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

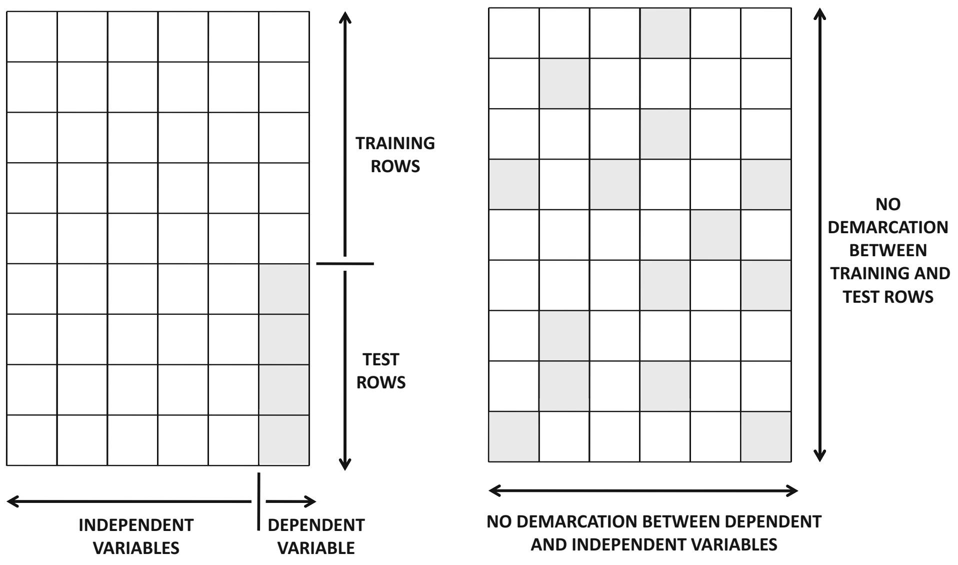
The neighborhood-based methods of the previous chapter can be viewed as generalizations of k-nearest neighbor classifiers, which are commonly used in machine learning. These methods are instance-based methods, whereby a model is not specifically created up front for prediction other than an optional 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.

© Springer International Publishing Switzerland 2016
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:

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

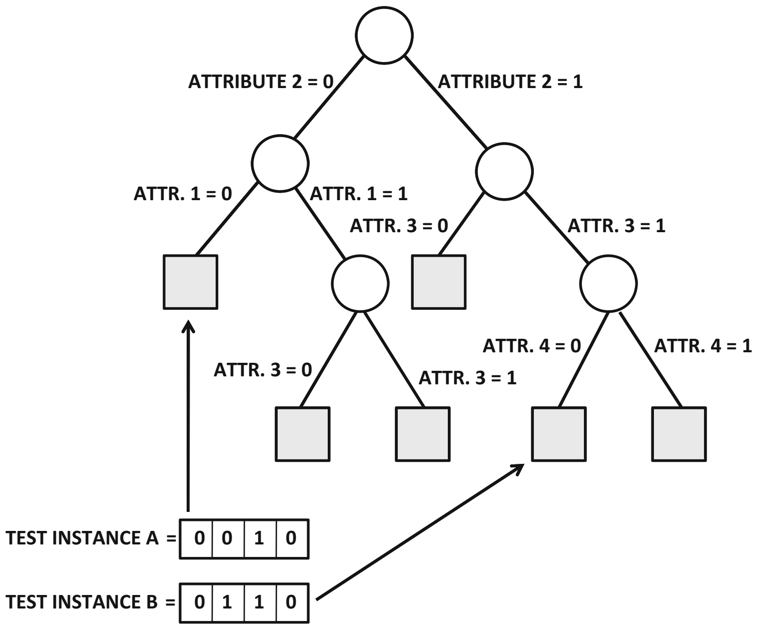


Figure 3.2: Example of a decision tree with four binary attributes

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 $\overline{e_1} \dots \overline{e_d}$ 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 $\mathcal{T} = \{T_1 \dots T_m\}$, 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 "$\Rightarrow$" 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 $v_1 \dots v_l$. 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 $\{v_1 \dots v_l\}$. 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 $s \in \{1 \dots l\}$, 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 $s \in \{1 \dots l\}$, 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 $s \in \{1 \dots l\}$ 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 $P(\mathit{Observed\ ratings\ in\ } I_u | r_{uj} = v_s)$ 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:

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

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:

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

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 $s \in \{1 \dots l\}$, 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:

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

   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 $P(r_{uj} = v_s | \mathit{Observed\ ratings\ in\ } I_u)$, 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:

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

   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

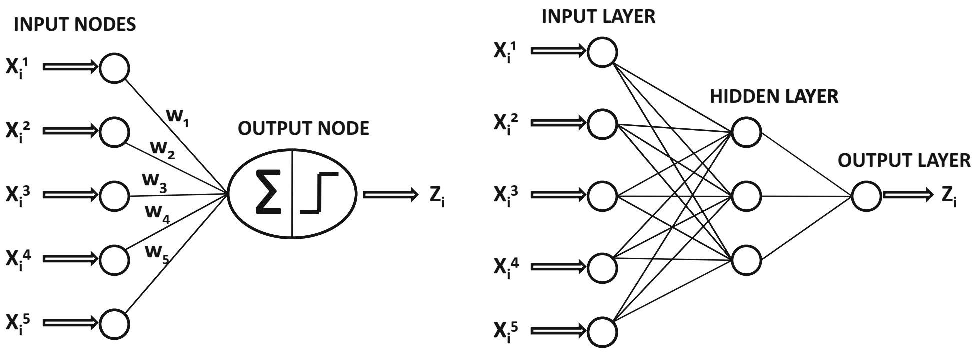
In this section, we will provide a simple example of the aforementioned approach, when neural networks are used as black-boxes to implement the approach. For the purpose of the following discussion, we will assume that the reader is already familiar with the basics of neural networks [87]. Nevertheless,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

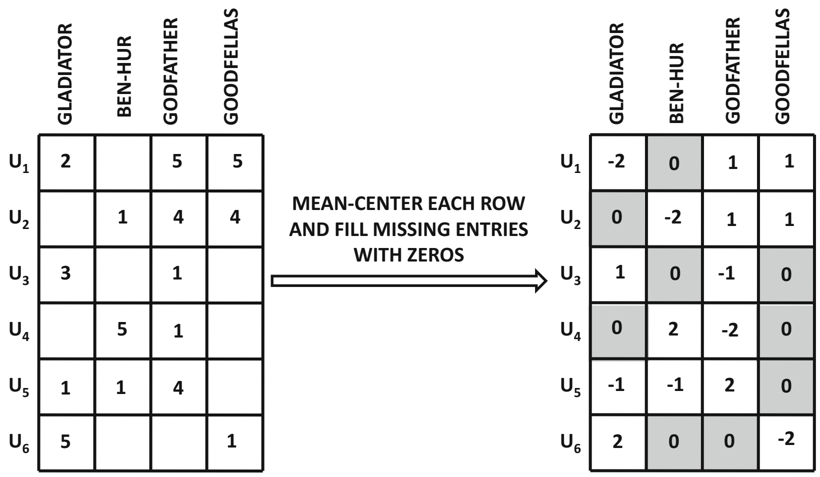
used by the learning algorithm. The most basic architecture of the neural network is the *perceptron*, which contains a set of input nodes and an output node. An example of a perceptron is shown in Figure 3.3(a). For a data set containing d different dimensions, there are d different input units. The 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


MEAN-CENTER EACH ROW AND FILL MISSING ENTRIES WITH ZEROS

	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

	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.

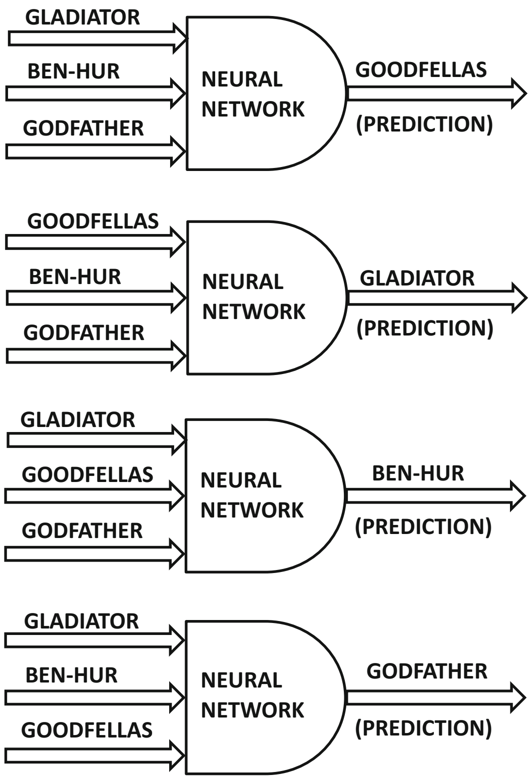


Figure 3.5: Neural networks for predicting and updating missing entries. Shaded entries of Figure 3.4 are iteratively updated by the neural networks.

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

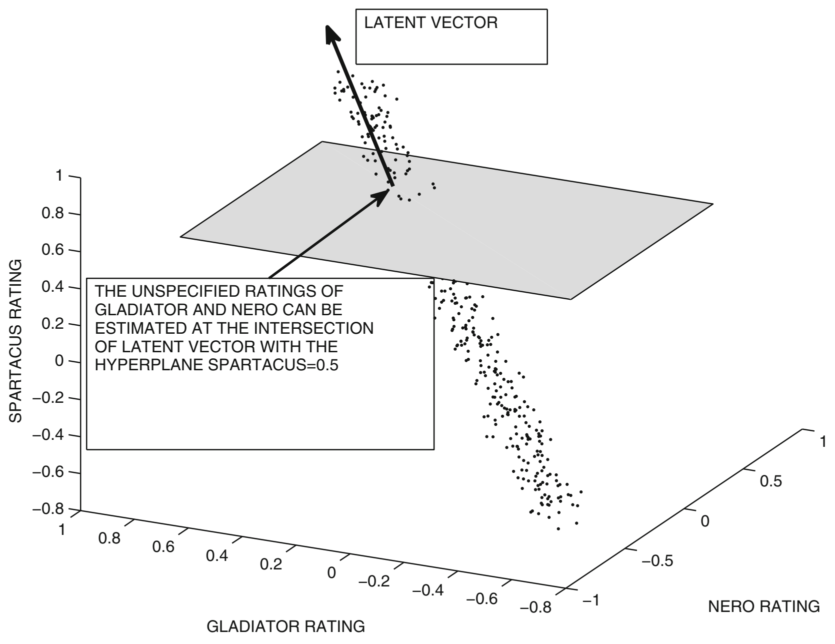


Figure 3.6: Leveraging correlation-based redundancies in missing data estimation for a user whose only specified rating is a value of 0.5 for the movie *Spartacus*

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:

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

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

$$\mathbf{R} \qquad \mathbf{U} \qquad \mathbf{V^T}$$

(a) Example of rank-2 matrix factorization

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

$$\mathbf{R}$$

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

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

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:

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

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:

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

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 $V$;
  $S = \{(i, j) : r_{ij}$ 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.

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

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:

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

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:

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

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:

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

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 $v_{j1} \dots v_{jk}$. The terms $u_{i1} \dots u_{ik}$ are treated as constant values, whereas $v_{j1} \dots v_{jk}$ 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 $u_{i1} \dots u_{ik}$. The terms $v_{j1} \dots v_{jk}$ are treated as constant values, whereas $u_{i1} \dots u_{ik}$ 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:

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

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:

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

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

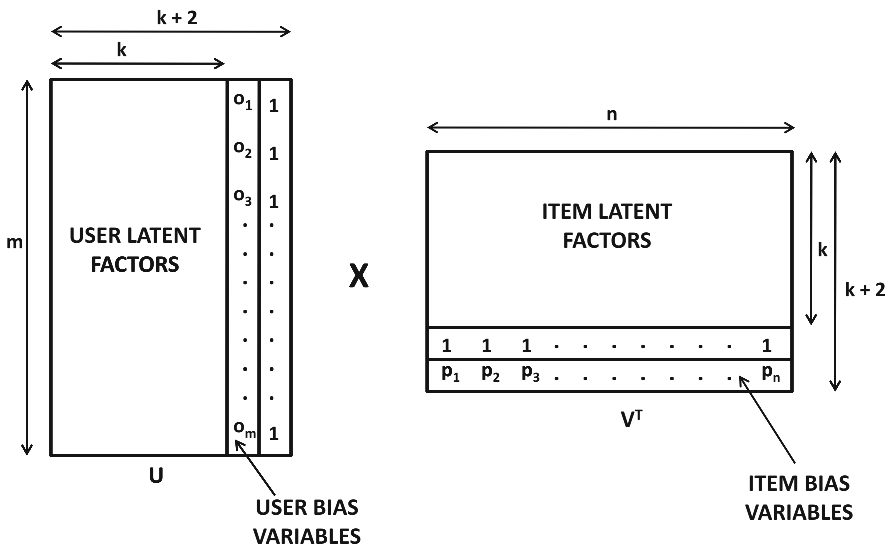


Figure 3.11: Incorporating user and item biases in the latent factor model

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:

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

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 eigen-decomposition 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 $R^T R$, 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 $R^T R$ 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 $R^T R$, 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:

$$\begin{aligned} R_f &\approx \begin{pmatrix} 1.129 & -2.152 \\ 1.937 & 0.640 \\ 1.539 & 0.873 \\ -2.400 & -0.341 \\ -2.105 & 0.461 \end{pmatrix} \begin{pmatrix} 0.431 & 0.246 & 0.386 & -0.518 & -0.390 & -0.431 \\ -0.266 & 0.668 & -0.249 & 0.124 & -0.578 & 0.266 \end{pmatrix} \\ &= \begin{pmatrix} 1.0592 & -1.1604 & 0.9716 & -0.8515 & 0.8040 & -1.0592 \\ 0.6636 & 0.9039 & 0.5881 & -0.9242 & -1.1244 & -0.6636 \\ 0.4300 & 0.9623 & 0.3764 & -0.6891 & -1.1045 & -0.4300 \\ -0.9425 & -0.8181 & -0.8412 & 1.2010 & 1.1320 & 0.9425 \\ -1.0290 & -0.2095 & -0.9270 & 1.1475 & 0.5535 & 1.0290 \end{pmatrix} \end{aligned}$$

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.

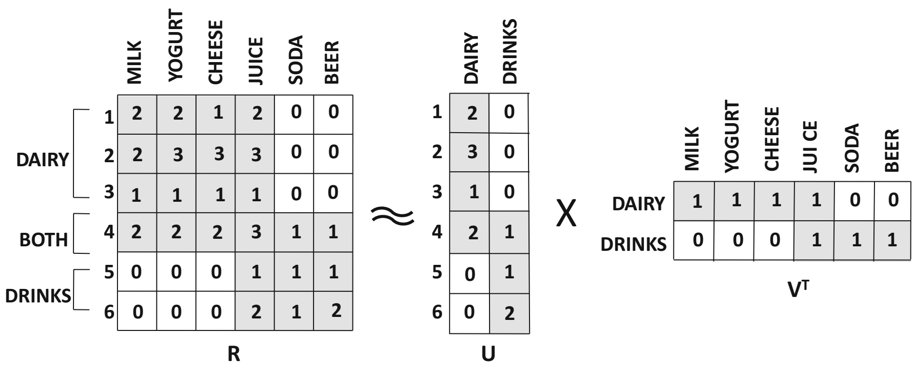


Figure 3.12: An example of non-negative matrix factorization

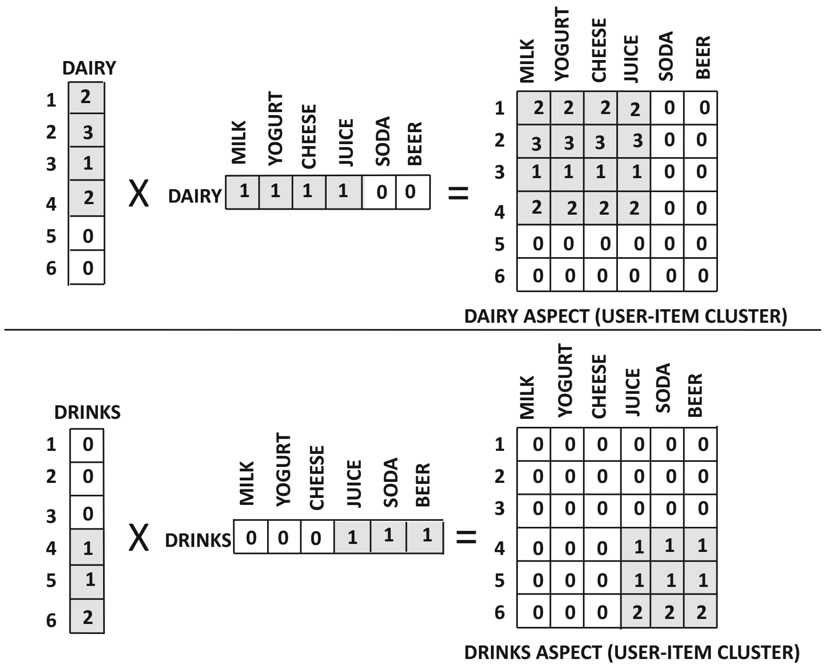
This sum-of-parts decomposition can be represented mathematically as follows. The rank-k matrix factorization UV^T can be decomposed into k components by expressing the matrix product in terms of the k columns $\overline{U_i}$ and $\overline{V_i}$, respectively, of U and V:

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

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

$$\begin{array}{c|c} & \text{DAIRY} \\ \hline 1 & 2 \\ 2 & 3 \\ 3 & 1 \\ 4 & 2 \\ 5 & 0 \\ 6 & 0 \end{array} \times \begin{array}{c|cccccc} & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUICE} & \text{SODA} & \text{BEER} \\ \hline \text{DAIRY} & 1 & 1 & 1 & 1 & 0 & 0 \end{array} = \begin{array}{c|cccccc} & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUICE} & \text{SODA} & \text{BEER} \\ \hline 1 & 2 & 2 & 2 & 2 & 0 & 0 \\ 2 & 3 & 3 & 3 & 3 & 0 & 0 \\ 3 & 1 & 1 & 1 & 1 & 0 & 0 \\ 4 & 2 & 2 & 2 & 2 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{c|c} & \text{DRINKS} \\ \hline 1 & 0 \\ 2 & 0 \\ 3 & 0 \\ 4 & 1 \\ 5 & 1 \\ 6 & 2 \end{array} \times \begin{array}{c|cccccc} & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUICE} & \text{SODA} & \text{BEER} \\ \hline \text{DRINKS} & 0 & 0 & 0 & 1 & 1 & 1 \end{array} = \begin{array}{c|cccccc} & \text{MILK} & \text{YOGURT} & \text{CHEESE} & \text{JUICE} & \text{SODA} & \text{BEER} \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 1 & 1 \\ 5 & 0 & 0 & 0 & 1 & 1 & 1 \\ 6 & 0 & 0 & 0 & 2 & 2 & 2 \end{array}$$

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

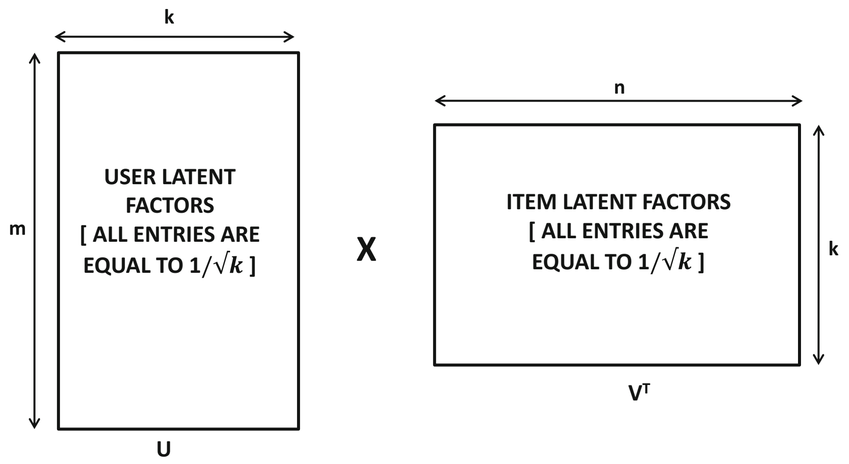

k
m
USER LATENT FACTORS [ALL ENTRIES ARE EQUAL TO $1/\sqrt{k}$]
U
X
n
ITEM LATENT FACTORS [ALL ENTRIES ARE EQUAL TO $1/\sqrt{k}$]
k
V^T

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:

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

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:

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

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:

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

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

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?