed edge exists from u to v in the horting graph if u horts v. On the other hand, an edge exists in the predictability graph if u horts v *and* v predicts u. Therefore, the predictability graph is obtained from the horting graph by dropping a few edges. This graph is set up in an offline phase and it is repeatedly queried for recommendations. In addition, a number of index data structures are set up from the ratings matrix during the offline setup phase. These data structures are used along with the predictability graph in order to resolve the queries efficiently. More details on the horting approach may be found in [33].

This approach can work for very sparse matrices because it uses transitivity to predict ratings. An important challenge in neighborhood methods is the lack of coverage of rating prediction. For example, if none of John's immediate neighbors have rated *Terminator*, it is impossible to provide a rating prediction for John. However, structural transitivity allows us to check whether the *indirect* neighbors of John have rated *Terminator*. Therefore, the main advantage of this approach is that it has better coverage compared to competing methods.



	GLADIATOR	GODFATHER	BEN-HUR	GOODFELLAS
U_1	1			5
U_2		5		
U_3	5	3		1
U_4			3	
U_5				3
U_6	5		4	

(a) Ratings matrix

(b) Unnormalized correlation graph (c) Normalized correlation graph

Figure 2.5: A ratings matrix and its correlation graphs

2.7.3 Item-Item Graphs

It is also possible to leverage item-item graphs to perform the recommendations. Such a graph is also referred to as the *correlation graph* [232]. In this case, a weighted and directed network $G = (N, A)$ is constructed, in which each node in N corresponds to an item, and each edge in A corresponds to a relationship between items. The weight w_{ij} is associated with each edge (i, j). If items i and j have both been rated by at least one common user, then both the directed edges (i, j) and (j, i) exist in the network. Otherwise, no edges exist between nodes i and j. The directed network is, however, asymmetric because the weight of edge (i, j) is not necessarily the same as that of edge (j, i). Let U_i be the set of users that have specified ratings for item i and U_j be the set of users that have specified ratings for item j. Then, the weight of the edge (i, j) is computed using the following simple algorithm.

First, we initialize the weight w_{ij} of each edge (i, j) to $|U_i \cap U_j|$. At this point, the edge weights are symmetric because $w_{ij} = w_{ji}$. Then, the weights of the edges are normalized, so that the sum of the weights of the outgoing edges of a node is equal to 1. This normalization is achieved by dividing w_{ij} with the sum of the outgoing weights from node i. The normalization step results in asymmetric weights, because each of the weights w_{ij} and w_{ji} are divided by different quantities. This results in a graph in which the weights on edges correspond to *random-walk probabilities.* An example of the correlation graph for a ratings matrix is illustrated in Figure 2.5. It is clear that the weights on the *normalized* correlation graph are not symmetric because of the scaling of the weights to transition probabilities. Furthermore, it is noteworthy that the rating values are not used in the construction of the correlation graph. Only the *number* of observed ratings in common between two items is used. This is sometimes not desirable. It is, of course, possible to define the correlation graph in other ways, such as with the use of the cosine function between the *rating* vectors of the two items.

As discussed in Chapter 10, random-walk methods can be used to determine the neighborhood of a given item. The resulting neighborhood can be used for item-based collaborative filtering methods. Furthermore, personalized *PageRank* methods can be used to directly determine the ratings on the item-item graph. This method is referred to as *ItemRank*, and it is discussed in section 10.2.3.3 of Chapter 10.

2.8 Summary

Because collaborative filtering can be viewed as a generalization of classification and regression problems, the methodologies for the latter classes of problems can also be applied to the former. Neighborhood-based methods derive their inspiration from nearest neighbor classification and regression methods. In user-based methods, the first step is to determine the neighborhood of the target user. In order to compute the neighborhood, a variety of similarity functions, such as the Pearson correlation coefficient or the cosine, are used. The neighborhood is used in order to extrapolate the unknown ratings of a record. In item-based methods, the most similar items are computed with respect to a target item. Then, the user's own ratings on these similar items are used in order to make a rating prediction. Item-based methods are likely to have more relevant recommendations, but they are less likely to yield diverse recommendations. In order to speed up neighborhood-based methods, clustering is often used.

Neighborhood-based methods can be viewed as linear models, in which the weights are chosen in a heuristic way with the use of similarity values. One can also learn these weights with the use of linear regression models. Such methods have the advantage that they can be combined with other optimization models, such as matrix factorization, for better prediction. Such methods are discussed in the next chapter.

Neighborhood-based methods face numerous challenges because of data sparsity. Users often specify only a small number of ratings. As a result, a pair of users may often have specified only a small number of ratings. Such situations can be addressed effectively with the use of both dimensionality reduction and graph-based models. While dimensionality reduction methods are often used as standalone methods for collaborative filtering, they can also be combined with neighborhood-based methods to improve the effectiveness and efficiency of collaborative filtering. Various types of graphs can be extracted from rating patterns, such as user-item graphs, user-user graphs, or item-item graphs. Typically, random-walk or shortest-path methods are used in these cases.

2.9 Bibliographic Notes

Neighborhood-based methods were among the earliest techniques used in the field of recommender systems. The earliest user-based collaborative filtering models were studied in [33, 98, 501, 540]. A comprehensive survey of neighborhood-based recommender systems may be found in [183]. Sparsity is a major problem in such systems, and various graph-based systems have been designed to alleviate the problem of sparsity [33, 204, 647]. Methods that are specifically designed for the long tail in recommender algorithms are discussed in [173, 463, 648].

User-based methods utilize the ratings of *similar* users on the *same* item in order to make predictions. While such methods were initially quite popular, they are not easily scalable and sometimes inaccurate. Subsequently, item-based methods [181, 360, 524] were proposed, which compute predicted ratings as a function of the ratings of the *same* user on *similar* items. Item-based methods provide more accurate but less diverse recommendations.

The notion of mean-centering for improving recommendation algorithms was proposed in [98, 501]. A comparison of the use of the Z-score with mean-centering is studied in [245, 258], and these two studies provide somewhat conflicting results. A number of methods which do not use the absolute ratings, but instead focus on ordering the ratings in terms of preference weights, are discussed in [163, 281, 282]. The significance-weighting methods of de-emphasizing the neighbors who have too few common ratings with a given neighbor are discussed in [71, 245, 247, 380]. Many different variants of the similarity function are used for computing the neighbor. Two such examples are the *mean-squared distance* [540] and the *Spearman rank correlation* [299]. The specific advantage of these distance measures is not quite clear because conflicting results have been presented in the literature [247, 258]. Nevertheless, the consensus seems to be that the Pearson rank correlation provides the most accurate results [247]. Techniques for adjusting for the impact of very popular items are discussed in [98, 280]. The use of exponentiated amplification for prediction in neighborhood-based methods is discussed in [98]. A discussion of the use of voting techniques in nearest neighbor methods may be found in [183]. Voting methods can be viewed as a direct generalization of the nearest neighbor classifier, as opposed to a generalization of nearest neighbor regression modeling.

Methods for item-based collaborative filtering were proposed in [181, 524, 526]. A detailed study of different variations of item-based collaborative filtering algorithms is provided in [526], along with a comparison with respect to user-based methods. The item-based method in [360] is notable because it describes one of Amazon.com's collaborative filtering methods. The user-based and item-based collaborative filtering methods have also been unified with the notion of similarity fusion [622]. A more generic unification framework may be found in [613]. Clustering methods are used frequently to improve the efficiency of neighborhood-based collaborative filtering. A number of clustering methods are described in [146, 167, 528, 643, 644, 647]. The extension of neighborhood methods to very large-scale data sets has been studied in [51].

Dimensionality reduction techniques have a rich history of being used in missing-value estimation [24, 472] and recommender systems [71, 72, 228, 252, 309, 313, 500, 517, 525]. In fact, most of these techniques directly use such latent models to predict the ratings without relying on neighborhood models. However, some of these dimensionality reduction techniques [71, 72, 309, 525] are specifically designed to improve the effectiveness and efficiency of neighborhood-based techniques. A key contribution of [72] is to provide an insight about the relationship between neighborhood methods and regression-based methods. This relationship is important because it shows how one can formulate neighborhood-based methods as model-based methods with a crisp optimization formulation. Note that many other model-based methods, such as latent factor models, can also be expressed as optimization formulations. This observation paves the way for combining neighborhood methods with latent factor models in a unified framework [309] because one can now combine the two objective functions. Other regression-based models for recommender systems, such as *slope-one predictors* and ordinary least-squares methods, are proposed in [342, 620]. Methods for learning pairwise preferences over itemsets are discussed in [469]. Item-item regression models have also been studied in the context of *Sparse Linear Models (SLIM)* [455], where an elastic-net regularizer is used on the linear model without restricting the coefficients to the neighborhood of the item. Higher-order sparse learning methods, which model the effects of using combinations of items, are discussed in [159]. Efficient methods for training linear models and tuning regularization parameters are discussed in [347]. Constrained linear regression methods are discussed in [430].

A general examination of linear classifiers, such as least-squares regression and support vector machines, is provided in [669]. However, the approach is designed for implicit feedback data sets in which only positive preferences are specified. It was observed that collaborative filtering, in such cases, is similar to text categorization. However, because of the noise in the data and the imbalanced nature of the class distribution, a direct use of SVM methods is sometimes not effective. Changes to the loss function are suggested in [669] in order to provide more accurate results.

Many graph-based methods have been proposed for improving collaborative filtering algorithms. Most of these methods are based on either user-item graphs, but a few are also based on user-user graphs. An important observation from the perspective of graph-based methods is that they show an interesting relationship between the problems of ranking, recommendation, and link-prediction. The use of random walks for determining the neighborhood in recommendation systems is discussed in [204, 647]. A method, which uses the *number* of discounted paths between a pair of nodes in a user-item graph for recommendations, was proposed in [262]. This approach is equivalent to using the Katz measure between user-user pairs in order to determine whether they reside in each other's neighborhoods. This approach is related to link-prediction [354], because the Katz measure is often used to determine the linkage affinity between a pair of nodes. A survey on link prediction methods may be found in [17]. Some graph-based methods do not directly use neighborhoods. For example, the *ItemRank* method proposed in [232] shows how to use ranking directly to make predictions, and the method in [261] shows how to use link-prediction methods directly for collaborative filtering. These methods are also discussed in Chapter 10 of this book. Techniques for leveraging user-user graphs are discussed in [33]. These methods have the advantage that they directly encode the user-user similarity relationships in the edges of the graph. As a result, the approach provides better coverage than competing methods.

2.10 Exercises

1. Consider the ratings matrix of Table 2.1. Predict the absolute rating of item 3 for user 2 using:

(a) User-based collaborative filtering with Pearson correlation and mean-centering

(b) Item-based collaborative filtering with adjusted cosine similarity

Use a neighborhood of size 2 in each case.

2. Consider the following ratings table between five users and six items:

Item-Id ⇒	1	2	3	4	5	6
1	5	6	7	4	3	?
2	4	?	3	?	5	4
3	?	3	4	1	1	?
4	7	4	3	6	?	4
5	1	?	3	2	2	5

(a) Predict the values of unspecified ratings of user 2 using user-based collaborative filtering algorithms. Use the Pearson correlation with mean-centering.

(b) Predict the values of unspecified ratings of user 2 using item-based collaborative filtering algorithms. Use the adjusted cosine similarity.

Assume that a peer group of size at most 2 is used in each case, and negative correlations are filtered out.

3. Discuss the similarity between a k-nearest neighbor classifier in traditional machine learning and the user-based collaborative filtering algorithm. Describe an analogous classifier to item-based collaborative filtering.

4. Consider an algorithm that performs clustering of users based on their ratings matrix and reports the average ratings within a cluster as the predicted items ratings for every user within a cluster. Discuss the effectiveness and efficiency trade-offs of such an approach compared to a neighborhood model.

5. Propose an algorithm that uses random walks on a user-user graph to perform neighborhood-based collaborative filtering. [This question requires a background in ranking methods.]

6. Discuss various ways in which graph clustering algorithms can be used to perform neighborhood-based collaborative filtering.

7. Implement the user-based and item-based collaborative filtering algorithms.

8. Suppose you had content-based profiles associated with users indicating their interests and profiles associated with items corresponding to their descriptions. At the same time, you had a ratings matrix between users and items. Discuss how you can incorporate the content-based information within the framework of graph-based algorithms.

9. Suppose that you had a unary ratings matrix. Show how collaborative filtering algorithms can be solved using content-based methods by treating the ratings of an item as its features. Refer to Chapter 1 for a description of content-based methods. What type of a content-based classifier does an item-based collaborative filtering algorithm correspond to?

Chapter 3

Model-Based Collaborative Filtering

"*Do not quench your inspiration and your imagination; do not become the slave of your model.*"– Vincent van Gogh

3.1 Introduction

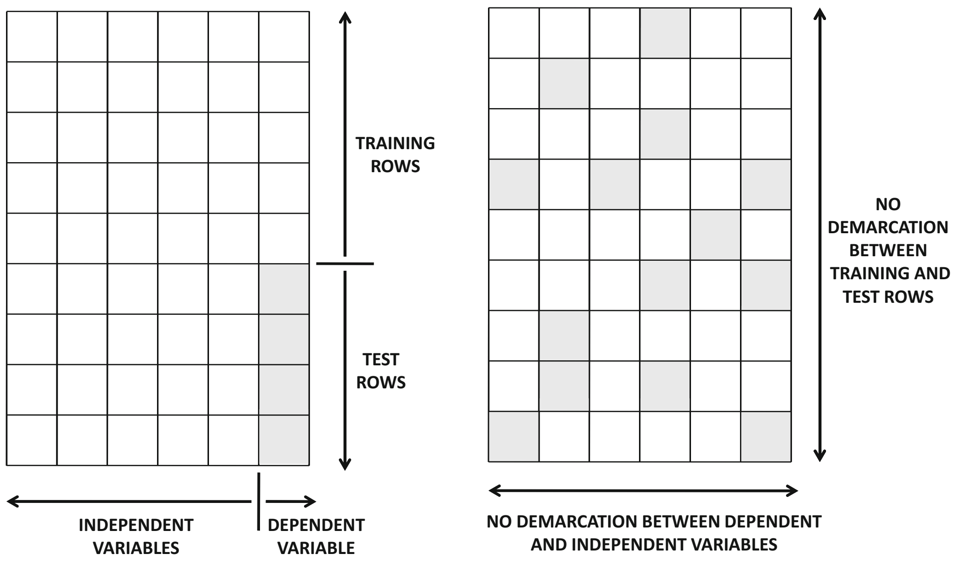
The neighborhood-based methods of the previous chapter can be viewed as generalizations of k-nearest neighbor classifiers, which are commonly used in machine learning. These methods are instance-based methods, whereby a model is not specifically created up front for prediction other than an optional preprocessing[1] phase, which is required to ensure efficient implementation. Neighborhood-based methods are generalizations of *instance-based learning methods* or *lazy learning methods* in which the prediction approach is *specific to the instance being predicted*. For example, in user-based neighborhood methods, the peers of the *target* user are determined in order to perform the prediction.

In model-based methods, a summarized model of the data is created up front, as with supervised or unsupervised machine learning methods. Therefore, the training (or *model-building* phase) is clearly separated from the prediction phase. Examples of such methods in traditional machine learning include decision trees, rule-based methods, Bayes classifiers, regression models, support vector machines, and neural networks [22]. Interestingly, almost all these models can be generalized to the collaborative filtering scenario, just as k-nearest neighbor classifiers can be generalized to neighborhood-based models for collaborative filtering. This is because the traditional classification and regression problems are special cases of the matrix completion (or collaborative filtering) problem.

In the data classification problem, we have an $m \times n$ matrix, in which the first $(n-1)$ columns are feature variables (or independent variables), and the last (i.e., nth) column is

[1]From a practical point of view, preprocessing is essential for efficiency. However, one could implement the neighborhood method without a preprocessing phase, albeit with larger latencies at query time.

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_3


TRAINING ROWS
TEST ROWS
INDEPENDENT VARIABLES
DEPENDENT VARIABLE
NO DEMARCATION BETWEEN TRAINING AND TEST ROWS
NO DEMARCATION BETWEEN DEPENDENT AND INDEPENDENT VARIABLES

(a) Classification (b) Collaborative filtering

Figure 3.1: Revisiting Figure 1.4 of Chapter 1. Comparing the traditional classification problem with collaborative filtering. Shaded entries are missing and need to be predicted.

the class variable (or dependent variable). All entries in the first $(n-1)$ columns are fully specified, whereas only a subset of the entries in the nth column is specified. Therefore, a subset of the rows in the matrix is fully specified, and these rows are referred to as the *training data*. The remaining rows are referred to as the *test data*. The values of the missing entries need to be learned for the test data. This scenario is illustrated in Figure 3.1(a), where the shaded values represent missing entries in the matrix.

Unlike data classification, any entry in the ratings matrix may be missing, as illustrated by the shaded entries in Figure 3.1(b). Thus, it can be clearly seen that the matrix completion problem is a generalization of the classification (or regression modeling) problem. Therefore, the crucial differences between these two problems may be summarized as follows:

1. In the data classification problem, there is a clear separation between feature (independent) variables and class (dependent) variables. In the matrix completion problem, this clear separation does not exist. Each column is both a dependent and independent variable, depending on which entries are being considered for predictive modeling at a given point.

2. In the data classification problem, there is a clear separation between the training and test data. In the matrix completion problem, this clear demarcation does not exist among the *rows* of the matrix. At best, one can consider the specified (observed) *entries* to be the training data, and the unspecified (missing) entries to be the test data.

3. In data classification, columns represent features, and rows represent data instances. However, in collaborative filtering, it is possible to apply the same approach to either the ratings matrix or to its transpose because of how the missing entries are distributed. For example, user-based neighborhood models can be viewed as direct

generalizations of nearest neighbor classifiers. When such methods are applied to the transpose of the ratings matrix, they are referred to as *item*-based neighborhood models. In general, many classes of collaborative filtering algorithms have both user-wise and item-wise versions.

These differences between data classification and collaborative filtering are illustrated in Figure 3.1. The greater generality of the collaborative filtering problem leads to a richer number of algorithmic *possibilities* in collaborative filtering, as compared to data classification.

The similarity between the collaborative filtering problem and the data classification problem is useful to keep in mind when designing learning algorithms for the former. This is because data classification is a relatively well-studied field, and the various types of solutions to classification also provide important hints for the design of collaborative filtering algorithms. In fact, most machine learning and classification algorithms have direct analogs in the collaborative filtering literature. Understanding recommender systems in a similar way to classification models enables the application of a significant number of meta-algorithms from the classification literature. For example, classical meta-algorithms from the classification literature, such as bagging, boosting or model combination, can be extended to collaborative filtering. Interestingly, much of the theory developed for ensemble methods in classification continues to apply to recommender systems. In fact, the ensemble-based methods [311, 704] were among the best performing methods in the Netflix challenge. Ensemble methods are discussed in detail in Chapter 6.

It is not always easy, however, to generalize data classification models directly to the matrix completion problem, especially when the vast majority of the entries are missing. Furthermore, the relative effectiveness of the various models are different in different settings. For example, a number of recent collaborative filtering models, such as latent factor models, are particularly well suited to collaborative filtering. Such models are, however, not considered competitive models in the context of data classification.

Model-based recommender systems often have a number of advantages over neighborhood-based methods:

1. *Space-efficiency:* Typically, the size of the learned model is much smaller than the original ratings matrix. Thus, the space requirements are often quite low. On the other hand, a user-based neighborhood method might have $O(m^2)$ space complexity, where m is the number of users. An item-based method will have $O(n^2)$ space complexity.

2. *Training speed and prediction speed:* One problem with neighborhood-based methods is that the pre-processing stage is quadratic in either the number of users or the number of items. Model-based systems are usually much faster in the preprocessing phase of constructing the trained model. In most cases, the compact and summarized model can be used to make predictions efficiently.

3. *Avoiding overfitting: Overfitting* is a serious problem in many machine learning algorithms, in which the prediction is overly influenced by random artifacts in the data. This problem is also encountered in classification and regression models. The summarization approach of model-based methods can often help in avoiding overfitting. Furthermore, *regularization* methods can be used to make these models robust.

Even though neighborhood-based methods were among the earliest collaborative filtering methods and were also among the most popular because of their simplicity, they are not necessarily the most accurate models available today. In fact, some of the most accurate methods are based on model-based techniques in general, and on latent factor models in particular.

This chapter is organized as follows. Section 3.2 discusses the use of decision and regression trees for recommender systems. Rule-based collaborative filtering methods are discussed in section 3.3. The use of the naive Bayes model for recommender systems is discussed in section 3.4. A general discussion of how other classification methods are extended to collaborative filtering is provided in section 3.5. Latent factor models are discussed in section 3.6. The integration of latent factor models with neighborhood models is discussed in section 3.7. A summary is given in section 3.8.

3.2 Decision and Regression Trees

Decision and regression trees are frequently used in data classification. Decision trees are designed for those cases in which the dependent variable is categorical, whereas regression trees are designed for those cases in which the dependent variable is numerical. Before discussing the generalization of decision trees to collaborative filtering, we will first discuss the application of decision trees to classification.

Consider the case in which we have an $m \times n$ matrix R. Without loss of generality, assume that the first $(n-1)$ columns are the independent variables, and the final column is the dependent variable. For ease in discussion, assume that all variables are binary. Therefore, we will discuss the creation of a decision tree rather than a regression tree. Later, we will discuss how to generalize this approach to other types of variables.

The decision tree is a hierarchical partitioning of the data space with the use of a set of hierarchical decisions, known as the *split criteria* in the independent variables. In a *univariate decision tree*, a single feature is used at one time in order to perform a split. For example, in a binary matrix R, in which the feature values are either 0 or 1, all the data records in which a carefully chosen feature variable takes on the value of 0 will lie in one branch, whereas all the data records in which the feature variable takes on the value of 1 will lie in the other branch. When the feature variable is chosen in such a way, so that it is correlated with the class variable, the data records within each branch will tend to be *purer*. In other words, most of the records belonging to the different classes will be separated out. In other words, one of the two branches will predominantly contain one class, whereas the other branch will predominantly contain the other class. When each node in a decision tree has two children, the resulting decision tree is said to be a binary decision tree.

The quality of the split can be evaluated by using the weighted average *Gini index* of the child nodes created from a split. If $p_1 \ldots p_r$ are the fractions of data records belonging to r different classes in a node S, then the Gini index $G(S)$ of the node is defined as follows:

$$G(S) = 1 - \sum_{i=1}^{r} p_i^2 \tag{3.1}$$

The Gini index lies between 0 and 1, with smaller values being more indicative of greater discriminative power. The overall Gini index of a split is equal to the weighted average of the Gini index of the children nodes. Here, the weight of a node is defined by the number of data points in it. Therefore, if S_1 and S_2 are the two children of node S in a binary decision tree, with n_1 and n_2 data records, respectively, then the Gini index of the split $S \Rightarrow (S_1, S_2)$ may be evaluated as follows:

$$\text{Gini}\,(S \Rightarrow [S_1, S_2]) = \frac{n_1 \cdot G(S_1) + n_2 \cdot G(S_2)}{n_1 + n_2} \tag{3.2}$$

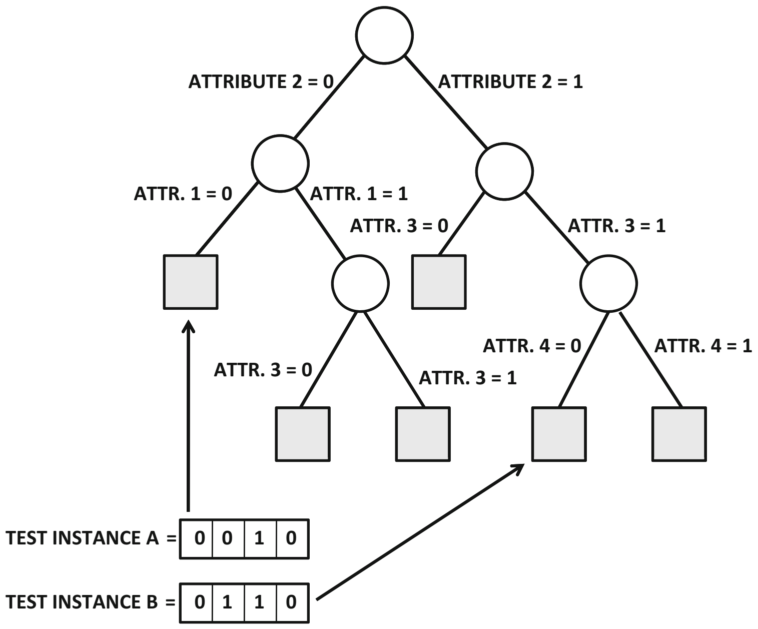


Figure 3.2: Example of a decision tree with four binary attributes

The Gini index is used for selecting the appropriate attribute to use for performing the split at a given level of the tree. One can test each attribute to evaluate the Gini index of its split according to Equation 3.2. The attribute with the smallest Gini index is selected for performing the split. The approach is executed hierarchically, in top-down fashion, until each node contains only data records belonging to a particular class. It is also possible to stop the tree growth early, when a minimum fraction of the records in the node belong to a particular class. Such a node is referred to as a leaf node, and it is labeled with the dominant class of the records in that node. To classify a test instance with an unknown value of the dependent variable, its independent variables are used to map a path in the decision tree from the root to the leaf. Because the decision tree is a hierarchical partitioning of the data space, the test instance will follow exactly one path from the root to the leaf. The label of the leaf is reported as the relevant one for the test instance. An example of a decision tree, constructed on four binary attributes, is illustrated in Figure 3.2. The leaf nodes of the tree are shaded in the figure. Note that all attributes are not necessarily used for splits by the decision tree. For example, the leftmost path uses attributes 1 and 2, but it does not use attributes 3 and 4. Furthermore, different paths in the decision tree may use different sequences of attributes. This situation is particularly common with high-dimensional data. Examples of the mappings of test instances A= 0010 and B= 0110 to respective leaf nodes are illustrated in Figure 3.2. Each of these test instances is mapped to a unique leaf node because of the hierarchical nature of the data partitioning.

The approach can be extended to numerical dependent and independent variables with minor modifications. To handle numerical independent (feature) variables, the attribute values can be divided into intervals in order to perform the splits. Note that this approach might result in a multi-way split, where each branch of the split corresponds to a different interval. The split is then performed by choosing the attribute on the basis of the Gini index criterion. Such an approach also applies to categorical feature variables, wherein each value of the categorical attribute corresponds to a branch of the split.

To handle numeric *dependent* variables, the split criterion is changed from the Gini index to a measure better suited to numeric attributes. Specifically, the variance of the numeric dependent variable is used instead of the Gini index. A lower variances is more desirable because it means that the node contains training instances that are discriminatively mapped in the locality of the dependent variable. Either the average value in the leaf node, or a linear regression model, is used at the leaf node to perform the prediction [22].

In many cases, the tree is pruned to reduce overfitting. In this case, a portion of the training data is not used during the tree construction phase. Then, the effect of pruning the node is tested on the portion of the training data that is held out. If the removal of the node improves the accuracy of the decision tree prediction on the held out data, then the node is pruned. Additionally, other variations of the split criteria, such as error rates and entropy, are commonly used. Detailed discussions of various design choices in decision tree construction may be found in [18, 22].

3.2.1 Extending Decision Trees to Collaborative Filtering

The main challenge in extending decision trees to collaborative filtering is that the predicted entries and the observed entries are not clearly separated in column-wise fashion as feature and class variables. Furthermore, the ratings matrix is very sparsely populated, where the majority of entries are missing. This creates challenges in hierarchically partitioning the training data during the tree-building phase. Furthermore, since the dependent and independent variables (items) are not clearly demarcated in collaborative filtering, what item should be predicted by the decision tree?

The latter issue is relatively easy to address by constructing separate decision trees to predict the rating of each item. Consider an $m \times n$ ratings matrix R with m users and n items. A separate decision tree needs to be constructed by fixing each attribute (item) to be dependent and the remaining attributes as independent. Therefore, the number of decision trees constructed is exactly equal to the number n of attributes (items). While predicting the rating of a particular item for a user, the decision tree corresponding to the relevant item is used for prediction.

On the other hand, the issue of missing independent features is more difficult to address. Consider the case, where a particular item (say, a particular movie) is used as a splitting attribute. All users whose rating is less than a threshold are assigned to one branch of the tree, whereas the users whose ratings are larger than the threshold are assigned to the other branch. Because ratings matrices are sparse, most users will not have specified ratings for this item. Which branch should such users be assigned to? Logic dictates that such users should be assigned to *both* branches. However, in such a case, the decision tree no longer remains a strict partitioning of the training data. Furthermore, according to this approach, test instances will map to multiple paths in the decision tree, and the possibly conflicting predictions from the various paths will need to combined into a single prediction.

A second (and more reasonable) approach is to create a lower-dimensional representation of the data using the dimensionality reduction methods discussed in section 2.5.1.1 of Chapter 2. Consider the scenario, where the rating of the jth item needs to be predicted. At the very beginning, the $m \times (n-1)$ ratings matrix, excluding the jth column, is converted into a lower-dimensional $m \times d$ representation, in which $d \ll n-1$ and all attributes are fully specified. The covariance between each pair of items in the $m \times (n-1)$ ratings matrix is estimated using the methods discussed in section 2.5.1.1 of Chapter 2. The top-d

eigenvectors $\overline{e_1} \ldots \overline{e_d}$ of the estimated $(n-1) \times (n-1)$ covariance matrix are determined. Note that each eigenvector is a vector containing $(n-1)$ elements. Equation 2.17 is used for projecting the ratings of each user on the eigenvectors, except that the jth item is not included on the right-hand side of Equation 2.17. This results in a d-dimensional vector of ratings for each user, which is completely specified. This reduced representation is used to construct the decision tree for the jth item by treating the problem as a standard classification or regression modeling problem. This approach is repeated by varying the value of j from 1 to n, in order to construct a total of n decision trees. Therefore, the jth decision tree is useful only for predicting the ratings of the jth item. Both the eigenvectors and the trees for each of the n cases are stored as a part of the model.

To predict the rating of item j for a user i, the ith row of the $m \times d$ matrix is used as the test instance, and the jth decision/regression tree is used as the model to predict the value of the corresponding rating. The first step is to use the remaining $n-1$ items (except for the jth item) in order to create the reduced d-dimensional representation of the test instance according to Equation 2.17. Note that the jth set of eigenvectors are used for the projection and reduction process. This representation is then used with the corresponding decision or regression tree for the jth item to perform the prediction. It is noteworthy that this broader approach of combining dimensionality reduction with a classification model is not restricted to decision trees. It is relatively easy to use this approach in conjunction with virtually any classification model. Furthermore, dimensionality reduction methods are also used in isolation to predict ratings in recommender systems. Both these issues are discussed later in this chapter.

3.3 Rule-Based Collaborative Filtering

The relationship between *association rules* [23] and collaborative filtering is a natural one because the association rule problem was first proposed in the context of discovering relationships between supermarket data. Association rules are naturally defined over binary data, although the approach can be extended to categorical and numerical data by converting these data types to binary data. For the purpose of this discussion, we will assume the simplified case of unary data, which are common in supermarket transactions and in implicit feedback data sets.

Consider a transaction database $\mathcal{T} = \{T_1 \ldots T_m\}$, containing m transactions, which are defined on n items I. Therefore, I is the universal set of items, and each transaction T_i is a subset of the items in I. The key in association rule mining is to determine sets of items that are closely correlated in the transaction database. This is achieved with the notions of *support* and *confidence*. These measures quantify the relationships between sets of items.

Definition 3.3.1 (Support) *The support of an itemset $X \subseteq I$ is the fraction of transactions in $\mathcal{T}$, of which X is a subset.*

If the support of an itemset is at least equal to a predefined threshold s, then the itemset is said to be frequent. This threshold is referred to as the *minimum support*. These itemsets are referred to as *frequent itemsets* or *frequent patterns*. Frequent itemsets can provide important insights about correlations in customer buying behavior.

For example, consider the data set illustrated in Table 3.1. In this table, the rows correspond to customers and columns correspond to items. The 1s correspond to cases in which a particular customer has bought an item. Although this data set is unary, and the 0s correspond to missing values, a common practice in such implicit feedback data sets is to

Table 3.1: Example of market basket data

Item ⇒ Customer ⇓	*Bread*	*Butter*	*Milk*	*Fish*	*Beef*	*Ham*
Jack	1	1	1	0	0	0
Mary	0	1	1	0	1	0
Jane	1	1	0	0	0	0
Sayani	1	1	1	1	1	1
John	0	0	0	1	0	1
Tom	0	0	0	1	1	1
Peter	0	1	0	1	1	0

approximate missing values with 0s. It is evident that the columns of the table can be partitioned into two sets of closely related items. One of these sets is $\{Bread, Butter, Milk\}$, and the other set is $\{Fish, Beef, Ham\}$. These are the only itemsets with at least 3 items, which also have a support of at least 0.2. Therefore, both of these itemsets are frequent itemsets or frequent patterns. Finding such patterns with high support is useful to the merchant, because she can use them to make recommendations and other target marketing decisions. For example, it is reasonable to conclude that Mary is likely to eventually buy *Bread*, because she has already bought $\{Butter, Milk\}$. Similarly, John is likely to buy *Beef* because he has also bought $\{Fish, Ham\}$. Such inferences are very useful from the point of view of a recommendation system.

A further level of insight may be obtained in terms of the *directions* of these correlations by using the notion of *association rules* and *confidence*. An association rule is denoted in the form $X \Rightarrow Y$, where the "⇒" is intended to give a direction to the nature of the correlation between the set of items X and Y. For example, a rule such as $\{Butter, Milk\} \Rightarrow \{Bread\}$ would be very useful to recommend $Bread$ to Mary, because it is already known that she has bought $Milk$ and $Butter$. The strength of such a rule is measured by its *confidence*.

Definition 3.3.2 (Confidence) *The confidence of the rule $X \Rightarrow Y$ is the conditional probability that a transaction in $\mathcal{T}$ contains Y, given that it also contains X. Therefore, the confidence is obtained by dividing the support of $X \cup Y$ with the support of X.*

Note that the support of $X \cup Y$ will always be less than the support of X. This is because if a transaction contains $X \cup Y$, then it will always contain X. However, the reverse might not be true. Therefore, the confidence of a rule must always lie in the range $(0, 1)$. Higher values of the confidence are always indicative of greater strength of the rule. For example, if a rule $X \Rightarrow Y$ is true, then a merchant, who knows that a specific set of customers has bought the set of items X, can also target these customers with the set of items Y. An association rule is defined on the basis of a minimum support s and minimum confidence c:

Definition 3.3.3 (Association Rules) *A rule $X \Rightarrow Y$ is said to be an association rule at a minimum support of s and minimum confidence of c, if the following two conditions are satisfied:*

1. *The support of $X \cup Y$ is at least s.*

2. *The confidence of $X \Rightarrow Y$ is at least c.*

The process of finding association rules is a two-phase algorithm. In the first phase, all the itemsets that satisfy a minimum support threshold s are determined. From each of these itemsets Z, all possible 2-way partitions $(X, Z - X)$ are used to create a potential rule $X \Rightarrow Z - X$. Those rules satisfying the minimum confidence are retained. The first phase of determining the frequent itemsets is the computationally intensive one, especially when the underlying transaction database is very large. Numerous computationally efficient algorithms have been devoted to the problem of efficient frequent itemset discovery. The discussion of these algorithms is beyond the scope of this book, because it is a distinct field of data mining in its own right. Interested readers are referred to [23] for a detailed discussion of frequent pattern mining. In this book, we will show how to use these algorithms as tools for collaborative filtering.

3.3.1 Leveraging Association Rules for Collaborative Filtering

Association rules are particularly useful for performing recommendations in the context of *unary* ratings matrices. As discussed in Chapters 1 and 2, unary ratings matrices are created by customer activity (e.g., buying behavior), wherein there is a natural mechanism for the customer to specify a liking for an item, but no mechanism to specify a dislike. In these cases, the items bought by a customer are set to 1, whereas the missing items are set to 0 as an approximation. Setting missing values to 0 is not common for most types of ratings matrices because doing so would cause bias in the predictions. However, it is generally considered an acceptable practice in sparse unary matrices because the most common value of an attribute is usually 0 in these cases. As a result, the effect of bias is relatively small, and one can now treat the matrix as a binary data set.

The first step of rule-based collaborative filtering is to discover all the association rules at a pre-specified level of minimum support and minimum confidence. The minimum support and minimum confidence can be viewed as parameters, which are tuned[2] to maximize predictive accuracy. Only those rules are retained in which the consequent contains exactly one item. This set of rules is the model, which can be used to perform recommendations for specific users. Consider a given customer A to which it desired to recommend relevant items. The first step is to determine all association rules that have been *fired* by customer A. An association rule is said to be fired by a customer A, if the itemset in the antecedent of the rule is a subset of the items preferred by that customer. All of the fired rules are then sorted in order of reducing confidence. The first k items discovered in the consequents of these sorted rules are recommended as the top-k items to customer A. The approach described here is a simplification of the algorithm described in [524]. Numerous other variations of this basic approach are used in the recommender systems literature. For example, sparsity can be addressed using dimensionality reduction methods [524].

The aforementioned association rules are based on unary ratings matrices, which allow the ability to specify likes, but they do not allow the ability to specify dislikes. However, numeric ratings can be easily handled by using variations of this basic methodology. When the number of possible ratings is small, each value of the rating-item combination can be treated as a pseudo-item. An example of such as pseudo-item is $(Item = Bread, Rating = Dislike)$. A new set of transactions is created in terms of these pseudo-items. The rules are then constructed in terms of these pseudo-items by using the approach as discussed earlier.

[2]Parameter-tuning methods, such as hold-out and cross-validation, are discussed in Chapter 7.

Therefore, such rules could appear as follows:

$$(Item = Bread, Rating = Like) \Rightarrow (Item = Eggs, Rating = Like)$$

$$(Item = Bread, Rating = Like) \text{ AND } (Item = Fish, Rating = Dislike) \\ \Rightarrow (Item = Eggs, Rating = Dislike)$$

For a given customer, the set of fired rules is determined by identifying the rules whose antecedents contain a subset of the pseudo-items for that user. The rules are sorted in decreasing order of confidence. These sorted rules can be used to predict ratings for items by selecting the top-k pseudo-items in the consequents of these rules. An additional step that might be required in this case is to resolve the conflicts between the various rules because different pseudo-items in the rules fired by a customer might be conflicting. For example, the pseudo-items $(Item = Bread, Rating = Like)$ and $(Item = Bread, Rating = Dislike)$ are conflicting pseudo-items. Such conflicts can be resolved by finding a way of aggregating the ratings in the consequents in order to create the final sorted list of recommendations. It is also possible to numerically aggregate the ratings in the consequents by using a variety of heuristics. For example, one can first determine all the fired rules in which the consequents correspond to an item of interest. The item ratings in the consequents of these fired rules are voted on in a weighted way in order to make a prediction for that user-item combination. One can weight the ratings in the fired rules by the corresponding confidence in the averaging process. For example, if two rules contain the rating "*like*" in the consequent (for a particular item), with confidences of 0.9 and 0.8, respectively, then the total number of votes for "*like*" for that item is $0.9 + 0.8 = 1.7$. The votes can be used to predict an average value of the rating for that item. Such predicted values can be determined for all items in the consequents of the fired rules. The resulting values can be used to sort the items in reducing order of priority. The voting approach is more appropriate when the granularity of the rating scale is very limited (e.g., *like* or *dislike*). In the case of interval-based ratings with high granularity, it is possible to discretize the ratings into a smaller number of intervals, and then use the same approach discussed above. Other heuristic methods for aggregating the predictions from rule-based methods are discussed in [18]. In many cases, it has been shown that the most effective results are not necessarily obtained by using the same support level for each item. Rather, it is often desirable to make the support level specific to the item whose rating is being predicted [358, 359, 365].

3.3.2 Item-Wise Models versus User-Wise Models

The dual relationship between user-wise and item-wise models is a recurrent theme in collaborative filtering. The neighborhood models of Chapter 2 provide the most well-known example of this duality. In general, every user-wise model can be converted to an item-wise model by applying it to the transpose of the rating matrix, and vice versa. Minor adjustments might sometimes be required to account for the varying semantic interpretations in the two cases. For example, one uses the adjusted cosine for similarity computation in item-based neighborhood models rather than the Pearson correlation coefficient.

The aforementioned discussion focuses on item-wise models for rule-based collaborative filtering. It is also possible to create user-wise models. These methods leverage user associations rather than item associations [358, 359]. In these cases, the rules associate the user tastes with one another rather than associating the item tastes with one another. Therefore, one works with *pseudo-users* corresponding to user-rating combinations. Examples of such rules are as follows:

$$(User = Alice, Rating = Like) \Rightarrow (User = Bob, Rating = Disike)$$

$$(User = Alice, Rating = Like) \text{ AND } (User = Peter, Rating = Dislike) \\ \Rightarrow (User = John, Rating = Like)$$

The first rule implies that Bob is likely to dislike items that Alice likes. The second rule implies that John is likely to like items that Alice likes and Peter dislikes. Such rules can be mined by applying exactly the same approach as the previous case on the *transpose* of the transaction matrix constructed from the pseudo-users. In other words, each list of pseudo-users for an item is now treated as a "transaction." Association rules are mined from this database at the required level of minimum support and confidence. In order to predict the rating of a user-item combination, the pseudo-user-based "transaction" for the relevant item is determined. Rules are fired by this transaction when the antecedent of this rule contains a subset of the pseudo-users in the transaction. All the fired rules are determined. Among these fired rules, all those in which the consequents correspond to the user of interest are determined. The ratings in the consequents of the fired rules may be averaged or voted on to make a prediction. The averaging process can be weighted with the confidence of the corresponding rule to provide a more robust prediction. Thus, the user-based approach is exactly analogous to the item-based approach. It is noteworthy that the two ways of performing collaborative filtering with association rules share a complimentary relationship, which is reminiscent of user-based and item-based neighborhood algorithms.

The association rule approach is useful not only for collaborative filtering, but also for content-based recommender systems, in which customer profiles are matched to specific items. These rules are referred to as *profile association rules*, and are used popularly for profile-based recommendations. It has been shown in [31, 32] how an efficient interactive interface can be constructed for performing profile-based recommendations for a variety of different types of queries.

Association rule-based recommender systems can be viewed as generalizations of rule-based systems that are used commonly for the classification problem [18]. The main difference is that consequents of the generated rules in the classification problem always contain the class variable. However, in the case of recommender systems, the consequents of the generated rules might contain[3] any item. Furthermore, the heuristics for sorting the fired rules and combining the possibly conflicting results from the rules are also similar in collaborative filtering and classification. This natural relationship between these methods is a direct result of the relationship between the classification and collaborative filtering problems. The main distinction between the two cases is that there is no clear demarcation between the feature variables and the class variables in collaborative filtering. This is why any association rule can be generated, rather than simply rules that contain the class variable in the consequent.

A number of comparative studies have shown [358, 359] that association rule systems can provide accurate results in certain types of settings. This is particularly true of unary data, which is commonly encountered in Web recommender systems. Association rule-based systems have found significant applications in Web-based personalization and recommender systems [441, 552]. The approach is naturally suited to Web personalization systems because it is specifically designed for sparse transaction data, which is commonly encountered in Web click behavior. Such methods can even be extended to include temporal information by using *sequential pattern mining models* [23].

[3]In the case of user-based associations, the consequents might contain any *user*.

3.4 Naive Bayes Collaborative Filtering

In the following, we will assume that there are a small number of distinct ratings, each of which can be treated as a categorical value. Therefore, the orderings among the ratings will be ignored in the following discussion. For example, three ratings, such as *Like*, *Neutral*, and *Dislike*, will be treated as unordered discrete values. In the case where the number of distinct ratings is small, such an approximation can be reasonably used without significant loss in accuracy.

Assume that there are l distinct values of the ratings, which are denoted by $v_1 \ldots v_l$. As in the case of the other models discussed in this chapter, we assume that we have an $m \times n$ matrix R containing the ratings of m users for n items. The (u, j)th entry of the matrix is denoted by r_{uj}.

The naive Bayes model is a generative model, which is commonly used for classification. One can treat the items as features and users as instances in order to infer the missing entries with a classification model. The main challenge in using this approach for collaborative filtering is that any feature (item) can be the target class in collaborative filtering, and one also has to work with incomplete feature variables. These differences can be handled with minor modifications to the basic methodology of the naive Bayes model.

Consider the uth user, who has specified ratings for the set of items I_u. In other words, if the uth row has specified ratings for the first, third, and fifth columns, then we have $I_i = \{1, 3, 5\}$. Consider the case where the Bayes classifier needs to predict the unobserved rating r_{uj} of user u for item j. Note that r_{uj} can take on any one of the discrete possibilities in $\{v_1 \ldots v_l\}$. Therefore, we would like to determine the probability that r_{uj} takes on any of these values *conditional on the observed ratings in* I_u. Therefore, for each value of $s \in \{1 \ldots l\}$, we would like to determine the probability $P(r_{uj} = v_s | \textit{Observed ratings in } I_u)$. This expression appears in the form $P(A|B)$, where A and B are events corresponding to the value of r_{uj}, and the values of the observed ratings in I_u, respectively. The expression can be simplified using the well-known Bayes rule in probability theory:

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)} \tag{3.3}$$

Therefore, for each value of $s \in \{1 \ldots l\}$, we have the following:

$$P(r_{uj} = v_s | \textit{Observed ratings in } I_u) = \frac{P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)}{P(\textit{Observed ratings in } I_u)} \tag{3.4}$$

We need to determine the value of s in the aforementioned expression for which the value of $P(r_{uj} = v_s | \textit{Observed ratings in } I_u)$ on the left-hand side is as large as possible. It is noteworthy that the denominator on the right-hand side of Equation 3.4 is independent of the value of s. Therefore, in order to determine the value of s at which the right-hand side takes on the maximum value, one can ignore the denominator and express the aforementioned equation in terms of a constant of proportionality:

$$P(r_{uj} = v_s | \textit{Observed ratings in } I_u) \propto P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s) \tag{3.5}$$

If desired, the constant of proportionality can be derived by ensuring that all the resulting probability values $P(r_{uj} = v_s | \textit{Observed ratings in } I_u)$ for $s \in \{1 \ldots l\}$ sum to 1. A key observation is that all the expressions on the right-hand side of Equation 3.5 can be estimated easily in a data-driven manner. The value of $P(r_{uj} = v_s)$, which is also referred to as the *prior probability* of rating r_{uj}, is estimated to the fraction of the users that have specified the rating v_s for the jth item. Note that the fraction is computed only out of those users that have rated item j, and the other users are ignored. The expression $P(\textit{Observed ratings in } I_u | r_{uj} = v_s)$ is estimated with the use of the *naive assumption*. The naive assumption is based on *conditional independence* between the ratings. The conditional independence assumption says that the ratings of user u for various items in I_u are independent of one another, *conditional* of the fact that the value of r_{uj} was observed to be v_s. This condition may be mathematically expressed as follows:

$$P(\textit{Observed ratings in } I_u | r_{uj} = v_s) = \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s) \tag{3.6}$$

The value of $P(r_{uk}|r_{uj} = v_s)$ is estimated as the fraction of users that have specified the rating of r_{uk} for the kth item, given that they have specified the rating of their jth item to v_s. By plugging in the estimation of the prior probability $P(r_{uj} = v_s)$ and that of Equation 3.6 into Equation 3.5, it is possible to obtain an estimate of the posterior probability of the rating of item j for user u as follows:

$$P(r_{uj} = v_s | \textit{Observed ratings in } I_u) \propto P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s) \tag{3.7}$$

This estimate of the posterior probability of the rating r_{uj} can be used to estimate its value in one of the following two ways:

1. By computing each of the expressions on the right-hand side of Equation 3.7 for each $s \in \{1 \ldots l\}$, and determining the value of s at which it is the largest, one can determine the most likely value $\hat{r}_{uj}$ of the missing rating r_{uj}. In other words, we have:

$$\begin{aligned}\hat{r}_{uj} &= \text{argmax}_{v_s} P(r_{uj} = v_s | \textit{Observed ratings in } I_u) \\ &= \text{argmax}_{v_s} P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)\end{aligned}$$

   Such an approach, however, treats a rating purely as a categorical value and ignores all ordering among the various ratings. When the number of possible ratings is small, this is a reasonable approach to use.

2. Rather than determining the rating that takes on the maximum probability, one can estimate the predicted value as the weighted average of all the ratings, where the weight of a rating is its probability. In other words, the weight of the rating v_s is proportional to the value of $P(r_{uj} = v_s|\textit{Observed ratings in } I_u)$, as computed in Equation 3.7. Note that the constant of proportionality in the equation is irrelevant for computing the weighted average. Therefore, the estimated value $\hat{r}_{uj}$ of the missing rating r_{uj} in the matrix R is as follows:

$$\begin{aligned}\hat{r}_{uj} &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s | \textit{Observed ratings in } I_u)}{\sum_{s=1}^{l} P(r_{uj} = v_s | \textit{Observed ratings in } I_u)} \\ &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)}{\sum_{s=1}^{l} P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)} \\ &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)}{\sum_{s=1}^{l} P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)}\end{aligned}$$

   This approach is preferable when the granularity of the ratings distribution is greater.

For a given user u, all her unobserved ratings are estimated using this approach. The items with the top-k estimated values of the ratings are reported.

It is noteworthy that this approach computes the conditional probability of a rating, based on the ratings of the other *items* (or dimensions). Therefore, this approach is an *item*-based Bayes approach. This approach is a straightforward adaptation of traditional classification methods, except that the predicted (class) dimension is fixed in traditional classification, whereas the predicted dimension varies in collaborative filtering. This difference occurs because collaborative filtering is a generalization of classification (cf. Figure 3.1). In the particular case of collaborative filtering, it is also possible to compute the probability of a rating based on the ratings of the other *users* for the same item (see Exercise 4). Such an approach can be viewed as a *user*-based Bayes approach. It is even possible to combine the predictions from the user-based and item-based Bayes methods. In virtually all forms of collaborative filtering, such as neighborhood-based and rule-based methods, it is possible to provide a solution from the user-based perspective, the item-based perspective, or a combination of the two methods.

3.4.1 Handling Overfitting

A problem arises when the underlying ratings matrix is sparse and the number of observed ratings is small. In such cases, the data-driven estimations may not remain robust. For example, the estimation of the prior probability $P(r_{uj} = v_s)$ is unlikely to be robust if a small number of users have specified ratings for the jth item. For example, if no user has specified a rating for the jth item, the estimation is of the form $0/0$, which is indeterminate. Furthermore, the estimation of each value $P(r_{uk}|r_{uj} = v_s)$ on the right-hand side of Equation 3.6 is likely to be even less robust than the estimation of the prior probability. This is because only a small portion of the ratings matrix will be conditional on the event $r_{uj} = v_s$. In this case, the portion of the ratings matrix that needs to be analyzed is only those users that have specified the rating v_s for item j. If the number of such users is small, the estimation will be inaccurate and the multiplicative terms in Equation 3.6 will produce a large error. For example, for any value of $k \in I_u$, if no user has specified the rating r_{uk} in cases where the rating of the jth item is set to v_s, the entire expression of Equation 3.6 will be set to 0 because of its multiplicative nature. This is, of course, an erroneous and overfitting result, which is obtained because of the estimation of the model parameters from a small amount of data.

In order to handle this problem, the method of Laplacian smoothing is commonly used. For example, let $q_1 \ldots q_l$ be the number of users that have respectively specified the ratings $v_1 \ldots v_l$ for the jth item. Then, instead of estimating $P(r_{uj} = v_s)$ in a straightforward way to $q_s / \sum_{t=1}^{l} q_t$, it is smoothed with a Laplacian smoothing parameter α:

$$P(r_{uj} = v_s) = \frac{q_s + \alpha}{\sum_{t=1}^{l} q_t + l \cdot \alpha} \tag{3.8}$$

Note that if no ratings are specified for the jth item, then such an approach will set the prior probability of each possible rating to $1/l$. The value of α controls the level of smoothing. Larger values of α will lead to more smoothing, but the results will become insensitive to the underlying data. An exactly similar approach can be used to smooth the estimation of $P(r_{uk}|r_{uj} = v_s)$, by adding α and $l \cdot \alpha$ to the numerator and denominator, respectively.

Table 3.2: Illustration of the Bayes method with a binary ratings matrix

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6
1	1	-1	1	-1	1	-1
2	1	1	?	-1	-1	-1
3	?	1	1	-1	-1	?
4	-1	-1	-1	1	1	1
5	-1	?	-1	1	1	1

3.4.2 Example of the Bayes Method with Binary Ratings

In this section, we will illustrate the Bayes method with a binary ratings matrix on 5 users and 6 items. The ratings are drawn from $\{v_1, v_2\} = \{-1, 1\}$. This matrix is shown in Table 3.2. For ease in discussion, we will not use Laplacian smoothing although it is essential to do so in practice. Consider the case in which we wish to predict the ratings of the two unspecified items of user 3. Therefore, we need to compute the probabilities of the unspecified ratings r_{31} and r_{36} taking on each of the values from $\{-1, 1\}$, conditional on the observed values of the other ratings of user 3. By using Equation 3.7, we obtain the following posterior probability for the rating of item 1 by user 3:

$$P(r_{31} = 1|r_{32}, r_{33}, r_{34}, r_{35}) \propto P(r_{31} = 1) \cdot P(r_{32} = 1|r_{31} = 1) \cdot P(r_{33} = 1|r_{31} = 1) \cdot \\ \cdot P(r_{34} = -1|r_{31} = 1) \cdot P(r_{35} = -1|r_{31} = 1)$$

The values of the individual terms on the right-hand side of the aforementioned equation are estimated using the data in Table 3.2 as follows:

$$\begin{aligned} &P(r_{31} = 1) = 2/4 = 0.5 \\ &P(r_{32} = 1|r_{31} = 1) = 1/2 = 0.5 \\ &P(r_{33} = 1|r_{31} = 1) = 1/1 = 1 \\ &P(r_{34} = -1|r_{31} = 1) = 2/2 = 1 \\ &P(r_{35} = -1|r_{31} = 1) = 1/2 = 0.5 \end{aligned}$$

Upon substituting these values in the aforementioned equation, we obtain the following:

$$P(r_{31} = 1|r_{32}, r_{33}, r_{34}, r_{35}) \propto (0.5)(0.5)(1)(1)(0.5) = 0.125$$

Upon performing the same steps for the probability of r_{31} taking on the value of -1, we obtain:

$$P(r_{31} = -1|r_{32}, r_{33}, r_{34}, r_{35}) \propto (0.5)\left(\frac{0}{1}\right)\left(\frac{0}{2}\right)\left(\frac{0}{2}\right)\left(\frac{0}{2}\right) = 0$$

Therefore, the rating r_{31} has a higher probability of taking on the value of 1, as compared to -1, and its predicted value is set to 1. One can use a similar argument to show that the predicted value of the rating r_{36} is -1. Therefore, in a top-1 recommendation scenario, item 1 should be prioritized over item 6 in a recommendation to user 3.

3.5 Using an Arbitrary Classification Model as a Black-Box

Many other classification (or regression modeling) methods can be extended to the case of collaborative filtering. The main challenge in these methods is the incomplete nature of the underlying data. In the case of some classifiers, it is more difficult to adjust the model to handle the case of missing attribute values. An exception is the case of *unary* data, in which missing values are often estimated to be 0, and the specified entries are set to 1. Therefore, the underlying matrix resembles sparse binary data of high dimensionality. In such cases, the data can be treated as a complete data set and any classifiers that are designed for sparse and high dimensional data can be used. Fortunately, many forms of data, including customer transaction data, Web click data, or other activity data, can be formulated as a unary matrix. It is noteworthy that text data is also sparse and high-dimensional; as a result, many of the classification algorithms used in text mining can be directly adapted to these data sets. In fact, it has been shown in [669] that one can directly leverage the success of support vector machines in text data to (unary) collaborative filtering, albeit with a squared form of the loss function. The squared form of the loss function makes the model more akin to regularized linear regression. It has also been suggested in [669] that the use of rare class learning methods can be effective in collaborative filtering due to the imbalanced nature of the class distribution. For example, one might use different loss functions for the majority and minority classes while adapting the support vector machine to the collaborative filtering scenario. Numerous ad hoc methods have also been proposed to extend various classification and regression methods to collaborative filtering. For example, smoothing support vector machines [638] have been used to estimate the missing values in the user-item matrix in an iterative way.

For cases in which the ratings matrix is not unary, it is no longer possible to fill in the missing entries of the matrix with 0s without causing significant bias. This issue is discussed in detail in section 2.5 of Chapter 2. Nevertheless, as discussed in the same section, several dimensionality reduction methods can be used to create a low-dimensional representation of the data, which is fully specified. In such cases, any known classification method can be used effectively by treating the low-dimensional representation as the feature variables of the training data. Any column that needs to be completed is treated as the class variable. The main problem with this approach is a loss of interpretability in the classification process. When the reduced representation represents a linear combination of the original columns, it is difficult to provide any type of explanation of the predictions.

In order to work in the original feature space, it is possible to use classification methods as meta-algorithms in conjunction with iterative methods. In other words, an off-the-shelf classification algorithm is used as a *black-box* to predict the ratings of one of the items with the ratings of other items. How does one overcome the problem that the training columns haven been incompletely specified? The trick is to iteratively fill in the missing values of the training columns with successive refinement. This successive refinement is achieved with the use of our black-box, which is an off-the-shelf classification (or regression modeling) algorithm.

Consider an arbitrary classification/regression modeling algorithm $\mathcal{A}$, which is designed to work with a completely specified matrix. The first step is to initialize the missing entries in the matrix with row averages, column averages, or with any simple collaborative filtering algorithm. For example, one might use a simple user-based algorithm for the initialization process. As an optional enhancement, one might center each row of the ratings matrix as a preprocessing step to remove user bias. In this case, the bias of each user needs to be added back to the predicted values in a post-processing phase. Removing user bias during pre-processing often makes[4] the approach more robust. If the user bias is removed, then the missing entries are always filled in with row averages, which are 0.

These simple initializations and bias removal methods will still lead to prediction bias, when one attempts to use the artificially filled in values as training data. Then, the bias in the predicted entries can be iteratively reduced by using the following two-step iterative approach:

1. **(Iterative step 1):** Use algorithm $\mathcal{A}$ to estimate the missing entries of each column by setting it as the target variable and the remaining columns as the feature variables. For the remaining columns, use the current set of filled in values to create a complete matrix of feature variables. The observed ratings in the target column are used for training, and the missing ratings are predicted.

2. **(Iterative step 2):** Update all the missing entries based on the prediction of algorithm $\mathcal{A}$ on each target column.

These two steps are iteratively executed to convergence. The approach can be sensitive to the quality of the initialization and the algorithm $\mathcal{A}$. Nevertheless, the merit of the approach is that it is a simple method that can easily be implemented with any off-the-shelf classification or regression model. Numerical ratings can also be handled with a linear regression model. The work in [571] uses a similar approach in which the ratings matrix is imputed with artificial entries predicted by an ensemble of different classifiers.

3.5.1 Example: Using a Neural Network as a Black-Box

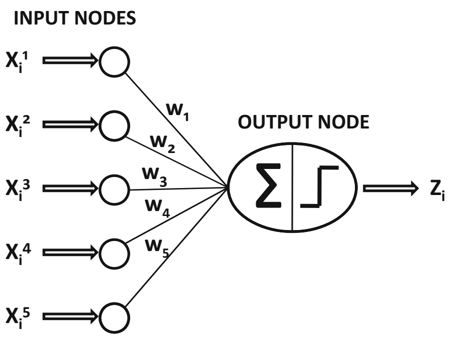
In this section, we will provide a simple example of the aforementioned approach, when neural networks are used as black-boxes to implement the approach. For the purpose of the following discussion, we will assume that the reader is already familiar with the basics of neural networks [87]. Nevertheless, we will introduce them very briefly to ensure continuity of discussion.

Neural networks simulate the human brain with the use of *neurons*, which are connected to one another via *synaptic connections*. In biological systems, learning is performed by changing the strength of synaptic connections in response to external stimuli. In artificial neural networks, the basic computation unit is also referred to as a neuron, and the strengths of the synaptic connections correspond to *weights*. These weights define the parameters

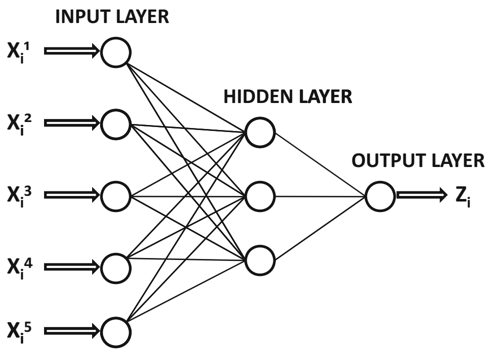
[4]It is also possible to use more sophisticated ways of removing bias for better performance. For example, the bias B_{ij}, which is specific to user i and item j, can be computed using the approach discussed in section 3.7.1. This bias is subtracted from observed entries and all missing entries are initialized to 0s during pre-processing. After computing the predictions, the biases B_{ij} are added back to the predicted values during postprocessing.

used by the learning algorithm. The most basic architecture of the neural network is the *perceptron*, which contains a set of input nodes and an output node. An example of a perceptron is shown in Figure 3.3(a). For a data set containing d different dimensions, there are d different input units. The output node is associated with a set of weights W, which is used to compute a function $f(\cdot)$ of the d inputs. A typical example of such a function is the signed linear function, which would work well for binary output:

$$z_i = \text{sign}\{\overline{W} \cdot \overline{X_i} + b\} \tag{3.9}$$

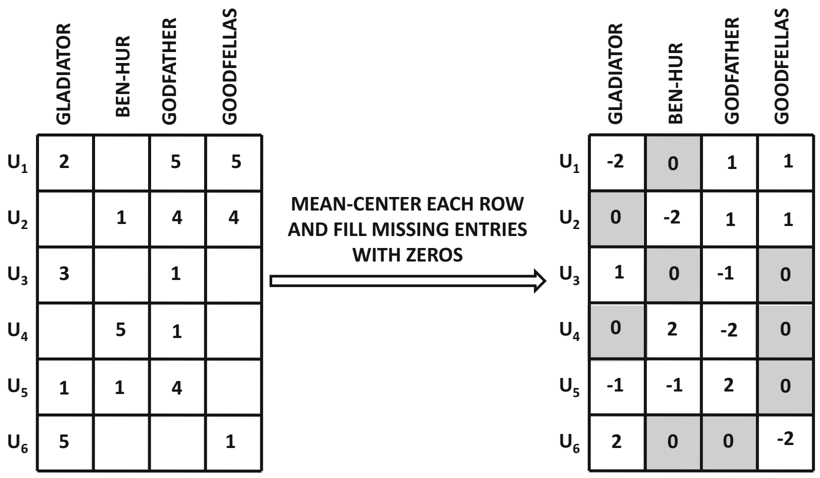


(a) Perceptron



(b) Multilayer

Figure 3.3: Single and multilayer neural networks



	GLADIATOR	BEN-HUR	GODFATHER	GOODFELLAS
U_1	2		5	5
U_2		1	4	4
U_3	3		1	
U_4		5	1	
U_5	1	1	4	
U_6	5			1

MEAN-CENTER EACH ROW AND FILL MISSING ENTRIES WITH ZEROS

	GLADIATOR	BEN-HUR	GODFATHER	GOODFELLAS
U_1	-2	0	1	1
U_2	0	-2	1	1
U_3	1	0	-1	0
U_4	0	2	-2	0
U_5	-1	-1	2	0
U_6	2	0	0	-2

Figure 3.4: Pre-processing the ratings matrix. Shaded entries are iteratively updated.

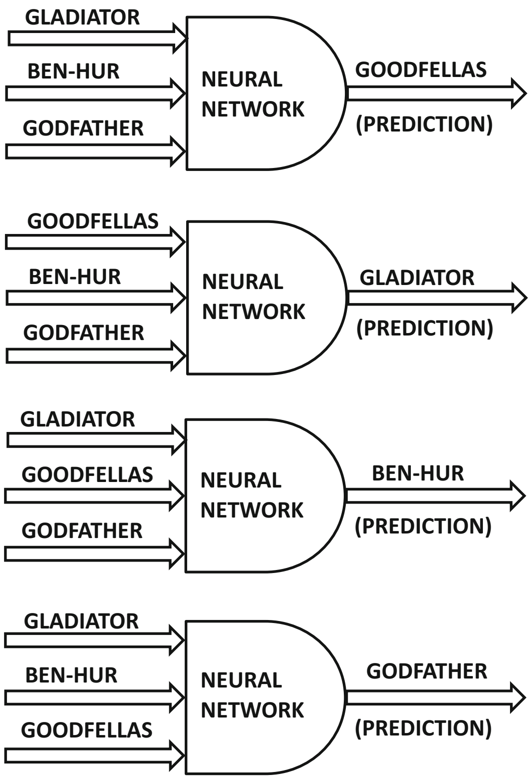


Figure 3.5: Neural networks for predicting and updating missing entries. Shaded entries of Figure 3.4 are iteratively updated by the neural networks.

Here, $\overline{X_i}$ is a d-dimensional row vector defining the d inputs of the ith training instance, and $\overline{W}$ is the coefficient vector. In the context of collaborative filtering, the d inputs correspond to the $(n-1)$ items, which are used to predict the rating of the remaining item. Assume that the label of the ith instance is y_i. In the context of collaborative filtering, y_i represents the observed ratings of the items being predicted. The parameter b denotes the bias. One can already notice the similarity of this approach with linear regression although the prediction function is slightly different. The value of z_i is the predicted output, and the error $(z_i - y_i)^2$ of this predicted output is used to update the weights in $\overline{W}$ in a manner similar to linear regression. This update is similar to the updates in gradient descent, which are made for least-squares optimization. In the case of neural networks, the update function is as follows:

$$\overline{W}^{t+1} = \overline{W}^{t} + \alpha(y_i - z_i)\overline{X_i} \tag{3.10}$$

Here, $\alpha > 0$ denotes the learning rate and $\overline{W}^t$ is the value of the weight vector in the tth iteration. It is not difficult to show that the incremental update vector, is the negative gradient of $(y_i - z_i)^2$ with respect to $\overline{W}$. We iterate through all the observed ratings in the item being predicted in order to make these updates. Since it was assumed that y_i is binary, this approach is designed for binary ratings matrices. One can also design neural networks in which the output need not be binary, and the prediction function need not be linear.

In general, a neural network can have multiple layers, and the intermediate nodes can compute nonlinear functions. An example of such a multi-layer neural network is illustrated in Figure 3.3(b). Of course, such a network would have a larger number of learning parameters. The corresponding learning algorithm is referred to as the *back-propagation algorithm* [87]. The main advantage of neural networks is that the multi-layer architecture provides the ability to compute complex nonlinear functions that are not easily computable with other classification methods. Therefore, neural networks are also referred to as *universal function approximators*. For noisy data like ratings matrices, regularization can be used to reduce the impact of noise.

Consider a ratings matrix with four items, illustrated on the left-hand side of Figure 3.4. In this example, the items correspond to movies. The first step is to mean-center each row, in order to remove user biases. The resulting mean-centered matrix is shown on the right-hand side of Figure 3.4. Note that the missing values are replaced with the corresponding row average, which is 0 after mean-centering. Since there are four items, there are four possible neural network models, whereby each model is constructed by using the ratings input of the other three items as training columns, and the fourth as the test column. These four neural networks are shown in Figure 3.5. The completed matrix of Figure 3.4 is used to train each of these neural networks in the first iteration. For each column of the ratings matrix, the relevant neural network in Figure 3.5 is used for prediction purposes. The resulting predictions made by the neural networks are then used to create a new matrix in which the missing entries are updated with the predicted values. In other words, the neural networks are used only to update the values in the shaded entries of Figure 3.4 with the use of an off-the-shelf neural network training and prediction procedure. After the update, the shaded entries of Figure 3.4 will no longer be zeros. This matrix is now used to predict the entries for the next iteration. This approach is repeated iteratively until convergence. Note that each iteration requires the application of n training procedures, where n is the number of items. However, one does not need to learn the parameters of the neural networks from scratch in each iteration. The parameters from the previous iteration can be used as a good starting point. It is important to use regularization because of the high dimensionality of the underlying data [220].

This model can be considered an *item-wise* model, in which the inputs represent the ratings of various items. It is also possible to create a *user-wise* model [679], in which the inputs correspond to the ratings of various users. The main challenge with such an approach is that the number of inputs to the neural network becomes very large. Therefore, it is recommended in [679] that not all users should be used as input nodes. Rather, only users who have rated at least a minimum threshold number of items are used. Furthermore, the users should not all be highly similar to one another. Therefore, heuristics are proposed in [679] to preselect mutually diverse users in the initial phase. This approach can be considered a type of feature selection for neural networks, and it can also be used in the item-wise model.

3.6 Latent Factor Models

In section 2.5 of Chapter 2, we discussed some dimensionality reduction methods to create a new fully specified representation of an incomplete data set. In Chapter 2, a number of heuristic methods were discussed, which create a full dimensional representation for enabling the use of neighborhood algorithms [525]. Such data reduction techniques are also used to enable other model-based methods, which use classification algorithms as a subroutine. Therefore, in all the methods previously discussed, dimensionality reduction only plays an *enabling* role of creating a more convenient data representation for other model-based methods. In this chapter, more sophisticated methods will be discussed, because the goal is to use dimensionality reduction methods to directly estimate the data matrix in one shot.

The earliest discussions on the use of latent factor models as a direct method for matrix completion may be found in [24, 525]. The basic idea is to exploit the fact that significant portions of the rows and columns of data matrices are highly correlated. As a result, the data has built-in redundancies and the resulting data matrix is often approximated quite well by a *low-rank* matrix. Because of the inherent redundancies in the data, the *fully specified* low-rank approximation can be determined even with a small subset of the entries in the original matrix. This fully-specified low rank approximation often provides a robust estimation of the missing entries. The approach in [24] combines the expectation-maximization (EM) technique with dimensionality reduction to reconstruct the entries of the incomplete data matrix.

Latent factor models are considered to be state-of-the-art in recommender systems. These models leverage well-known dimensionality reduction methods to fill in the missing entries. Dimensionality reduction methods are used commonly in other areas of data analytics to represent the underlying data in a small number of dimensions. The basic idea of dimensionality reduction methods is to rotate the axis system, so that pairwise correlations between dimensions are removed. The key idea in dimensionality reduction methods is that the reduced, rotated, and completely specified representation can be robustly estimated from an incomplete data matrix. Once the completely specified representation has been obtained, one can rotate it back to the original axis system in order to obtain the fully specified representation [24]. Under the covers, dimensionality reduction methods leverage the row and column correlations to create the fully specified and reduced representation. The use of such correlations is, after all, fundamental to all collaborative filtering methods, whether they are neighborhood methods or model-based methods. For example, user-based neighborhood methods leverage user-wise correlations, whereas item-based neighborhood methods leverage item-wise correlations. Matrix factorization methods provide a neat way to leverage all row and column correlations in one shot to estimate the entire data matrix. This sophistication of the approach is one of the reasons that latent factor models have become the state-of-the-art in collaborative filtering. In order to understand why latent factor models are effective, we will provide two pieces of intuition, one of which is geometric and the other elucidates the semantic interpretation directly. Both these intuitions show how data redundancies in highly correlated data can be exploited to create a low-rank approximation.

3.6.1 Geometric Intuition for Latent Factor Models

We will first provide a geometric intuition for latent factor models, based on a discussion provided in [24]. In order to understand the intuition of how the notions of low-rank, redundancy, and correlation are related, consider a ratings matrix with three items, in which all three items are positively correlated. Assume a movie rating scenario, in which the three items correspond to *Nero*, *Gladiator*, and *Spartacus*. For ease of discussion, assume that the ratings are continuous values, which lie in the range $[-1, 1]$. If the ratings are positively correlated, then the 3-dimensional scatterplot of the ratings might be roughly arranged along a 1-dimensional line, as shown in Figure 3.6. Since the data is mostly arranged along a 1-dimensional line, it means that the original data matrix has a rank of approximately 1 after removing the noisy variations. For example, the rank-1 approximation of Figure 3.6 would be the 1-dimensional line (or *latent vector*) that passes through the center of the data and aligned with the elongated data distribution. Note that dimensionality reduction methods such as Principal Component Analysis (PCA) and (mean-centered) Singular Value Decomposition (SVD) typically represent the projection of the data along this line as an approximation. When the $m \times n$ ratings matrix has a rank of $p \ll \min\{m, n\}$ (after

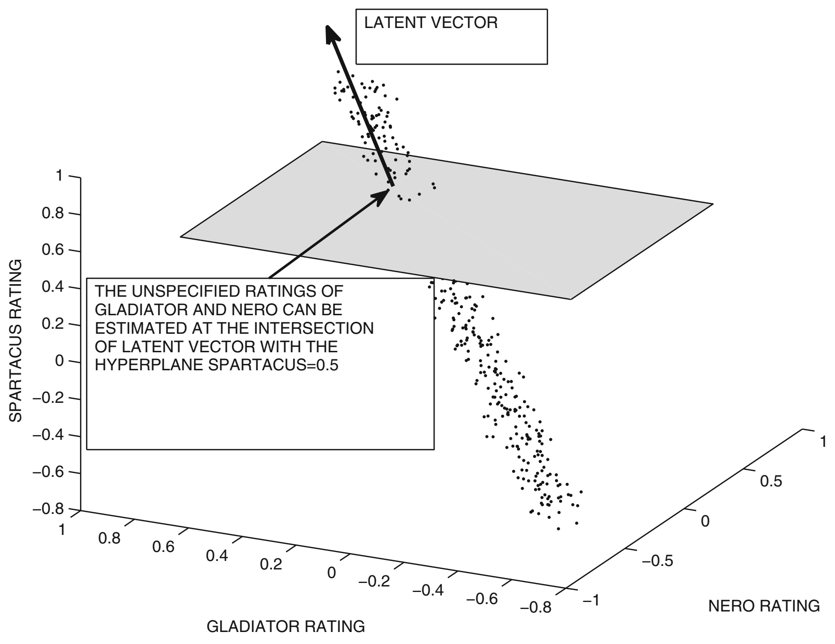


Figure 3.6: Leveraging correlation-based redundancies in missing data estimation for a user whose only specified rating is a value of 0.5 for the movie *Spartacus*

removing noisy variations), the data can be approximately represented on a p-dimensional hyperplane. In such cases, the missing ratings of a user can often be robustly estimated with as few as p specified entries as long as the p-dimensional hyperplane is known. For example, in the case of Figure 3.6, only one rating needs to be specified in order to determine the other two ratings, because the rank of the ratings matrix is only 1 after noise removal. For example, if the rating of *Spartacus* is fixed at 0.5, then the ratings of *Nero* and *Gladiator* can be estimated[5] as the intersection of the 1-dimensional latent vector with the axis-parallel hyperplane, in which the rating of *Spartacus* is fixed to 0.5. This hyperplane is illustrated in Figure 3.6. Therefore, dimensionality reduction methods such as SVD leverage the inter-attribute correlations and redundancies in order to infer unspecified entries.

In this case, it was assumed that a specified data matrix was available to estimate the relevant latent vector. In practice, the data matrix does not need to be fully specified in order to estimate the *dominant* latent vectors, such as the line aligned with the elongated shape of the data distribution in Figure 3.6. The ability to estimate such latent vectors with missing data is the key to the success of the latent factor approach. The basic idea in all these methods is to find a set of latent vectors, in which the average squared distance of the data points (representing individual user ratings) from the hyperplane defined by these latent vectors is as small as possible. Therefore, *we must use a partially specified data set to recover the low-dimensional hyperplane on which the data approximately lies.* By doing so, we can implicitly capture the underlying redundancies in the correlation structure of the data and reconstruct all the missing values in one shot. It is the knowledge of these implicit redundancies that helps us to predict the missing entries in the matrix. It is noteworthy that if the data does not have any correlations or redundancies, then a latent factor model will simply not work.

[5] A detailed description of the method used for performing this estimation in various scenarios is discussed in section 3.6.5.3.

3.6.2 Low-Rank Intuition for Latent Factor Models

The *geometric* intuition of the previous section is helpful in understanding the impact of latent vectors when they are mutually orthogonal. However, latent vectors are not always mutually orthogonal. In such cases, it is helpful to obtain some intuition from linear algebra. One way of understanding the effectiveness of latent factor models is by examining the role that *factorization* plays in such matrices. Factorization is, in fact, a more general way of approximating a matrix when it is prone to dimensionality reduction because of correlations between columns (or rows). Most dimensionality reduction methods can also be expressed as matrix factorizations.

First, let us consider the simple case in which all entries in the ratings matrix R are observed. The key idea is that any $m \times n$ matrix R of rank $k \ll \min\{m, n\}$ can always be expressed in the following product form of rank-k factors:

$$R = UV^T \tag{3.11}$$

Here, U is an $m \times k$ matrix, and V is an $n \times k$ matrix. Note that the rank of both the row space[6] and the column space of R is k. Each column of U be viewed as one of the k basis vectors of the k-dimensional column space of R, and the jth row of V contains the corresponding coefficients to combine these basis vectors into the jth column of R. Alternatively, one can view the columns of V as the basis vectors of the row space of R, and the rows of U as the corresponding coefficients. The ability to factorize any rank-k matrix in this form is a fundamental fact of linear algebra [568], and there are an infinite number of such factorizations corresponding to various sets of basis vectors. SVD is one example of such a factorization in which the basis vectors represented by the columns of U (and the columns of V) are orthogonal to one another.

Even when the matrix R has rank larger than k, it can often be *approximately* expressed as the product of rank-k factors:

$$R \approx UV^T \tag{3.12}$$

As before, U is an $m \times k$ matrix, and V is an $n \times k$ matrix. The error of this approximation is equal to $||R - UV^T||^2$, where $|| \cdot ||^2$ represents the sum of the squares of the entries in the resulting *residual matrix* $(R - UV^T)$. This quantity is also referred to as the (squared) *Frobenius norm* of the residual matrix. The residual matrix typically represents the noise in the underlying ratings matrix, which cannot be modeled by the low-rank factors. For simplicity in discussion, let us consider the straightforward case in which R is fully observed. We will first examine the intuition behind the factorization process, and then we will discuss the implication of this intuition in the context of matrices with missing entries.

What is the implication of the factorization process, and its impact on a matrix with highly correlated rows and columns? In order to understand this point, consider the ratings matrix illustrated in Figure 3.7. In this figure, a 7×6 ratings matrix with 7 users and 6 items is illustrated. All ratings are drawn from $\{1, -1, 0\}$, which correspond to like, dislike, and neutrality. The items are movies, and they belong to the romance and history genres, respectively. One of the movies, titled *Cleopatra*, belongs to both genres. Because of the nature of the genres of the underlying movies, users also show clear trends in their ratings. For example, users 1 to 3 typically like historical movies, but they are neutral to the romance genre. User 4 likes movies of both genres. Users 5 to 7 like movies belonging to the romance genre, but they explicitly dislike historical movies. Note that this matrix has a significant

[6]The row space of a matrix is defined by all possible linear combinations of the rows of the matrix. The column space of a matrix is defined by all possible linear combinations of the columns of the matrix.

number of correlations among the users and items, although the ratings of movies belonging to the two distinct genres seem to be relatively independent. As a result, this matrix can be approximately factorized into rank-2 factors, as shown in Figure 3.7(a). The matrix U is a 7×2 matrix, which shows the proclivity of users towards the two genres, whereas the matrix V is a 6×2 matrix, which shows the membership of the movies in the two genres. In other words, the matrix U provides the basis for the column space, whereas the matrix V provides the basis for the row space. For example, the matrix U shows that user 1 likes history movies, whereas user 4 likes both genres. A similar inference can be made using the rows of V. The columns of V correspond to the latent vectors, such as those shown in Figure 3.6. Unlike SVD, however, the latent vectors in this case are not mutually orthogonal.

The corresponding residual matrix for the factorization is shown in Figure 3.7(b). The residual matrix typically corresponds to the ratings of users for *Cleopatra*, which do not follow the set pattern. It needs to be pointed out that in real-world applications, the matrix entries in the factors are typically real numbers (rather than integral). An example with integral factors is shown here for visual simplicity. Furthermore, a neat semantic interpretation of the factors in terms of genres or categories is sometimes not possible, especially when the factors contain both positive and negative values. For example, if we multiply both U and V with -1 in Figure 3.7, the factorization is still valid, but the interpretation becomes more difficult. Nevertheless, the k columns of U and V do represent key correlations among the users and items, respectively, and they can be viewed abstractly as *latent concepts*, whether or not they are semantically interpretable. In some forms of factorization, such as non-negative matrix factorization, the interpretability of these concepts is retained to a greater degree.

In this example, the matrix R was fully specified, and therefore the factorization is not particularly helpful from the perspective of missing value estimation. The key usefulness of the approach arises when the matrix R is not fully specified, but one can still robustly estimate *all* entries of the latent factors U and V, respectively. *For low values of the rank*, this is still possible from sparsely specified data. This is because one does not need too many observed entries to estimate the latent factors from inherently redundant data. Once the matrices U and V have been estimated, the entire ratings matrix can be estimated as UV^T in one shot, which provides all the missing ratings.

3.6.3 Basic Matrix Factorization Principles

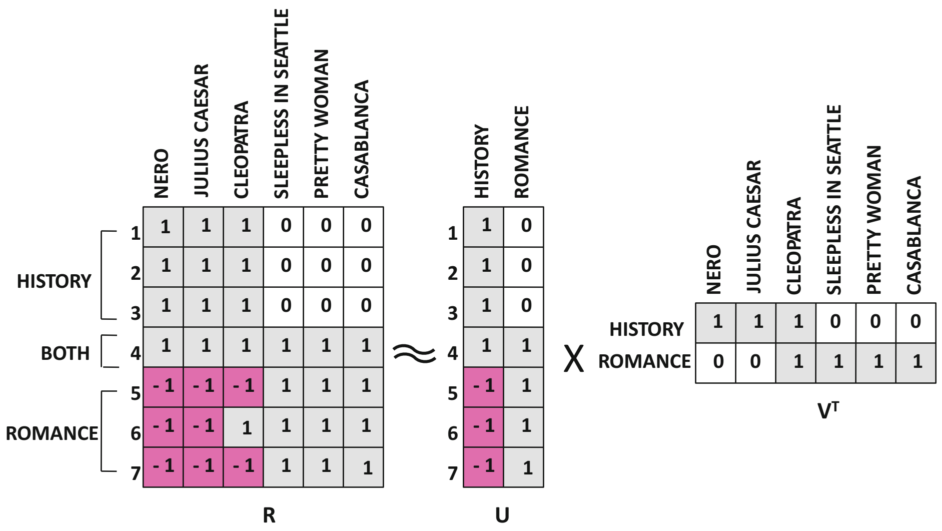
In the basic matrix factorization model, the $m \times n$ ratings matrix R is approximately factorized into an $m \times k$ matrix U and an $n \times k$ matrix V, as follows:

$$R \approx UV^T \tag{3.13}$$

Each column of U (or V) is referred to as a latent *vector* or latent *component*, whereas each row of U (or V) is referred to as a latent *factor*. The ith row $\overline{u_i}$ of U is referred to as a *user factor*, and it contains k entries corresponding to the affinity of user i towards the k concepts in the ratings matrix. For example, in the case of Figure 3.7, $\overline{u_i}$ is a 2-dimensional vector containing the affinity of user i towards the history and romance genres in the ratings matrix. Similarly, each row $\overline{v_i}$ of V is referred to as an *item factor*, and it represents the affinity of the ith item towards these k concepts. In Figure 3.7, the item factor contains the affinity of the item towards the two categories of movies.

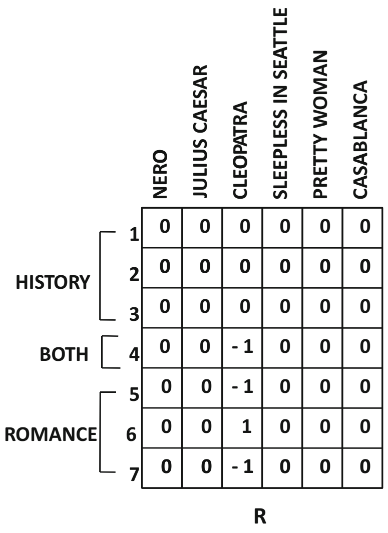
From Equation 3.13, it follows that each rating r_{ij} in R can be approximately expressed as a dot product of the ith user factor and jth item factor:

$$r_{ij} \approx \overline{u_i} \cdot \overline{v_j} \tag{3.14}$$



$$\underbrace{\begin{array}{ll|cccccc} & & \text{NERO} & \text{JULIUS CAESAR} & \text{CLEOPATRA} & \text{SLEEPLESS IN SEATTLE} & \text{PRETTY WOMAN} & \text{CASABLANCA} \\ \hline \text{HISTORY} & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ & 3 & 1 & 1 & 1 & 0 & 0 & 0 \\ \text{BOTH} & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\ \text{ROMANCE} & 5 & -1 & -1 & -1 & 1 & 1 & 1 \\ & 6 & -1 & -1 & 1 & 1 & 1 & 1 \\ & 7 & -1 & -1 & -1 & 1 & 1 & 1 \end{array}}_{R} \approx \underbrace{\begin{array}{l|cc} & \text{HISTORY} & \text{ROMANCE} \\ \hline 1 & 1 & 0 \\ 2 & 1 & 0 \\ 3 & 1 & 0 \\ 4 & 1 & 1 \\ 5 & -1 & 1 \\ 6 & -1 & 1 \\ 7 & -1 & 1 \end{array}}_{U} \times \underbrace{\begin{array}{l|cccccc} & \text{NERO} & \text{JULIUS CAESAR} & \text{CLEOPATRA} & \text{SLEEPLESS IN SEATTLE} & \text{PRETTY WOMAN} & \text{CASABLANCA} \\ \hline \text{HISTORY} & 1 & 1 & 1 & 0 & 0 & 0 \\ \text{ROMANCE} & 0 & 0 & 1 & 1 & 1 & 1 \end{array}}_{V^T}$$

(a) Example of rank-2 matrix factorization



$$\underbrace{\begin{array}{ll|cccccc} & & \text{NERO} & \text{JULIUS CAESAR} & \text{CLEOPATRA} & \text{SLEEPLESS IN SEATTLE} & \text{PRETTY WOMAN} & \text{CASABLANCA} \\ \hline \text{HISTORY} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{BOTH} & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \text{ROMANCE} & 5 & 0 & 0 & -1 & 0 & 0 & 0 \\ & 6 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 7 & 0 & 0 & -1 & 0 & 0 & 0 \end{array}}_{R}$$

(b) Residual matrix

Figure 3.7: Example of a matrix factorization and its residual matrix

Since the latent factors $\overline{u_i} = (u_{i1} \ldots u_{ik})$ and $\overline{v_j} = (v_{j1} \ldots v_{jk})$ can be viewed as the affinities of the users for k different concepts, an intuitive way of expressing Equation 3.14 would be as follows:

$$\begin{aligned} r_{ij} &\approx \sum_{s=1}^{k} u_{is} \cdot v_{js} \\ &= \sum_{s=1}^{k} (\textit{Affinity of user } i \textit{ to concept } s) \times (\textit{Affinity of item } j \textit{ to concept } s) \end{aligned}$$

In the case of Figure 3.7, the two concepts in the aforementioned summation correspond to the romance and historical genres. Therefore, the summation may be expressed as follows:

$$\begin{aligned} r_{ij} \approx &(\textit{Affinity of user } i \textit{ to history}) \times (\textit{Affinity of item } j \textit{ to history}) \\ &+ (\textit{Affinity of user } i \textit{ to romance}) \times (\textit{Affinity of item } j \textit{ to romance}) \end{aligned}$$

It needs to be pointed out that the notion of concepts is often not semantically interpretable, as illustrated in Figure 3.7. A latent vector may often be an arbitrary vector of positive and negative values and it becomes difficult to give it a semantic interpretation. However, it does represent a dominant correlation pattern in the ratings matrix, just as the latent vector of Figure 3.6 represents a geometric correlation pattern. As we will see later, some forms of factorization, such as non-negative matrix factorization, are explicitly designed to achieve greater interpretability in the latent vectors.

The key differences among various matrix factorization methods arise in terms of the constraints imposed on U and V (e.g., orthogonality or non-negativity of the latent vectors) and the nature of the objective function (e.g., minimizing the Frobenius norm or maximizing the likelihood estimation in a generative model). These differences play a key role in the usability of the matrix factorization model in various real-world scenarios.

3.6.4 Unconstrained Matrix Factorization

The most fundamental form of matrix factorization is the unconstrained case, in which no constraints are imposed on the factor matrices U and V. Much of the recommendation literature refers to unconstrained matrix factorization as singular value decomposition (SVD). Strictly speaking, this is technically incorrect; in SVD, the columns of U and V must be orthogonal. However, the use of the term "SVD" to refer to unconstrained matrix factorization[7] is rather widespread in the recommendation literature, which causes some confusion to practitioners from outside the field. In this chapter, we will deviate from this incorrect practice and treat unconstrained matrix factorization and SVD in a distinct way. This section will discuss unconstrained matrix factorization, and the following section will discuss SVD.

Before discussing the factorization of incomplete matrices, let us first visit the problem of factorizing fully specified matrices. How can one determine the factor matrices U and V,

[7]In SVD [568], the basis vectors are also referred to as *singular* vectors, which, by definition, must be mutually orthonormal.

so that the fully specified matrix R matches UV^T as closely as possible? One can formulate an optimization problem with respect to the matrices U and V in order to achieve this goal:

$$\begin{aligned}&\text{Minimize } J = \frac{1}{2}||R - UV^T||^2\\&\text{subject to:}\\&\qquad\text{No constraints on } U \text{ and } V\end{aligned}$$

Here, $||.||^2$ represents the squared Frobenius norm of the matrix, which is equal to the sum of the squares of the matrix entries. Thus, the objective function is equal to the sum of the squares of the entries in the residual matrix $(R - UV^T)$. The smaller the objective function is, the better the quality of the factorization $R \approx UV^T$ will be. This objective function can be viewed as a *quadratic loss* function, which quantifies the loss of accuracy in estimating the matrix R with the use of low-rank factorization. A variety of gradient descent methods can be used to provide an optimal solution to this factorization.

However, in the context of a matrix with *missing entries*, only a subset of the entries of R are known. Therefore, the objective function, as written above, is undefined as well. After all, one cannot compute the Frobenius norm of a matrix in which some of the entries are missing! The objective function, therefore, needs to be rewritten only in terms of the observed entries in order to learn U and V. The nice part about this process is that once the latent factors U and V are learned, *the entire ratings matrix can be reconstructed as UV^T in one shot.*

Let the set of all user-item pairs (i, j), which are observed in R, be denoted by S. Here, $i \in \{1 \dots m\}$ is the index of a user, and $j \in \{1 \dots n\}$ is the index of an item. Therefore, the set S of observed user-item pairs is defined as follows:

$$S = \{(i, j) : r_{ij} \text{ is observed}\} \tag{3.15}$$

If we can somehow factorize the incomplete matrix R as the approximate product UV^T of fully specified matrices $U = [u_{is}]_{m \times k}$ and $V = [v_{js}]_{n \times k}$, then all the entries in R can be predicted as well. Specifically, the (i, j)th entry of matrix R can be predicted as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{3.16}$$

Note the "hat" symbol (i.e., circumflex) on the rating on the left-hand side to indicate that it is a predicted value rather than an observed value. The difference between the observed and predicted value of a specified entry (i, j) is given by $e_{ij} = (r_{ij} - \hat{r}_{ij}) = (r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js})$. Then, the modified objective function, which works with incomplete matrices, is computed only over the observed entries in S as follows:

$$\begin{aligned}&\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)^2\\&\text{subject to:}\\&\qquad\text{No constraints on } U \text{ and } V\end{aligned}$$

Note that the aforementioned objective function sums up the error *only over the observed entries in S*. Furthermore, each of the terms $(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js})^2$ is the squared error e_{ij}^2 between the observed and predicted values of the entry (i, j). Here, u_{is} and v_{js} are the

```
Algorithm GD(Ratings Matrix: $R$, Learning Rate: $\alpha$)
begin
  Randomly initialize matrices $U$ and $V$;
  $S = \{(i,j) : r_{ij}$ is observed$\}$;
  while not(convergence) do
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$;
    for each user-component pair $(i,q)$ do $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq}$;
    for each item-component pair $(j,q)$ do $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq}$;
    for each user-component pair $(i,q)$ do $u_{iq} \Leftarrow u_{iq}^{+}$;
    for each item-component pair $(j,q)$ do $v_{jq} \Leftarrow v_{jq}^{+}$;
    Check convergence condition;
  end
end
```

Figure 3.8: Gradient descent

unknown variables, which need to be learned to minimize the objective function. This can be achieved simply with gradient descent methods. Therefore, one needs to compute the partial derivative of J with respect to the decision variables u_{iq} and v_{jq}:

$$\begin{aligned}
\frac{\partial J}{\partial u_{iq}} &= \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\} \\
&= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\} \\
\frac{\partial J}{\partial v_{jq}} &= \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\} \\
&= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}
\end{aligned}$$

Note that the entire vector of partial derivatives provides us with the gradient with respect to the vector of $(m \cdot k + n \cdot k)$ decision variables in the matrices U and V. Let this gradient vector be denoted by $\overline{\nabla J}$. Let the vector of $(m \cdot k + n \cdot k)$ decision variables corresponding to the entries in U and V be denoted by $\overline{VAR}$. Then, one can update the entire vector of decision variables as $\overline{VAR} \Leftarrow \overline{VAR} - \alpha \cdot \overline{\nabla J}$. Here, $\alpha > 0$ is the step size, which can be chosen using standard numerical methods in nonlinear programming [76]. In many cases, the step sizes are set to small constant values. The iterations are executed to convergence. This approach is referred to as *gradient descent*. The algorithmic framework for gradient-descent is illustrated in Figure 3.8. It is noteworthy that the intermediate variables u_{iq}^{+} and v_{jq}^{+} are used to ensure that all updates to the entries in U and V are performed simultaneously.

One can also perform the updates in Figure 3.8 using a matrix representation. The first step is to compute an error matrix $E = R - UV^T$ in which the unobserved entries of E (i.e., entries not in S) are set to 0. Note that E is a very sparse matrix, and it makes sense to compute the value of e_{ij} for only the observed entries $(i,j) \in S$ and store the matrix using a sparse data structure. Subsequently, the updates can be computed as follows:

$$U \Leftarrow U + \alpha E V$$
$$V \Leftarrow V + \alpha E^T U$$

These updates can be executed to convergence, while taking care to update all entries in both matrices simultaneously with the use of intermediate variables (as in Figure 3.8).

3.6.4.1 Stochastic Gradient Descent

The aforementioned method is referred to as the *batch update method.* An important observation is that the updates are linear functions of the errors in the observed entries of the ratings matrix. The update can be executed in other ways by *decomposing* it into smaller components associated with the errors in *individual* observed entries rather than all entries. This update can be *stochastically approximated* in terms of the error in a (randomly chosen) observed entry (i, j) as follows:

$$u_{iq} \Leftarrow u_{iq} - \alpha \cdot \left[\frac{\partial J}{\partial u_{iq}}\right]_{\text{Portion contributed by } (i,j)} \quad \forall q \in \{1 \dots k\}$$
$$v_{jq} \Leftarrow v_{jq} - \alpha \cdot \left[\frac{\partial J}{\partial v_{jq}}\right]_{\text{Portion contributed by } (i,j)} \quad \forall q \in \{1 \dots k\}$$

One can cycle through the observed entries in R one at a time (in random order) and update only the relevant set of $2 \cdot k$ entries in the factor matrices rather than all $(m \cdot k + n \cdot k)$ entries in the factor matrices. In such a case, the $2 \cdot k$ updates *specific to the observed entry* $(i, j) \in S$, are as follows:

$$u_{iq} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq} \quad \forall q \in \{1 \dots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq} \quad \forall q \in \{1 \dots k\}$$

For each observed rating r_{ij}, the error e_{ij} is used to update the k entries in row i of U and the k entries in the row j of V. Note that $e_{ij} \cdot v_{jq}$ is the component of partial derivative of J with respect to u_{iq}, that *is specific to* a single observed entry (i, j). For better efficiency, each of these k entries can be updated simultaneously in vectorized form. Let $\overline{u_i}$ be the ith row of U and $\overline{v_j}$ be the jth row of V. Then, the aforementioned updates can be rewritten in k-dimensional vectorized form as follows:

$$\overline{u_i} \Leftarrow \overline{u_i} + \alpha \, e_{ij} \, \overline{v_j}$$
$$\overline{v_j} \Leftarrow \overline{v_j} + \alpha \, e_{ij} \, \overline{u_i}$$

We cycle through all the observed entries multiple times (i.e., use multiple iterations) until convergence is reached. This approach is referred to as *stochastic gradient descent* in which the gradient is approximated by that computed on the basis of the error of a single randomly chosen entry in the matrix. The pseudo-code for the stochastic gradient descent method is illustrated in Figure 3.9. It is noteworthy that temporary variables u_{iq}^+ and v_{jq}^+ are used to store intermediate results during an update, so that the $2 \cdot k$ updates do not affect each other. This is a general approach that should be used in all group-wise updates discussed in this book, although we might not state it explicitly.

In practice, faster convergence is achieved by the stochastic gradient descent method as compared to the batch method, although the convergence is much smoother in the latter.

```
**Algorithm** *SGD*(Ratings Matrix: $R$, Learning Rate: $\alpha$)
begin
  Randomly initialize matrices $U$ and $V$;
  $S = \{(i, j) : r_{ij} \text{ is observed}\}$;
  while not(convergence) do
  begin
    Randomly shuffle observed entries in $S$;
    for each $(i, j) \in S$ in shuffled order do
    begin
      $e_{ij} \Leftarrow r_{ij} - \sum_{s=1}^{k} u_{is} v_{js}$;
      for each $q \in \{1 \ldots k\}$ do $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$;
      for each $q \in \{1 \ldots k\}$ do $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$;
      for each $q \in \{1 \ldots k\}$ do $u_{iq} = u_{iq}^{+}$ and $v_{jq} = v_{jq}^{+}$;
    end
    Check convergence condition;
  end
end
```

Figure 3.9: Stochastic gradient descent

This is because the entries of U and V are updated simultaneously in the latter case with the use of all observed entries, rather than a single randomly chosen observed entry. This noisy approximation of stochastic gradient descent can sometimes impact solution quality and smoothness of convergence. In general, stochastic gradient descent is preferable when the data size is very large and computational time is the primary bottleneck. In other "compromise" methods, mini-batches are used in which a subset of observed entries is used to construct the update. These different methods provide different trade-offs between solution quality and computational efficiency.

As one repeatedly cycles through the observed entries in the matrix to update the factor matrices, convergence will eventually be reached. In general, the global method is known to have guaranteed convergence, even though it is generally slower than the local method. A typical value of the step size (or *learning rate*) is a small constant value such as $\alpha = 0.005$. A more effective approach to avoid local minima and speed up convergence is to use the *bold driver algorithm* [58, 217] to select α adaptively in each iteration. It is also possible, in principle, to use different step sizes for different factors [586]. An interesting observation about some of these models is that executing them until convergence for too many iterations can sometimes lead to slight worsening of the solution quality on the unobserved entries. Therefore, it is sometimes advisable not to set the convergence criteria too strictly.

Another issue with these latent factor models is that of *initialization*. For example, one can initialize the factor matrices to small numbers in $(-1, 1)$. However, the choice of initialization can affect the final solution quality. It is possible to use a number of heuristics to improve quality. For example, one can use some simple SVD-based heuristics, discussed later in this section, to create an approximate initialization.

3.6.4.2 Regularization

One of the main problems with this approach arises when the ratings matrix R is sparse and relatively few entries are observed. This is almost always the case in real settings.

In such cases, the observed set S of ratings is small, which can cause overfitting. Note that overfitting is also a common problem in classification when training data are limited. A common approach for addressing this problem is to use *regularization.* Regularization reduces the tendency of the model to overfit at the expense of introducing a *bias*[8] in the model.

In regularization, the idea is to discourage very large values of the coefficients in U and V in order to encourage stability. Therefore, a regularization term, $\frac{\lambda}{2}(||U||^2 + ||V||^2)$, is added to the objective function, where $\lambda > 0$ is the regularization parameter. Here, $|| \cdot ||^2$ denotes the (squared) Frobenius norm of the matrix. The basic idea is to create a bias in favor of simpler solutions by penalizing large coefficients. This is a standard approach, which is used in many forms of classification and regression, and also leveraged by collaborative filtering. The parameter λ is always non-negative and it controls the weight of the regularization term. The method for choosing λ is discussed later in this section.

As in the previous case, assume that $e_{ij} = (r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js})$ represents the difference between the observed value and predicted value of specified entry $(i, j) \in S$. The regularized objective function is as follows:

$$\begin{aligned}
\text{Minimize } J &= \frac{1}{2} \sum_{(i,j)\in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 \\
&= \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2
\end{aligned}$$

Upon taking the partial derivative of J with respect to each of the decision variables, one obtains almost the same result as the unregularized case, except that the terms λu_{iq} and λv_{jq}, respectively, are added to the corresponding gradients in the two cases.

$$\begin{aligned}
\frac{\partial J}{\partial u_{iq}} &= \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\} \\
&= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\} \\
\frac{\partial J}{\partial v_{jq}} &= \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\} \\
&= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}
\end{aligned}$$

The steps for performing the gradient descent remain similar to those discussed in the case without regularization. Either the batch or the local methods may be used. For example, consider the global update method. Let the vector of $(m \cdot k + n \cdot k)$ decision variables corresponding to the entries in U and V be denoted by $\overline{VAR}$ and let the corresponding gradient vector be denoted by $\overline{\nabla J}$. Then, one can update the entire vector of decision variables as $\overline{VAR} \Leftarrow \overline{VAR} - \alpha \cdot \overline{\nabla J}$. This can be effectively achieved by modifying the

[8]Refer to Chapter 6 for a discussion of the bias-variance trade-off.

(unregularized) updates in Figure 3.8 to include regularization terms. The modified updates may be written as follows:

$$u_{iq} \Leftarrow u_{iq} + \alpha\left(\sum_{j:(i,j)\in S} e_{ij}\cdot v_{jq} - \lambda\cdot u_{iq}\right) \quad \forall q \in \{1\dots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha\left(\sum_{i:(i,j)\in S} e_{ij}\cdot u_{iq} - \lambda\cdot v_{jq}\right) \quad \forall q \in \{1\dots k\}$$

The updates can be executed to convergence. One can also write these updates in terms of the $m \times n$ error matrix $E = [e_{ij}]$ in which unobserved entries of E are set to 0:

$$U \Leftarrow U(1-\alpha\cdot\lambda) + \alpha E V$$
$$V \Leftarrow V(1-\alpha\cdot\lambda) + \alpha E^T U$$

Note that the multiplicative term $(1-\alpha\cdot\lambda)$ shrinks the parameters in each step, which is a result of regularization. If the matrix form is to be used for updates, care must be taken to compute and use sparse representations of E. It makes sense to compute the value of e_{ij} for only the observed entries $(i,j)\in S$ and store E using a sparse data structure.

In the case of local updates (i.e., stochastic gradient descent), the partial derivatives are computed with respect to the error in a randomly chosen observed entry (i,j) rather than all the entries. The following $2\cdot k$ updates may be executed for each observed entry $(i,j)\in S$, which are processed in random order:

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij}\cdot v_{jq} - \lambda\cdot u_{iq}) \quad \forall q \in \{1\dots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha(e_{ij}\cdot u_{iq} - \lambda\cdot v_{jq}) \quad \forall q \in \{1\dots k\}$$

For better efficiency, these updates are executed in vectorized form over the k-dimensional factor vectors of user i and item j as follows:

$$\overline{u_i} \Leftarrow \overline{u_i} + \alpha(e_{ij}\overline{v_j} - \lambda\overline{u_i})$$
$$\overline{v_j} \Leftarrow \overline{v_j} + \alpha(e_{ij}\overline{u_i} - \lambda\overline{v_j})$$

These updates are used within the framework of the algorithm described in Figure 3.9. It is noteworthy that the local updates are not exactly equivalent[9] to the vectorized global updates in terms of how the regularization term is treated. This is because the regularization components of the updates, which are $-\lambda u_{iq}$ and $-\lambda v_{jq}$, are used multiple times in a cycle of local updates through *all* the observed entries; updates are executed to u_{iq} for each observed entry in row i and updates are executed to v_{jq} for each observed entry in column j. Furthermore, different rows and columns may have different numbers of observed entries, which can further affect the relative level of regularization of various user and

[9]A more precise update should be $\overline{u_i} \Leftarrow \overline{u_i} + \alpha(e_{ij}\overline{v_j} - \lambda\overline{u_i}/n_i^{user})$ and $\overline{v_j} \Leftarrow \overline{v_j} + \alpha(e_{ij}\overline{u_i} - \lambda\overline{v_j}/n_j^{item})$. Here, n_i^{user} represents the number of observed ratings for user i and n_j^{item} represents the number of observed ratings for item j. Here, the regularization terms for various user/item factors are divided equally among the corresponding observed entries for various users/items. In practice, the (simpler) heuristic update rules discussed in the chapter are often used. We have chosen to use these (simpler) rules throughout this chapter to be consistent with the research literature on recommender systems. With proper parameter tuning, λ will automatically adjust to a smaller value in the case of the simpler update rules.

item factors. In the vectorized global method, the regularization is done more gently and uniformly because each entry u_{iq} and v_{jq} is updated only once. Nevertheless, since λ is chosen adaptively during parameter tuning, the local update method will automatically select smaller values of λ than the global method. From a heuristic point of view, the two methods provide roughly similar results, but with different trade-offs between quality and efficiency.

As before, $\alpha > 0$ represents the step size, and $\lambda > 0$ is the regularization parameter. For example, a small constant value of α, such as 0.005, is known to work reasonably well in the case of the Netflix Prize data set. Alternatively, one might use the bold driver algorithm [58, 217] to select α adaptively in each iteration in order to avoid local optima and speed up convergence. It remains to discuss how the regularization parameter λ is selected. The simplest method is to hold out a fraction of the observed entries in the ratings matrix and not use them to train the model. The prediction accuracy of the model is tested over this subset of held out entries. Different values of λ are tested, and the value of λ that provides the highest accuracy is used. If desired, the model can be retrained on the entire set of specified entries (with no hold outs), once the value of λ is selected. This method of parameter tuning is referred to as the *hold out* method. A more sophisticated approach is to use a method referred to as *cross-validation*. This method is discussed in Chapter 7 on evaluating recommender systems. For better results, different regularization parameters λ_1 and λ_2 may be used for the user factors and item factors.

Often, it can be expensive to try different values of λ on the hold-out set in order to determine the optimal value. This restricts the ability to try many choices of λ. As a result, the values of λ are often not well-optimized. One approach, proposed in [518], is to treat the entries of matrices U and V as parameters, and the regularization parameters as hyper-parameters, which are optimized *jointly* with a probabilistic approach. A Gibbs sampling approach is proposed in [518] to jointly learn the parameters and hyper-parameters.

3.6.4.3 Incremental Latent Component Training

One variant of these training methods is to train the latent components incrementally. In other words, we first perform the updates $u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq})$ and $v_{jq} \Leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq})$ only for $q = 1$. The approach repeatedly cycles through all the observed entries in S while performing these updates for $q = 1$ until convergence is reached. Therefore, we can learn the first pair of columns, $\overline{U_1}$ and $\overline{V_1}$, of U and V, respectively. Then, the $m \times n$ *outer-product*[10] matrix $\overline{U_1}\,\overline{V_1}^T$ is subtracted from R (for observed entries). Subsequently, the updates are performed for $q = 2$ with the (residual) ratings matrix to learn the second pair of columns, $\overline{U_2}$ and $\overline{V_2}$, of U and V, respectively. Then, $\overline{U_2}\,\overline{V_2}^T$ is subtracted from R. This process is repeated each time with the residual matrix until $q = k$. The resulting approach provides the required matrix factorization because the overall rank-k factorization can be expressed as the sum of k rank-1 factorizations:

$$R \approx UV^T = \sum_{q=1}^{k} \overline{U_q}\,\overline{V_q}^T \tag{3.17}$$

[10]The inner-product of two column-vectors $\overline{x}$ and $\overline{y}$ is given by the scalar $\overline{x}^T\overline{y}$, whereas the outer-product is given by the rank-1 matrix $\overline{x}\,\overline{y}^T$. Furthermore, $\overline{x}$ and $\overline{y}$ need not be of the same size in order to compute an outer-product.

```
**Algorithm** *ComponentWise-SGD*(Ratings Matrix: $R$, Learning Rate: $\alpha$)
begin
  Randomly initialize matrices $U$ and $V$;
  $S = \{(i, j) : r_{ij}$ is observed$\}$;
  for $q = 1$ to $k$ do
  begin
    while not(convergence) do
    begin
      Randomly shuffle observed entries in $S$;
      for each $(i, j) \in S$ in shuffled order do
      begin
        $e_{ij} \Leftarrow r_{ij} - u_{iq}v_{jq}$;
        $u^+_{iq} \Leftarrow u_{iq} + \alpha \cdot (e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq})$;
        $v^+_{jq} \Leftarrow v_{jq} + \alpha \cdot (e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq})$;
        $u_{iq} = u^+_{iq}$; $v_{jq} = v^+_{jq}$;
      end
      Check convergence condition;
    end
    { Element-wise implementation of $R \Leftarrow R - \overline{U_q}\,\overline{V_q}^T$ }
    for each $(i, j) \in S$ do $r_{ij} \Leftarrow r_{ij} - u_{iq}v_{jq}$;
  end
end
```

Figure 3.10: Component-wise implementation of stochastic gradient descent

A description of this procedure is illustrated in Figure 3.10. The differences of this approach from the version discussed earlier can be understood in terms of the differences in their nested loop structures. Incremental component training loops through various values of q in the outermost loops and cycles through the observed entries repeatedly in the inner loops to reach convergence for each value of q (cf. Figure 3.10). The earlier method loops through the observed entries repeatedly to reach convergence in the outer loops and cycles though various values of q in the inner loop (cf. Figure 3.9). Furthermore, the incremental method needs to adjust the ratings matrix between two executions of the outer loop. This approach leads to faster and more stable convergence in each component because a smaller number of variables is optimized at one time.

It is noteworthy that different strategies for gradient descent will lead to solutions with different properties. This particular form of incremental training will lead to the earlier latent components being the dominant ones, which provides a similar flavor to that of SVD. However, the resulting columns in U (or V) might not be mutually orthogonal. It is also possible to force mutual orthogonality of the columns of U (and V) by using *projected* gradient descent for $q > 1$. Specifically, the gradient vector with respect to the variables in column $\overline{U_q}$ (or $\overline{V_q}$) is projected in an orthogonal direction to the $(q-1)$ columns of U (or V) found so far.

3.6.4.4 Alternating Least Squares and Coordinate Descent

The stochastic gradient method is an efficient methodology for optimization. On the other hand, it is rather sensitive, both to the initialization and the way in which the step sizes are chosen. Other methods for optimization include the use of *alternating least squares (ALS)* [268, 677], which is generally more stable. The basic idea of this approach to use the following iterative approach, starting with an initial set of matrices U and V:

1. Keeping U fixed, we solve for each of the n rows of V by treating the problem as a least-squares regression problem. Only the observed ratings in S can be used for building the least-squares model in each case. Let $\overline{v_j}$ be the jth row of V. In order to determine the optimal vector $\overline{v_j}$, we wish to minimize $\sum_{i:(i,j)\in S}(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^2$, which is a least-squares regression problem in $v_{j1} \ldots v_{jk}$. The terms $u_{i1} \ldots u_{ik}$ are treated as constant values, whereas $v_{j1} \ldots v_{jk}$ are treated as optimization variables. Therefore, the k latent factor components in $\overline{v_j}$ for the jth item are determined with least-squares regression. A total of n such least-squares problems need to be executed, and each least-squares problem has k variables. Because the least-squares problem for each item is independent, this step can be parallelized easily.

2. Keeping V fixed, solve for each of the m rows of U by treating the problem as a least-squares regression problem. Only the specified ratings in S can be used for building the least-squares model in each case. Let $\overline{u_i}$ be the ith row of U. In order to determine the optimal vector $\overline{u_i}$, we wish to minimize $\sum_{j:(i,j)\in S}(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^2$, which is a least-squares regression problem in $u_{i1} \ldots u_{ik}$. The terms $v_{j1} \ldots v_{jk}$ are treated as constant values, whereas $u_{i1} \ldots u_{ik}$ are treated as optimization variables. Therefore, the k latent factor components for the ith user are determined with least-squares regression. A total of m such least-squares problems need to be executed, and each least-squares problem has k variables. Because the least-squares problem for each user is independent, this step can be parallelized easily.

These two steps are iterated to convergence. When regularization is used in the objective function, it amounts to using Tikhonov regularization [22] in the least-squares approach. The value of the regularization parameter $\lambda > 0$ can be fixed across all the independent least-squares problems, or it can be chosen differently. In either case, one might need to determine the optimal value of λ by using a hold-out or cross-validation methodology. A brief discussion of linear regression with Tikhonov regularization is provided in section 4.4.5 of Chapter 4. Although the linear regression discussion in Chapter 4 is provided in the context of content-based models, the basic regression methodology is invariant across the different scenarios in which it is used.

Interestingly, a weighted version *ALS* is particularly well-suited to implicit feedback settings in which the matrix is assumed to be fully specified with many zero values. Furthermore, the nonzero entries are often weighted more heavily in these settings. In such cases, stochastic gradient descent becomes too expensive. When most of the entries are zeros, some tricks can be used to make weighted *ALS* an efficient option. The reader is referred to [260].

The drawback of *ALS* is that it is not quite as efficient as stochastic-gradient descent in large-scale settings with explicit ratings. Other methods such as coordinate descent can effectively address the trade-off between efficiency and stability [650]. In coordinate-descent, the approach of fixing a subset of variables (as in *ALS*) is pushed to the extreme. Here, all entries in U and V are fixed except for a single entry (or *coordinate*) in one of the two matrices, which is optimized using the objective function of section 3.6.4.2. The resulting optimization solution can be shown to have closed form because it is a quadratic objective function in a single variable. The corresponding value of u_{iq} (or v_{jq}) can be determined efficiently according to one of the following two updates:

$$u_{iq} \Leftarrow \frac{\sum_{j:(i,j)\in S}(e_{ij} + u_{iq}v_{jq})v_{jq}}{\lambda + \sum_{j:(i,j)\in S} v_{jq}^2}$$

$$v_{jq} \Leftarrow \frac{\sum_{i:(i,j)\in S}(e_{ij} + u_{iq}v_{jq})u_{iq}}{\lambda + \sum_{i:(i,j)\in S} u_{iq}^2}$$

Here, S denotes the set of observed entries in the ratings matrix and $e_{ij} = r_{ij} - \hat{r}_{ij}$ is the prediction error of entry (i, j). One cycles through the $(m + n) \cdot k$ parameters in U and V with these updates until convergence is reached. It is also possible to combine coordinate descent with incremental latent component training just as stochastic gradient descent is combined with increment component training (cf. section 3.6.4.3).

3.6.4.5 Incorporating User and Item Biases

A variation on the unconstrained model was introduced by Paterek [473] to incorporate variables that can learn user and item biases. Assume for the purpose of discussion that the ratings matrix is mean-centered by subtracting the *global* mean μ of the *entire* ratings matrix from all the entries as a preprocessing step. After predicting the entries with the latent factor model, the value μ is added back to the predicted values as a postprocessing step. Therefore, in this section, we will simply assume that the ratings matrix R has already been centered in this way, and ignore the preprocessing and postprocessing steps.

Associated with each user i, we have a variable o_i, which indicates the general bias of users to rate items. For example, if user i is a generous person, who tends to rate all items highly, then the variable o_i will be a positive quantity. On the other hand, the value of o_i will be negative for a curmudgeon who rates most items negatively. Similarly, the variable p_j denotes the bias in the ratings of item j. Highly liked items (e.g., a box-office hit) will tend to have larger (positive) values of p_j, whereas globally disliked items will have negative values of p_j. It is the job of the factor model to learn the values of o_i and p_j in a data-driven manner. The main change to the original latent factor model is that a part of the (i, j)th rating is explained by $o_i + p_j$ and the remainder by the (i, j)th entry of the product UV^T of the latent factor matrices. Therefore, the predicted value of the rating of entry (i, j) is given by the following:

$$\hat{r}_{ij} = o_i + p_j + \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{3.18}$$

Thus, the error e_{ij} of an observed entry $(i, j) \in S$ is given by the following:

$$e_{ij} = r_{ij} - \hat{r}_{ij} = r_{ij} - o_i - p_j - \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{3.19}$$

Note that the values o_i and p_j are also variables that need to be learned in a data-driven manner along with the latent factor matrices U and V. Then, the minimization objective function J may be formulated by aggregating the squared errors over the observed entries of the ratings matrix (i.e., set S) as follows:

$$\begin{aligned} J &= \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} o_i^2 + \frac{\lambda}{2} \sum_{j=1}^{n} p_j^2 \\ &= \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - o_i - p_j - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda}{2} \left(\sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2 + \sum_{i=1}^{m} o_i^2 + \sum_{j=1}^{n} p_j^2 \right) \end{aligned}$$

It turns out that this problem is different from unconstrained matrix factorization to only a minor degree. Instead of having separate bias variables o_i and p_j for users and items, we can increase the size of the factor matrices to incorporate these bias variables. We need to add two additional columns to each factor matrix U and V, to create larger factor matrices of size $m \times (k+2)$ and $n \times (k+2)$, respectively. The last two columns of each factor matrix are special, because they correspond to the bias components. Specifically, we have:

$$\begin{aligned} u_{i,k+1} &= o_i \quad \forall i \in \{1 \dots m\} \\ u_{i,k+2} &= 1 \quad \forall i \in \{1 \dots m\} \\ v_{j,k+1} &= 1 \quad \forall j \in \{1 \dots n\} \\ v_{j,k+2} &= p_j \quad \forall j \in \{1 \dots n\} \end{aligned}$$

Note that the conditions $u_{i,k+2} = 1$ and $v_{j,k+1} = 1$ are constraints on the factor matrices. *In other words, we need to constrain the last column of the user-factor matrix to all 1s, and the second last column of the item-factor matrix to all 1s.* This scenario is pictorially shown in Figure 3.11. Then, the modified optimization problem with these enlarged factor matrices is as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k+2} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda}{2} \sum_{s=1}^{k+2} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 \right)$$

subject to:

$(k+2)$th column of U contains only 1s

$(k+1)$th column of V contains only 1s

It is noteworthy that the summations in the objective are up to $(k + 2)$ rather than k. Note that this problem is virtually identical to the unconstrained case except for the minor

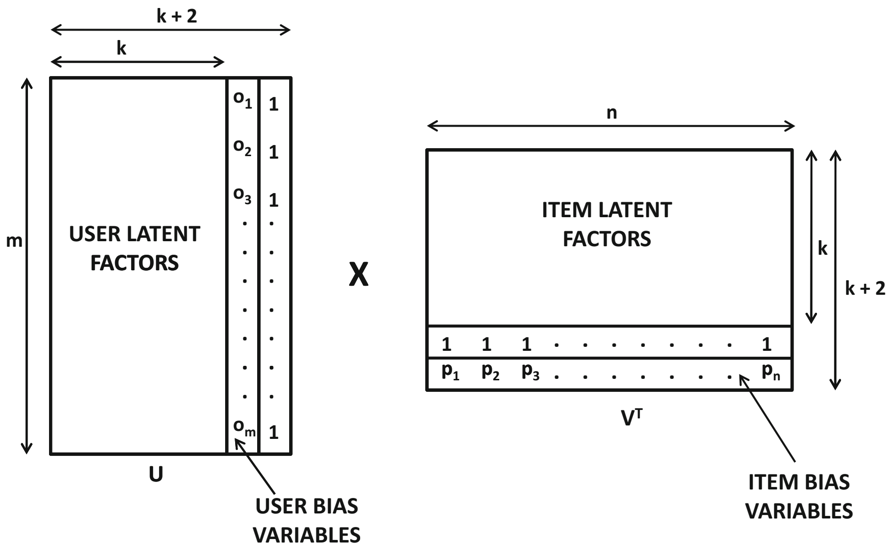


Figure 3.11: Incorporating user and item biases in the latent factor model

constraints on the factors. The other change is the increase in the sizes of the factor matrices to incorporate the user and item bias variables. Because of the minor change in the problem formulation, one only needs to make corresponding changes to the gradient descent method. For initialization, the $(k+1)$th column of V and the $(k+2)$th column of U are set to 1s. Exactly the same (local) update rules are used as in the unconstrained case, except that the two perturbed entries in the $(k+1)$th column of V and the $(k+2)$th column of U are reset to their fixed values after each update (or simply not updated). The following updates may be executed by cycling over each specified entry $(i,j) \in S$:

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \dots k+2\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}) \quad \forall q \in \{1 \dots k+2\}$$

Reset perturbed entries in $(k+2)$th column of U and $(k+1)$th column of V to 1s

This group of updates is performed simultaneously as a group. It is also possible to use the alternating least-squares method with minor variations (see Exercise 11). The aforementioned discussion uses the same regularization parameters and learning rates for each type of variable. It is sometimes recommended to use different regularization parameters and learning rates for the user biases, item biases, and factor variables [586]. This can be achieved with minor modifications of the aforementioned updates.

A natural question that arises is why this formulation should perform better than unconstrained matrix factorization. The addition of constraints on the last two columns of the factor matrices should only reduce the global solution quality, because one is now optimizing over a smaller space of solutions. However, in many cases, adding such constraints biases the solution while reducing overfitting. In other words, the addition of such intuitive constraints can often improve the generalizability of the learning algorithm to *unseen* entries, even though the error over the *specified* entries may be higher. This is particularly helpful when the number of observed ratings for a user or for an item is small [473]. Bias variables add a component to the ratings that are global to either the users or the items. Such global properties are useful when limited data is available. As a specific example, consider the case in which a user has provided ratings for only a small number (1 or 2) items. In such cases, many recommendation algorithms, such as neighborhood-based methods, will not give reliable predictions for the user. On the other hand, the (non-personalized) predictions of the item bias variables will be able to give reasonable predictions. After all, if a particular movie is a box-office hit on a global basis, then the relevant user is also more likely to appreciate it. The bias variables will also reflect this fact and incorporate it into the learning algorithm.

In fact, it has been shown [73, 310, 312] that using *only* the bias variables (i.e., $k = 0$) can often provide reasonably good rating predictions. This point was emphasized as one of the practical lessons learned from the Netflix Prize contest [73]:

> "Of the numerous new algorithmic contributions, I would like to highlight one – those humble baseline predictors (or biases), which capture main effects in the data. While the literature mostly concentrates on the more sophisticated algorithmic aspects, we have learned that an accurate treatment of main effects is probably at least as significant as coming up with modeling breakthroughs."

This means that a significant part of the ratings can be explained by user generosity and item popularity, rather than any specific *personalized* preferences of users for items. Such a non-personalized model is discussed in section 3.7.1, which is equivalent to setting $k = 0$ in the aforementioned model. As a result, only the biases of users and items are learned, and a *baseline* rating B_{ij} is predicted for user i and item j by summing their biases. One can use such a baseline rating to enhance any off-the-shelf collaborative filtering model. To do so, one can simply subtract each B_{ij} from the (i, j)th (observed) entry of the ratings matrix before applying collaborative filtering. These values are added back in a postprocessing phase to the predicted values. Such an approach is especially useful for models in which one cannot easily parameterize bias variables. For example, (traditional) neighborhood models accomplish these bias-correction goals with row-wise mean-centering, although the use of B_{ij} to correct the matrix entries would be a more sophisticated approach because it adjusts for both user and item biases.

3.6.4.6 Incorporating Implicit Feedback

Generally, implicit feedback scenarios correspond to the use of unary ratings matrices in which users express their interests by buying items. However, even in cases in which users explicitly rate items, the *identity of the items they rate* can be viewed as an implicit feedback. In other words, a significant predictive value is captured by the identity of the items that users rate, *irrespective of the actual values of the ratings*. A recent paper [184] describes this phenomenon elegantly in the context of the music domain:

> "Intuitively, a simple process could explain the results [showing the predictive value of implicit feedback]: users chose to rate songs they listen to, and listen to music they expect to like, while avoiding genres they dislike. Therefore, most of the songs that would get a bad rating are not voluntarily rated by the users. Since people rarely listen to random songs, or rarely watch random movies, we should expect to observe in many areas a difference between the distribution of ratings for random items and the corresponding distribution for the items selected by the users."

Various frameworks such as *asymmetric factor models* and *SVD++* have been proposed to incorporate implicit feedback. These algorithms use two different item factor matrices V

and Y, corresponding to explicit and implicit feedback, respectively. The user latent factors are either wholly or partially derived using a linear combination of those rows of the (implicit) item latent factor matrix Y that correspond to rated items of the user. The idea is that user factors correspond to user preferences, and user preferences should therefore be influenced by the items they have chosen to rate. In the simplest version of asymmetric factor models, a linear combination of the (implicit) factor vectors of the rated items is used to create the user factors. This results in an asymmetric approach in which we no longer have independent variables for user factors. Instead, we have *two* sets of independent *item* factors (i.e., explicit and implicit), and user factors are derived as a linear combination of the implicit item factors. Many variants [311] of this methodology are discussed in the literature, although the original idea is credited to Paterek [473]. The SVD++ model further combines this asymmetric approach with (explicit) user factors and a traditional factorization framework. The asymmetric approach can, therefore, be viewed as a simplified precursor to SVD++. For clarity in exposition, we will first discuss the asymmetric model briefly.

Asymmetric Factor Models: To capture the implicit feedback information, we first derive an *implicit feedback matrix* from the explicit ratings matrix. For an $m \times n$ ratings matrix R, the $m \times n$ implicit feedback matrix $F = [f_{ij}]$ is defined by setting it to 1, if the value r_{ij} is observed, and 0, if it is missing. The feedback matrix F is subsequently normalized so that the L_2-norm of each row is 1. Therefore, if I_i is the set of indices of the items rated by user i, then each nonzero entry in the ith row is $1/\sqrt{|I_i|}$. An example of a ratings matrix R together with its corresponding implicit feedback matrix F is illustrated below:

$$\underbrace{\begin{pmatrix} 1 & -1 & 1 & ? & 1 & 2 \\ ? & ? & -2 & ? & -1 & ? \\ 0 & ? & ? & ? & ? & ? \\ -1 & 2 & -2 & ? & ? & ? \end{pmatrix}}_{R} \Rightarrow \underbrace{\begin{pmatrix} 1/\sqrt{5} & 1/\sqrt{5} & 1/\sqrt{5} & 0 & 1/\sqrt{5} & 1/\sqrt{5} \\ 0 & 0 & 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 1/\sqrt{1} & 0 & 0 & 0 & 0 & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} & 0 & 0 & 0 \end{pmatrix}}_{F}$$

An $n \times k$ matrix $Y = [y_{ij}]$ is used as the implicit item-factor matrix and the matrix F provides the linear combination coefficients to create a user-factor matrix from it. The variables in Y encode the propensity of each factor-item combination to contribute to implicit feedback. For example, if $|y_{ij}|$ is large, then it means that simply the *act of rating* item i contains significant information about the affinity of that *action* for the jth latent component, no matter what the actual value of the rating might be. In the simplified asymmetric model, user factors are encoded as linear combinations of the implicit item factors of rated items; the basic idea is that linear combinations of user *actions* are used to define their preferences (factors). Specifically, the matrix product FY is an $m \times k$ user-factor matrix, and each (user-specific) row in it is a (user-specific) linear combination of implicit item factors depending on the items rated by the user. The matrix FY is used in lieu of the user-factor matrix U, and the ratings matrix is factorized as $R \approx [FY]V^T$, where V is the $n \times k$ *explicit* item-factor matrix. If desired, bias variables can be incorporated in the model by mean-centering the ratings matrix and appending two additional columns to each of Y and V, as discussed in section 3.6.4.5 (see Exercise 13).

This simple approach often provides excellent[11] results because it reduces the redundancy in user factors by deriving them as linear combinations of item factors. The basic

[11]In many cases, this approach can outperform SVD++, especially when the number of observed ratings is small.

idea here is that two users will have similar user factors if they have rated similar items, *irrespective of the values of the ratings.* Note that the $n \times k$ matrix Y contains fewer parameters than an $m \times k$ user-factor matrix U because $n \ll m$. Another advantage of this approach is that it is possible to incorporate other types of *independent* implicit feedback (such as buying or browsing behavior) by incorporating it in the implicit feedback matrix F. In such cases, the approach can usually do better than most other forms of matrix factorization (with explicit ratings) because of its ability to use *both* explicit and implicit ratings. Nevertheless, even in cases where no independent implicit feedback is available, this model seems to be perform better than straightforward variations of matrix factorization for very sparse matrices with a large number of users (compared to the number of items). An additional advantage of this model is that no user parameterizations are needed; therefore, the model can work well for out-of-sample users, although it cannot be used for out-of-sample items. In other words, the model is at least partially inductive unlike most matrix factorization methods. We omit discussing the gradient-descent steps of this model, because the generalization of this model is discussed in the next section. The corresponding steps are, nevertheless, enumerated in the problem statement of Exercise 13.

The item-based parametrization of asymmetric factor models also provides it the merit of *explainability.* Note that one can re-write the factorization $[FY]V^T$ as $F[YV^T]$. The matrix YV^T can be viewed as an $n \times n$ item-to-item prediction matrix in which $[YV^T]_{ij}$ tells us how much the act of rating item i contributes to the predicted rating of item j. The matrix F provides the corresponding $m \times n$ user-to-item coefficients and, therefore, multiplying F with $[YV^T]$ provides user-to-item predictions. Therefore, one can now explain, which items previously consumed/rated by the user contribute most to the prediction in $F[YV^T]$. This type of explainability is inherent to item-centric models.

SVD++: The derivation of user factors *purely* on the basis of the identities of rated items seems like a rather extreme use of implicit feedback in asymmetric factor models. This is because such an approach does not discriminate *at all* between pairs of users who have rated exactly the same set of items but have very different observed values of the ratings. Two such users will receive exactly the same rating prediction for an item that is not rated by both.

In SVD++, a more nuanced approach is used. The implicit user-factor matrix FY is used only to *adjust* the explicit user-factor matrix U rather than to create it. Therefore, FY needs to be added to U before multiplying with V^T. Then, the reconstructed $m \times n$ ratings matrix R is given by $(U + FY)V^T$, and the implicit feedback component of the predicted rating is given by $(FY)V^T$. The price for the additional modeling flexibility in SVD++ is that the number of parameters is increased, which can cause overfitting in very sparse ratings matrices. The implicit feedback matrix can be derived from the ratings matrix (as in asymmetric factor models), although other forms of implicit feedback (e.g., buying or browsing behavior) can also be included.

The user and item biases are included in this model in a manner similar to section 3.6.4.5. We can assume, without loss[12] of generality, that the ratings matrix is mean-centered around the global mean μ of all the entries. Therefore, we will work with $m \times (k+2)$ and $n \times (k+2)$ factor matrices U and V, respectively, in which the last two columns contain either 1s or bias variables according to section 3.6.4.5. We also assume[13] that Y is an $n \times (k+2)$ matrix,

[12]For matrices, which are not mean-centered, the global mean can be subtracted during preprocessing and then added back at prediction time.

[13]We use a slightly different notation than the original paper [309], although the approach described here is equivalent. This presentation simplifies the notation by introducing fewer variables and viewing bias

and the last two columns of Y contain 0s. This is because the bias component is already addressed by the last two columns of U, but we need the last two dummy columns in Y to ensure that we can add U and FY as matrices of the same dimensions. Therefore, the predicted rating $\hat{r}_{ij}$ can be expressed in terms of these variables as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k+2} (u_{is} + [FY]_{is}) \cdot v_{js} \tag{3.20}$$

$$= \sum_{s=1}^{k+2} \left(u_{is} + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right) \cdot v_{js} \tag{3.21}$$

The first term $\sum_{s=1}^{k+2} u_{is} v_{js}$ on the right-hand side of the aforementioned equation is the (i,j)th term of UV^T, and the second term $\sum_{s=1}^{k+2} \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} v_{js}$ is the (i,j)th term of $[FY]V^T$. Note that the (i,s)th entry of $[FY]$ is given by $\sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}}$. One can view this model as a combination of the unconstrained matrix factorization model (with biases) and the asymmetric factorization model discussed in the previous section. Therefore, it combines the strengths of both models.

The corresponding optimization problem, which minimizes the aggregate squared error $e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2$ over all observed entries (denoted by set S) in the ratings matrix, may be stated as follows:

$$\text{Min. } J = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k+2} \left[u_{is} + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right] \cdot v_{js} \right)^2 + \frac{\lambda}{2} \sum_{s=1}^{k+2} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 + \sum_{j=1}^{n} y_{js}^2 \right)$$

subject to:

$(k+2)$th column of U contains only 1s

$(k+1)$th column of V contains only 1s

Last two columns of Y contain only 0s

Note that this optimization formulation is different from that in the previous section in terms of its having an implicit feedback term together with its regularizer. One can use the partial derivative of this objective function to derive the update rules for matrices U and V, as well as the variables in Y. The update rules are then expressed in terms of the error values $e_{ij} = r_{ij} - \hat{r}_{ij}$ of the observed entries. The following updates[14] may be used for each

variables as constraints on the factorization process.

[14]The literature often describes these updates in vectorized form. These updates may be applied to the rows of U, V, and Y as follows:

$$\overline{u_i} \Leftarrow \overline{u_i} + \alpha(e_{ij}\overline{v_j} - \lambda \overline{u_i})$$

$$\overline{v_j} \Leftarrow \overline{v_j} + \alpha \left(e_{ij} \cdot \left[\overline{u_i} + \sum_{h \in I_i} \frac{\overline{y_h}}{\sqrt{|I_i|}} \right] - \lambda \cdot \overline{v_j} \right)$$

$$\overline{y_h} \Leftarrow \overline{y_h} + \alpha \left(\frac{e_{ij} \cdot \overline{v_j}}{\sqrt{|I_i|}} - \lambda \cdot \overline{y_h} \right) \quad \forall h \in I_i$$

Reset perturbed entries in fixed columns of U, V, and Y

observed entry $(i,j) \in S$ in the ratings matrix:

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \dots k+2\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(e_{ij} \cdot \left[u_{iq} + \sum_{h \in I_i} \frac{y_{hq}}{\sqrt{|I_i|}} \right] - \lambda \cdot v_{jq} \right) \quad \forall q \in \{1 \dots k+2\}$$

$$y_{hq} \Leftarrow y_{hq} + \alpha \left(\frac{e_{ij} \cdot v_{jq}}{\sqrt{|I_i|}} - \lambda \cdot y_{hq} \right) \quad \forall q \in \{1 \dots k+2\}, \forall h \in I_i$$

Reset perturbed entries in fixed columns of U, V, and Y

The updates are executed by repeatedly looping over all the observed ratings in S. The perturbed entries in the fixed columns of U, V, and Y are reset by these rules to either 1s and 0s. A more efficient (and practical) alternative would be to simply not update the fixed entries by keeping track of them during the update. Furthermore, these columns are always initialized to fixed values that respect the constraints of the optimization model. The nested loop structure of stochastic-gradient descent is similar across the family of matrix factorization methods. Therefore, the basic framework described in Figure 3.9 may be used, although the updates are based on the aforementioned discussion. Better results may be obtained by using different regularization parameters for different factor matrices. A fast variation of stochastic gradient descent is described in [151]. It is also possible to develop an alternating least-squares approach to solve the aforementioned problem (see Exercise 12). Although this model is referred to as SVD++ [309], the name is slightly misleading because the basis vectors of the factorized matrices are not orthogonal. Indeed, the term "SVD" is often loosely applied in the literature on latent factor models. In the next section, we will discuss the use of singular value decomposition with orthogonal vectors.

3.6.5 Singular Value Decomposition

Singular value decomposition (SVD) is a form of matrix factorization in which the columns of U and V are constrained to be mutually orthogonal. Mutual orthogonality has the advantage that the concepts can be completely independent of one another, and they can be geometrically interpreted in scatterplots. However, the semantic interpretation of such a decomposition is generally more difficult, because these latent vectors contain both positive and negative quantities, and are constrained by their orthogonality to other concepts. For a *fully specified matrix*, it is relatively easy to perform SVD with the use of eigendecomposition methods. We will first briefly recap the discussion on singular value decomposition in section 2.5.1.2 of Chapter 2.

Consider the case in which the ratings matrix is fully specified. One can *approximately* factorize the ratings matrix R by using *truncated* SVD of rank $k \ll \min\{m,n\}$. Truncated SVD is computed as follows:

$$R \approx Q_k \Sigma_k P_k^T \tag{3.22}$$

Here, Q_k, Σ_k, and P_k are matrices of size $m \times k$, $k \times k$, and $n \times k$, respectively. The matrices Q_k and P_k respectively contain the k largest eigenvectors of RR^T and R^TR, whereas the (diagonal) matrix Σ_k contains the (non-negative) square roots of the k largest eigenvalues of either matrix along its diagonal. It is noteworthy that the nonzero eigenvalues of RR^T and R^TR are the same, even though they will have a different number of zero eigenvalues when $m \neq n$. The matrix P_k contains the top eigenvectors of R^TR, which is the *reduced* basis representation required for dimensionality reduction of the row space. These eigenvectors contain information about the directions of item-item correlations among ratings, and therefore they provide the ability to represent each user in a reduced number of dimensions in a rotated axis system. For example, in Figure 3.6, the top eigenvector corresponds to the latent vector representing the dominant directions of item-item correlations. Furthermore, the matrix $Q_k\Sigma_k$ contains the transformed and reduced $m \times k$ representation of the original ratings matrix in the basis corresponding to P_k. Therefore, in Figure 3.6, the matrix $Q_k\Sigma_k$ would be a 1-dimensional column vector containing the coordinates of the ratings along the dominant latent vector.

It is easy to see from Equation 3.22 that SVD is inherently defined as a matrix factorization. Of course, the factorization here is into *three* matrices rather than *two*. However, the diagonal matrix Σ_k can be absorbed in either the user factors Q_k or the item factors P_k. By convention, the user factors and item factors are defined as follows:

$$U = Q_k\Sigma_k$$
$$V = P_k$$

As before, the factorization of the ratings matrix R is defined as $R = UV^T$. As long as the user and item factor matrices have orthogonal columns, it is easy to convert the resulting factorization into a form that is compliant with SVD (see Exercise 9). Therefore, the goal of the factorization process is to discover matrices U and V with orthogonal columns. Therefore, SVD can be formulated as the following optimization problem over the matrices U and V:

$$\begin{aligned}\text{Minimize } J = \frac{1}{2}||R - UV^T||^2 \\ \text{subject to:} \\ \text{Columns of } U \text{ are mutually orthogonal} \\ \text{Columns of } V \text{ are mutually orthogonal}\end{aligned}$$

It is easy to see that the only difference from the case of unconstrained factorization is the presence of orthogonality constraints. In other words, the same objective function is being optimized over a smaller space of solutions compared to unconstrained matrix factorization. Although one would expect that the presence of constraints would increase the error J of the approximation, it turns out that the optimal value of J is identical in the case of SVD and unconstrained matrix factorization, if the matrix R is fully specified and regularization is not used. Therefore, for fully specified matrices, the optimal solution to SVD is one of the alternate optima of unconstrained matrix factorization. This is not necessarily true in the cases in which R is not fully specified, and the objective function $J = \frac{1}{2}||R - UV^T||^2$ is computed *only over the observed entries.* In such cases, unconstrained matrix factorization will typically provide lower error on the observed entries. However, the relative performance on the unobserved entries can be unpredictable because of varying levels of *generalizability* of different models.

3.6.5.1 A Simple Iterative Approach to SVD

In this section, we discuss how to solve the optimization problem when the matrix R is incompletely specified. The first step is to mean-center each row of R by subtracting the average rating μ_i of the user i from it. These row-wise averages are stored because they will eventually be needed to reconstruct the raw ratings of the missing entries. Let the centered matrix be denoted by R_c. Then, the missing entries of R_c are set to 0. This approach effectively sets the missing entries to the average rating of the corresponding user, because the missing entries of the centered matrix are set to 0. SVD is then applied to R_c to obtain the decomposition $R_c = Q_k \Sigma_k P_k^T$. The resulting user factors and item factors are given by $U = Q_k \Sigma_k$ and $V = P_k$. Let the ith row of U be the k-dimensional vector denoted by $\overline{u_i}$ and the jth row of V be the k-dimensional vector denoted by $\overline{v_j}$. Then, the rating $\hat{r}_{ij}$ of user i for item j is estimated as the following adjusted dot product of $\overline{u_i}$ and $\overline{v_j}$:

$$\hat{r}_{ij} = \overline{u_i} \cdot \overline{v_j} + \mu_i \tag{3.23}$$

Note that the mean μ_i of user i needs to added to the estimated rating to account for the mean-centering applied to R in the first step.

The main problem with this approach is that the substitution of missing entries with row-wise means can lead to considerable bias. A specific example of how column-wise mean substitution leads to bias is provided in section 2.5.1 of Chapter 2. The argument for row-wise substitution is exactly similar. There are several ways of reducing this bias. One of the methods is to use maximum-likelihood estimation [24, 472], which is discussed in section 2.5.1.1 of Chapter 2. Another approach is to use a method, which reduces the bias iteratively by improving the estimation of the missing entries. The approach uses the following steps:

1. **Initialization:** Initialize the missing entries in the ith row of R to be the mean μ_i of that row to create R_f.
2. **Iterative step 1:** Perform rank-k SVD of R_f in the form $Q_k \Sigma_k P_k^T$.
3. **Iterative step 2:** Readjust only the (originally) missing entries of R_f to the corresponding values in $Q_k \Sigma_k P_k^T$. Go to iterative step 1.

The iterative steps 1 and 2 are executed to convergence. In this method, although the initialization step causes bias in the early SVD iterations, later iterations tend to provide more robust estimates. This is because the matrix $Q_k \Sigma_k P_k^T$ will differ from R to a greater degree in the biased entries. The final ratings matrix is then given by $Q_k \Sigma_k P_k^T$ at convergence.

The approach can become stuck in a local optimum when the number of missing entries is large. In particular, the local optimum at convergence can be sensitive to the choice of initialization. It is also possible to use the baseline predictor discussed in section 3.7.1 to perform more robust initialization. The idea is to compute an initial predicted value B_{ij} for user i and item j with the use of *learned* user and item biases. This approach is equivalent to applying the method in section 3.6.4.5 at $k = 0$, and then adding the bias of user i to that of item j to derive B_{ij}. The value of B_{ij} is subtracted from each observed entry (i, j) in the ratings matrix, and missing entries are set to 0 at initialization. The aforementioned iterative approach is applied to this adjusted matrix. The value of B_{ij} is added back to entry (i, j) at prediction time. Such an approach tends to be more robust because of better initialization.

Regularization can be used in conjunction with the aforementioned iterative method. The idea is to perform regularized SVD of R_f in each iteration rather than using only vanilla SVD. Because the matrix R_f is fully specified in each iteration, it is relatively easy to apply regularized SVD methods to these intermediate matrices. Regularized singular value decomposition methods for complete matrices are discussed in [541]. The optimal values of the regularization parameters λ_1 and λ_2 are chosen adaptively by using either the hold-out or the cross-validation methods.

3.6.5.2 An Optimization-Based Approach

The iterative approach is quite expensive because it works with fully specified matrices. It is simple to implement for smaller matrices but does not scale well in large-scale settings. A more efficient approach is to add orthogonality constraints to the optimization model of the previous sections. A variety of gradient-descent methods can be used for solve the model. Let S be the set of specified entries in the ratings matrix. The optimization problem (with regularization) is stated as follows:

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda_1}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda_2}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

subject to:

Columns of U are mutually orthogonal

Columns of V are mutually orthogonal

The primary difference of this model from unconstrained matrix factorization is the addition of orthogonality constraints, which makes the problem more difficult. For example, if one tries to directly use the update equations of the previous section on unconstrained matrix factorization, the orthogonality constraints will be violated. However, a variety of modified update methods exist to handle this case. For example, one can use a *projected gradient descent* [76] method, wherein all components of a particular column of U or V are updated at one time. In projected gradient descent, the descent direction for the pth column of U (or V), as indicated by the equations of the previous section, is projected in a direction that is orthogonal to the first $(p-1)$ columns of U (or V). For example, the implementation of section 3.6.4.3 can be adapted to learn orthogonal factors by projecting each factor in a direction orthogonal to those learned so far at each step. One can easily incorporate user and item biases by computing the baseline predictions B_{ij} (discussed in the previous section) and subtracting them from the observed entries in the ratings matrix before modeling. Subsequently, the baseline values can be added back to the predicted values as a postprocessing step.

3.6.5.3 Out-of-Sample Recommendations

Many matrix completion methods like matrix factorization are inherently *transductive*, in which predictions can be made only for users and items already included in the ratings matrix at the time of training. It is often not easy to make predictions for new users and items from the factors U and V, if they were not included in the original ratings matrix R at factorization time. One advantage of orthogonal basis vectors is that they can be leveraged more easily to perform out-of-sample recommendations for new users and items. This problem is also referred to as *inductive matrix completion.*

The geometric interpretation provided in Figure 3.6 is helpful in understanding why orthogonal basis vectors are helpful in predicting missing ratings. Once the latent vectors have been obtained, one can project the information in the specified ratings on the corresponding latent vectors; this is much easier when the vectors are mutually orthogonal. Consider a situation where SVD has obtained latent factors U and V, respectively. The columns of V define a k-dimensional hyperplane, $\mathcal{H}_1$, passing through the origin. In Figure 3.6, the number of latent factors is 1, and therefore the single latent vector (i.e., 1-dimensional hyperplane) is shown. If two factors had been used, it would have been a plane.

Now imagine a new user whose ratings have been added into the system. Note that this new user is not represented in the latent factors in U or V. Consider the scenario in which the new user has specified a total of h ratings. The space of possibilities of ratings for this user is an $(n-h)$-dimensional hyperplane in which h values are fixed. An example is illustrated in Figure 3.6, where one rating for *Spartacus* is fixed, and the hyperplane is defined on the other two dimensions. Let this hyperplane be denoted by $\mathcal{H}_2$. The goal is then to determine the point on $\mathcal{H}_2$, which is as close to $\mathcal{H}_1$ as possible. That point on $\mathcal{H}_2$ yields the values of all the other ratings. Three possibilities arise:

1. *$\mathcal{H}_1$ and $\mathcal{H}_2$ do not intersect:* The point on $\mathcal{H}_2$ that is closest to $\mathcal{H}_1$ is returned. The smallest distance between a pair of hyperplanes can be formulated as a simple sum-of-squares optimization problem.

2. *$\mathcal{H}_1$ and $\mathcal{H}_2$ intersect at a unique point:* This case is similar to that of Figure 3.6. In that case, the values of the ratings of the intersection point can be used.

3. *$\mathcal{H}_1$ and $\mathcal{H}_2$ intersect on an t-dimensional hyperplane, where $t \geq 1$:* All ratings, which are as close as possible to the t-dimensional hyperplane, should be found. The average values of the ratings of the corresponding users are returned. Note that this approach combines latent factor and neighborhood methods. The main difference from neighborhood methods is that the neighborhood is discovered in a more refined way with the use of feedback from latent factor models.

Orthogonality has significant advantages in terms of geometric interpretability. The ability to discover out-of-sample recommendations is one example of such an advantage.

3.6.5.4 Example of Singular Value Decomposition

In order to illustrate the use of singular value decomposition, let us apply this approach to the example of Table 3.2. We will use the iterative approach of estimating the missing entries repeatedly. The first step is to fill in the missing entries with the average of each row. As a result, the filled in ratings matrix R_f becomes:

$$R_f = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -0.2 & -1 & -1 & -1 \\ 0 & 1 & 1 & -1 & -1 & 0 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & 0.2 & -1 & 1 & 1 & 1 \end{pmatrix}$$

Upon applying rank-2 truncated SVD to the matrix, and absorbing the diagonal matrix within the user factors, we obtain the following:

$$R_f \approx \begin{pmatrix} 1.129 & -2.152 \\ 1.937 & 0.640 \\ 1.539 & 0.873 \\ -2.400 & -0.341 \\ -2.105 & 0.461 \end{pmatrix} \begin{pmatrix} 0.431 & 0.246 & 0.386 & -0.518 & -0.390 & -0.431 \\ -0.266 & 0.668 & -0.249 & 0.124 & -0.578 & 0.266 \end{pmatrix}$$

$$= \begin{pmatrix} 1.0592 & -1.1604 & 0.9716 & -0.8515 & 0.8040 & -1.0592 \\ 0.6636 & 0.9039 & 0.5881 & -0.9242 & -1.1244 & -0.6636 \\ 0.4300 & 0.9623 & 0.3764 & -0.6891 & -1.1045 & -0.4300 \\ -0.9425 & -0.8181 & -0.8412 & 1.2010 & 1.1320 & 0.9425 \\ -1.0290 & -0.2095 & -0.9270 & 1.1475 & 0.5535 & 1.0290 \end{pmatrix}$$

Note that even after the first iteration, a reasonable estimate is obtained of the missing entries. In particular, the estimated values are $\hat{r}_{23} \approx 0.5581$, $\hat{r}_{31} \approx 0.43$, $\hat{r}_{36} \approx -0.43$, and $\hat{r}_{52} \approx -0.2095$. Of course, these entries are biased by the fact that the initial filled-in entries were based on the row averages, and thus did not accurately reflect the correct values. Therefore, in the next iteration, we fill in these four missing values in the original matrix to obtain the following matrix:

$$R_f = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 0.5581 & -1 & -1 & -1 \\ 0.43 & 1 & 1 & -1 & -1 & -0.43 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -0.2095 & -1 & 1 & 1 & 1 \end{pmatrix}$$

This matrix is still biased, but it is better than filling in missing entries with row averages. In the next iteration, we apply SVD with this new matrix, which is clearly a better starting point. Upon applying the entire process of rank-2 SVD again, we obtain the following matrix in the next iteration:

$$R_f = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 0.9274 & -1 & -1 & -1 \\ 0.6694 & 1 & 1 & -1 & -1 & -0.6694 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -0.5088 & -1 & 1 & 1 & 1 \end{pmatrix}$$

Note that the newly estimated entries have further changed in the next iteration. The new estimated values are $\hat{r}_{23} \approx 0.9274$, $\hat{r}_{31} \approx 0.6694$, $\hat{r}_{36} \approx -0.6694$, and $\hat{r}_{52} \approx -0.5088$. Furthermore, the entries have changed to a smaller degree than in the first iteration. Upon applying the process for one more iteration to the latest value of R_f, we obtain the following:

$$R_f = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 0.9373 & -1 & -1 & -1 \\ 0.7993 & 1 & 1 & -1 & -1 & -0.7993 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -0.6994 & -1 & 1 & 1 & 1 \end{pmatrix}$$

The estimated values are now $\hat{r}_{23} \approx 0.9373$, $\hat{r}_{31} \approx 0.7993$, $\hat{r}_{36} \approx -0.7993$, and $\hat{r}_{52} \approx -0.6994$. Note that the change is even smaller than in the previous iteration. In fact, the change in entry $\hat{r}_{23}$ is very small. Over successive iterations, the changes in the entries tend to become smaller and smaller, until convergence is reached. The resulting entries can be used as the predicted values. A large number of iterations are typically not required in the process. In fact, for *ranking* the items for a given user, only 5 to 10 iterations might be sufficient. In this particular example, one can correctly rank the two missing ratings for user 3 after the very first iteration. The approach can also be applied after mean-centering the rows or columns, or both. This approach has the effect of removing user and item biases before the estimation process. Applying such bias correction methods often has a positive effect on prediction.

The approach is not guaranteed to converge to a global optimum, especially if poor initialization points have been used. This is especially true when a large fraction of the entries in the matrix are missing. In these cases, the initial bias can be significant enough to affect the quality of the final solution. Therefore, it is sometimes advisable to use a simple heuristic, such as a neighborhood model, in order to obtain a first estimate of the missing entries. Choosing such a robust estimate as a starting point will speed up the convergence, and it will also lead to more accurate results. Furthermore, one could easily apply this entire process with regularized singular value decomposition of the filled-in matrices. The main difference is that each iteration uses regularized singular value decomposition of the current matrix, which is filled in with the estimated values. The work in [541] may be used as the relevant subroutine for regularized singular value decomposition.

3.6.6 Non-negative Matrix Factorization

Non-negative matrix factorization (NMF) may be used for ratings matrices that are non-negative. The major advantage of this approach is not necessarily one of accuracy, but that of the high level of interpretability it provides in understanding the user-item interactions. The main difference from other forms of matrix factorization is that the factors U and V must be non-negative. Therefore, the optimization formulation in non-negative matrix factorization is stated as follows:

$$\begin{gathered}\text{Minimize } J = \frac{1}{2}||R - UV^T||^2 \\ \text{subject to:} \\ U \geq 0 \\ V \geq 0\end{gathered}$$

Although non-negative matrix factorization can be used for any non-negative ratings matrix (e.g., ratings from 1 to 5), its greatest interpretability advantages arise in cases in which users have a mechanism to specify a liking for an item, but no mechanism to specify a dislike. Such matrices include unary ratings matrices or matrices in which the non-negative entries correspond to the activity frequency. These data sets are also referred to as *implicit feedback data sets* [260, 457]. Some examples of such matrices are as follows:

1. In customer transaction data, the purchase of an item corresponds to expressing a liking for an item. However, not buying an item does not necessarily imply a dislike because the user might have purchased the item elsewhere or they may not be aware of the item. When amounts are associated with transactions, the matrix R may contain arbitrary non-negative numbers. However, all these numbers specify the degree of liking for an item, but they do not indicate dislike. In other words, the numerical quantities in implicit feedback indicate *confidence*, whereas the numerical quantities in explicit feedback indicate *preference*.

2. Similar to the case of purchasing an item, the browsing of an item may be indicative of a like. In some cases, the frequency of the buying or browsing behavior can be quantified as a non-negative value.

3. In Web click data, the selection of an item corresponds to a unary rating of liking an item.

4. A "*like*" button on Facebook can be considered a mechanism to provide a unary rating for an item.

The implicit feedback setting can be considered the matrix completion analog to the positive-unlabeled (PU) learning problem in classification and regression modeling. In classification and regression modeling, reasonable results can often be obtained by treating the unlabeled entries as belonging to the negative class when the positive class is already known to be a very small minority class. Similarly, a helpful aspect of such matrices and problem settings is that it is often reasonably possible to set the unspecified entries to 0, rather than treat them as missing values. For example, consider a customer transaction data set, in which values indicate quantities purchased by a customer. In such a case, it is reasonable to set a value to 0, when that item has not been bought by the customer. Therefore, in this case, one only has to perform non-negative matrix factorization of a fully specified matrix, which is a standard problem in the machine learning literature. This problem is also referred to as *one class collaborative filtering*. Although some recent works argue that the missing values should not be set to 0 in such cases [260, 457, 467, 468] to reduce bias, a considerable amount of work in the literature shows that reasonably robust solutions can be obtained by treating the missing entries as 0 in the modeling process. This is especially the case when the prior probability of an entry to be 0 is very large. For instance, in the supermarket scenario, a customer would typically never buy the vast majority of items in the store. In such cases, setting the missing values to 0 (in the initial matrix for factorization purposes but not in the final prediction) would result in a small amount of bias, but explicitly treating the entries as unspecified in the initial matrix will lead to greater solution complexity. Unnecessary complexity always leads to overfitting. These effects are especially significant[15] in smaller data sets.

Note that the optimization formulation of non-negative matrix factorization is a constrained optimization formulation, which can be solved using standard methods such as Lagrangian relaxation. Although a detailed derivation of the algorithm used for non-negative matrix factorization is beyond the scope of this book, we refer the reader to [22] for details. Here, we present only a brief discussion of how non-negative matrix factorization is performed.

An iterative approach is used to update the matrices U and V. Let u_{ij} and v_{ij}, respectively, be the (i, j)th entries of the matrices U and V. The following multiplicative update rules for u_{ij} and v_{ij} are used:

$$u_{ij} \Leftarrow \frac{(RV)_{ij} u_{ij}}{(UV^T V)_{ij} + \epsilon} \quad \forall i \in \{1 \dots m\}, \forall j \in \{1 \dots k\} \tag{3.24}$$

$$v_{ij} \Leftarrow \frac{(R^T U)_{ij} v_{ij}}{(VU^T U)_{ij} + \epsilon} \quad \forall i \in \{1 \dots n\}, \forall j \in \{1 \dots k\} \tag{3.25}$$

Here, ϵ is a small value such as 10^{-9} to increase numerical stability. All entries in U and V on the right-hand side of the update equations are fixed to the values obtained at the end of the previous iteration during the course of a particular iteration. In other words, all entries in U and V are updated "simultaneously." Small values are sometimes added to the denominator of the update equations to prevent division by 0. The entries in U and V are initialized to random values in $(0, 1)$, and the iterations are executed to convergence. It is possible to obtain better solutions by performing the initialization in a more judicious way [331, 629].

As in the case of other types of matrix factorization, regularization can be used to improve the quality of the underlying solution. The basic idea is to add the penalties $\frac{\lambda_1 ||U||^2}{2} + \frac{\lambda_2 ||V||^2}{2}$ to the objective function. Here $\lambda_1 > 0$ and $\lambda_2 > 0$ are the regularization

[15]These effects are best understood in terms of the bias-variance trade-off in machine learning [22]. Setting the unspecified values to 0 increases bias, but it reduces variance. When a large number of entries are unspecified, and the prior probability of a missing entry to be 0 is very high, the variance effects can dominate.

parameters. This results in a modification [474] of the update equations as follows:

$$u_{ij} \Leftarrow \max\left\{\left[\frac{(RV)_{ij} - \lambda_1 u_{ij}}{(UV^TV)_{ij} + \epsilon}\right] u_{ij}, 0\right\} \quad \forall i \in \{1 \dots m\}, \forall j \in \{1 \dots k\} \tag{3.26}$$

$$v_{ij} \Leftarrow \max\left\{\left[\frac{(R^TU)_{ij} - \lambda_2 v_{ij}}{(VU^TU)_{ij} + \epsilon}\right] v_{ij}, 0\right\} \quad \forall i \in \{1 \dots n\}, \forall j \in \{1 \dots k\} \tag{3.27}$$

The maximization function is used to impose non-negativity and the small additive term $\epsilon \approx 10^{-9}$ in the denominator is used to ensure numerical stability. The parameters λ_1 and λ_2 can be determined using the same approach as described earlier. Instead of using the gradient-descent method, one can also use alternating least-squares methods in which non-negative linear regression is used. Tikhonov regularization can be used within the regression model to prevent overfitting. Details of the alternating least-squares method for non-negative matrix factorization may be found in [161, 301]. The main challenges with these off-the-shelf methods is their lack of computational efficiency with large ratings matrices, because all entries are treated as observed. In section 3.6.6.3, we will discuss how these issues can be addressed.

3.6.6.1 Interpretability Advantages

The main advantage of non-negative matrix factorization is that a high degree of interpretability is achieved in the solution. It is always useful to pair recommender systems with explanations for the recommendations, and this is provided by non-negative matrix factorization. In order to better understand this point, consider a situation in which the preference matrix contains quantities of items bought by customers. An example of a toy 6×6 matrix with 6 items and 6 customers is illustrated in Figure 3.12. It is clear that there are two classes of products corresponding to dairy products and drinks, respectively. It is clear that the customer buying behavior is highly correlated on the basis of the classes of items although all customers seem to like juice. These classes of items are referred to as *aspects*. The corresponding factor matrices also provide a clear interpretability about the affinity of customers and items to these aspects. For example, customers 1 to 4 like dairy products, whereas customers 4 to 6 like drinks. This is clearly reflected in the 6×2 user-factor matrix U. In this simplified example, we have shown all the factored values in U and V to be integral for visual simplicity. In practice, the optimal values are almost always real numbers. The magnitude of the entry of a user in each of the two columns quantifies her level of interest in the relevant aspect. Similarly, the factor matrix V shows how the items are related to the various aspects. Therefore, in this case, the condition $r_{ij} \approx \sum_{s=1}^{k} u_{is} \cdot v_{js}$ can be semantically interpreted in terms of the $k = 2$ aspects:

$$\begin{aligned} r_{ij} \approx &(\textit{Affinity of user } i \textit{ to dairy aspect}) \times (\textit{Affinity of item } j \textit{ to dairy aspect}) \\ &+ (\textit{Affinity of user } i \textit{ to drinks aspect}) \times (\textit{Affinity of item } j \textit{ to drinks aspect}) \end{aligned}$$

This way of predicting the value of r_{ij} shows a "sum-of-parts" decomposition of the matrix. Each of these parts can also be viewed as a user-item co-cluster. This is also one of the reasons that non-negative matrix factorization is often used in clustering. In practical applications, it is often possible to look at each of these clusters and semantically interpret the associations between users and items. When semantic labels can be manually attached to the various clusters, the factorization process provides a neat explanation of the ratings in terms of the contributions of various semantic "genres" of items.

$$\begin{array}{ll|cccccc}
 & & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUICE} & \text{SODA} & \text{BEER} \\
\hline
\text{DAIRY} & 1 & 2 & 2 & 1 & 2 & 0 & 0 \\
 & 2 & 2 & 3 & 3 & 3 & 0 & 0 \\
 & 3 & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{BOTH} & 4 & 2 & 2 & 2 & 3 & 1 & 1 \\
\text{DRINKS} & 5 & 0 & 0 & 0 & 1 & 1 & 1 \\
 & 6 & 0 & 0 & 0 & 2 & 1 & 2 \\
\end{array}_{\textbf{R}}
\approx
\begin{array}{l|cc}
 & \text{DAIRY} & \text{DRINKS} \\
\hline
1 & 2 & 0 \\
2 & 3 & 0 \\
3 & 1 & 0 \\
4 & 2 & 1 \\
5 & 0 & 1 \\
6 & 0 & 2 \\
\end{array}_{\textbf{U}}
\times
\begin{array}{l|cccccc}
 & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUI CE} & \text{SODA} & \text{BEER} \\
\hline
\text{DAIRY} & 1 & 1 & 1 & 1 & 0 & 0 \\
\text{DRINKS} & 0 & 0 & 0 & 1 & 1 & 1 \\
\end{array}_{\textbf{V}^{\textbf{T}}}$$

Figure 3.12: An example of non-negative matrix factorization

This sum-of-parts decomposition can be represented mathematically as follows. The rank-k matrix factorization UV^T can be decomposed into k components by expressing the matrix product in terms of the k columns $\overline{U_i}$ and $\overline{V_i}$, respectively, of U and V:

$$UV^T = \sum_{i=1}^{k} \overline{U_i}\ \overline{V_i}^T \tag{3.28}$$

Each $m \times n$ matrix $\overline{U_i}\ \overline{V_i}^T$ is a rank-1 matrix that corresponds to an aspect in the data. Because of the interpretable nature of non-negative decomposition, it is easy to map these aspects to clusters. For example, the two latent components of the aforementioned example corresponding to dairy products and drinks, respectively, are illustrated in Figure 3.13. Note that Equation 3.28 decomposes the factorization in terms of the *columns* of U and V, whereas Equation 3.14 is a different way of understanding the factorization in terms of the *rows* of U and V. For a given user-item combination, the rating prediction is given by the sum of the contributions of these aspects, and one can even gain a better understanding of why a rating is predicted in a certain way by the approach.

3.6.6.2 Observations about Factorization with Implicit Feedback

Non-negative matrix factorization is particularly well suited to implicit feedback matrices in which ratings indicate positive preferences. Unlike explicit feedback data sets, it is not possible to ignore the missing entries in the optimization model because of the lack of negative feedback in such data. It is noteworthy that the non-negative matrix factorization model treats missing entries as negative feedback by setting them to 0s. Not doing so would grossly increase the error on the unobserved entries. In order to understand this point, consider a unary ratings matrix in which likes are specified by 1s. The factorization shown in Figure 3.14 will provide 100% accuracy on an *arbitrary* unary matrix when computed only over observed entries. This is because the multiplication of U and V^T in Figure 3.14 leads to a matrix containing only 1s and no 0s. Of course, such a factorization will have very high error on the unobserved entries because many unobserved entries may correspond to negative preferences. This example is a manifestation of overfitting caused by lack of negative feedback data. Therefore, for ratings matrices in which negative preferences are missing and it is known that negative preferences vastly outnumber positive preferences, it is important to treat missing entries as 0s. For example, in a customer transaction data set,

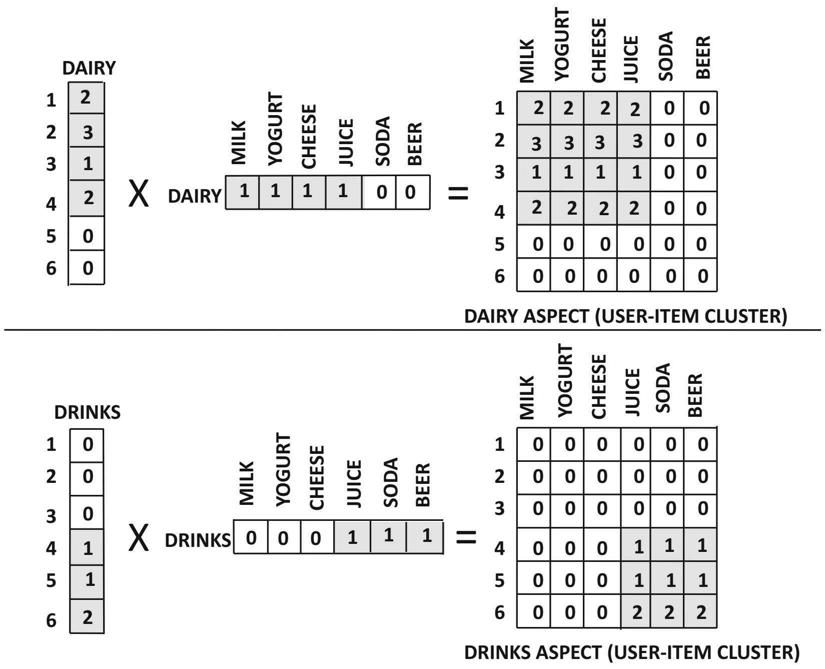

DAIRY
X
=
DAIRY ASPECT (USER-ITEM CLUSTER)
DRINKS
DRINKS ASPECT (USER-ITEM CLUSTER)
MILK
YOGURT
CHEESE
JUICE
SODA
BEER

Figure 3.13: The sum-of-parts interpretation of NMF

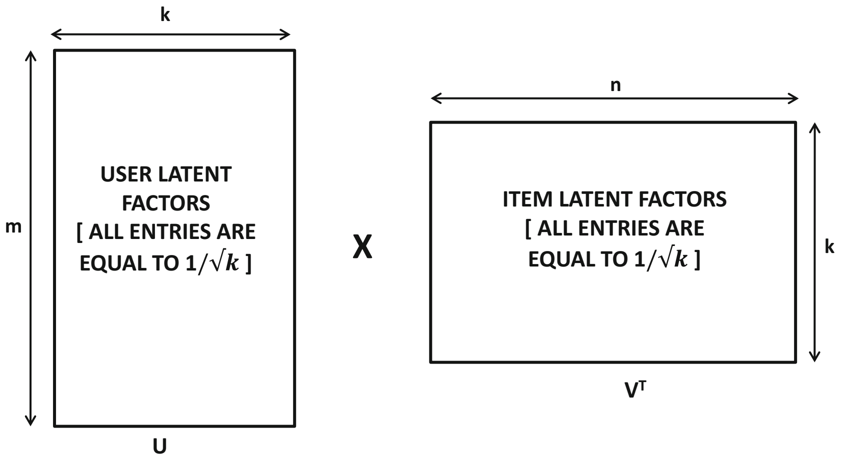

k
m
USER LATENT FACTORS [ALL ENTRIES ARE EQUAL TO $1/\sqrt{k}$]
U
X
n
ITEM LATENT FACTORS [ALL ENTRIES ARE EQUAL TO $1/\sqrt{k}$]
k
V^T

Figure 3.14: Overfitting caused by ignoring missing entries in a unary matrix

if the values indicate the amounts bought by various users and most items are not bought by default, then one can approximate the value of a missing entry as 0.

3.6.6.3 Computational and Weighting Issues with Implicit Feedback

The treatment of missing entries as 0s leads to computational challenges with large matrices. There are several solutions to this dilemma. For example, a sample of the missing entries can be treated as 0s. The gradient-descent solution for the sampled case is similar to that discussed in the next section. It is possible to further improve the accuracy with an ensemble approach. The matrix is factorized multiple times with a different sample of 0s, and each factorization is used to predict (a slightly different) value of the rating. The different predictions of a particular rating are then averaged to create the final result. By using samples of varying sizes, it is also possible to weight the negative feedback entries differently from the positive feedback entries. Such an approach can be important in cost-sensitive settings where false positives and false negatives are weighted differently. Typically, the zero entries should be weighted less than the nonzero entries, and therefore down-sampling the zero entries is useful.

It is also possible to incorporate such weights directly in the objective function and treat all missing entries as 0s. The errors on the zero entries should be weighted less than those on the nonzero entries in the objective function to prevent the zero entries from dominating the optimization. The relative weights can be determined using cross-validation with respect to a particular accuracy measure. Alternatively, the work in [260] suggests the following heuristic to select the weight w_{ij} of entry (i, j):

$$w_{ij} = 1 + \theta \cdot r_{ij} \tag{3.29}$$

Note that all missing values of r_{ij} are treated as 0s in Equation 3.29, and a typical value of θ is 40. This approach also works in settings, where the ratings r_{ij} represent quantities that are bought, rather than binary indicators. In such cases, the weights w_{ij} are computed by treating these quantities as the ratings in Equation 3.29, but the factorized matrix is a derivative binary indicator matrix R_I of the quantity matrix $R = [r_{ij}]$. This indicator matrix R_I is derived from R by copying the zero entries and substituting the nonzero entries with 1s. This approach of weighted factorization of the indicator matrix is therefore slightly different from the example of Figure 3.12, which was presented purely for illustrative purposes.

When working with weighted entries, it is possible to modify stochastic gradient descent methods with weights (cf. section 6.5.2.1 of Chapter 6). However, the problem is that implicit feedback matrices are fully specified, and many of the gradient-descent methods no longer remain computationally viable in large-scale settings. An efficient (weighted) *ALS* method was proposed in [260] for the factorization process in order to avoid the computational challenge of handing the large number of zero entries. Although this approach does not impose non-negativity on the factors, it can be easily generalized to the non-negative setting.

3.6.6.4 Ratings with Both Likes and Dislikes

Our discussion of non-negative matrix factorization so far has focussed only on implicit feedback matrices in which there is a mechanism to specify a liking for an item but no mechanism to specify a dislike. As a result, the underlying "ratings" matrices are always non-negative. Although one can use non-negative matrix factorization for *nominally* non-negative ratings (e.g., from 1 to 5), which explicitly specify both likes and dislikes, there are no special interpretability advantages of using non-negative matrix factorization in such cases. For example, the rating scale might be from 1 to 5, wherein a value of 1 indicates extreme dislike. In this case, one cannot treat the unspecified entries as 0, and one must work only with the set of observed entries. As before, we denote the set of observed entries in the ratings matrix $R = [r_{ij}]$ by S:

$$S = \{(i, j) : r_{ij} \text{ is observed}\} \tag{3.30}$$

The optimization problem (with regularization) is stated in terms of these observed entries as follows:

$$\begin{aligned}\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2 \\ \text{subject to:} \\ U \geq 0 \\ V \geq 0\end{aligned}$$

This formulation is similar to the regularized formulation in unconstrained matrix factorization. The only difference is the addition of the non-negativity constraints. In such cases, modifications are required to the update equations that are used for unconstrained matrix factorization. First, one must initialize the entries of U and V to non-negative values in $(0, 1)$. Then, a similar update can be made, as in the section on unconstrained matrix factorization. In fact, the update equations in section 3.6.4.2 can be used directly. The main modification is to ensure that non-negativity is maintained during updates. If any component of U or V violates the non-negativity constraint as a result of the update, then it is set to 0. The updates are performed to convergence as in all stochastic gradient descent methods.

Other solution methodologies are often used to compute optimal solutions to such models. For example, it is possible to adapt an alternating least-squares approach to non-negative matrix factorization. The main difference is that the coefficients of the least-squares regression are constrained to be non-negative. A wide variety of projected gradient descent, coordinate descent, and nonlinear programming methods are also available to handle such optimization models [76, 357].

In the setting where ratings specify both likes and dislikes, non-negative matrix factorization has no special advantages over unconstrained matrix factorization in terms of interpretability. This is because one can no longer interpret the solution from a sum-of-parts perspective. For example, the addition of three dislike ratings cannot be interpreted as leading to a like rating. Furthermore, because of the addition of non-negativity constraints, the quality of the solution is reduced over that of unconstrained matrix factorization when computed over the observed entries. However, this does not always mean that the quality of the solution will be worse when computed over the unobserved entries. In real settings, positive relationships between users and items are more important than negative relationships between users and items. As a result, non-negativity constraints often introduce a bias which is beneficial in avoiding overfitting. As in the case of unconstrained matrix factorization, one can also incorporate user and item biases to further improve the generalization performance.

3.6.7 Understanding the Matrix Factorization Family

It is evident that the various forms of matrix factorization in the previous sections share a lot in common. All of the aforementioned optimization formulations minimize the Frobenius norms of the residual matrix $(R - UV^T)$ subject to various constraints on the factor matrices U and V. Note that the goal of the objective function is to make UV^T approximate the ratings matrix R as closely as possible. The constraints on the factor matrices achieve different interpretability properties. In fact, the broader family of matrix factorization models can use any other objective function or constraint to force a good approximation. This broader family can be written as follows:

$$\begin{aligned}
\text{Optimize } J = [&\text{Objective function quantifying matching between } R \text{ and } UV^T]\\
&\text{subject to:}\\
&\qquad \text{Constraints on } U \text{ and } V
\end{aligned}$$

The objective function of a matrix factorization method is sometimes referred to as the loss function, when it is in minimization form. Note that the optimization formulation may be either a minimization or a maximization problem, but the goal of the objective function is always to force R to match UV^T as closely as possible. The Frobenius norm is an example of a minimization objective, and some probabilistic matrix factorization methods use a maximization formulation such as the maximum-likelihood objective function. In most cases, regularizers are added to the objective function to prevent overfitting. The various constraints often impose different types of interpretability on the factors. Two examples of such interpretability are orthogonality (which provides geometric interpretability) and non-negativity (which provides sum-of-parts interpretability). Furthermore, even though constraints increase the error on the observed entries, they can sometimes improve the errors on the unobserved entries when they have a meaningful semantic interpretation. This is because constraints reduce the variance[16] on the unobserved entries while increasing bias. As a result, the model has better *generalizability*. For example, fixing the entries in a column in each of U and V to ones almost always results in better performance (cf. section 3.6.4.5). Selecting the right constraints to use is often data-dependent and requires insights into the application-domain at hand.

Other forms of factorization exist in which one can assign probabilistic interpretability to the factors. For example, consider a scenario in which a non-negative unary ratings matrix R is treated as a relative frequency distribution, whose entries sum to 1.

$$\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} = 1 \tag{3.31}$$

Note that it is easy to scale R to sum to 1 by dividing it with the sum of its entries. Such a matrix can be factorized in a similar way to SVD:

$$\begin{aligned}
R &\approx (Q_k \Sigma_k) P_k^T\\
&= UV^T
\end{aligned}$$

As in SVD, the diagonal matrix Σ_k is absorbed in the user factor matrix $U = Q_k \Sigma_k$, and the item factor matrix V is set to P_k. The main difference from SVD is that the columns of Q_k and P_k are not orthogonal, but they are non-negative values summing to 1. Furthermore, the entries of the diagonal matrix Σ_k are non-negative and they also sum to 1.

[16]Refer to Chapter 6 for a discussion of the bias-variance trade-off in collaborative filtering.

Table 3.3: The family of matrix factorization methods

Method	**Constraints**	**Objective**	**Advantages/Disadvantages**
Unconstrained	No constraints	Frobenius + regularizer	Highest quality solution Good for most matrices Regularization prevents overfitting Poor interpretability
SVD	Orthogonal Basis	Frobenius + regularizer	Good visual interpretability Out-of-sample recommendations Good for dense matrices Poor semantic interpretability Suboptimal in sparse matrices
Max. Margin	No constraints	Hinge loss + margin regularizer	Highest quality solution Resists overfitting Similar to unconstrained Poor interpretability Good for discrete ratings
NMF	Non-negativity	Frobenius + regularizer	Good quality solution High semantic interpretability Loses interpretability with both like/dislike ratings Less overfitting in some cases Best for implicit feedback
PLSA	Non-negativity	Maximum Likelihood + regularizer	Good quality solution High semantic interpretability Probabilistic interpretation Loses interpretability with both like/dislike ratings Less overfitting in some cases Best for implicit feedback

Such a factorization has a probabilistic interpretation; the matrices Q_k, P_k and Σ_k contain the probabilistic parameters of a generative process that creates the ratings matrix. The objective function learns the parameters of this generative process so that the likelihood of the generative process creating the ratings matrix is as large as possible. Therefore, the objective function is in *maximization form.* Interestingly, this method is referred to as *Probabilistic Latent Semantic Analysis (PLSA)*, and it can be viewed as a probabilistic variant of non-negative matrix factorization. Clearly, the probabilistic nature of this factorization provides it with a different type of interpretability. A detailed discussion of PLSA may be found in [22]. In many of these formulations, optimization techniques such as gradient descent (or ascent) are helpful. Therefore, most of these methods use very similar ideas in terms of formulating the optimization problem and the underlying solution methodology.

Similarly, maximum margin factorization [180, 500, 569, 624] borrows ideas from *support vector machines* to add a maximum margin regularizer to the objective function and some of its variants [500] are particularly effective for discrete ratings. This approach shares a number of conceptual similarities with the regularized matrix factorization method discussed in section 3.6.4. In fact, the maximum margin regularizer is not very different than that used in unconstrained matrix factorization. However, *hinge loss* is used to quantify the errors in the approximation, rather than the Frobenius norm. While it is beyond the scope of this book to discuss these variants in detail, a discussion may be found in [500, 569]. The focus on maximizing the margin often provides higher quality factorization than some of the other models in the presence of overfitting-prone data. In Table 3.3, we have provided a list of various factorization models and their characteristics. In most cases, the addition of constraints such as non-negativity can reduce the quality of the underlying solution on the observed entries, because it reduces the space of feasible solutions. This is the reason that unconstrained and maximum margin factorization are expected to have the highest quality of global optima. Nevertheless, since the global optimum cannot be easily found in most cases by the available (iterative) methods, a constrained method can sometimes perform better than an unconstrained method. Furthermore, the accuracy over observed entries may be different from that over unobserved entries because of the effects of overfitting. In fact, non-negativity constraints can sometimes improve the accuracy over unobserved entries in some domains. Some forms of factorization such as NMF cannot be applied to matrices with negative entries. Clearly, the choice of the model depends on the problem setting, the noise in the data, and the desired level of interpretability. There is no single solution that can achieve all these goals. A careful understanding of the problem domain is important for choosing the correct model.

3.7 Integrating Factorization and Neighborhood Models

Neighborhood-based methods are generally considered inherently different from other optimization models because of their heuristic nature. Nevertheless, it was shown in section 2.6 of Chapter 2 that neighborhood methods can also be understood in the context of optimization models. This is a rather convenient framework because it paves the way for integration of neighborhood models with other optimization models, such as latent factor models. The approach in [309] integrates the item-wise model of section 2.6.2 in Chapter 2 with the SVD++ model of section 3.6.4.6.

Assume that the ratings matrix R is mean centered. In other words, the global mean μ of the ratings matrix is already subtracted from all the entries, and all predictions will be performed on mean-centered values. The global mean μ can be added back to the predicted values in a postprocessing phase. With this assumption on the ratings matrix $R = [r_{ij}]$, we will re-visit the various portions of the model.

3.7.1 Baseline Estimator: A Non-Personalized Bias-Centric Model

The non-personalized bias-centric model predicts the (mean-centered) ratings in R purely as an addition of user and item biases. In other words, ratings are explained completely by user generosity and item popularity, rather than specific and *personalized* interests of users in items. Let b_i^{user} be the bias variable for user i, and b_j^{item} be the bias variable for item j. Then, the prediction of such a model is as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} \tag{3.32}$$

Let S be the pairs of indices corresponding to the observed entries in the ratings matrix.

$$S = \{(i, j) : r_{ij} \text{ is observed}\} \tag{3.33}$$

Then, b_i^{user} and b_j^{item} can be determined by formulating an objective function over the errors $e_{ij} = r_{ij} - \hat{r}_{ij}$ in the observed entries as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda}{2} \left(\sum_{u=1}^{m} (b_u^{user})^2 + \sum_{j=1}^{n} (b_j^{item})^2 \right)$$

This optimization problem can be solved via gradient descent using the following update rules over each observed entry (i, j) in S in a stochastic gradient descent method:

$$b_i^{user} \Leftarrow b_i^{user} + \alpha(e_{ij} - \lambda b_i^{user})$$
$$b_j^{item} \Leftarrow b_j^{item} + \alpha(e_{ij} - \lambda b_j^{item})$$

The basic framework of the gradient-descent method is similar to that in Figure 3.9, except for the differences in the choice of optimization variables and corresponding update steps. Interestingly, a pure bias-centric model can often provide reasonable predictions in spite of its non-personalized nature. This is especially the case when the amount of ratings data is limited. After solving for the values of b_i^{user} and b_j^{item}, we set B_{ij} to the predicted value of $\hat{r}_{ij}$ according to Equation 3.32. This value of B_{ij} is then *treated as a constant* throughout this section rather than as a variable. Therefore, the first step of the integrated model solution is to determine the *constant value* B_{ij} by solving the non-personalized model. This non-personalized model can also be viewed as a *baseline* estimator because B_{ij} is a rough baseline estimate to the values of the rating r_{ij}. In general, subtracting the value of B_{ij} from each observed entry r_{ij} results in a new matrix that can often be estimated more robustly by most of the models discussed in earlier sections and chapters. This section provides a specific example of how neighborhood models may be adjusted with the use of the baseline estimator though its applicability is much broader.

3.7.2 Neighborhood Portion of Model

We replicate the neighborhood-based prediction relationship of Equation 2.29 (cf. section 2.6.2 of Chapter 2) as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} + \frac{\sum_{l \in Q_j(i)} w_{lj}^{item} \cdot (r_{il} - b_i^{user} - b_l^{item})}{\sqrt{|Q_j(i)|}} \tag{3.34}$$

Although the aforementioned equation is that same as Equation 2.29 of Chapter 2, the subscript notations have been changed to ensure consistency with the latent factor models in this section. Here b_i^{user} is the user bias and b_j^{item} is the item bias. The variable w_{lj}^{item} represents the item-item regression coefficient between item l and item j. The set $Q_j(i)$ represents[17] the subset of the K nearest items to item j, that have been rated by user i. Furthermore, one of the occurrences of $b_i^{user} + b_l^{item}$ in Equation 3.34 is replaced with the *constant* value B_{il} (derived using the approach of the previous section). The resulting prediction is as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} + \frac{\sum_{l \in Q_j(i)} w_{lj}^{item} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} \tag{3.35}$$

It is noteworthy that the bias variables b_i^{user} and b_j^{item} are parameters to be optimized, whereas B_{il} is a constant. One can set up an optimization model that sums up the squared errors $e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2$ in addition to regularization terms. A stochastic gradient-descent approach can be used to determine a solution to the neighborhood portion of the model.

[17]Note that we use upper-case variable K to represent the size of the neighborhood that defines $Q_j(i)$. This is a deviation from section 2.6.2 of Chapter 2. We use lower-case variable k to represent the dimensionality of the factor matrices. The values of k and K are generally different.

The resulting gradient-descent steps are as follows:

$$\begin{aligned}
b_i^{user} &\Leftarrow b_i^{user} + \alpha(e_{ij} - \lambda b_i^{user}) \\
b_j^{item} &\Leftarrow b_j^{item} + \alpha(e_{ij} - \lambda b_j^{item}) \\
w_{lj}^{item} &\Leftarrow w_{lj}^{item} + \alpha_2 \left(\frac{e_{ij} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} - \lambda_2 \cdot w_{lj}^{item} \right) \quad \forall l \in Q_j(i)
\end{aligned}$$

This neighborhood model can be enhanced further with implicit feedback by introducing item-item implicit feedback variables c_{lj}. The basic idea is that if item j is rated together with many neighboring items by the same user i, then it should have an impact on the predicted rating $\hat{r}_{ij}$. This impact is irrespective of the actual values of the ratings of these neighbor items of j. This impact is equal to $\frac{\sum_{l \in Q_j(i)} c_{lj}}{\sqrt{|Q_j(i)|}}$. Note that the scaling of the expression with $\sqrt{|Q_j(i)|}$ is done in order to adjust for varying levels of sparsity in different user-item combinations. Then, the neighborhood model with implicit feedback can be written as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} + \frac{\sum_{l \in Q_j(i)} w_{lj}^{item} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} + \frac{\sum_{l \in Q_j(i)} c_{lj}}{\sqrt{|Q_j(i)|}} \tag{3.36}$$

On creating a least-squares optimization model with respect to the error $e_{ij} = r_{ij} - \hat{r}_{ij}$, one can compute the gradient and derive the stochastic gradient-descent steps. This results in the following modified set of updates:

$$\begin{aligned}
b_i^{user} &\Leftarrow b_i^{user} + \alpha(e_{ij} - \lambda b_i^{user}) \\
b_j^{item} &\Leftarrow b_j^{item} + \alpha(e_{ij} - \lambda b_j^{item}) \\
w_{lj}^{item} &\Leftarrow w_{lj}^{item} + \alpha_2 \left(\frac{e_{ij} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} - \lambda_2 \cdot w_{lj}^{item} \right) \quad \forall l \in Q_j(i) \\
c_{lj} &\Leftarrow c_{lj} + \alpha_2 \left(\frac{e_{ij}}{\sqrt{|Q_j(i)|}} - \lambda_2 \cdot c_{lj} \right) \quad \forall l \in Q_j(i)
\end{aligned}$$

The work in [309] assumes a more general framework, in which the implicit feedback matrix is not necessarily derived from only the ratings matrix. For example, a retailer might create the implicit ratings matrix based on users who have browsed, rated, or bought an item. This generalization is relatively straightforward to incorporate in our models by changing the final term of Equation 3.36 to $\frac{\sum_{l \in Q'_j(i)} c_{lj}}{\sqrt{|Q'_j(i)|}}$. Here, $Q'_j(i)$ is the set of closest neighbors of user i (based on explicit ratings), who have also provided some form of implicit feedback for item j. This modification can also be applied to the latent factor portion of the model, although we will consistently work with the simplified assumption that the implicit feedback matrix is derived from the ratings matrix.

3.7.3 Latent Factor Portion of Model

The aforementioned prediction is made on the basis of the neighborhood model. A corresponding latent factor model is introduced in section 3.6.4.6, in which implicit feedback is integrated with ratings information to make predictions. We replicate Equation 3.21 from that section here:

$$\hat{r}_{ij} = \sum_{s=1}^{k+2} \left(u_{is} + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right) \cdot v_{js} \tag{3.37}$$

As in section 3.6.4.6, I_i represents the set of items rated by user i. The $m \times (k+2)$ matrix $Y = [y_{hs}]$ contains the implicit feedback variables, and its construction is described in section 3.6.4.6. Furthermore, the $(k+2)$th column of U contains only 1s, the $(k+1)$th column of V contains only 1s, and the last two columns of Y are 0s. Note that the right-hand side of Equation 3.37 already accounts for the user and item biases. Since the last two columns of the factor matrices contain the bias variables, the component $\sum_{s=1}^{k+2} u_{is} v_{js}$ of Equation 3.37 includes the bias terms.

3.7.4 Integrating the Neighborhood and Latent Factor Portions

One can now integrate the two models in Equations 3.36 and 3.37 to create a single predicted value as follows:

$$\hat{r}_{ij} = \underbrace{\frac{\sum_{l \in Q_j(i)} w_{lj}^{item} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} + \frac{\sum_{l \in Q_j(i)} c_{lj}}{\sqrt{|Q_j(i)|}}}_{\text{Neighborhood Component}} + \underbrace{\sum_{s=1}^{k+2} \left(u_{is} + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right) \cdot v_{js}}_{\text{Latent Factor Comp.+Bias}} \tag{3.38}$$

Note that the initial bias terms $b_i^{user} + b_j^{item}$ of Equation 3.36 are missing here because they are included in the final term corresponding to the latent factor model. The same user and item biases are now shared by both components of the model.

The corresponding optimization problem, which minimizes the aggregated squared error $e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2$ over the entries in (observed set) S is as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda}{2} \sum_{s=1}^{k+2} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 + \sum_{j=1}^{n} y_{js}^2 \right) +$$
$$+ \frac{\lambda_2}{2} \sum_{j=1}^{n} \sum_{l \in \cup_i Q_j(i)} \left[(w_{lj}^{item})^2 + c_{lj}^2 \right]$$

subject to:

$(k+2)$th column of U contains only 1s

$(k+1)$th column of V contains only 1s

Last two columns of Y contain only 0s

The value of $\hat{r}_{ij}$ in the aforementioned objective function can be materialized with the help of Equation 3.38. As in all latent factor models, the sum of squares of the optimization variables are included for regularization. Note that the different parameters λ and λ_2 are used for regularizing the sets of variables from the latent factor model and the neighborhood model, respectively, for better flexibility in the optimization process.

3.7.5 Solving the Optimization Model

As in the case of all the other optimization models discussed in this chapter, a gradient descent approach is used to solve the optimization problem. In this case, the optimization model is rather complex because it contains a relatively large number of terms, and a large number of variables. Nevertheless, the approach for solving the optimization model is exactly the same as in the case of the latent factor model of section 3.6.4.6. A partial derivative with respect to each optimization variable is used to derive the update step. We omit the derivation of the gradient descent steps, and simply state them here in terms of the error values $e_{ij} = r_{ij} - \hat{r}_{ij}$. The following rules may be used for each observed entry $(i, j) \in S$ in the ratings matrix:

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \dots k+2\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha\left(e_{ij} \cdot \left[u_{iq} + \sum_{h \in I_i} \frac{y_{hq}}{\sqrt{|I_i|}}\right] - \lambda \cdot v_{jq}\right) \quad \forall q \in \{1 \dots k+2\}$$

$$y_{hq} \Leftarrow y_{hq} + \alpha\left(\frac{e_{ij} \cdot v_{jq}}{\sqrt{|I_i|}} - \lambda \cdot y_{hq}\right) \quad \forall q \in \{1 \dots k+2\}, \forall h \in I_i$$

$$w_{lj}^{item} \Leftarrow w_{lj}^{item} + \alpha_2\left(\frac{e_{ij} \cdot (r_{il} - B_{il})}{\sqrt{|Q_j(i)|}} - \lambda_2 \cdot w_{lj}^{item}\right) \quad \forall l \in Q_j(i)$$

$$c_{lj} \Leftarrow c_{lj} + \alpha_2\left(\frac{e_{ij}}{\sqrt{|Q_j(i)|}} - \lambda_2 \cdot c_{lj}\right) \quad \forall l \in Q_j(i)$$

Reset perturbed entries in fixed columns of U, V, and Y

The first three updates can also be written in $(k+2)$-dimensional vectorized form. Refer to the section on SVD++ for a footnote containing these updates. We repeatedly loop over all the observed ratings in S with a stochastic gradient descent method. The basic algorithmic framework for stochastic gradient descent is described in Figure 3.9. The value of α regulates the step size for variables associated with the latent factor portion of the model, whereas α_2 regulates the step size for variables associated with the neighborhood portion of the model. The fixed columns of U, V, and Y should not be updated by these rules, according to the constraints in the optimization model. This is achieved in practice by always resetting them to their fixed values at the end of an iteration. Furthermore, these columns are always initialized to their fixed values, as required by the constraints of the optimization model. The regularization parameters can be selected by holding out a fraction of the observed entries during training, and tuning the accuracy on the held out entries. A more effective approach is to use the cross-validation method discussed in Chapter 7. It is particularly important to use different step-sizes and regularization parameters for the neighborhood and latent factor portions of the model to avoid poor performance.

3.7.6 Observations about Accuracy

It was shown in [309] that the combined model provided superior results to those of each of the individual models. This is a result of the ability of the combined model to adapt to varying characteristics of different portions of the data set. The basic idea is similar to that used often in hybrid recommender systems (cf. Chapter 6) for combining different types of models. One can try to approximate the results of the integrated model by using a weighted average of the predictions of the two different component models. The relative weights can be learned using the aforementioned hold-out or cross-validation techniques. However, compared to the averaged model, the integrated model of this section is more powerful. One reason is that the bias variables are shared by the two components, which prevents the overfitting of the bias variables to the specific nuances of each model. Furthermore, the use of the prediction function of Equation 3.38 implicitly regulates the importance of each portion of the model by automatically choosing appropriate values for each of the variables in the optimization process. As a result, this type of integration often provides superior accuracy. However, the model provides only slightly superior performance to that given by SVD++, and the results are data-set dependent. One issue to keep in mind is that the neighborhood model has more parameters to be optimized than SVD++. Significant advantages will not be obtained by the neighborhood component unless the data set is sufficiently large. For smaller data sets, increasing the number of parameters often leads to overfitting. In this sense, the proper choice between asymmetric factor models, pure SVD with biases, SVD++, and neighborhood-integrated factorization, often depends on the size of the data set at hand. More complex models require larger data sets to avoid overfitting. For very small data sets, one would do best with asymmetric factor models. For very large data sets, the neighborhood-integrated factorization model is best. SVD++ generally does better than pure SVD (with biases) in most settings.

3.7.7 Integrating Latent Factor Models with Arbitrary Models

The integration of latent factor models with neighborhood-based models provides useful hints about integrating the former with other types of models such as content-based methods. Such an integration naturally leads to the creation of *hybrid recommender systems*. In general, item profiles may be available in the form of descriptions of products. Similarly, users might have explicitly created profiles describing their interests. Assume that the profile for user i is denoted by the keyword vector $\overline{C}_i^{user}$ and the profile for item j is denoted by the keyword vector $\overline{C}_j^{item}$. Furthermore, assume that the observed ratings of user i are denoted by $\overline{R}_i^{user}$, and the observed ratings of item j are denoted by $\overline{R}_j^{item}$. Then, one can write the following general form of the prediction function:

$$\hat{r}_{ij} = \underbrace{[(U + FY)V^T]_{ij}}_{\text{Latent Factor Portion}} + \beta \underbrace{F(\overline{C}_i^{user}, \overline{C}_j^{item}, \overline{R}_i^{user}, \overline{R}_j^{item})}_{\text{Another Prediction Model}} \tag{3.39}$$

Here, β is a balancing factor that controls the relative importance of the two models. The second term, which is $F(\overline{C}_i^{user}, \overline{C}_j^{item}, \overline{R}_i^{user}, \overline{R}_j^{item})$, is a parameterized function of the user profile, item profile, user ratings, and item ratings. One can optimize the parameters of this function jointly with the latent factors to minimize the error of prediction in Equation 3.39.

The integration of neighborhood and latent factor models can be viewed as a special case of this method in which the function $F()$ is a linear regression function that uses only $\overline{R}_j^{item}$ and ignores all the other arguments. It is, however, possible to design an almost infinite number of variants of this broader approach by varying the choice of function $F()$. It is also possible to broaden the scope of $F()$ by using other sources of data such as social data, location, or time. In fact, virtually any collaborative filtering model, which is posed in the form of a parameterized prediction function, can be integrated with the latent factor model. Many methods have indeed been proposed in the research literature that integrate various types of feature-based regression, topic modeling, or other novel data sources with latent factor models. For example, a social regularization method (cf. section 11.3.8 of Chapter 11) integrates the latent factor model with social trust information to improve predictions. There is significant scope in improving the state of the art in recommender systems by identifying new sources of data, whose predictive power can be integrated with latent factor models using the aforementioned framework.

3.8 Summary

This chapter discusses a number of models for collaborative filtering. The collaborative filtering problem can be viewed as a generalization of the problem of classification. Therefore, many of the models that apply to classification also apply to collaborative filtering with some generalization. A notable exception is that of latent factor models, which are highly tailored to the collaborative filtering problem. Latent factor models use different types of factorization in order to predict ratings. These different types of factorization differ in the nature of their objective functions and the constraints on their basis matrices. Furthermore, they may have different trade-offs in terms of accuracy, overfitting, and interpretability. Latent factor models are the state-of-the-art in collaborative filtering. A wide variety of latent factor models have been proposed, based on the choices of the objective function and optimization constraints. Latent factor models can also be combined with neighborhood methods to create integrated models, which can benefit from the power of both latent factor models and neighborhood methods.

3.9 Bibliographic Notes

The problem of collaborative filtering is closely related to that of classification. Numerous recommender systems have been proposed in the literature; these modify the various classification models to perform recommendations. The relationship between collaborative filtering and classification is discussed in [82]. The earliest association-based methods are described in [524]. Various enhancements of the method, which use support levels specific to the item at hand are discussed in [358, 359, 365]. The first two of these methods leverage user associations rather than item associations [358, 359]. Association rule-based systems have found significant uses in Web-based personalization and recommender systems [441, 552]. Association rule methods can be combined with neighborhood methods in order to extract *localized* associations [25] between items or between users. Localized associations generally provide more refined recommendations than is possible with global rule-based methods. A method for performing collaborative filtering with the use of the Bayes method is discussed in [437]. The use of probabilistic relational models for collaborative filtering is proposed in [219]. Support vector machines for recommender systems are discussed in [638].

Neural networks have also been used recently for collaborative filtering [519, 679]. The *restricted Boltzmann machine (RBM)* is a neural network with one input layer and one hidden layer. This kind of network has been used for collaborative filtering [519], in which the visible units correspond to items, and the training is done over all users in each epoch. The rating of items by users results in the activation of the visible units. Since RBMs can use nonlinearity within the units, they can sometimes achieve superior performance to latent factor models. RBMs use factorized representations of the large parameter space to reduce overfitting, and have been shown to be very accurate in the Netflix Prize contest. The basic idea of factorized parameter representations has also been used other recent methods such as factorization machines [493].

A detailed discussion of various dimensionality reduction methods may be found in [22]. The use of dimensionality reduction methods for neighborhood-based filtering was proposed in [525]. The works in [24, 525], which were proposed independently, also discuss the earliest uses of latent factor models as stand-alone methods for recommendation and missing data imputation. The work in [24] combines an EM-algorithm with latent factor models to impute missing entries. Stand-alone latent factor methods are particularly effective for collaborative filtering and are the state-of-the-art in the literature. Methods for regularizing latent factor methods are discussed by Paterek in [473]. The same work also introduces the notion of user and item biases in latent factor models. An *asymmetric factor model* is discussed in this work, in which users are not explicitly represented by latent factors. In this case, a user factor is represented as a linear combination of the implicit factors of items she has rated. As a result, the number of parameters to be learned is reduced. In fact, Paterek's (relatively under-appreciated) work [473] introduced almost all the basic innovations that were later combined and refined in various ways [309, 311, 313] to create state-of-the-art methods such as SVD++.

The early works [133, 252, 300, 500, 569, 666] showed how different forms of matrix factorization could be used for recommendations. The difference between various forms of matrix factorization is in the nature of the objective (loss) functions and the constraints on the factor matrices. The method in [371] proposes the notion of *kernel collaborative filtering*, which discovers *nonlinear* hyper-planes on which the ratings are distributed. This approach is able to model more complex ratings distributions. These different types of factorization lead to different trade-offs in quality, overfitting, and interpretability. Incremental methods for collaborative filtering for matrix factorization are discussed in [96].

Many variations of the basic objective function and constraints are used in different forms of matrix factorization. The works in [180, 500, 569, 624] explore maximum margin factorization, which is very closely related to unconstrained matrix factorization. The main difference is that a maximum margin regularizer is used with hinge loss in the objective function, rather than using the Frobenius norm of the error matrix to quantify the loss. The works in [252, 666] are non-negative forms of matrix factorization. A detailed discussion of non-negative matrix factorization methods for complete data may be found in [22, 537]. The work in [666] explores the conventional non-negative factorization method with the Frobenius norm, whereas the work in [252, 517] explores probabilistic forms of matrix factorization. Some of the probabilistic versions also minimize the Frobenius norm but also optimize the regularization simultaneously. Methods for combining Bayesian methods with matrix factorization methods (in order to judiciously determine regularization parameters) are discussed in [518]. Gibbs sampling is used to achieve this goal. Initialization techniques for non-negative matrix factorization methods are discussed in [331]. After the popularization of latent factor models by the Netflix Prize contest [73], other factorization-based methods were also proposed for collaborative filtering [309, 312, 313]. One of the earliest latent factor models, which works with implicit feedback data, is presented in [260]. The SVD++ description in this book is borrowed from [309]. A recent work [184] imposes a penalty proportional to the Frobenius norm of UV^T to force unobserved values to have lower ratings. The idea is to penalize higher ratings. This approach imposes stronger biases than [309] because it explicitly assumes that the unobserved ratings have lower values. Furthermore, the ratings in [184] need to be non-negative quantities, so that the Frobenius norm penalizes higher ratings to a greater degree. Some of the latent factor methods [309] show how techniques such as SVD++ can be combined with regression-based neighborhood methods (cf. section 3.7). Therefore, these methods combine linear regression with factorization models. A matrix factorization method that uses singular value decomposition is discussed in [127]. The use of inductive matrix completion methods on collaborative filtering matrices with side information is discussed in [267].

Various regression-based models are discussed in [72, 309, 342, 434, 620, 669]. A general examination of linear classifiers, such as least-squares regression and support vector machines (SVMs), is provided in [669]. This work was one of the earliest evaluations of linear methods, although it was designed only for implicit feedback data sets, such as Web click data or sales data, in which only positive preferences are available. It was observed that collaborative filtering in such cases is similar in form to text categorization. However, because of the noise in the data and the imbalanced nature of the class distribution, a direct use of SVM methods is sometimes not effective. Changes to the loss function are suggested in [669] in order to provide more accurate results. The approach shows that by using a quadratic loss function in SVM optimization, one gets a form that is more similar to the least-squares approach. The modified SVM performs either competitively to, or better than the least-squares approach. The methods in [72, 309] are closely associated with neighborhood-based methods, and they are discussed in section 2.6 of Chapter 2. The work in [620] uses collections of linear models, which are modeled as ordinary least-squares problems. The use of regression-based models, such as *slope-one predictors*, is discussed in [342]. As discussed in section 2.6 of Chapter 2, regression models can be used to show the formal connection between model-based methods and neighborhood-based methods [72, 309]. Other methods for combining regression with latent factor models are discussed in [13]. The works in [321, 455] develop various types of sparse linear models (*SLIM*) that combine the neighborhood approach with regression and matrix factorization. The *SLIM* approach is primarily designed for implicit feedback data sets.

A significant amount of work has been devoted to the choice of methodology for determining the solution to the underlying optimization problems. For example, a discussion of the trade-offs between gradient-descent and stochastic gradient descent is provided in [351], and mini-batches are proposed to bridge the gap between the two. Alternating least-squares methods are discussed in [268, 677]. The original idea of alternating least squares are proposed in the positive matrix factorization of complete matrices [460]. Methods for large-scale and distributed stochastic gradient descent in latent factor models are proposed in [217]. The main trade-off between stochastic descent and alternating least squares is the trade-off between stability and efficiency. The former method is more efficient, whereas the latter is more stable. It has been suggested that coordinate descent methods [650] can be efficient, while retaining stability. It has also been shown [651] that non-parametric methods have several advantages for large-scale collaborative filtering with latent factor models. Methods for addressing cold-start issues in latent factor models are discussed in [676]. The Netflix Prize competition was particularly notable in the history of latent factor models because it resulted in several useful lessons [73] about the proper implementation of such models. Recently, latent factor models have been used to model richer user preferences. For example, the work in [322] shows how one might combine global preferences with interest-specific preferences to make recommendations.

3.10 Exercises

1. Implement a decision tree-based predictor of ratings for an incomplete data sets. Use the dimensionality reduction approach described in the chapter.

2. How would you use a rule-based collaborative filtering system in the case where ratings are real-numbers in $[-1, 1]$.

3. Design an algorithm that combines association rule methods with clustering for recommendations in order to discover localized associations in unary data. What is the advantage of this approach over a vanilla rule-based method?

4. The naive Bayes model discussed in this chapter predicts the ratings of each item using the user's other ratings as a conditional. Design a Bayes model that uses the

item's other ratings as a condition. Discuss the advantages and disadvantages of each model. Identify a case in which each approach would work better. How would you combine the predictions of the two models?

5. Suppose that a merchant had a unary matrix containing the buying behavior of various customers. Each entry in the matrix contains information about whether or not a customer has bought a particular item. Among the users that have not bought an item yet, the merchant wishes to rank all the users in order of their propensity to buy it. Show how to use the Bayes model to achieve this goal.

6. Use the Bayes model on Table 3.1 to determine the probability that John might buy *Bread* in the future. Treat 0s in the table as values that are actually specified for the ratings, rather than as missing values (except for John's ratings for *Bread* and *Beef*). Determine the probability that he might buy *Beef* in the future. Is John more likely to buy *Bread* or *Beef* in the future?

7. Implement the naive Bayes model for collaborative filtering.

8. Perform a straightforward rank-2 SVD of the matrix in Table 3.2 by treating missing values as 0. Based on the use of SVD, what are the predicted ratings for the missing values of user 3? How does this compare with the results shown in the example of section 3.6.5.4, which uses a different initialization? How do the results compare to those obtained using the Bayes model described in the chapter?

9. Suppose you are given a matrix R which can be factorized as $R = UV^T$, where the columns of U are mutually orthogonal and the columns of V are mutually orthogonal. Show how to factorize R into three matrices in the form $Q\Sigma P^T$, where the columns of P and Q are orthonormal and Σ is a non-negative diagonal matrix.

10. Implement the unconstrained matrix factorization method with stochastic gradient descent and batch updates.

11. Discuss the changes required to the alternating least-squares method for unconstrained matrix factorization, when one constrains the last column of the user-factor matrix to contain only 1s, and the second-last column of the item-factor matrix to contain only 1s. This method is useful for incorporating user and item biases in unconstrained matrix factorization.

12. Discuss how you might apply the alternating least-squares method for designing latent factor models with implicit feedback.

13. Let the $m \times k$ matrix F, $n \times k$ matrix V, and and $n \times k$ matrix Y be defined as discussed in the asymmetric factor model portion of section 3.6.4.6. Assume a simplified setting of asymmetric factor models in which we do not need to account for user and item biases.

(a) Show that the stochastic gradient-descent updates for each observed entry (i, j) in the ratings matrix R are as follows:

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(e_{ij} \cdot \left[\sum_{h \in I_i} \frac{y_{hq}}{\sqrt{|I_i|}} \right] - \lambda \cdot v_{jq} \right) \quad \forall q \in \{1 \dots k\}$$

$$y_{hq} \Leftarrow y_{hq} + \alpha \left(\frac{e_{ij} \cdot v_{jq}}{\sqrt{|I_i|}} - \lambda \cdot y_{hq} \right) \quad \forall q \in \{1 \dots k\}, \forall h \in I_i$$

Here, $e_{ij} = r_{ij} - \hat{r}_{ij}$ is the error of observed entry (i, j) and I_i is the set of items for which user i has specified ratings.

(b) What changes would need to be made to the definitions of various matrices and to the updates to account for user and item biases?

Chapter 4

Content-Based Recommender Systems

"*Form must have a content, and that content must be linked with nature.*"– Alvar Aalto

4.1 Introduction

The collaborative systems discussed in the previous chapters use the correlations in the ratings patterns across users to make recommendations. On the other hand, these methods do not use item attributes for computing predictions. This would seem rather wasteful; after all, if John likes the futuristic science fiction movie *Terminator*, then there is a very good chance that he might like a movie from a similar genre, such as *Aliens*. In such cases, the ratings of other users may not be required to make meaningful recommendations. Content-based systems are designed to exploit scenarios in which items can be described with descriptive sets of attributes. In such cases, a user's *own* ratings and actions on other movies are sufficient to discover meaningful recommendations. This approach is particularly useful when the item is new, and there are few ratings available for that item.

Content-based recommender systems try to match users to items that are similar to what they have liked in the past. This similarity is not necessarily based on rating correlations across users but on the basis of the *attributes* of the objects liked by the user. Unlike collaborative systems, which explicitly leverage the ratings of *other* users in addition to that of the target user, content-based systems largely focus on the target user's *own* ratings and the attributes of the items liked by the user. Therefore, the other users have little, if any, role to play in content-based systems. In other words, the content-based methodology leverages a different source of data for the recommendation process. As we will see in Chapter 6, many recommender systems leverage the power of both sources. Such recommender systems are referred to as *hybrid* recommender systems.

© Springer International Publishing Switzerland 2016
C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_4

At the most basic level, content-based systems are dependent on two sources of data:

1. The first source of data is a description of various items in terms of content-centric attributes. An example of such a representation could be the text description of an item by the manufacturer.

2. The second source of data is a *user profile*, which is generated from user feedback about various items. The user feedback might be explicit or implicit. Explicit feedback may correspond to ratings, whereas implicit feedback may correspond to user actions. The ratings are collected in a way similar to collaborative systems.

   The user profile relates the attributes of the various items to user interests (ratings). A very basic example of a user profile might simply be a set of labeled training documents of item descriptions, the user ratings as the labels, and a classification or regression model relating the item attributes to the user ratings. The specific user profile is heavily dependent on the methodology at hand. For example, explicit ratings might be used in one setting, and implicit feedback might be used in another. It is also possible for the user to specify her own profile in terms of keywords of interest, and this approach shares some characteristics with *knowledge-based recommender systems.*

It is noteworthy that the ratings of the *other* users usually play no role in a content-based recommendation algorithm. This is both an advantage and a disadvantage, depending on the scenario at hand. On the one hand, in *cold-start scenarios*, where little information about the ratings of other users is available, such an approach can still be used as long as sufficient information about the user's own interests are available. This, at least, partially alleviates the cold-start problem when the number of other users in the recommender system is small. Furthermore, when an item is new, it is not possible to obtain the ratings of other users for that item. Content-based methods enable recommendations in such settings because they can extract the attributes from the new item, and use them to make predictions. On the other hand, the cold-start problem for new *users* cannot be addressed with content-based recommender systems. Furthermore, by not using the ratings of other users, one reduces the diversity and novelty of the recommended items. In many cases, the recommended items may be obvious items for the user, or they may be other items that the user has consumed before. This is because the content attributes will always recommend items with similar attributes to what the user has seen in the past. A recommended item with similar attributes often presents little surprise to the user. These advantages and disadvantages will be discussed in a later section of this chapter.

Content-based systems are largely used in scenarios in which a significant amount of attribute information is available at hand. In many cases, these attributes are keywords, which are extracted from the product descriptions. In fact, the vast majority of content-based systems extract text attributes from the underlying objects. Content-based systems are, therefore, particularly well suited to giving recommendations in text-rich and *unstructured* domains. A classical example of the use of such systems is in the recommendation of Web pages. For example, the previous browsing behavior of a user can be utilized to create a content-based recommender system. However, the use of such systems is not restricted only to the Web domain. Keywords from product descriptions are used to create item and user profiles for the purposes of recommendations in other e-commerce settings. In other settings, relational attributes such as manufacturer, genre, and price, may be used in addition to keywords. Such attributes can be used to create *structured* representations, which can be stored in a relational database. In these cases, it is necessary to combine the structured and unstructured attributes in a single structured representation. The basic principles of content-based systems, however, remain invariant to whether a structured or unstructured representation is used. This is because most learning methods in the structured domain have direct analogs in the unstructured domain, and vice versa. To preserve uniformity in exposition, our discussion in this chapter will be focused on unstructured settings. However, most of these methods can be easily adapted to structured settings.

Content-based systems are closely related to knowledge-based recommender systems. A summary of the relationship between the various types of systems is provided in Table 1.2 of Chapter 1. Like content-based systems, knowledge-based recommender systems use the content attributes of the items to make recommendations. The main difference is that knowledge-based systems support the *explicit specification* of user requirements in conjunction with interactive interfaces between the user and the recommender systems. Knowledge bases are used in conjunction with this interactivity to match user requirements to items. On the other hand, content-based systems generally use a learning-based approach based on *historical ratings*. Therefore, knowledge-based systems provide better control to the user in the recommendation process, whereas content-based systems leverage past behavior more effectively. Nevertheless, these differences are not so significant, and some content-based methods also allow users to explicitly specify their interest profiles. A number of systems leverage both the learning and interactive aspects within a unified framework. Such systems are referred to as *hybrid recommender systems*. Knowledge-based recommender systems are discussed in Chapter 5, whereas hybrid recommender systems are discussed in Chapter 6.

This chapter is organized as follows. The next section provides an overview of the basic components of a content-based recommender system. Feature extraction and selection methods are discussed in section 4.3. The process of learning user profiles and leveraging them for recommendations is discussed in section 4.4. A comparison of the main properties of collaborative and content-based systems is provided in section 4.5. The connections between collaborative filtering and content-based methods are explored in section 4.6. A summary is given in section 4.7.

4.2 Basic Components of Content-Based Systems

Content-based systems have certain basic components, which remain invariant across different instantiations of such systems. Since content-based systems work with a wide variety of item descriptions and knowledge about users, one must convert these different types of unstructured data into standardized descriptions. In most cases, it is preferred to convert the item descriptions into keywords. Therefore, content-based systems largely, but not exclusively, operate in the text domain. Many natural applications of content-based systems are also text-centric. For example, news recommender systems are often content-based systems, and they are also text-centric systems. In general, text classification and regression modeling methods remain the most widely used tools for creating content-based recommender systems.

The main components of content-based systems include the (offline) preprocessing portion, the (offline) learning portion, and the online prediction portion. The offline portions are used to create a summarized model, which is often a classification or regression model. This model is then used for the online generation of recommendations for users. The various components of content-based systems are as follows:

1. *Preprocessing and feature extraction:* Content-based systems are used in a wide variety of domains, such as Web pages, product descriptions, news, music features, and so on. In most cases, features are extracted from these various sources to convert them into

a keyword-based vector-space representation. This is the first step of any content-based recommendation system, and it is highly domain-specific. However, the proper extraction of the most informative features is essential for the effective functioning of any content-based recommender system.

2. *Content-based learning of user profiles:* As discussed earlier, a content-based model is specific to a given user. Therefore, a user-specific *model* is constructed to predict user interests in items, based on their past history of either buying or rating items. In order to achieve this goal, user feedback is leveraged, which may be manifested in the form of previously specified ratings (explicit feedback) or user activity (implicit feedback). Such feedbacks are used in conjunction with the attributes of the items in order to construct the training data. A learning model is constructed on this training data. This stage is often not very different from classification or regression modeling, depending on whether the feedback is categorical (e.g., binary act of selecting an item), or whether the feedback is numerical (e.g., ratings or buying frequency). The resulting model is referred to as the *user profile* because it conceptually relates user interests (ratings) to item attributes.

3. *Filtering and recommendation:* In this step, the learned model from the previous step is used to make recommendations on items for specific users. It is important for this step to be very efficient because the predictions need to be performed in real time.

In the following sections, we will describe each of these phases in detail. The second phase of learning often utilizes off-the-shelf classification models. The field of data classification is a vast area in its own right, and it is not the goal of this book to discuss classification models in detail. Therefore, throughout this chapter, we will assume a working familiarity with classification models. The goal will be to show how a specific classification model can be used as a black-box in the recommender system and the kinds of classification models that are specially suited to content-based recommender systems. A brief description of two of the most commonly used models is included, but this is by no means an exhaustive description. For the reader who is unfamiliar with classification models, pointers to several useful resources are included in the bibliographic notes.

4.3 Preprocessing and Feature Extraction

The first phase in all content-based models is to extract discriminative features for representing the items. Discriminative features are those, which are highly predictive of user interests. This phase is highly dependent on the specific application at hand. For example, a Web page recommendation system will be very different from a product recommendation system.

4.3.1 Feature Extraction

In the feature extraction phase, the descriptions of various items are extracted. Although it is possible to use any kind of representation, such as a multidimensional data representation, the most common approach is to extract keywords from the underlying data. This choice is because unstructured text descriptions are often widely available in a variety of domains, and they remain the most natural representations for describing items. In many cases, the items may have multiple fields describing various aspects of the item. For example, a merchant selling books may have descriptions of the books and keywords describing the content, title, and author. In some cases, these descriptions can be converted into a bag of keywords. In other cases, one might work directly with a multidimensional (structured) representation. The latter is necessary when the attributes contain numerical quantities (e.g., price) or fields that are drawn from a small universe of possibilities (e.g., color).

The various fields need to be weighted appropriately in order to facilitate their use in the classification process. *Feature weighting* is closely related to *feature selection*, in that the former is a soft version of the latter. In the latter case, attributes are either included or not included depending on their relevance, whereas in the former case, features are given differential weight depending on their importance. The issue of feature selection will be discussed in detail in section 4.3.4. Because the feature extraction phase is highly application-specific, we provide the reader with a flavor of the types of features that may need to be extracted in the context of various applications.

4.3.1.1 Example of Product Recommendation

Consider a movie recommendation site[1] such as IMDb [699], that provides personalized recommendations for movies. Each movie is usually associated with a description of the movie such as its synopsis, the director, actors, genre, and so on. A short description of *Shrek* at the IMDb Website is as follows:

> "After his swamp is filled with magical creatures, an ogre agrees to rescue a princess for a villainous lord in order to get his land back."

Many other attributes, such as user tags, are also available, which can be treated as content-centric keywords.

In the case of *Shrek*, one might simply concatenate all the keywords in the various fields to create a text description. The main problem is that the various keywords may not have equal importance in the recommendation process. For example, a particular actor might have greater importance in the recommendation than a word from the synopsis. This can be achieved in two ways:

1. Domain-specific knowledge can be used to decide the relative importance of keywords. For example, the title of the movie and the primary actor may be given more weight than the words in the description. In many cases, this process is done in a heuristic way with trial and error.

2. In many cases, it may be possible to *learn* the relative importance of various features in an automated way. This process is referred to as feature weighting, which is closely related to feature selection. Both feature weighting and feature selection are described in a later section.

4.3.1.2 Example of Web Page Recommendation

Web documents require specialized preprocessing techniques because of some common properties of their structure and the richness of the links inside them. Two major aspects of Web document preprocessing include the removal of specific parts of the documents (e.g., tags) that are not useful and the leveraging of the actual structure of the document.

All fields in a Web document are not equally important. HTML documents have numerous fields in them, such as the title, the meta-data, and the body of the document.

[1]The exact recommendation method used by IMDb is proprietary and not known to the author. The description here is intended only for illustrative purposes.

Typically, analytical algorithms treat these fields with different levels of importance, and therefore weight them differently. For example, the title of a document is considered more important than the body and is weighted more heavily. Another example of a specially processed portion of a Web document is anchor text. Anchor text contains a description of the Web page pointed to by a link. Because of its descriptive nature, it is considered important, but it is sometimes not relevant to the topic of the page itself. Therefore, it is often removed from the text of the document. In some cases, where possible, anchor text could even be added to the text of the document *to which it points*. This is because anchor text is often a summary description of the document to which it points. The learning of the importance of these various features can be done through automated methods, as discussed in section 4.3.4.

A Web page may often be organized into content blocks that are not related to the primary subject matter of the page. A typical Web page will have many irrelevant blocks, such as advertisements, disclaimers, or notices, which are not very helpful for mining. It has been shown that the quality of mining results improves when only the text in the main block is used. However, the (automated) determination of main blocks from Web-scale collections is itself a data mining problem of interest. While it is relatively easy to decompose the Web page into blocks, it is sometimes difficult to identify the main block. Most automated methods for determining main blocks rely on the fact that a particular site will typically utilize a similar layout for all its documents. Therefore, the structure of the layout is learned from the documents at the site by extracting *tag trees* from the site. Other main blocks are then extracted through the use of the *tree-matching algorithm* [364, 662]. Machine learning methods can also be used for this task. For example, the problem of labeling the main block in a page can be treated as a classification problem. The bibliographic notes contain pointers to methods for extracting the main block from a Web document.

4.3.1.3 Example of Music Recommendation

Pandora Internet radio [693] is a well-known music recommendation engine, which associates tracks with features extracted from the Music Genome Project [703]. Examples of such features of tracks could be "feature trance roots," "synth riffs," "tonal harmonies," "straight drum beats," and so on. Users can initially specify a single example of a track of their interest to create a "station." Starting with this single training example, similar songs are played for the user. The users can express their likes or dislikes to these songs.

The user feedback is used to build a more refined model for music recommendation. It is noteworthy, that even though the underlying features are quite different in this case, they can still be treated as keywords, and the "document" for a given song corresponds to the bag of keywords associated with it. Alternatively, one can associate specific attributes with these different keywords, which leads to a structural multidimensional representation.

The initial specification of a track of interest is more similar to a knowledge-based recommender system than a content-based recommender system. Such types of knowledge-based recommender systems are referred to as case-based recommender systems. However, when ratings are leveraged to make recommendations, the approach becomes more similar to that of a content-based recommender system. In many cases, Pandora also provides an explanation for the recommendations in terms of the item attributes.

4.3.2 Feature Representation and Cleaning

This process is particularly important when the unstructured format is used for representation. The feature extraction phase is able to determine bags of words from the unstructured descriptions of products or Web pages. However, these representations need to be cleaned and represented in a suitable format for processing. There are several steps in the cleaning process:

1. *Stop-word removal:* Much of the text that is extracted from free-form descriptions of items, will contain many words that are not specific to the item but that are a common part of English vocabulary. Such words are typically high-frequency words. For example, words such as '*a*," "*an*," and "*the*" will not be particularly specific to the item at hand. In the movie recommendation application, it is common to find such words in the synopsis. In general, articles, prepositions, conjunctions, and pronouns are treated as stop-words. In most cases, standardized lists of stop-words are available in various languages.

2. *Stemming:* In stemming, variations of the same word are consolidated. For example, singular and plural forms of a word or different tenses of the same word are consolidated. In some cases, common roots are extracted from various words. For example, words such as "hoping" and "hope" are consolidated into the common root "hop." Of course, stemming can sometimes have a detrimental effect, because a word such as "hop" has a different meaning of its own. Many off-the-shelf tools [710–712] are available for stemming.

3. *Phrase extraction:* The idea is to detect words that occur together in documents on a frequent basis. For example, a phrase such as "hot dog" means something different from its constituent words. Manually defined dictionaries are available for phrase extraction, although automated methods can also be used [144, 364, 400].

After executing these steps, the keywords are converted into a *vector-space representation*. Each word is also referred to as a *term*. In the vector-space representation, documents are represented as bags of words, together with their frequencies. Although it might be tempting to use the raw frequency of word occurrence, this is often not desirable. This is because commonly occurring words are often statistically less discriminative. Therefore, such words are often discounted by down-weighting. This is similar to the principle of stop-words, except that it is done in a soft way by discounting the word, rather than completely removing it.

How are words discounted? This is achieved by using the notion of *inverse document frequency*. The inverse document frequency id_i of the ith term is a decreasing function of the number of documents n_i in which it occurs.

$$id_i = \log(n/n_i) \tag{4.1}$$

Here, the number of documents in the collection is denoted by n.

Furthermore, care needs to be taken that the excessive occurrence of a single word in the collection is not given too much importance. For example, when item descriptions are collected from unreliable sources or open platforms, such as the Web, they are liable to contain a significant amount of spam. To achieve this goal, a damping function $f(\cdot)$, such as the square root or the logarithm, is optionally applied to the frequencies before similarity computation.

$$f(x_i) = \sqrt{x_i}$$
$$f(x_i) = \log(x_i)$$

Frequency damping is optional and is often omitted. Omitting the damping process is equivalent to setting $f(x_i)$ to x_i. The *normalized* frequency $h(x_i)$ for the ith word is defined by combining the inverse document frequency with the damping function:

$$h(x_i) = f(x_i) id_i \tag{4.2}$$

This model is popularly referred to as the tf-idf model, where tf represents the term frequency and idf represents the inverse document frequency.

4.3.3 Collecting User Likes and Dislikes

Aside from the content about the items, it is also necessary to collect data about the user likes and dislikes for the recommendation process. The data collection is done during the offline phase, whereas recommendations are determined during the online phase when a specific user is interacting with the system. The user for whom the prediction is performed at any given time is referred to as the *active user*. During the online phase, the user's own preferences are combined with the content to create the predictions. The data about user likes and dislikes can take on any of the following forms:

1. *Ratings:* In this case, users specify ratings indicating their preference for the item. Ratings can be binary, interval-based, or ordinal. In rare cases, ratings can even be real valued. The nature of the rating has a significant impact on the model used for learning the user profiles.

2. *Implicit feedback:* Implicit feedback refers to user actions, such as buying or browsing an item. In most cases, only the positive preferences of a user are captured with implicit feedback but not negative preferences.

3. *Text opinions:* In many cases, users may express their opinions in the form of text descriptions. In such cases, implicit ratings can be extracted from these opinions. This form of rating extraction is related to the field of *opinion mining* and *sentiment analysis*. This area is beyond the scope of this book. Interested readers are referred to [364].

4. *Cases:* Users might specify examples (or *cases*) of items that they are interested in. Such cases can be used as implicit feedback with nearest neighbor or Rocchio classifiers. However, when the similarity retrieval is used in conjunction with carefully designed utility functions, these methods are more closely related to case-based recommender systems. Case-based systems are a subclass of knowledge-based recommender systems in which domain knowledge is used to discover matching items, instead of learning algorithms (cf. section 5.3.1 of Chapter 5). It is often difficult to delineate where content-based recommender systems end and knowledge-based recommender systems begin. For example, Pandora Internet Radio often uses an initial case of an interesting music album to set up "radio stations" for users with similar music items. At a later stage, user feedback about likes and dislikes is utilized to refine the recommendations. Therefore, the first part of the approach can be viewed as a knowledge-based system, and the second part of the approach can be viewed as a content-based (or collaborative) system.

In all the aforementioned cases, the likes or dislikes of a user for an item are finally converted into a unary, binary, interval-based, or real rating. This rating can also be viewed as the extraction of a *class label* or *dependent variable*, which is eventually leveraged for learning purposes.

4.3.4 Supervised Feature Selection and Weighting

The goal of feature selection and weighting is to ensure that only the most informative words are retained in the vector-space representation. In fact, many well-known recommender systems [60, 476] explicitly advocate that a size cut-off should be used on the number of keywords. The experimental results in [476], which were performed in a number of domains, suggested that the number of extracted words should be somewhere between 50 and 300. The basic idea is that the noisy words often result in overfitting and should therefore be removed *a priori*. This is particularly important, considering the fact that the number of documents available to learn a particular user profile is often not very large. When the number of documents available for learning is small, the tendency of the model to overfit will be greater. Therefore, it is crucial to reduce the size of the feature space.

There are two distinct aspects to incorporating the feature informativeness in the document representation. One is feature *selection*, which corresponds to the removal of words. The second is feature *weighting*, which involves giving greater importance to words. Note that stop-word removal and the use of inverse-document frequency are examples of feature selection and weighting, respectively. However, these are unsupervised ways of feature selection and weighting, where user feedback is given no importance. In this section, we will study supervised methods for feature selection, which take the user ratings into account for evaluating feature informativeness. Most of these methods evaluate the sensitivity of the dependent variable to a feature in order to evaluate its informativeness.

The measures for computing feature informativeness can be used to either perform a hard selection of features or to heuristically weight the features with a function of the computed quantification of informativeness. The measures used for feature informativeness are also different, depending on whether the user rating is treated as a numeric or categorical value. For example, in the context of binary ratings (or ratings with a small number of discrete values), it makes sense to use categorical rather than numeric representations. We will also describe a few methods that are commonly used for feature weighting. In most of the following descriptions, we will assume an unstructured (textual) representation, although the methods can be generalized easily to structured (multidimensional) representations. This is because the vector-space representation of text can be viewed as a special case of the multidimensional representation. The bibliographic notes contain pointers to more details of feature selection methods.

4.3.4.1 Gini Index

The Gini index is one of the most commonly used measures for feature selection. It is a simple and intuitive measure, which is easy to understand. The Gini index is inherently suited to binary ratings, ordinal ratings, or ratings which are distributed into a small number of intervals. The latter case may sometimes be obtained by discretizing the ratings. The ordering among the ratings is ignored, and each possible value of the rating is treated as an instance of a categorical attribute value. This might seem like a disadvantage because it loses information about the relative ordering of the ratings. However, in practice, the number of possible ratings is usually small and therefore significant accuracy is not lost.

Let t be the total number of possible values of the rating. Among documents containing a particular word w, let $p_1(w) \dots p_t(w)$ be the fraction of the items rated at each of these t possible values. Then, the Gini index of the word w is defined as follows:

$$\text{Gini}(w) = 1 - \sum_{i=1}^{t} p_i(w)^2 \tag{4.3}$$

The value of Gini(w) always lies in the range $(0, 1-1/t)$, with smaller values being indicative of greater discriminative power. For example, when the presence of the word w always results in the document being rated at the jth possible rating value (i.e., $p_j(w) = 1$), then such a word is very discriminative for rating predictions. Correspondingly, the value of the Gini index in such a case is $1 - 1^2 = 0$. When each value of $p_j(w)$ takes on the same value of $1/t$, the Gini index takes on its maximum value of $1 - \sum_{i=1}^{t}(1/t^2) = 1 - 1/t$.

4.3.4.2 Entropy

Entropy is very similar in principle to the Gini index except that information-theoretic principles are used to design the measure. As in the previous case, let t be the total number of possible values of the rating and $p_1(w) \ldots p_t(w)$ be the fraction of the documents containing a particular word w, which are rated at each of these t possible values. Then, the entropy of the word w is defined as follows:

$$\text{Entropy}(w) = -\sum_{i=1}^{t} p_i(w)\log(p_i(w)) \tag{4.4}$$

The value of Entropy(w) always lies in the range $(0, 1)$, with smaller values being more indicative of discriminative power. It is easy to see that entropy has similar characteristics with the Gini index. In fact, these two measures often yield very similar results although they have different probabilistic interpretations. The Gini index is easier to understand, whereas entropy measures are more firmly grounded in mathematical principles from information theory.

4.3.4.3 χ^2-Statistic

The χ^2-statistic can be computed by treating the co-occurrence between the word and class as a *contingency table*. For example, consider a scenario where we are trying to determine whether a particular word is relevant to a user's buying interests. Assume that the user has bought about 10% of the items in the collection, and the word w occurs in about 20% of the descriptions. Assume that the total number of items (and corresponding documents) in the collection is 1000. Then, the *expected* number of occurrences of each possible combination of word occurrence and class contingency is as follows:

	Term occurs in description	Term does not occur
User bought item	$1000 * 0.1 * 0.2 = 20$	$1000 * 0.1 * 0.8 = 80$
User did not buy item	$1000 * 0.9 * 0.2 = 180$	$1000 * 0.9 * 0.8 = 720$

The aforementioned expected values are computed under the assumption that the occurrence of the term in the description and the user interest in the corresponding item are independent of one another. If these two quantities are independent, then clearly the term will be irrelevant to the learning process. However, in practice, the item may be highly related to the item at hand. For example, consider a scenario where the contingency table deviates from expected values and the user is very likely to buy the item containing the term. In such a case, the contingency table may appear as follows:

	Term occurs in description	Term does not occur
User bought item	$O_1 = 60$	$O_2 = 40$
User did not buy item	$O_3 = 140$	$O_4 = 760$

The χ^2-statistic measures the normalized deviation between observed and expected values across the various cells of the contingency table. In this case, the contingency table contains $p = 2 \times 2 = 4$ cells. Let O_i be the observed value of the ith cell and E_i be the expected value of the ith cell. Then, the χ^2-statistic is computed as follows:

$$\chi^2 = \sum_{i=1}^{p} \frac{(O_i - E_i)^2}{E_i} \tag{4.5}$$

Therefore, in the particular example of this table, the χ^2-statistic evaluates to the following:

$$\begin{aligned}\chi^2 &= \frac{(60-20)^2}{20} + \frac{(40-80)^2}{80} + \frac{(140-180)^2}{180} + \frac{(760-720)^2}{720}\\ &= 80 + 20 + 8.89 + 2.22\\ &= 111.11\end{aligned}$$

It is also possible to compute the χ^2-statistic as a function of the observed values in the contingency table without explicitly computing expected values. This is possible because the expected values are functions of the aggregate observed values across rows and columns. A simple arithmetic formula to compute the χ^2-statistic in a 2×2 contingency table is as follows (see Exercise 8):

$$\chi^2 = \frac{(O_1 + O_2 + O_3 + O_4) \cdot (O_1 O_4 - O_2 O_3)^2}{(O_1 + O_2) \cdot (O_3 + O_4) \cdot (O_1 + O_3) \cdot (O_2 + O_4)} \tag{4.6}$$

Here, $O_1 \ldots O_4$ are the observed frequencies according to the table above. It is easy to verify that this formula yields the same χ^2-statistic of 111.11. Note that the χ^2-test can also be interpreted in terms of the probabilistic level of significance with the use of a χ^2 distribution. However, for practical purposes, it is sufficient to know that larger values of the χ^2-statistic indicate that a particular term and item are related to a greater degree. Note that if the observed values are exactly equal to the expected values, then it implies that the corresponding term is irrelevant to the item at hand. In such a case, the χ^2-statistic will evaluate to its least possible value of 0. Therefore, the top-k features with the largest χ^2-statistic are retained.

4.3.4.4 Normalized Deviation

The problem with most of the aforementioned measures is that they lose information about the relative ordering of ratings. For cases in which the ratings have high granularity, the normalized deviation is an appropriate measure.

Let σ^2 be the variance of the ratings in all the documents. Furthermore, let $\mu^+(w)$ be the average rating of all documents containing the word w, and $\mu^-(w)$ be the average rating of all documents that do not contain the word w. Then, the normalized deviation of the word w is defined as follows:

$$\text{Dev}(w) = \frac{|\mu^+(w) - \mu^-(w)|}{\sigma} \tag{4.7}$$

Larger values of Dev(w) are indicative of more discriminatory words.

The aforementioned quantification is based on the relative distribution of the ratings for documents containing a specific word with respect to the ratings distribution of all documents. Such an approach is particularly suitable when ratings are treated as numerical quantities. A related measure is the Fisher's discrimination index, which computes the ratio of the inter-class separation to the intra-class separation in the *feature space* (rather than on the ratings dimension). This measure is described in detail in [22]. Fisher's discriminant index is however, better suited to categorical dependent variables rather than numerical dependent variables, such as ratings.

4.3.4.5 Feature Weighting

Feature Weighting can be viewed as a soft version of feature selection. In the earlier section on feature representation in this chapter, it was already discussed how measures such as the inverse document frequency can be used to weight documents. However, the inverse document frequency is an unsupervised measure that does not depend on user likes or dislikes. A supervised measure can also be used to further weight the vector-space representation in order to yield differential importance to different words. For example, in a movie recommendation application, keywords describing a movie genre or actor name are more important than words selected from the synopsis of the movie. On the other hand, the words in the synopsis are also somewhat indicative of tastes. Therefore, they cannot be excluded either. Feature weighting is a more refined approach for discriminating between various words by using a weight rather than a hard binary decision. The simplest approach to feature weighting is to take any of the feature selection measures and use them to derive the weights. For example, the inverse of the Gini index or entropy could be used. In many cases, a heuristic function can be further applied on the selection measure to control the sensitivity of the weighting process. For example, consider the following weighting function $g(w)$ for word w, where a is a parameter greater than 1.

$$g(w) = a - \text{Gini}(w) \tag{4.8}$$

The resulting weight $g(w)$ will always lie in the range $(a-1, a)$. By varying the value of a, the sensitivity of the weighting process can be controlled. Smaller values of a will lead to greater sensitivity. The weight of each word w in the vector-space representation is then multiplied by $g(w)$. Similar weighting functions can be defined with respect to the entropy and the normalized deviation. The process of selecting an appropriate feature weighting is a highly heuristic process that varies significantly corresponding to the application at hand. The value of a can be viewed as a parameter of the weighting function. It is also possible to learn the optimal parameters of such a function using *cross-validation* techniques. Such techniques are discussed in Chapter 7.

4.4 Learning User Profiles and Filtering

The learning of user profiles is closely related to the classification and regression modeling problem. When the ratings are treated as discrete values (e.g., "thumbs up" or "thumbs down"), the problem is similar to that of text classification. On the other hand, when the ratings are treated as a set of numerical entities, the problem is similar to that of regression modeling. Furthermore, the learning problem can be posed in both structured and unstructured domains. For homogeneity in presentation, we will assume that the descriptions of items are in the form of documents. However, the approach can easily be generalized to any type of multidimensional data because text is a special type of multidimensional data.

In each case, we assume that we have a set $\mathcal{D}_L$ of training documents, which are labeled by a specific user. This user is also referred to as the active user when that user obtains a recommendation from the system. The training documents correspond to the descriptions of items, which are extracted in the preprocessing and feature selection phases. Furthermore, the training data contain the ratings assigned by the active user to these documents. These documents are used to construct a training model. Note that the labels assigned by other users (than the active user) are not used in the training process. Therefore, the training models are specific to particular users, and they cannot be used for arbitrarily chosen users. This is different from traditional collaborative filtering, in which methods like matrix factorization build a single model across all users. The training model for a specific user represents the user profile.

The labels on the documents correspond to the numeric, binary, or unary ratings. Assume that the ith document in $\mathcal{D}_L$ has a rating denoted by c_i. We also have a set $\mathcal{D}_U$ of testing documents, which are unlabeled. Note that both $\mathcal{D}_L$ and $\mathcal{D}_U$ are specific to a particular (active) user. The testing documents might correspond to descriptions of items, which might be potentially recommended to the user but which have not yet been bought or rated by the user. In domains such as news recommendation the documents in $\mathcal{D}_U$ might correspond to candidate Web documents for recommendation to the active user. The precise definition of $\mathcal{D}_U$ depends on the domain at hand, but the individual documents in $\mathcal{D}_U$ are extracted in a similar way to those in $\mathcal{D}_L$. The training model on $\mathcal{D}_L$ is used to make recommendations from $\mathcal{D}_U$ to the active user. As in the case of collaborative filtering, the model can be used to provide either a predicted value of the rating or a ranked list of top-k recommendations.

It is immediately evident that this problem is similar to that of classification and regression modeling in the text domain. The reader is referred to a recent survey [21] for a detailed discussion of many of these techniques. In the following, we will discuss some of the common learning methods.

4.4.1 Nearest Neighbor Classification

The nearest neighbor classifier is one of the simplest classification techniques, and it can be implemented in a relatively straightforward way. The first step is to define a similarity function, which is used in the nearest neighbor classifier. The most commonly used similarity function is the cosine function. Let $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ be a pair of documents, in which the normalized frequencies of the ith word are given by x_i and y_i, respectively, in the two documents. Note that these frequencies are normalized or weighted with the use of unsupervised tf-idf weighting or the supervised methods discussed in the previous section. Then, the cosine measure is defined using these normalized frequencies as follows:

$$\text{Cosine}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}} \tag{4.9}$$

The cosine similarity is frequently used in the text domain because of its ability to adjust to the varying lengths of the underlying documents. When this approach is used for other types of structured and multidimensional data, other similarity/distance functions, such as the Euclidean distance and Manhattan distance, are used. For relational data with categorical attributes, various match-based similarity measures are available [22].

This similarity function is useful in making predictions for items (documents) in which the user preference is unknown. For each document in $\mathcal{D}_U$, its k-nearest neighbors in $\mathcal{D}_L$ are determined using the cosine similarity function. The average value of the rating for the k neighbors of each item in $\mathcal{D}_U$ is determined. This average value is the predicted rating for the corresponding item in $\mathcal{D}_U$. An additional heuristic enhancement is that one can weight each rating with the similarity value. In cases where ratings are treated as categorical values, the number of votes for each value of the rating is determined, and the rating value with the largest frequency is predicted. The documents in $\mathcal{D}_U$ are then ranked based on the predicted value of the rating, and the top items are recommended to the user.

The main challenge with the use of this approach is its high computational complexity. Note that the nearest neighbor of each document in $\mathcal{D}_U$ needs to be determined, and the time required for each nearest neighbor determination is linear to the size of $\mathcal{D}_L$. Therefore, the computational complexity is equal to $|\mathcal{D}_L| \times |\mathcal{D}_U|$. One way of making the approach faster is to use clustering to reduce the number of training documents in $\mathcal{D}_L$. For each distinct value of the rating, the corresponding subset of documents in $\mathcal{D}_L$ are clustered into $p \ll |\mathcal{D}_L|$ groups. Therefore, if there are s distinct values of the ratings, then the total number of groups is $p \cdot s$. Typically, a fast centroid-based (i.e., k-means) clustering is used to create each group of p clusters. Note that the number of groups $p \cdot s$ is significantly smaller than the number of training documents. In such cases, each group is converted into a larger document corresponding to the concatenation[2] of the documents in that group. The vector-space representation of this larger document can be extracted by adding up the word frequencies of its constituents. The corresponding rating label associated with the document is equal to the rating of the constituent documents. For each target document T, the closest $k < p$ documents are found from this newly created set of p documents. The average rating of this set of k documents is returned as the label for the target. As in the previous case, the rating is predicted for each item in $\mathcal{D}_U$, and the top-ranked items are returned to the active user. This approach speeds up the classification process, because one must compute the similarity between the target document and a relatively small number of aggregated documents. Even though this approach incurs an additional preprocessing overhead of clustering, this overhead is generally small compared to the savings at recommendation time when the sizes of $\mathcal{D}_L$ and $\mathcal{D}_U$ are large.

A special case of this clustering-based approach is one in which all documents belonging to a particular value of the rating are aggregated into a single group. Thus, the value of p is set to 1. The vector-space representation of the resulting vector of each group is also referred to as the *prototype* vector. For a test document, the rating of the closest document is reported as the relevant one for the target. This approach is closely related to Rocchio classification, which also allows for the notion of *relevance feedback* from the active user. The Rocchio method was originally designed for binary classes, which, in our case, translate to binary ratings. The bibliographic notes contain pointers to the Rocchio method.

4.4.2 Connections with Case-Based Recommender Systems

Nearest neighbor methods are connected to knowledge-based recommender systems in general, and case-based recommender systems in particular. Knowledge-based recommender systems are discussed in detail in Chapter 5. The main difference is that in case-based recommender systems, the user interactively specifies a *single* example of interest, and the nearest neighbors of this example are retrieved as possible items of interest for the user.

[2]For structured data, the centroid of the group may be used.

Furthermore, a significant amount of domain knowledge is used in the design of the similarity function, because only a single example is available. This single example can be more appropriately viewed as a user requirement rather than a historical rating, because it is specified interactively. In knowledge-based systems, there is less emphasis on using historical data or ratings. Like the Rocchio method, such methods are also interactive, although the interactivity is far more sophisticated in case-based systems.

4.4.3 Bayes Classifier

The Bayes classifier is discussed in section 3.4 of Chapter 3 in collaborative filtering. However, the discussion in Chapter 3 is a non-standard use of the Bayes model in which the missing entries are predicted from the specified ones. In the context of content-based recommender systems, the problem translates to a more conventional use of the Bayes model for text classification. Therefore, we will revisit the Bayes model in the context of text classification.

In this case, we have a set $\mathcal{D}_L$ containing the training documents, and a set $\mathcal{D}_U$ containing the test documents. For ease in discussion, we will assume that the labels are binary in which users specify either a like or a dislike rating as $+1$ or -1, respectively for each of the training documents in $\mathcal{D}_L$. It is, however, relatively easy to generalize this classifier to the case where the ratings take on more than two values.

As before, assume that the rating of the ith document in $\mathcal{D}_L$ is denoted by $c_i \in \{-1, 1\}$. Therefore, this labeled set represents the user profile. There are two models that are commonly used in text data, which correspond to the Bernoulli and the multinomial models, respectively. In the following, we will discuss only the Bernoulli model. The multinomial model is discussed in detail in [22].

In the Bernoulli model, the frequencies of the words are ignored, and only the presence or absence of the word in the document is considered. Therefore, each document is treated as a binary vector of d words containing only values of 0 and 1. Consider a target document $\overline{X} \in \mathcal{D}_U$, which might correspond to the description of an item. Assume that the d binary features in $\overline{X}$ are denoted by $(x_1 \ldots x_d)$. Informally, we would like to determine $P(\text{Active user likes } \overline{X} | x_1 \ldots x_d)$. Here, each x_i is a 0-1 value, corresponding to whether or not the ith word is present in the document $\overline{X}$. Then, if the class (binary rating) of $\overline{X}$ is denoted by $c(\overline{X})$, this is equivalent to determining the value of $P(c(\overline{X}) = 1 | x_1 \ldots x_d)$. By determining both $P(c(\overline{X}) = 1 | x_1 \ldots x_d)$ and $P(c(\overline{X}) = -1 | x_1 \ldots x_d)$ and selecting the larger of the two, one can determine whether or not the active user likes $\overline{X}$. These expressions can be evaluated by using the Bayes rule and then applying a *naive assumption* as follows:

$$\begin{aligned}
P(c(\overline{X}) = 1 | x_1 \ldots x_d) &= \frac{P(c(\overline{X}) = 1) \cdot P(x_1 \ldots x_d | c(\overline{X}) = 1)}{P(x_1 \ldots x_d)} \\
&\propto P(c(\overline{X}) = 1) \cdot P(x_1 \ldots x_d | c(\overline{X}) = 1) \\
&= P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i | c(\overline{X}) = 1) \quad \text{[Naive Assumption]}
\end{aligned}$$

The naive assumption states that the occurrences of words in documents are conditionally independent events (on a specific class), and therefore one can replace $P(x_1 \ldots x_d | c(\overline{X}) = 1)$ with $\prod_{i=1}^{d} P(x_i | c(\overline{X}) = 1)$. Furthermore, the constant of proportionality is used in the first relationship because the denominator is independent of the class. Therefore, the denominator does not play any role in deciding between the relative order of the classes.

The denominator, however, does play a role in terms of ranking the propensity of *different items (documents)* to be liked by the user. This is relevant to the problem of *ranking* items for a specific user, in order of $P(c(\overline{X}) = 1|x_1 \ldots x_d)$.

In cases where such a ranking of the items is needed, the constant of proportionality is no longer irrelevant. This is particularly common in recommendation applications where it is not sufficient to determine the relative probabilities of items belonging to different rating values, but to actually rank them with respect to one another. In such cases, the constant of proportionality needs to be determined. Assume that the constant of proportionality in the relationship above is denoted by K. The constant of proportionality K can be obtained by using the fact that the sum of the probabilities of all possible instantiations of $c(\overline{X})$ should always be 1. Therefore, we have:

$$K \cdot \left[P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) + P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) \right] = 1$$

Therefore, we can derive the following value for K:

$$K = \frac{1}{P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) + P(c(\overline{X}) = -1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = -1)}$$

This approach is used to determine the probability of a user liking each possible item in $\mathcal{D}_U$. The items in $\mathcal{D}_U$ are then ranked according to this probability and presented to the user. These methods are particularly well suited to binary ratings. There are other ways of using the probability to estimate the predicted value of the ratings and rank the items when dealing with ratings that are not necessarily binary. Such methods are discussed in detail in section 3.4 of Chapter 3.

4.4.3.1 Estimating Intermediate Probabilities

The Bayes method requires the computation of intermediate probabilities such as $P(x_i|c(\overline{X}) = 1)$. So far, we have not yet discussed how these probabilities can be estimated in a data-driven manner. The main utility of the aforementioned Bayes rule is that it expresses the prediction probabilities in terms of other probabilities [e.g., $P(x_i|c(\overline{X}) = 1)$] that can be estimated more easily in a data-driven way. We reproduce the Bayes condition above:

$$P(c(\overline{X}) = 1|x_1 \ldots x_d) \propto P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1)$$

$$P(c(\overline{X}) = -1|x_1 \ldots x_d) \propto P(c(\overline{X}) = -1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = -1)$$

In order to compute the Bayes probabilities, we need to estimate the probabilities on the right-hand side of the equations above. These include the prior class probabilities $P(c(\overline{X}) = 1)$ and $P(c(\overline{X}) = -1)$. Furthermore, the feature-wise conditional probabilities, such as $P(x_i|c(\overline{X}) = 1)$ and $P(x_i|c(\overline{X}) = -1)$, need to be estimated. The probability $P(c(\overline{X}) = 1)$ can be estimated as the fraction of positive training examples $\mathcal{D}_L^+$ in the labeled data $\mathcal{D}_L$. In order to reduce overfitting, Laplacian smoothing is performed by adding values proportional to a small parameter $\alpha > 0$ to the numerator and denominator.

$$P(c(\overline{X}) = 1) = \frac{|\mathcal{D}_L^+| + \alpha}{|\mathcal{D}_L| + 2 \cdot \alpha} \tag{4.10}$$

Table 4.1: Illustration of the Bayes method for a content-based system

Keyword ⇒ Song-Id ⇓	Drums	Guitar	Beat	Classical	Symphony	Orchestra	Like or Dislike
1	1	1	1	0	0	0	Dislike
2	1	1	0	0	0	1	Dislike
3	0	1	1	0	0	0	Dislike
4	0	0	0	1	1	1	Like
5	0	1	0	1	0	1	Like
6	0	0	0	1	1	0	Like
Test-1	0	0	0	1	0	0	?
Test-2	1	0	1	0	0	0	?

The value of $P(c(\overline{X}) = -1)$ is estimated in an exactly similar way. Furthermore, the conditional feature probability $P(x_i|c(\overline{X}) = 1)$ is estimated as the fraction of the instances in the positive class for which the ith feature takes on the value of x_i. Let $q^+(x_i)$ represent the number of instances in the positive class that take on the value of $x_i \in \{0, 1\}$ for the ith feature. Then, we can use a Laplacian smoothing parameter $\beta > 0$ to estimate the probability as follows:

$$P(x_i|c(\overline{X}) = 1) = \frac{q^+(x_i) + \beta}{|\mathcal{D}_L^+| + 2 \cdot \beta} \tag{4.11}$$

A similar approach can be used to estimate $P(x_i|c(\overline{X}) = -1)$. Note that the Laplacian smoothing is helpful for cases where little training data are available. In the extreme case, where $\mathcal{D}_L^+$ is empty, the probability $P(x_i|c(\overline{X}) = 1)$ would be (appropriately) estimated to be 0.5 as a kind of prior belief. Without smoothing, such an estimation would be indeterminate, because both the numerator and denominator of the ratio would be 0. Laplacian smoothing, like many regularization methods, can be interpreted in terms of the greater importance of prior beliefs when the amount of training data is limited. Although we have presented the aforementioned estimation for the case of binary ratings, it is relatively easy to generalize the estimation when there are k distinct values of the ratings. A similar type of estimation is discussed in the context of collaborative filtering in section 3.4 of Chapter 3.

4.4.3.2 Example of Bayes Model

We provide an example of the use of the Bayes model for a set of 6 training examples and two test examples. In Table 4.1, the columns correspond to features representing properties of various songs. The user like or dislike is illustrated in the final column of the table. Therefore, the final column can be viewed as the rating. The first 6 rows correspond to the training examples, which correspond to the user profile. The final pair of rows correspond to two candidate music tracks that need to be ranked for the specific user at hand. In machine learning parlance, these rows are also referred to as test instances. Note that the final (dependent variable) column is specified only for the training rows because the user like or dislike (ratings) are not known for the test rows. These values need to be predicted.

By examining the features in Table 4.1, it becomes immediately evident that the first three features (columns) might often occur in many popular music genres such as rock music, whereas the final three features typically occur in classical music. The user profile, represented by Table 4.1 clearly seems to suggest a preference for classical music over rock music. Similarly, among the test examples, only the first of the two examples seems to match the user's interests. Let us examine how the Bayes approach is able to derive this fact in a data-driven way. For ease in computation, we will assume that Laplacian smoothing is not used, although it is important to use such smoothing methods in real applications.

By using the Bayes model, we can derive the conditional probabilities for likes and dislikes based on the observed features of the test examples:

$$\begin{aligned}
P(\text{Like}|\,\textit{Test-1}) &\propto 0.5 \prod_{i=1}^{6} P(\text{Like}|x_i) \\
&= (0.5) \cdot \frac{3}{4} \cdot \frac{2}{2} \cdot \frac{3}{4} \cdot \frac{3}{3} \cdot \frac{1}{4} \cdot \frac{1}{3} \\
&= \frac{3}{128} \\
P(\text{Dislike}|\,\textit{Test-1}) &\propto 0.5 \prod_{i=1}^{6} P(\text{Dislike}|x_i) \\
&= (0.5) \cdot \frac{1}{4} \cdot \frac{0}{2} \cdot \frac{1}{4} \cdot \frac{0}{3} \cdot \frac{3}{4} \cdot \frac{2}{3} \\
&= 0
\end{aligned}$$

By normalizing the two probabilities to sum to 1, we obtain the result that $P(\text{Like}|\,\textit{Test-1})$ is 1 and $P(\text{Dislike}|\,\textit{Test-1})$ is 0. In the case of *Test-2*, exactly the opposite result is obtained where $P(\text{Like}|\,\textit{Test-2})$ is 0. Therefore, *Test-1* should be recommended to the active user over *Test-2*. This is the same result that we obtained on visual inspection of this example.

When Laplacian smoothing is used, we will not obtain such binary probability values for the various classes, although one of the classes will obtain a much higher probability than the other. In such cases, all the test examples can be ranked in order of their predicted probability of a "Like" and recommended to the user. Laplacian smoothing is advisable because a single 0-value in the product-wise form of the expression on the right-hand side of the Bayes rule can result in a conditional probability value of 0.

4.4.4 Rule-based Classifiers

Rule-based classifiers can be designed in a variety of ways, including leave-one-out methods, as well as associative methods. A detailed description of the various types of rule-based classifiers is provided in [18, 22]. In the following, we will discuss only associative classifiers because they are based on the simple principles of association rules. A discussion of rule-based methods is provided in section 3.3 of Chapter 3. Refer to that section for the basic definitions of association rules and their measures, such as *support* and *confidence*. The support of a rule defines the fraction of rows satisfying both the antecedent and the consequent of a rule. The confidence of a rule is the fraction of rows satisfying the consequent, from the rows already known to satisfy the antecedent. The concept of a row "satisfying" the antecedent or consequent is described in more detail below.

Rule-based classifiers in content-based systems are similar to rule-based classifiers in collaborative filtering. In the item-item rules of collaborative filtering, both the antecedents and consequents of rules correspond to ratings of items. The main difference is that the antecedents of the rules in collaborative filtering correspond[3] to the ratings of various items,

[3] A different approach in collaborative filtering is to leverage user-user rules. For user-user rules, the antecedents and consequents may both contain the ratings of specific users. Refer to section 3.3 of Chapter 3.

whereas the antecedents of the rules in content-based methods correspond to the presence of specific keywords in item descriptions. Therefore, the rules are of the following form:

$$\text{Item contains keyword set A} \Rightarrow \text{Rating= Like}$$
$$\text{Item contains keyword set B} \Rightarrow \text{Rating=Dislike}$$

Therefore, an antecedent of a rule is said to "satisfy" a particular row (keyword representation of item), if all keywords in the antecedent are contained in that row. The consequents correspond to the various ratings, which we have assumed to be binary likes or dislikes for simplicity. A row is said to satisfy the consequent of that rule if the rating value in the consequent matches the dependent variable (rating) of that row.

The first step is to leverage the active user profile (i.e., training documents) to mine all the rules at a desired level of support and confidence. As in all content-based methods, the rules are specific to the active user at hand. For example, in the case of Table 4.1, the active user seems to be interested in classical music. In this case, an example of a relevant rule, which has 33% support and 100% confidence, is as follows:

$$\{\text{Classical, Symphony}\} \Rightarrow \text{Like}$$

Therefore, the basic idea is to mine all such rules for a given active user. Then, for target items where the user's interests are unknown, it is determined which rules are *fired*. A rule is fired by a target item description if the former's antecedent keywords are included in the latter. Once all such fired rules have been determined for the active user, the average rating in the consequents of these rules is reported as the rating of the target item. Many different heuristics exist for combining the ratings of the consequents. For example, we can choose to weight the rating with the confidence of the rule while computing the average. In the event that no rule is fired, default heuristics need to be used. For example, one can determine the average rating of the active user over all items and also determine the average rating of the target item by all users. The average of these two quantities is reported. Therefore, the overall approach for rule-based classification can be described as follows:

1. **(Training phase:)** Determine all the relevant rules from the user profile at the desired level of minimum support and confidence from the training data set $\mathcal{D}_L$.

2. **(Testing phase)** For each item description in $\mathcal{D}_U$, determine the fired rules and an average rating. Rank the items in $\mathcal{D}_U$ on the basis of this average rating.

One advantage of rule-based systems is the high level of interpretability they provide. For example, for a recommended item, one can use the keywords in the antecedent of the fired rules to give a recommendation to the target user about why she might like a particular item.

4.4.4.1 Example of Rule-based Methods

In order to illustrate the use of rule-based methods, we will provide an example of the rules generated for the active user in Table 4.1. At a support level of 33% and confidence level of

75%, the following rules are generated along with their support-confidence values:

Rule 1: $\{\text{Classical}\} \Rightarrow \text{Like} \quad (50\%, 100\%)$

Rule 2: $\{\text{Symphony}\} \Rightarrow \text{Like} \quad (33\%, 100\%)$

Rule 3: $\{\text{Classical, Symphony}\} \Rightarrow \text{Like} \quad (33\%, 100\%)$

Rule 4: $\{\text{Drums, Guitar}\} \Rightarrow \text{Dislike} \quad (33\%, 100\%)$

Rule 5: $\{\text{Drums}\} \Rightarrow \text{Dislike} \quad (33\%, 100\%)$

Rule 6: $\{\text{Beat}\} \Rightarrow \text{Dislike} \quad (33\%, 100\%)$

Rule 7: $\{\text{Guitar}\} \Rightarrow \text{Dislike} \quad (50\%, 75\%)$

The aforementioned rules are primarily sorted in order of decreasing confidence, with ties broken in order of decreasing support. It is evident that rule 2 is fired by *Test-1*, whereas rules 5 and 6 are fired by *Test-2*. Therefore, *Test-1* should be preferred over *Test-2* as a recommendation to the active user. Note that the rules fired by *Test-1* also provide an understanding of why it should be considered the best recommendation for the active user. Such explanations are often very useful in recommender systems both from the perspective of the customer and the perspective of the merchant.

4.4.5 Regression-Based Models

Regression-based models have the merit that they can be used for various types of ratings such as binary ratings, interval-based ratings, or numerical ratings. Large classes of regression models such as linear models, logistic regression models, and ordered probit models can be used to model various types of ratings. Here, we will describe the simplest model, which is referred to as *linear regression*. The bibliographic notes contain pointers to more sophisticated regression methods.

Let D_L be an $n \times d$ matrix representing the n documents in the labeled training set $\mathcal{D}_L$ on a lexicon of size d. Similarly, let $\overline{y}$ be an n-dimensional column vector containing the ratings of the active user for the n documents in the training set. The basic idea in linear regression is to assume that the ratings can be modeled as a linear function of the word frequencies. Let $\overline{W}$ be a d-dimensional row vector representing the coefficients of each word in the linear function relating word frequencies to the rating. Then, the linear regression model assumes that the word frequencies in the training matrix D_L are related to rating vectors as follows:

$$\overline{y} \approx D_L \overline{W}^T \tag{4.12}$$

Therefore, the vector $(D_L\overline{W}^T - \overline{y})$ is an n-dimensional vector of prediction errors. In order to maximize the quality of the prediction, one must minimize the squared norm of this vector. Furthermore, a regularization term $\lambda||\overline{W}||^2$ may be added to the objective function in order to reduce overfitting. This form of regularization is also referred to as *Tikhonov regularization*. Here, $\lambda > 0$ is the regularization parameter. Therefore, the objective function O can be expressed as follows:

$$\text{Minimize } O = ||D_L\overline{W}^T - \overline{y}||^2 + \lambda||\overline{W}||^2 \tag{4.13}$$

The problem can be solved by setting the gradient of this objective function with respect to $\overline{W}$ to 0. This results in the following condition:

$$D_L^T(D_L\overline{W}^T - \overline{y}) + \lambda\overline{W}^T = 0$$
$$(D_L^T D_L + \lambda I)\overline{W}^T = D_L^T\overline{y}$$

Table 4.2: The family of regression models and applicability to various types of ratings

Regression Model	Nature of Rating (Target Variable)
Linear Regression	Real
Polynomial Regression	Real
Kernel Regression	Real
Binary Logistic Regression	Unary, Binary
Multiway Logistic regression	Categorical, Ordinal
Probit	Unary, Binary
Multiway Probit	Categorical, Ordinal
Ordered Probit	Ordinal, Interval-based

The matrix $(D_L^T D_L + \lambda I)$ can be shown to be positive-definite, and therefore invertible (see Exercise 7). Therefore, we can directly solve for the weight vector $\overline{W}$ as follows:

$$\overline{W}^T = (D_L^T D_L + \lambda I)^{-1} D_L^T \overline{y} \tag{4.14}$$

Here, I is a $d \times d$ identity matrix. Therefore, a closed-form solution always exists for $\overline{W}^T$. For any given document vector (item description) $\overline{X}$ from the unlabeled set $\mathcal{D}_U$, its rating can be predicted as the dot product between $\overline{W}$ and $\overline{X}$. Tikhonov regularization uses the L_2-regularization term $\lambda \cdot ||W||^2$. It is also possible to use L_1-regularization, in which this term is replaced with $\lambda \cdot ||W||$. The resulting optimization problem does not have a closed-form solution, and gradient descent methods must be used. This form of regularization, also known as *Lasso* [242], can be used in the dual role of feature selection. This is because such methods have the tendency to select sparse coefficient vectors for $\overline{W}$, in which most components of $\overline{W}$ take on the value of 0. Such features can be discarded. Therefore, L_1-regularization methods provide highly interpretable insights about important subsets of features for the recommendation process. A detailed discussion of these models can be found in [22].

The linear model is one example of a regression model that is suitable for real-valued ratings. In practice, ratings might be unary, binary, interval-based, or categorical (small number of ordinal values). Various linear models have been designed for different types of target class variables. Some examples include logistic regression, probit regression, ordered probit regression, and nonlinear regression. Unary ratings are often treated as binary ratings, in which the unlabeled items are treated as negative instances. However, specialized positive-unlabeled (PU) models exist for such cases [364]. Ordered probit regression is especially useful for interval-based ratings. Furthermore, nonlinear regression models, such as polynomial regression and kernel regression, may be used in cases where the dependency between the features and target variables is nonlinear. When the number of features is large and the number of training samples is small, linear models usually perform quite well and may, in fact, outperform nonlinear models. This is because linear models are less prone to overfitting. Table 4.2 shows the mapping between the various regression models and the nature of the target variable (rating).

4.4.6 Other Learning Models and Comparative Overview

As the problem of content-based filtering is a direct application of classification and regression modeling, many other techniques can be used from the literature. A detailed discussion of various classification models can be found in [18, 86, 242, 436]. The decision-tree model discussed in Chapter 3 can also be applied to content-based methods. However, for very high-dimensional data, such as text, decision trees often do not provide very effective results. Experimental results [477] have shown the poor performance of decision trees compared to other classification methods. Even though rule-based classifiers are closely related to decision trees, they can often provide superior results because they do not assume a strict partitioning of the feature space. Successful results have been obtained with rule-based classifiers for email classification [164, 165]. Among the various models, the Bayes approach has the advantage that it can handle all types of feature variables with the use of an appropriate model. Regression-based models are very robust, and they can handle all forms of target variables. Logistic regression and ordered probit regression are particularly useful for binary and interval-based ratings.

In the case of binary ratings, support vector machines [114] are a popular choice. Support vector machines are very similar to logistic regression; the main difference is that the loss is quantified as a *hinge-loss* rather than with the use of the *logit* function. Support vector machines are highly resistant to overfitting, and numerous off-the-shelf implementations exist. Both linear and kernel-based support vector machines have been used in the literature. For the case of high-dimensional data, such as text, it has been observed that linear support vector machines are sufficient. For such cases, specialized methods with linear performance [283] have been designed. Although neural networks [87] can be used for building arbitrarily complex models, they are not advisable when the amount of available data is small. This is because neural networks are sensitive to the noise in the underlying data, and they can overfit the training data when its size is small.

4.4.7 Explanations in Content-Based Systems

Since content-based systems extract models based on content features, they often provide highly interpretable insights for the recommendation process. For example, in a movie recommendation system, it is often useful to present the user with a reason as to why they might like a specific movie, such as the presence of a particular genre feature, actor feature, or an informative set of keywords. As a result, the active user will be able to make a more informed choice about whether they should watch that movie. Similarly, a descriptive set of keywords in a music recommendation system can provide a better understanding of why a user might like a specific track. As a specific example, Pandora Internet radio [693] provides explanations for recommended tracks, such as the following:

> "We are playing this track because it features trance roots, four-on-the-floor beats, disco influences, a knack for catchy hooks, beats made for dancing, straight drum beats, clear pronunciation, romantic lyrics, storytelling lyrics, subtle buildup/breakdown, a rhythmic intro, use of modal harmonies, the use of chordal patterning, light drum fills, emphasis on instrumental performance, a synth bass riff, synth riffs, subtle use of arpeggiatted synths, heavily effected synths, and synth swoops."

Each of these reported characteristics can be viewed as an important feature, which are responsible for the classification of the test instance as a "like." Note that such detailed explanations are often lacking in collaborative systems, where a recommendation can be explained only in terms of similar items, rather than in terms of detailed *characteristics* of these items. The nature and extent of the insights are, however, highly sensitive to the specific model used. For example, the Bayes model and rule-based systems are very highly interpretable in terms of the specific causality of the classification. Consider the example of Table 4.1 in which the following rule is fired for the example *Test-1*:

$$\{\text{Symphony}\} \Rightarrow \text{Like}$$

It is evident that the user has been recommended the item described by *Test-1* because it is a symphony. Similarly, in the Bayes classification model, it is evident that the contribution of $P(\text{Symphony}|\text{Like})$ is largest in the multiplicative formula for classification. Other models, such as linear and nonlinear regression models, are harder to interpret. Nevertheless, certain instances of these models, such as *Lasso*, provide important insights about the most relevant features for the classification process.

4.5 Content-Based Versus Collaborative Recommendations

It is instructive to compare content-based methods with the collaborative methods discussed in Chapters 2 and 3. Content-based methods have several advantages and disadvantages as compared to collaborative methods. The advantages of content-based methods are as follows:

1. When a new item is added to a ratings matrix, it has no ratings from the various users. None of the memory-based and model-based collaborative filtering methods would recommend such an item, because sufficient ratings are not available for recommendation purposes. On the other hand, in the case of content-based methods, the previous items rated by a given user are leveraged to make recommendations. Therefore, as long as the *user* is not new, meaningful recommendations can always be made in a way that treats the new item in a fair way in comparison to other items. Collaborative systems have cold-start problems *both* for new users and new items, whereas content-based systems have cold-start problems only for new users.

2. As discussed in the previous section, content-based methods provide explanations in terms of the features of items. This is often not possible with collaborative recommendations.

3. Content-based methods can generally be used with off-the-shelf text classifiers. Furthermore, each user-specific classification problem is generally not very large, as in the case of collaborative systems. Therefore, they are particularly easy to use with relatively little engineering effort.

On the other hand, content-based methods also have several disadvantages that are not present in collaborative recommenders.

1. Content-based systems tend to find items that are similar to those the user has seen so far. This problem is referred to as *overspecialization.* It is always desirable to have a certain amount of novelty and serendipity in the recommendations. Novelty refers to the fact that the item is different from one the user has seen in the past. Similarly, serendipity implies that the user would like to discover *surprisingly relevant* items that they might otherwise not have found. This is a problem for content-based systems in which attribute-based classification models tend to recommend very similar items. For example, if a user has never listened to or rated classical music, a content-based

system will typically not recommend such an item to her because classical music will be described by very different attribute values than those that the user has rated so far. On the other hand, a collaborative system might recommend such items by leveraging the interests of her *peer* group. For example, a collaborative system might automatically infer a *surprising* association between certain pop songs and classical songs and recommend the corresponding classical songs to a user who is a pop music lover. Overspecialization and lack of serendipity are the two most significant challenges of content-based recommender systems.

2. Even though content-based systems help in resolving cold-start problems for new *items*, they do not help in resolving these problems for new *users*. In fact, for new users, the problem in content-based systems may be more severe because a text classification model usually requires a sufficient number of training documents to avoid overfitting. It would seem rather wasteful that the training data for all the other users is discarded and only the (small) training data set specific to a single user is leveraged.

In spite of these disadvantages, content-based systems often complement collaborative systems quite well because of their ability to leverage content-based knowledge in the recommendation process. This complementary behavior is often leveraged in *hybrid recommender systems* (cf. Chapter 6), in which the goal is to combine the best of both worlds to create an even more robust recommender system. In general, content-based systems are rarely used in isolation, and they are generally used in combination with other types of recommender systems.

4.6 Using Content-Based Models for Collaborative Filtering

There is an interesting connection between collaborative filtering models and content-based methods. It turns out that content-based methods can be directly used for collaborative filtering. Although the content description of an item refers to its descriptive keywords, it is possible to envision scenarios, where the ratings of users are leveraged to define content-based descriptions. For each item, one can concatenate the user name (or identifier) of a user who has rated the item with the value of this rating to create a new "keyword." Therefore, each item would be described in terms of as many keywords as the number of ratings of that item. For example, consider a scenario where the descriptions of various movies are as follows:

Terminator: John#Like, Alice#Dislike, Tom#Like
Aliens: John#Like, Peter#Dislike, Alice#Dislike, Sayani#Like
Gladiator: Jack#Like, Mary#Like, Alice#Like

The "#" symbol is used to denote the demarcation of the concatenation and ensure a unique keyword for each user-rating combination. This approach is generally more effective, when the number of possible ratings is small (e.g., unary or binary ratings). After such a content-based description has been constructed, it can be used in conjunction with an off-the-shelf content-based algorithm. There is almost a one-to-one mapping between the resulting methods and various collaborative filtering models, depending on the base method used for classification. Although each such technique maps to a collaborative filtering model, the converse is not true because many collaborative filtering methods cannot be captured by this approach. Nevertheless, we provide some examples of the mapping:

1. A nearest neighbor classifier on this representation approximately maps to an item-based neighborhood model for collaborative filtering (cf. section 2.3.2 of Chapter 2).

2. A regression model on the content approximately maps to a user-wise regression model for collaborative filtering (cf. section 2.6.1 of Chapter 2).

3. A rule-based classifier on the content approximately maps to an user-wise rule-based classifier for collaborative filtering (cf. section 3.3.2 of Chapter 3).

4. A Bayes classifier on the content approximately maps to a user-wise Bayes model for collaborative filtering (cf. Exercise 4 of Chapter 3).

Therefore, many methods for collaborative filtering can be captured by defining an appropriate content representation and directly using off-the-shelf content-based methods. These observations are important because they open up numerous opportunities for hybridization. For example, one can combine the ratings-based keywords with actual descriptive keywords to obtain an even more robust model. In fact, this approach is often used in some hybrid recommendation systems. Such an approach no longer wastes the available ratings data from other users, and it combines the power of content-based and collaborative models within a unified framework.

4.6.1 Leveraging User Profiles

Another case in which collaborative filtering-like models can be created with content attributes is when user profiles are available in the form of specified keywords. For example, users may choose to specify their particular interests in the form of keywords. In such cases, instead of creating a local classification model for each user, one can create a global classification model over all users by using the user features. For each user-item *combination*, a content-centric representation can be created by using the Kronecker-product of the attribute vectors of the corresponding user and item [50]. A classification or regression model is constructed on this representation to map user-item *combinations* to ratings. Such an approach is described in detail in section 8.5.3 of Chapter 8.

4.7 Summary

This chapter introduces the methodology of content-based recommender systems in which user-specific training models are created for the recommendation process. The content attributes in item descriptions are combined with user ratings to create user profiles. Classification models are created on the basis of these models. These models are then used to classify item descriptions that have as of yet not been rated by the user. Numerous classification and regression models are used by such systems, such as nearest-neighbor classifiers, rule-based methods, the Bayes method, and linear models. The Bayes method has been used with great success in a variety of scenarios because of its ability to handle various types of content. Content-based systems have the advantage that they can handle cold-start problems with respect to new items, although they cannot handle cold-start problems with respect to new users. The serendipity of content-based systems is relatively low because content-based recommendations are based on the content of the items previously rated by the user.

4.8 Bibliographic Notes

The earliest content-based systems were attributed to the work in [60] and the *Syskill & Webert* [82, 476–478] systems. *Fab*, however, uses a partial hybridization design in which the peer group is determined using content-based methods, but the ratings of other users are leveraged in the recommendation process. The works in [5, 376, 477] provide excellent overview articles on content-based recommender systems. The latter work was designed for finding interesting Websites, and therefore numerous text classifiers were tested for their effectiveness. In particular, the work in [82] provides a number of useful pointers about the relative performance of various content-based systems. Probabilistic methods for user modeling are discussed in [83]. The work in [163, 164] is notable for its use of rule-based systems in e-mail classification. Rocchio's relevance feedback [511] was also used during the early years, although the work does not have theoretical underpinnings, and it can often perform poorly in many scenarios. Numerous text classification methods, which can be used for content-based recommendations, are discussed in [21, 22, 400]. A discussion of the notion of serendipity in the context of information retrieval is provided in [599]. Some content-based systems explicitly filter out very similar items in order to improve serendipity [85]. The work in [418] discusses how one can go beyond accuracy metrics to measure the quality of a recommender system.

Methods for feature extraction, cleaning, and feature selection in text classification are discussed in [21, 364, 400]. The extraction of the main content block from a Web page containing multiple blocks is achieved with the help of the tree-matching algorithm can be found in [364, 662]. The use of visual representations for extracting content structure from Web pages is described in [126]. A detailed discussion of feature selection measures for classification may be found in [18]. A recent text classification survey [21] discusses feature selection algorithms for the specific case of text data.

Numerous real-world systems have been designed with the use of content-based systems. Some of the earliest are *Fab* [60] and *Syskill & Webert* [477]. An early system, referred to as *Personal WebWatcher* [438, 439], makes recommendations by learning the interests of users from the Web pages that they visit. In addition, the Web pages that are linked to by the visited page are used in the recommendation process. The *Letizia* system [356] uses a Web-browser extension to track the user's browsing behavior, and uses it to make recommendations. A system known as *Dynamic-Profiler* uses a pre-defined taxonomy of categories to make news recommendations to users in real time [636]. In this case, user Web logs are used to learn the preferences and make personalized recommendations. The *IfWeb* system [55] represents the user interests in the form of a semantic network. The *WebMate* system [150] learns user profiles in the form of keyword vectors. This system is designed for keeping track of positive user interests rather than negative ones. The general principles in Web recommendations are not very different from those of news filtering. Methods for performing news recommendations are discussed in [41, 84, 85, 392, 543, 561]. Some of these methods use enhanced representations, such as *WordNet*, to improve the modeling process. Web recommender systems are generally more challenging than news recommender systems because the underlying text is often of lower quality. The *Citeseer* system [91] is able to discover interesting publications in a bibliographic database by identifying the common citations among the papers. Thus, it explicitly uses citations as a content mechanism for determination of similarity.

Content-based systems have also been used in other domains such as books, music, and movies. Content-based methods for book recommendations are discussed in [448]. The main challenge in music recommendations is the semantic gap between easily available features and the likelihood of a user appreciating the music. This is a common characteristic between the music and the image domains. Some progress in bridging the semantic gap has been made in [138, 139]. *Pandora* [693] uses the features extracted in the Music Genome Project to make recommendations. The *ITR* system discusses how one might use text descriptions [178] of items (e.g., book descriptions or movie plots) to make recommendations. Further work [179] shows how one might integrate tags in a content-based recommender. The approach uses linguistic tools such as *WordNet* to extract knowledge for the recommendation process. A movie recommendation system that uses text categorization is the *INTIMATE* system [391]. A method that combines content-based and collaborative recommender systems is discussed in [520]. A broader overview of hybrid recommender systems is provided in [117]. A potential direction of work, mentioned in [376], is to enhance content-based recommender systems with encyclopedic knowledge [174, 210, 211], such as that gained from Wikipedia. A few methods have been designed that use Wikipedia for movie recommendation [341]. Interestingly, this approach does not improve the accuracy of the recommender system. The application of advanced semantic knowledge in content-based recommendations has been mentioned as a direction of future work in [376].

4.9 Exercises

1. Consider a scenario in which a user provides like/dislike ratings of a set of 20 items, in which she rates 9 items as a "like" and the remaining as a "dislike." Suppose that 7 item descriptions contain the word "thriller," and the user dislikes 5 of these items. Compute the Gini index with respect to the original data distribution, and with respect to the subset of items containing the word "thriller." Should feature selection algorithms retain this word in the item descriptions?

2. Implement a rule-based classifier with the use of association pattern mining.

3. Consider a movie recommender system in which movies belong to one or more of the genres illustrated in the table, and a particular user provides the following set of ratings to each of the movies.

Genre ⇒ Movie-Id ⇓	Comedy	Drama	Romance	Thriller	Action	Horror	Like or Dislike
1	1	0	1	0	0	0	Dislike
2	1	1	1	0	1	0	Dislike
3	1	1	0	0	0	0	Dislike
4	0	0	0	1	1	0	Like
5	0	1	0	1	1	1	Like
6	0	0	0	0	1	1	Like
Test-1	0	0	0	1	0	1	?
Test-2	0	1	1	0	0	0	?

Mine all the rules with at least 33% support and 75% confidence. Based on these rules, would you recommend the item *Test-1* or *Test-2* to the user?

4. Implement a Bayes classifier with Laplacian smoothing.

5. Repeat Exercise 3 with the use of a Bayes classifier. Do not use Laplacian smoothing. Explain why Laplacian smoothing is important in this case.

6. Repeat Exercise 3 with the use of a 1-nearest neighbor classifier.

7. For a training data matrix D, regularized least-squares regression requires the inversion of the matrix $(D^T D + \lambda I)$, where $\lambda > 0$. Show that this matrix is always invertible.

8. The χ^2 distribution is defined by the following formula, as discussed in the chapter:

$$\chi^2 = \sum_{i=1}^{p} \frac{(O_i - E_i)^2}{E_i}$$

Show that for a 2×2 contingency table, the aforementioned formula can be rewritten as follows:

$$\chi^2 = \frac{(O_1 + O_2 + O_3 + O_4) \cdot (O_1 O_4 - O_2 O_3)^2}{(O_1 + O_2) \cdot (O_3 + O_4) \cdot (O_1 + O_3) \cdot (O_2 + O_4)}$$

Here, $O_1 \ldots O_4$ are defined in the same way as in the tabular example in the text.