Chapter 8

Context-Sensitive Recommender Systems

"*For me context is the key – from that comes the understanding of everything.*"– Kenneth Noland

8.1 Introduction

Context-sensitive recommender systems tailor their recommendations to additional information that defines the specific situation under which recommendations are made. This additional information is referred to as the *context*. Some examples of context are as follows:

1. *Time:* Recommendations can be affected by many aspects of time, such as weekdays, weekends, holidays, and so on. A recommendation that is relevant to the morning context, may not be relevant in the evening and vice versa. Clothing recommendations during summer and winter may be different. A number of time-sensitive recommendation methods are discussed in Chapter 9. In fact, some of the methods discussed in this chapter, such as pre-filtering and post-filtering, are re-examined in Chapter 9 in the temporal context.

2. *Location:* With the increasing popularity of GPS-enabled mobile phones, location-sensitive recommendations have gained increasing importance in recent years. For example, a traveling user might wish to determine a recommendation for a restaurant in her locality. Context-sensitive systems can provide more relevant recommendations by using the location as a context. The next chapter will provide several examples of location-aware systems.

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_8

3. *Social information:* The social context is often important from the perspective of recommender systems. For example, the choice of a user's friends, tags, and social circles can affect the recommendation process. Similarly, a person might choose to watch a different movie depending on whether she is watching it with her parents or with her boyfriend [5]. Social recommender systems are discussed in Chapters 10 and 11. Some of these systems can be considered contextual systems as well.

The context of a user can be detected in a variety of ways. In some cases, it can be learned with little effort because the data is already available. For example, the GPS receiver on a mobile phone would indicate a customer's location, and the time-stamp of a customer transaction indicates the time. Such methods are referred to as implicit collection methods [466]. In other cases, the context is not quite as readily available. For example, it can be learned explicitly by gathering information through surveys or other means. Finally, in some cases, data mining and inference tools can be used to gather contextual information.

In traditional recommender systems with user set U and item set I, the set of possibilities in $U \times I$ is mapped to a rating. This mapping results in (an incompletely specified) ratings matrix of size $|U| \times |I|$. In a context-aware system, an additional set of contextual possibilities is present in the set C. For example, the set C might be $\{morning, afternoon, night\}$, with the context corresponding to the time of day. In this case, it is no longer possible to map the possibilities in $U \times I$ to the ratings, because the same user might have different preferences for an item depending on whether the time is in the morning, afternoon, or night. The context must be included in the mapping in order to provide a more refined and accurate recommendation. Therefore, in context-sensitive recommender systems, the possibilities in $U \times I \times C$ are mapped to the ratings. Formally, the function h_R, which maps the user, items, and context to the rating, can be written as follows:

$$h_R : U \times I \times C \to rating$$

The function h_R is subscripted with R to denote the data set to which it is applied. In this case, the ratings data R is a 3-dimensional ratings data *cube* corresponding to the user, item, and context. It is possible to use multiple types of context within a single recommendation application. For example, aside from the time, one might use location, weather, or the social context. Therefore, one might have multiple contextual *dimensions*. This would result in an multidimensional cube for representing the ratings. As we will see later in this chapter, the notion of multidimensional representation can be used to seamlessly represent a variety of different contexts. In this chapter, we will work with such an multidimensional contextual model.

This chapter is organized as follows. Section 8.2 discusses the multidimensional model for contextual recommendations. A contextual pre-filtering and reduction approach is described in section 8.3. Post-filtering methods are described in section 8.4. The process of incorporating context directly into the recommendation process is discussed in section 8.5. A summary is given in section 8.6.

8.2 The Multidimensional Approach

The traditional problem of recommendations can be viewed as that of learning a mapping function from the user-item combinations to the ratings. The corresponding function f_R may be defined as follows:

$$f_R : U \times I \to rating \tag{8.1}$$

The 2-dimensional ratings matrix is used by this function to create the mapping. Therefore, this function maps a data point in the 2-dimensional space of users and items to ratings. Of course, the dimensions could, in principle, correspond not just to users or items, but to any type of context. This general principle motivates the *multidimensional approach* [6] to recommendations, in which the rating problem is seen as that of mapping a set of w different dimensional values to a rating.

$$g_R : D_1 \times D_2 \ldots \times D_w \Rightarrow rating$$

In this case, the ratings data R contain w different dimensions that are mapped to ratings, just as the 2-dimensional user-item combinations are mapped to ratings in the traditional setting. This results in a w-dimensional cube rather than a 2-dimensional matrix. The w different dimensions are denoted by $D_1 \ldots D_w$. Note that two of these dimensions will always be users and items, as in the classical case of multidimensional recommendations, but the other values of D_i might correspond to other contexts. For example, these contexts could correspond to time, location, and so on. Therefore, the traditional recommendation problem can be viewed as a special case of the multidimensional approach in which the only two dimensions are users and items. A nice way of viewing this generalization is as an online analytical processing (OLAP) data cube [145], which is traditionally used in data warehousing. An example of such an OLAP cube with three dimensions is shown in Figure 8.1 corresponding to user, item (movie), and time. Each cell in this cube contains a rating for a particular user, item, and time combination. Although the context in this case is an ordered variable (time), it is usually treated as a discrete value during the analytical process. Furthermore, some representations of time, such as weekday, weekend, or season, are not ordered. Similarly, the contextual dimension could very well have been location, which is not an ordered variable. Treating the contextual dimensions in a discrete setting is essential to the data cube paradigm.

The rating function g_R is defined as a partial function, in which the number of arguments is equal to the number of dimensions w. In the example of Figure 8.1, the rating function $g_R(David, Terminator, 9\ PM)$ refers to the rating of user David when he watches the movie *Terminator* at 9 PM. This cell is shaded in Figure 8.1. The mapping function g_R is referred to as *partial* because it is defined only for the subset of cells corresponding to observed rating values. The remaining values need to be learned in a data-driven manner for making contextual recommendations. Note that the context can be a property of the user, a property of the item, a property of the user-item combination, or a completely independent property. For example, when David watches *Terminator* at 9 PM, the context of 9 PM relates to both, because the user watches the movie at that specific time and the time does not exclusively relate to either the user or the item. However, it is also possible for the context to relate to only one of the two. For example, consider a movie recommendation application in which the movies are recommended to a user based on the ratings matrix and also her demographic characteristics. In such a case, the context is clearly related to the user. In general, however, it is not important who the context is related to, because it is treated as a completely independent entity from the user or the item. Therefore, a separate dimension is assigned to each context, just as there are individual dimensions assigned to the user and item, respectively. This abstraction helps in addressing the most general cases of context-sensitive recommendations.

At a more general level, this idea can be related to that of querying for top-ranked combinations of values with the use of two disjoint subsets from $D_1 \ldots D_w$. The selected subsets of dimensions in $D_1 \ldots D_w$ are either "what" dimensions, or they are "for whom"

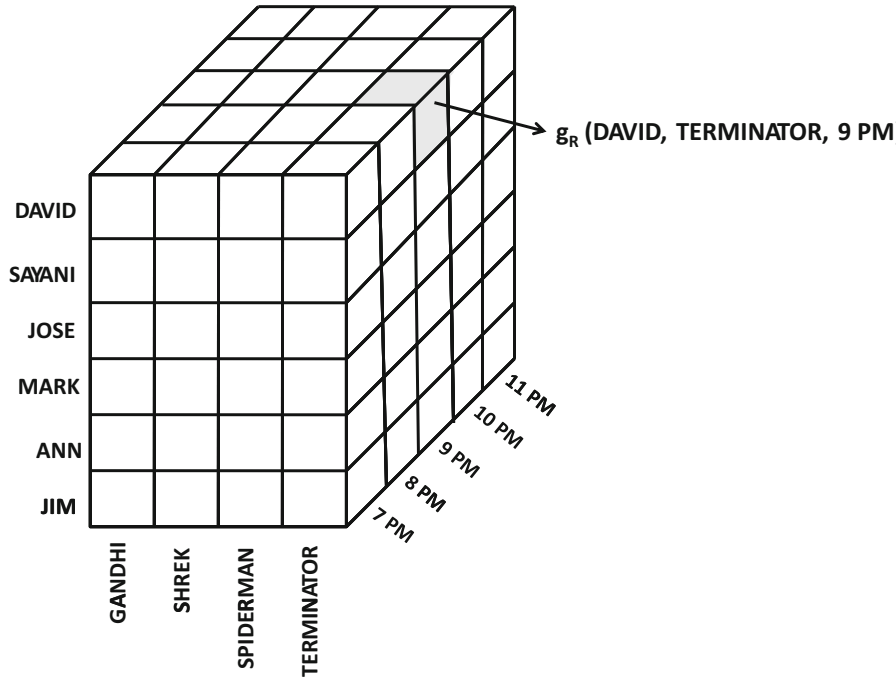

Figure 8.1: The multidimensional rating cube

dimensions. Each of the dimensions belongs to one of the two categories but it cannot belong to both categories. A typical query is of the following form:

Determine the top-k possibilities in the "what" dimensions for a particular set of specified values in the "for whom" dimensions.

In traditional recommender systems, the item dimension always belongs to the former category, whereas the user dimension always belongs to the latter category. However, in multidimensional recommender systems, this constraint does not apply. Formally, the problem of multidimensional recommendations can be defined as follows [6]:

Definition 8.2.1 (Multidimensional Recommendations) *Given the recommendation space* $D_1 \times D_2 \times \ldots D_w$ *and the rating function* $g_R : D_1 \times D_2 \ldots \times D_w \rightarrow$ rating*, the recommendation problem is defined by selecting certain "what" dimensions* $D_{i_1} \ldots D_{i_p}$ *and certain "for whom" dimensions* $D_{j_1} \ldots D_{j_q}$ *that do not overlap, and recommending for a query tuple* $(d_{j_1} \ldots d_{j_q}) \in D_{j_1} \times \ldots \times D_{j_q}$ *the top-k tuples* $(d_{i_1} \ldots d_{i_p}) \in D_{i_1} \times \ldots \times D_{i_p}$ *with the maximum predicted value of the rating* $g_R(d_1, d_2, \ldots, d_w)$*.*

In other words, a ranked list of the "what" dimension combinations are recommended in response to "for whom" queries. The traditional 2-dimensional model of recommendations is a special case of this scenario in which items are recommended to users. Therefore, items always belong to the "what" category and users always belong to the "for whom" category. In a multidimensional recommender system, a more general framework is used, where the segmentation between the "what" and the "for whom" items might be arbitrary. For example, one might recommend best item-time combinations for each user, or one might recommend best user-time combinations for each item. Alternatively, one might recommend the best

time(s) for each user-item combination. Note that both users and items belong to the "for whom" category in the last case. In a social application, one might wish to recommend the best companions to watch a movie with for a particular user-movie combination. Note that the union of the "what" and "for whom" dimensions might be a proper subset of the full set of w dimensions. For example, consider the case where $w = 4$ and we have the time and location context in addition to the user and item dimensions. It is possible for the query to completely ignore the time and only use the location context to make recommendations.

As is reflected in the above descriptions, the multidimensional model is particularly rich, and it allows broad leeway in deciding the formulation of the recommendations. In fact, a query language [9], referred to as *Recommendation Query Language (RQL)*, has been developed for formulating different types of recommendation requests in a multidimensional recommender system. Such query languages are particularly useful for selecting different subsets of "what" and "for whom" dimensions in the querying process, and in developing a systematic query response methodology.

8.2.1 The Importance of Hierarchies

In the traditional OLAP model, hierarchies are often defined over various dimensions. For example, in a sales application, the various cells of the data cube correspond to sales values, and the location dimensions may have various hierarchical levels, such as city, state, region, and so on. One can aggregate the sales at the level of state, region, or the country. Furthermore, one can combine the location dimension with the time dimension by aggregating the sales in a particular region over a particular period of time. Such aggregation can also be performed in multidimensional recommender systems. Hierarchies are also useful in the context-sensitive recommender systems because they provide various levels of abstraction in which one might perform aggregated analysis.

In order to perform aggregated analysis, it is assumed that some or all of the dimensions have hierarchies associated with them. These hierarchies are part of the input to the recommender system. The nature of the hierarchy is highly domain-specific, and it depends on the application at hand. Some examples are as follows:

1. The location dimension can have a hierarchy corresponding to city, state, region, country, and so on.
2. If demographic information is associated with users, then one can also arrange the person dimension in a hierarchy of demographic attributes, such as age or occupation. A dimension such as age can be discretized into various hierarchical levels of granularity.
3. The item dimension can use a standard industry hierarchy, such as the North American Industry Classification System (NAICS). Alternatively, one can use a variety of genres or subgenres to represent the items in a number of product domains (with the movie domain being an example).
4. Dimensions such as time can be represented in various granular levels of hierarchy, such as hours, days, weeks, months, and so on.

Clearly, the user needs to make careful choices up front about the hierarchy to use, so that the most relevant analysis may be performed in a given application. It is also important to select the most relevant contextual dimensions $D_1 \ldots D_w$ for the application at hand. This problem is closely related to that of feature selection [18, 22] in the traditional classification and machine learning literature. Alternatively, these dimensions can be selected by domain experts.

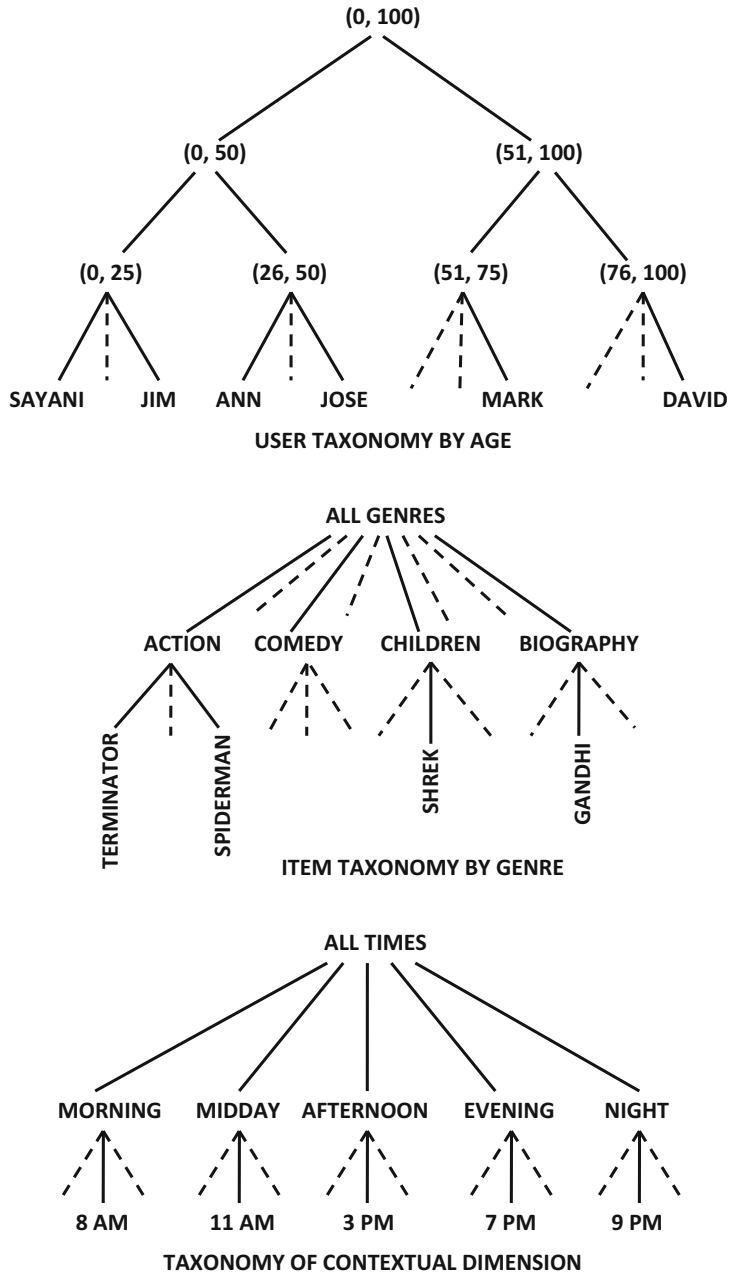

Figure 8.2: Taxonomies of users, items, and context

Examples of possible hierarchies for users, items (movies), and time are shown in Figure 8.2. The users are categorized by age, the movies are categorized by genre, and the time is categorized by the time-of-day. Now consider the case where these hierarchies are used in the example of Figure 8.1. With these hierarchies, one can now make more general (aggregated) queries, such as $g_R(David, Terminator, Evening)$, instead of

$g_R(David, Terminator, 7\ PM)$. The former provides an *average* prediction of how much David likes the movie *Terminator*, if he watches it at any time in the evening, whereas the latter provides a prediction of how much he would like it, if he saw the movie in the 7 PM show. At the extreme end, a query such as $g_R(David, Action, Any\ Time)$ corresponds to completely ignoring the time context and focusing on a specific genre of movies. This query estimates the average rating of David for action movies that are watched at any time. Therefore, the hierarchies are useful not only from the contextual perspective, but also from the perspective of hierarchical analysis on the user and item dimensions.

It is possible to combine the hierarchical analysis on the user and item dimensions. For example, one can aggregate further by querying how much users in the age range $[20, 30]$ like action movies rather than focusing on a specific user such as David. This is achieved with the function $g_R(Users \in Age[20, 30], Action, Any\ Time)$. Note that the hierarchy in Figure 8.2 groups the users by age. Such aggregated queries can be viewed as a sort of aggregation in multidimensional recommender systems. For example, one can view the aggregated rating $g_R(David, Action, Any\ Time)$ in terms of an aggregation function as follows:

$$g_R(David, Action, Any\ Time) = \text{AGGR}_{(x \in Action, \text{All}\ y)} g_R(David, x, y) \tag{8.2}$$

In traditional OLAP applications, one can obtain the relevant aggregation by summing the relevant values in the cells. This is also referred to as the "roll-up" operator in traditional OLAP systems. However, in recommender systems, it is more meaningful to talk of *averages* rather than additions. One might determine either the average rating of David for action movies or the average of his top-k ratings for action movies. The main challenge here is that the ratings are not completely observed in the original data cube, which is defined as a partial function. In most cases, the ratings are specified in a very sparse way at the bottom level of the hierarchy. In some cases, it is also possible for the observed ratings to be specified at the higher levels. For example, in some systems, David might be able to directly specify his interests for action or comedy movies, rather than providing ratings for individual movies. Multidimensional recommender systems are designed to address these scenarios as well. Therefore, a crucial step is to able to estimate the missing ratings at all levels of the hierarchy. These estimated ratings, together with the originally specified ratings, can be used to provide responses to various queries. Therefore, the *multi-level multidimensional rating estimation problem* is stated as follows:

Definition 8.2.2 (Multi - level Multidimensional Rating Estimation Problem) *Given an initial set of user-assigned ratings specified at different levels of the multidimensional cube of ratings, the task is to estimate all other ratings in the cube at all the levels of the OLAP hierarchies.*

Although full use of the hierarchical information is not always possible with various strategies, most techniques are able to predict ratings at the lowest level from other ratings. The techniques for performing contextual recommendation fall into one of three categories:

1. *Contextual pre-filtering:* In these methods, a segment of the ratings is pre-filtered corresponding to the relevant context. This relevant segment of ratings is then used to make targeted recommendations.

2. *Contextual post-filtering:* In these methods, the recommendations are first performed on the entire global set of ratings. Subsequently, the ranked recommendation lists are filtered or adjusted as a post-processing step with the use of temporal context.

3. *Contextual modeling:* In this case, the contextual information is incorporated directly into the prediction function, rather than as a pre-filtering or post-filtering step. This

is fundamentally different from the previous case, in which traditional 2-dimensional recommender systems are used under the covers. Contextual modeling is the most general approach in which one *directly* works with the w-dimensional representation of the ratings matrix in the modeling process. This approach provides the most integrated results, but it is sometimes computationally intensive or otherwise difficult to execute in a high-dimensional setting.

In the following sections, we will discuss these different classes of techniques for making recommendations. It is noteworthy that some of these techniques, such as post-filtering, use additional side-information about the various dimensions. These pieces of side information are referred[1] to as *attributes*. For example, a user might have demographic information associated with them, such as their name, address, age, gender, or profession. An item, such as a movie, might have side information associated with it, such as the title, actors, directors, and so on. Attributes are associated not just with the user and item dimensions, but also with the contextual dimensions. For example, consider a case where a user wishes to watch a movie with a specific companion. The companion dimension might contain a name, companion type (e.g., friend or parent), and age. As we will see later in this chapter, these types of side information are often important for some types of contextual recommendation applications. The set of attributes associated with a dimension is referred to as its *profile*. Note that item profiles and user profiles are used frequently for learning content-based recommendation models (cf. Chapter 4). Such attributes are also useful in many algorithms for the contextual setting.

8.3 Contextual Pre-filtering: A Reduction-Based Approach

Contextual pre-filtering is also referred to as *reduction* [6]. In the reduction-based approach, the idea is to reduce the w-dimensional estimation problem to a set of 2-dimensional estimations. The 2-dimensional estimation problem is equivalent to that in traditional collaborative filtering systems.

In order to understand this point, we will use an example of a 3-dimensional recommender system. Consider the case where the three attributes are users (U), movie items (I), and time (T). In such a case, the rating function g_R is defined as follows:

$$g_R : U \times I \times T \rightarrow rating$$

Note that the data set R is a 3-dimensional cube in this case. Consider a traditional 2-dimensional recommender system in which the mapping $f_{R'}$ is as follows:

$$f_{R'} : U \times I \rightarrow rating$$

In this case, the data cube R' is a 2-dimensional cube, in which only the two dimensions U and I are present. Clearly, ignoring the contextual dimension is equivalent to using a 2-dimensional recommender system. The 3-dimensional prediction function can be expressed in terms of the 2-dimensional prediction function by using a reduced derivative of the 3-dimensional ratings matrix. At any queried time t, this is achieved by deriving a

[1] In the traditional database context, the notions of dimension and attribute mean the same thing. In this case, however, they do not mean the same thing. A set of attributes is associated with a dimension.

2-dimensional ratings matrix $R'(t)$ from R with a pair of standard database operations:

$$R'(t) = \text{Project}_{U,I}(\text{Select}_{T=t}(R))$$
$$= \pi_{U,I}(\sigma_{T=t}(R))$$

Note that projection and selection are standard database operators. The matrix $R'(t)$ is obtained by first selecting the ratings in which the time is fixed to t, and then projecting down to the user and item dimensions. In other words, the 2-dimensional *slice* of the data cube in which the time is fixed to t corresponds to $R'(t)$. This is shown in Figure 8.3, where the entire user-item slice at 9 PM is shaded. Note that this 2-dimensional slice creates a user-item matrix, which can be used with traditional collaborative filtering algorithms. Such an approach can be used to perform the ratings prediction with the context fixed at 9 PM. In general, the 3-dimensional ratings estimation can be systematically reduced to 2-dimensional ratings estimation on this slice with the following relationship between the 3-dimensional function g_R and the traditional 2-dimensional collaborative filtering function $f_{R'(t)}$:

$$\forall (u, i, t) \in U \times I \times T, \quad g_R(u, i, t) = f_{R'(t)}(u, i)$$

This approach can easily be generalized to the case where there are $w > 3$ dimensions $D_1 \ldots D_w$ by fixing the remaining $w-2$ dimensions. The two dimensions, which are not fixed, are referred to as the *main* dimensions, whereas the other dimensions are the *contextual* dimensions. In typical applications, the users and items are the main dimensions. By fixing the values of the contextual dimensions, we can extract specific *slices* or *segments* of the data that are defined on only the two main dimensions. Traditional collaborative filtering algorithms can be used on such segments.

Because only a small subset of the ratings are used in a given slice, one may sometimes not have sufficient ratings to perform an accurate recommendation. In such cases, one may aggregate the rating at t with other adjacent time slices to create more accurate recommendations. For example, instead of using $t = 9$ PM, one might use all values of t in the evening, from 7 PM to 11 PM, and then average the ratings in these slices to create the resulting matrix. The 2-dimensional recommender is then applied to this averaged slice.

The main advantage of the approach is that it performs the collaborative filtering only over the *relevant* ratings in which the ratings have been selected with the use of the context. This can lead to improved accuracy in many cases, although the trade-off is that fewer ratings are now being used for prediction. Averaging over adjacent slices allows retention of a limited amount of local relevance, while reducing sparsity. The problem of sparsity can, nevertheless, be significant in many cases, and it may not always be possible to use such an averaging in order to increase the number of ratings. When fewer ratings are available, overfitting becomes more likely, and it is easy to envision scenarios in which the accuracy of the approach is not very high.

There are many natural solutions to reducing the sparsity problem at the expense of losing refinement. At one extreme, one might ignore the context altogether, and the ratings matrix is averaged over all the possible (combinations of) values of the contextual dimension(s). Such an approach will contain less relevant ratings than the *local* model that pre-selects the ratings based on context. This is clearly an extreme generalization of the approach discussed in the previous section where one averages only over adjacent values of the contextual variable (e.g., averaging the ratings in slices from 7 PM to 11 PM instead of using only the slice at 9 PM). We refer to this extreme approach as the *global* approach. Although the global approach uses less relevant ratings than the approach of using contextually localized slices, it will be able to use more ratings from the averaged slice. The comparative accuracy between the two alternatives depends on the nature of the

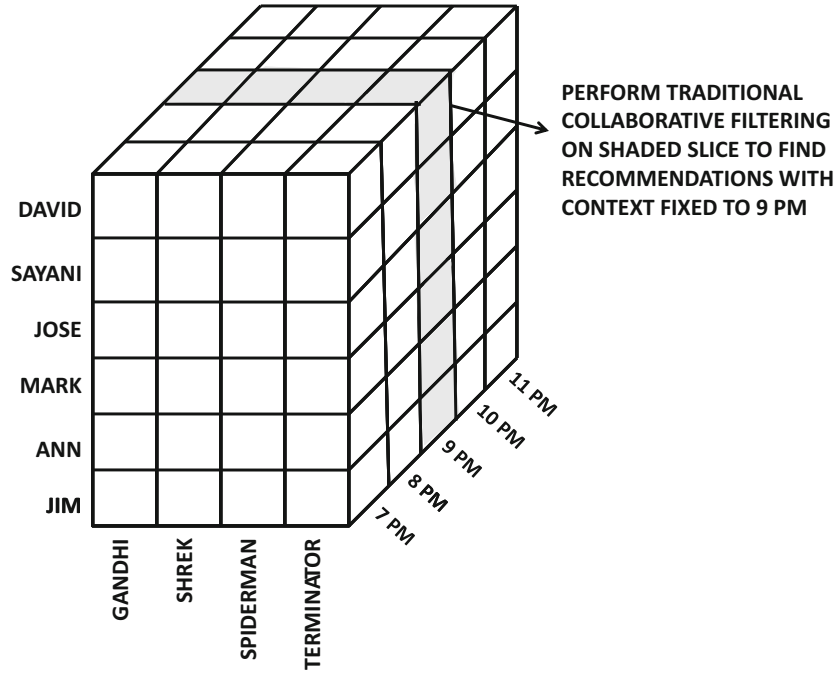

Figure 8.3: Extracting a 2-dimensional slice by fixing the context in reduction methods

trade-off between relevance and data sparsity. In many practical applications, it has been observed that either of the two alternatives might be better, depending on the part of the ratings matrix that one is looking at.

8.3.1 Ensemble-Based Improvements

Because of the unpredictability in the relative quality obtained with the global and local method, a question arises, as to how one might combine the two methods to obtain a technique that provides high accuracy in most scenarios. Although the local method provides more relevant results, it could also cause overfitting when there are too few relevant ratings for that context. We will discuss an ensemble-based method to improve the quality of the prediction. The goal of the ensemble-based method is to use the best of both worlds in the prediction process. In other words, either the local or the global matrix may be used, depending on the part of the ratings matrix that one is looking at. This approach results in the best trade-off between sparsity and local relevance. In this context, the *bucket-of-models* hybrid (cf. section 6.4.2 of Chapter 6) is very useful because it can help decide between alternative models of varying quality. The approach, however, needs to be tailored so that instead of selecting the best model, it selects the best data segment to train the model.

In the following discussion, a define a data segment of the ratings cube with the use of a combination of values of the contextual variables. For example, if the main variables are users and items, and the contextual variables are location and time, then each possible value of a location-time pair defines a data segment. When a recommendation problem is posed with a particular location-time context, it is important for the recommendation algorithm to determine whether or not using that context is indeed helpful.

In the training step, the approach first identifies the cross-validated accuracy on each data segment. For example, when the context is location and time, one determines the cross-validated accuracy of using all location-time pairs. In cases, where a hierarchical tree-like structure of the context is available, the higher-level nodes of the tree can be included as possibilities for the location-time pairs. A table is created for each location-time possibility, which contains its best generalization to use to obtain the highest accuracy. For example, if the contextual variables are location and time, examples of generalizations of (*9 PM*, *Boston*) might be (*night*, *Boston*), (*9 PM*, *Massachusetts*), (*night*, *Massachusetts*), (*9 PM*, $*$), ($*$, *Boston*), (*night*, $*$), ($*$, *Massachusetts*), and ($*$, $*$). For each contextual possibility such as (*9 PM*, *Boston*), the table will contain the correct level of generalization to use in order to obtain the best accuracy. This level is determined using a cross-validation procedure on the training data. Segments that contain too few ratings are ignored. In the testing step, the appropriate data segment is identified for the test instance using this table. Only the specific data segment is used, which provides the best quality results.

How is the cross-validation procedure performed? For example, for the context (*9 PM*, *Boston*), the relevant ratings in the training cube are identified, which correspond to the context of 9 PM and Boston. These ratings are segmented into folds using the cross-validation approach described in Chapter 7. The same folds are used to test the various alternative generalizations of (*9 PM*, *Boston*), which are enumerated above. The fold providing the best accuracy is retained. In practice, a more refined way of choosing the best segment is used, while keeping in mind that overfitting is more likely for more specialized segments. A local segment is selected over its generalization only if it *significantly* outperforms the latter according to standardized statistical tests.

One problem with this approach is that it can be very expensive when the number of contextual possibilities is large. For example, in the aforementioned example, one must test the accuracy of all generalizations of all possible local-time combinations. Clearly, this is possible only in the cases in which the number of possible contextual alternatives to the tested is small. Otherwise, the training phase can become too expensive. In some cases, simpler heuristics are used instead of explicitly learning the accuracy of various generalizations. Instead of using the *accuracy*, one might simply determine the number of training entries (i.e., ratings) for each possible generalization of a particular context. The lowest-level (i.e., most specific) generalization that contains the required minimum number of ratings is used. The basic idea is to ensure that overfitting is avoided because of limited training data.

8.3.2 Multi-level Estimation

So far, we have only discussed how to estimate the ratings at the lowest level of the hierarchy from other ratings at the lowest level. However, in some cases, a user might have expressed ratings at upper levels of the hierarchy. For example, a user might have specified ratings for genres of movies, as opposed to individual movies. A question arises as to how one might use these higher-level ratings to improve the estimation process. The basic idea here is to assign ratings at the lowest level, so that the computed average of the observed and predicted ratings at the lower levels (descendent nodes) is as close as possible to the observed rating at the higher level (ancestor node). For example, David might have specified ratings for action movies such as "*Terminator*" and also for the genre of action movies. How can one integrate the ratings at various levels to provide a holistic prediction?

Let the rating of David for action movies be r_a. In such cases, David's predicted ratings of action movies at the lowest level should be such that the average value of his observed and predicted ratings is as close as possible to his rating r_a of action movies. In the extreme case, we can impose exact equality. In other words, we impose the constraint that the sum of the predicted and observed ratings of David on action movies is equal to $n_a \cdot r_a$, where n_a is the total number of action movies. Note that this is a linear constraint on the variables of the collaborative filtering problem. There will be many such constraints for various users, and the ratings specified at various levels. Therefore, in addition to the standard techniques used for collaborative filtering, genre-specific constraints are introduced for predicting David's rating. This problem can be formulated as an optimization problem with added linear constraints. The work in [6] does not provide further details of how one might use this approach in an actual collaborative filtering algorithm, and leaves the solution of this problem to further research. This type of optimization modeling could provide a promising direction for future research; the main caveat is that sufficient ratings must be available to prevent overfitting.

8.4 Post-Filtering Methods

In pre-filtering methods, relevant segments (slices) of the data are extracted and the collaborative filtering algorithm is applied on these extracted slices. Therefore, the filtering is performed on the *input* data, *before* applying the collaborative filtering algorithm. The qualifier "pre-" in pre-filtering derives its name from this fact. In post-filtering, the filtering steps are applied to the *output* obtained *after* applying a global collaborative filtering algorithm that ignores the contextual information in the data set.

In post-filtering methods, the contextual information is ignored, and a global 2-dimensional ratings matrix is created by aggregating the ratings over all the possible contextual values. For example, the rating for each user-item combination may be derived by averaging the available ratings over all contextual alternatives of that combination. Subsequently the ratings are adjusted with the use of context. Therefore, the approach comprises two steps:

1. Recommendations are generated on all the data by applying a conventional collaborative filtering model on an *aggregated* user-item matrix. Thus, context is ignored in the first step.

2. Context is then used to adjust or filter the recommended list.

How is the multidimensional ratings cube aggregated into a 2-dimensional ratings matrix? In the case of explicit ratings, the aggregation process refers to the averaging of (observed) ratings, whereas in the case of implicit feedback matrices (e.g., units sold), the process of aggregation refers to the sum of the values. Note that the use of sums or averages will not yield the same result in general because of the varying number of observed values across different user-item combinations. In implicit feedback matrices, it is more appropriate to use the sum rather than averages because the number of nonzero values is highly indicative of the user interest in the item.

Consider a scenario where a user has provided a separate rating for the same item over three different contexts (e.g., morning, afternoon, evening). In this case, the ratings are averaged over these contexts in order to create a global 2-dimensional user-item ratings matrix. For implicit feedback matrices, the number of 1s over the different contexts need to

be added up. The resulting matrix no longer contains context-specific information because the contextual dimension has been aggregated. For the case of w-dimensional cubes, the rating values need to be aggregated over all the $(w-2)$-dimensional combinations. For example, if there are two contexts corresponding to location and time, then the user ratings of the same item over various location-time combinations need to be aggregated. If the user has never rated the item in any context, then the corresponding entry is also missing in the resulting aggregated matrix. The final result is always a 2-dimensional matrix, which is similar to that of traditional collaborative filtering.

Traditional collaborative filtering algorithms can be applied to this aggregated matrix in order to create predicted ratings $\hat{r}_{uj}$ and a corresponding ranked list of items for each user u. However, this ranked list is not sensitive to the contextual information, because the contextual dimension was ignored in the recommendation process. The post-filtering strategy adjusts the results *after* the estimations have been made. The adjustments can be made in one of two different ways. The first method corresponds to *filtering* out irrelevant items, and the second method corresponds to *adjusting* the ranking of the recommendations in the list based on the underlying context. The latter approach can be viewed as a soft version of the former. Both forms of post-filtering adjust the predicted rating $\hat{r}_{uj}$ for a given user-item combination.

One approach is to use heuristic methods to adjust or filter the recommended list based on the attributes associated with users and items. The notion of attributes associated with dimensions is discussed at the end of section 8.2. For example, if the context corresponds to $\{summer, winter\}$, then a clothing merchant might want to filter out sweaters and heavy jackets in the summer context, even if they are high in the list of recommended items. Such items can be detected with the use of attribute information.

For example, the attribute "wool" for a clothing item may be relevant to the context of the season attribute. One heuristic approach is to find the common item attributes that are relevant to a given context. Those items that do not have a significant number of these attributes are then filtered. In a more refined version of the approach, one can actually build a *predictive model* that uses the attributes to estimate the probability of relevance of the item to the context at hand. This approach is desirable because one can now use many of the traditional machine-learning techniques to build these predictive models. Those items that have very low probability of relevance are then filtered out. Such an approach is akin to determining a context-based probability $P(u, j, C)$ of the user u liking item j in context C, with the use of content-based models. The value of $P(u, j, C)$ need not be estimated with the use of content-based models. For example, one can even use a collaborative approach in conjunction with pre-filtering to estimate $P(u, j, C)$. This is identical to the pre-filtering approach discussed in the previous section. However, instead of using the pre-filtered prediction directly as the final result, it is normalized to the range $(0, 1)$, and then multiplied with the predicted rating $\hat{r}_{uj}$, which was estimated with the *global* data. The value of $P(u, j, C) \cdot \hat{r}_{uj}$ now defines an adjusted value of the prediction after post-filtering, and it may be used to adjust the ranking. Alternatively, one might simply remove the items j from the ranked list, for which the value of $P(u, j, C) \cdot \hat{r}_{uj}$ is very small. Post-filtering can be more robust than pre-filtering in a larger number of situations because one combines the local information $P(u, j, C)$ with the rating $\hat{r}_{uj}$ determined using all the data.

In cases where the amount of data available for context C is very limited, the value of $P(u, j, C)$ can be determined independent of the user u. In other words, the training data is used over all users to relate the item j to context C with a content-based model. For each item k, its attributes are treated as the feature variables, and the fraction of time that

the item k is consumed in context C is treated as a numeric dependent variable. A linear regression model is constructed to relate the attributes to context C. Then, for each item j, this linear regression model can be used to estimate $P(*, j, C)$. Note that we use a "*" (don't care) as the user argument, because this model is independent of the user at hand. The final predicted value of the rating of user i for item j after the post-filtering step is given by $P(*, j, C) \cdot \hat{r}_{uj}$.

8.5 Contextual Modeling

In both pre-filtering and post-filtering, the collaborative filtering problem is reduced to the 2-dimensional setting, and the context is used during pre-processing or post-processing. The main disadvantage of this approach is that context is not integrated very tightly into the recommendation algorithm. Such an approach prevents the full use of the relationships between various user-item combinations and contextual values. Contextual modeling methods have been designed to explore this possibility.

It is possible to incorporate context directly into the recommendation process by modifying existing models (such as neighborhood-based methods) to the w-dimensional setting. Such an approach provides the most flexible and generalized view of context-sensitive recommendations, which is independent of the shackles of 2-dimensional algorithms. The following sections will review some of these methods.

8.5.1 Neighborhood-Based Methods

It is possible to adapt ideas from existing neighborhood-based methods to perform context-sensitive recommendations. An example of such an approach was presented in [7, 8]. However, the approach can sometimes be subtly different from traditional user-user or item-item methods because of the use of contextual dimensions in the similarity computation process. For the purpose of discussion, let us consider the case where the specific context used is time. Therefore, we have three dimensions corresponding to users, items, and time. The first step is to compute the distances separately on the users, items, and time. Consider two points in the 3-dimensional cube, corresponding to $A = (u, i, t)$ and $B = (u', i', t')$ respectively. Then, the distance between A and B can be defined as the sum of the weighted distances between the individual dimensions. In other words, we have:

$$Dist(A, B) = w_1 \cdot Dist(u, u') + w_2 \cdot Dist(i, i') + w_3 \cdot Dist(t, t') \tag{8.3}$$

Here, w_1, w_2, and w_3, respectively, reflect the relative importance of the user, item, and context (time) dimensions. Note that one can add as many contextual dimensions as are of interest to the aforementioned summation, rather than only time. Alternatively, one can also use the weighted Euclidean metric:

$$Dist(A, B) = \sqrt{w_1 \cdot Dist(u, u')^2 + w_2 \cdot Dist(i, i')^2 + w_3 \cdot Dist(t, t')^2} \tag{8.4}$$

Then, for a given cell of the 3-dimensional matrix, the closest r (observed) ratings are determined by using this metric. The weighted average of these ratings is reported as the predicted rating. The weighting used is the similarity between A and B, which is also defined as $1/Dist(A, B)$. In order to perform the recommendation for a given user u and context t, one would need to apply this process to each item, and then report the top-k items as the recommendations.

A question arises as to how $Dist(u, u')$, $Dist(i, i')$, and $Dist(t, t')$ may be determined. There are several different ways of doing this:

1. *Collaborative:* In this case, one can use the Pearson method or the adjusted cosine to calculate $Dist(u, u')$, $Dist(i, i')$, and $Dist(t, t')$. For example, in order to compute the distance between users u and u', one can extract the 2-dimensional slices corresponding to the users u and u'. We can generalize the neighborhood-based similarity measure (cf. Chapter 2)) to compute the Pearson coefficients between all the ratings with the users fixed at u and u', respectively. Therefore, the individually observed ratings of users u and u' over the entire $Item \times Context$ grid are used in the Pearson computation. The inverse of the similarity is used to determine the distance values. A similar approach can be used to compute the item-wise and context-wise distances $Dist(i, i')$ and $Dist(t, t')$.

2. *Content-based:* In this case, the attributes associated with the dimensions (i.e., user profiles and item profiles) are used to compute the profile. A variety of text-based measures, such as the cosine, may be used. A similar approach may be used to compute $Dist(t, t')$ by associating each context with its frequently co-occurring content attributes. Alternatively, the attributes associated with a specific context, such as season, weekday, and so on, may be used. This approach may be viewed as a monolithic hybrid method because the representation is content-centric, but the overall approach uses the framework of collaborative methods.

3. *Combined:* It is possible to combine the collaborative and content-based measures to obtain a more robust measure of similarity. The relative weighting may be inferred with the use of cross-validation methods, so as to maximize the prediction accuracy.

Significant variations exist in terms of how the distance function may be designed for the specific application at hand. The aforementioned approach describes the broader idea although the specific implementation may vary with the application at hand. It is noteworthy that this approach can be viewed as a contextual generalization of the user-item approach discussed in section 2.3.6 of Chapter 2.

8.5.2 Latent Factor Models

Tensor factorization can be considered a generalization of matrix factorization, in which an n-dimensional data cube is factorized, rather than a 2-dimensional matrix. The traditional context-sensitive representation is indeed an w-dimensional cube, and therefore it is particularly well suited to tensor factorization. In this sense, tensor factorization methods can be considered contextual generalizations of conventional matrix factorization methods in recommender systems. While a detailed discussion of tensors is beyond the scope of this book, the reader is referred to [212, 294, 332, 495, 496] for details of such methods. A particularly notable example of a higher-order tensor factorization method is the *Multiverse Recommendation* model [294]. The Multiverse Recommendation model uses the higher-order Tucker decomposition [605], whose complexity increases *exponentially* with the order of the decomposition.

Such an application of tensor factorization is computationally very intensive, especially when the underlying data cubes are large. In most cases, the use of *higher-order* tensor factorization is an overkill in such settings [496]. There are, however, other simplified ways of applying the principles of latent factor models in multidimensional settings. Some simplified

forms of these factorization methods use only *pairwise* interactions between the different dimensions [496, 498].

Here, we describe one such pairwise interaction approach. A closely related ranking method, referred to as *Pairwise Interaction Tensor Factorization (PITF)* [496], has also been used for tag recommendation. This description can be viewed as a very special case of the notion of *factorization machines* discussed in [496]. Let $R = [r_{ijc}]$ be a 3-dimensional ratings cube of size $m \times n \times d$ with m users, n items, and d different values of the contextual dimension. For example, in Figure 8.1, we have $m = 6$, $n = 4$, and $d = 5$. Let $U = [u_{is}]$, $V = [v_{js}]$, and $W = [w_{cs}]$ be $m \times k$, $n \times k$, and $d \times k$, matrices. Here, U denotes the user-factor matrix, V denotes the item-factor matrix, and W denotes the context-factor matrix. The notation k denotes the rank of the latent factor model. Then, the basic principle of the simplified prediction function of the (i, j, k)th element of the data cube is based on pairwise interactions between users, items, and contexts. This implies the following prediction function:

$$\hat{r}_{ijc} = (UV^T)_{ij} + (VW^T)_{jc} + (UW^T)_{ic} \tag{8.5}$$

$$= \sum_{s=1}^{k} (u_{is}v_{js} + v_{js}w_{cs} + u_{is}w_{cs}) \tag{8.6}$$

It is easy to see that this prediction function is a straightforward generalization of latent factor models. One can now use this prediction function to set up the optimization problem as in all latent factor models. Let S be the set of all observed entries in R.

$$S = \{(i, j, c) : r_{ijc} \text{ is observed}\} \tag{8.7}$$

In cases where R is an implicit feedback matrix, a sample of unobserved entries also need to be included within S, under the assumption that these entries are observed as 0s. The detailed reasons for doing so are described in section 3.6.6.2 of Chapter 3.

Then, the errors over all the observed entries needs to be minimized as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j,c) \in S} (r_{ijc} - \hat{r}_{ijc})^2 + \frac{\lambda}{2} \sum_{s=1}^{k} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 + \sum_{c=1}^{d} w_{cs}^2 \right)$$

$$= \frac{1}{2} \sum_{(i,j,c) \in S} \left(r_{ijc} - \sum_{s=1}^{k} [u_{is}v_{js} + v_{js}w_{cs} + u_{is}w_{cs}] \right)^2 +$$

$$\frac{\lambda}{2} \sum_{s=1}^{k} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 + \sum_{c=1}^{d} w_{cs}^2 \right)$$

The last term is the regularization term, where $\lambda > 0$ is the regularization parameter.

We need to solve for the optimum values of parameters in U, V, and W. One can determine the partial derivative of J with respect to the individual elements in U, V,

and W, in order to derive the update directions for a gradient-descent method. Therefore, all elements of U, V, and W are updated *simultaneously* as follows:

$$u_{iq} \Leftarrow u_{iq} - \alpha \frac{\partial J}{\partial u_{iq}} \quad \forall i \quad \forall q \in \{1 \ldots k\}$$
$$v_{jq} \Leftarrow v_{jq} - \alpha \frac{\partial J}{\partial v_{jq}} \quad \forall j \quad \forall q \in \{1 \ldots k\}$$
$$w_{cq} \Leftarrow w_{cq} - \alpha \frac{\partial J}{\partial w_{cq}} \quad \forall c \quad \forall q \in \{1 \ldots k\}$$

Here, $\alpha > 0$ is the step size. As in traditional latent factor models, the descent direction depends on the error $e_{ijc} = r_{ijc} - \hat{r}_{ijc}$ over the observed entries in S. The corresponding updates are as follows:

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

A faster alternative is to use stochastic gradient descent. In stochastic-gradient descent, instead of descending over the gradient with respect to all the errors in S simultaneously, one descends with respect to the error in a single observed entry $(i, j, c) \in S$, which is randomly chosen:

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

On computing these contributions, the following steps may be executed for each specified entry $(i, j, c) \in S$ and the qth latent component $(1 \leq q \leq k)$:

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(e_{ijc} \cdot (v_{jq} + w_{cq}) - \frac{\lambda \cdot u_{iq}}{n_i^{user}} \right) \quad \forall q \in \{1 \ldots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha \left(e_{ijc} \cdot (u_{iq} + w_{cq}) - \frac{\lambda \cdot v_{jq}}{n_j^{item}} \right) \quad \forall q \in \{1 \ldots k\}$$
$$w_{cq} \Leftarrow w_{cq} + \alpha \left(e_{ijc} \cdot (u_{iq} + v_{jq}) - \frac{\lambda \cdot w_{cq}}{n_c^{context}} \right) \quad \forall q \in \{1 \ldots k\}$$

Here, n_i^{user}, n_j^{item}, and $n_c^{context}$ represent the number of observed entries in the data cube for user i, item j, and context c, respectively. Using these terms to normalize the regularization terms results in better convergence, although they can be (heuristically) omitted

and a smaller value of λ may be used instead. It is necessary to repeatedly cycle through the specified entries in S with each of the aforementioned updates. These gradient descent steps may be executed to convergence to obtain the matrices U, V, and W. The resulting updates are similar to the case of traditional matrix factorization discussed in section 3.6.4 of Chapter 3. Refer to Figure 3.9 for the algorithmic framework of stochastic gradient descent. The main change to that pseudocode is the use of an additional set of context factors, and the corresponding changes to the update equations. One now needs to cycle through each observed *triplet* (i, j, c) within the algorithmic framework of Figure 3.9 in order to execute the updates. Better convergence may be achieved by selecting different regularization parameters for each of the matrices U, V, and W. The values of these regularization parameters may be learned using cross-validation. One can also incorporate bias into the model by using a straightforward 3-dimensional generalization of the baseline predictors in section 3.7.1 of Chapter 3. The resulting baseline predictions B_{ijc} for user i, item j, and context c may be subtracted from the corresponding (observed) entries in the data cube before applying the factorization process. These values may be added back to the predictions in a post-processing phase.

This approach is less complex than higher-order tensor factorization models, and it can work particularly well in sparse matrices. It uses the 2-dimensional interactions in an additive way without going through the higher-order interactions, which can unnecessarily hamper the model both in terms of computational time and overfitting. In real settings, the ratings cube is generally too sparse to take full advantage of higher-order models. These issues are stated as the primary criticisms of *Multiverse Recommendation* methods [496].

This general principle can also be extended to w-dimensional cubes for $w > 3$. Consider an w-dimensional data cube R, in which a rating entry is denoted by $r_{i_1 \ldots i_w}$, with corresponding matrix dimensions $n_1 \ldots n_w$. Then, one can express the predicted rating value in terms of w different latent factor matrices U_{i_a} of respective sizes $n_a \times k$ $(a \in \{1 \ldots w\})$ as follows:

$$\hat{r}_{i_1 \ldots i_w} = \sum_{a<b \leq w} [U_a (U_b)^T]_{i_a i_b} \tag{8.8}$$

A least-squares optimization problem can be set up, as in the case of the 3-dimensional cube. A standard gradient descent approach can be used to solve this problem. The derivation of the update equations in this case is provided in Exercise 6.

8.5.2.1 Factorization Machines

The latent factor approach in the previous section can be viewed as a special case of *factorization machines*. Large classes of models (such as SVD and SVD++) are special cases of factorization machines. In factorization machines, the basic idea is to model each rating as a linear combination of interactions between input variables. The input variables are derived from the original ratings matrix. For example, consider the case in which we have a 3-dimensional cube containing m users, n items, and d values of the contextual dimension, and each rating is associated with a unique triplet. One can then "flatten" this 3-dimensional cube into a set of $(m + n + d)$-dimensional rows, such that each row corresponds to the user, item, and contextual value of an observed rating. Therefore, there are as many rows as the number of observed ratings. In this *specific example*, each row is a vector of binary indicator variables, in which exactly three of the values are 1s, depending on the specific user-item-context triplet relevant to that observed rating. All the remaining values are 0s. We represent the variables of the row by $x_1 \ldots x_{m+n+d}$, all of which are either 0s or 1s. Furthermore, the target variable for that row corresponds to the rating represented by

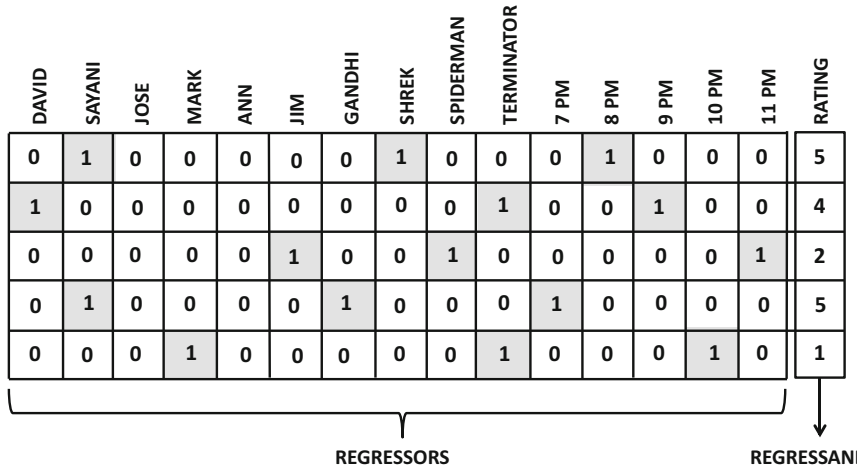

Figure 8.4: The flattened representation of five observed ratings in the data cube of Figure 8.1. Most recommendation problems can be transformed to sparse classification and regression problems.

that row. In Figure 8.4, we have shown the flattened representation of five observed ratings of the data cube in Figure 8.1. At first sight, it would seem that we could use a classification or regression predictor on this flattened representation in a straightforward way; however, it would not work very well because of the extraordinary data sparsity, in which there are only three nonzero entries in each row. It is here that factorization machines rescue us from the perils of sparsity.

The basic idea is to associate a k-dimensional latent factor with each of the $p = (m+n+d)$ decision variables $x_1 \ldots x_p$. Assume that the factor vector associated with the ith variable is denoted by $\overline{v_i} = (v_{i1} \ldots v_{ik})$. Similarly, the ith column has a bias b_i associated with it, and we also have a global bias variable g. The rating prediction $\hat{y}(\overline{x})$ of *second-order* factorization machines uses pairwise interactions between the factors as follows:

$$\hat{y}(\overline{x}) = g + \sum_{i=1}^{p} b_i x_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} (\overline{v_i} \cdot \overline{v_j}) x_i x_j \tag{8.9}$$

The variables to be learned are g, the different values of b_i, and each of the vectors $\overline{v_i}$. Although the number of interaction terms might seem alarmingly large, most of them will evaluate to zero in sparse settings. In the example shown in Figure 8.4, only three of the interaction terms will be nonzero, which seems suspiciously similar[2] to the three factorization terms in Equation 8.6. In fact, it can be easily shown that Equation 8.9 is a generalization of Equation 8.6 with added bias variables, and we can set up a similar least-squares model. As in matrix factorization, the stochastic-gradient descent method cycles through the observed ratings to estimate the aforementioned parameters. The update step with respect

[2]This similarity might not be obvious at first because the two equations do not use the same notation. Each k-dimensional factor vector $\overline{v_i}$ of the factorization machine is equivalent to one of the k-dimensional rows of either the user, item, or context factor matrix in Equation 8.6.

to any particular model parameter θ depends on the error $e(\overline{x}) = y(\overline{x}) - \hat{y}(\overline{x})$ between the predicted and observed values:

$$\theta \Leftarrow \theta(1 - \alpha \cdot \lambda) + \alpha \cdot e(\overline{x}) \frac{\partial \hat{y}(\overline{x})}{\partial \theta} \tag{8.10}$$

Here, $\alpha > 0$ is the learning rate, and $\lambda > 0$ is the regularization parameter. The partial derivative in the update equation is defined as follows:

$$\frac{\partial \hat{y}(\overline{x})}{\partial \theta} = \begin{cases} 1 & \text{if } \theta \text{ is } g \\ x_i & \text{if } \theta \text{ is } b_i \\ x_i \sum_{j=1}^{p} v_{js} \cdot x_j - v_{is} \cdot x_i^2 & \text{if } \theta \text{ is } v_{is} \end{cases} \tag{8.11}$$

The term $L_s = \sum_{j=1}^{p} v_{js} \cdot x_j$ in the third case is noteworthy. To avoid redundant effort, this term can be pre-stored while evaluating $\hat{y}(\overline{x})$ for computation of the error term $e(\overline{x}) = y(\overline{x}) - \hat{y}(\overline{x})$. This is because Equation 8.9 can be algebraically rearranged as follows:

$$\begin{aligned} \hat{y}(\overline{x}) &= g + \sum_{i=1}^{p} b_i x_i + \frac{1}{2} \sum_{s=1}^{k} \left([\sum_{j=1}^{p} v_{js} \cdot x_j]^2 - \sum_{j=1}^{p} v_{js}^2 \cdot x_j^2 \right) \\ &= g + \sum_{i=1}^{p} b_i x_i + \frac{1}{2} \sum_{s=1}^{k} \left(L_s^2 - \sum_{j=1}^{p} v_{js}^2 \cdot x_j^2 \right) \end{aligned}$$

Furthermore, the parameters $\overline{v_i}$ and b_i do not need to be updated when $x_i = 0$. This allows for an efficient update process in sparse settings, which is linear in both the number of nonzero entries and the value of k.

In this specific example, we have assumed that the vector $\overline{x}$ contains indicator variables with exactly three 1s. Factorization machines, however, allow arbitrary values of $\overline{x}$ to increase expressiveness. For example, it is possible for the values of $\overline{x}$ to be real, or to contain multiple nonzero values from the same dimension (e.g., context). This flexibility also allows the interaction between latent factors of pairs of users or pairs of contexts. There could be settings in which a context could correspond to a *set* of keywords or a set of entities. In the traditional data-cube model, there is no mechanism to represent such *set-wise* attributes. For example, consider the case where the context represents the companions with which a user watches a movie, and therefore each rating is associated with a set of companions (as context). In this case, the context variables x_i correspond to individual companions. If John watches a movie with Alice, Bob, and Jack, then the value of x_i for each of the three companions will be 1/3. This scenario is not quite so simple to represent with the straightforward latent factor approach, and it provides an example of the greater expressiveness of factorization machines. It is also relatively easy to see that this approach can be generalized to a setting in which each context is a document with associated word frequencies. For any given ratings matrix, all we have to do is to spend some time in a *feature engineering* effort. The observed ratings (target variables) are associated with a set of carefully designed attributes, some of which might be given (e.g., user, item, and context), and others might be extracted (e.g., implicit feedback). The versatility of factorization machines is striking. For example, by removing the contextual columns in Figure 8.4, one obtains traditional matrix factorization. By replacing the contextual columns in Figure 8.4 with implicit feedback variables, one roughly obtains SVD++ (with a few additional terms).

Factorization machines can be used for any (massively sparse) classification or regression task; ratings prediction in recommender systems is only one example of a natural application. Although the model is inherently designed for regression, binary classification can be handled by applying the logistic function on the numerical predictions to derive the probability whether $\hat{y}(\overline{x})$ is $+1$ or -1. Applications to classification and pairwise ranking are discussed in section 13.2.1 of Chapter 13. In fact, factorization machines can be shown to be sparsity-resistant generalizations of polynomial regression [493]. Note that Equation 8.9 is not very different from the prediction function of second-order polynomial regression. The most important difference is that the regression coefficients w_{ij} of pairwise interactions $x_i x_j$ are assumed to satisfy the low-rank assumption, and can therefore be expressed as $w_{ij} = \overline{v_i} \cdot \overline{v_j}$. For example, we could have tried to directly learn w_{ij} without making this low-rank assumption; this would be almost equivalent to using kernel regression with a second-order polynomial kernel. Since there are $O(p^2) = O([m + n + d]^2)$ such values of w_{ij}, overfitting is extremely likely. Factorization machines assume that the $p \times p$ regression coefficient matrix $W = [w_{ij}]$ is of low rank and can be expressed as VV^T. This reduces the $O(p^2)$ coefficients in $W = [w_{ij}]$ to the $O(p \cdot k)$ coefficients in $V = [v_{js}]$ and therefore helps in reducing overfitting. Under the covers, *factorization machines are polynomial regression models with an added low-rank assumption on the coefficients*. The basic idea here is that it would be difficult to accurately estimate the interaction coefficient w_{ij} between Jim and *Terminator* (with off-the-shelf polynomial regression) if Jim has never rated *Terminator*. However, the low-rank assumption enables accurate estimation of such regression coefficients by forcing inter-coefficient relationships in the parameter space. This is particularly useful in sparse settings.

The description in this section is based on second-order factorization machines that are popularly used in practice. In third-order polynomial regression, we would have $O(p^3)$ additional regression coefficients of the form w_{ijk}, which correspond to interaction terms of the form $x_i x_j x_k$. These coefficients would define a massive third-order *tensor*, which can be compressed with tensor factorization. Although higher-order factorization machines have also been developed, they are often impractical because of greater computational complexity and overfitting. A software library, referred to as *libFM* [494], provides an excellent set of factorization machine implementations. The main task in using *libFM* is an initial feature engineering effort, and the effectiveness of the model mainly depends on the skill of the analyst in extracting the correct set of features.

8.5.2.2 A Generalized View of Second-Order Factorization Machines

Although second-order factorization machines assume that all pairs of variables x_i and x_j interact with one another, this may not always be desirable. For example, when the contextual variables correspond to word frequencies of documents, it may not always be desirable for the word frequencies to interact with one another. In some cases, such as SVD++, implicit feedback variables might interact with item factors, but not with the user factors. Similarly, implicit feedback variables do not interact with one another in SVD++. In order to handle this setting, we define an interaction indicator δ_{ij}, which indicates pairs of variables that are allowed to interact with one another:

$$\delta_{ij} = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are allowed to interact} \\ 0 & \text{otherwise} \end{cases} \tag{8.12}$$

The interaction indicators are typically defined on the basis of the *block structure* of the variables, and therefore all p^2 values do not need to be explicitly stored. For example, user

variables might not be allowed to interact with context variables, contexts might not be allowed to interact with other contexts, and so on. This provides the analyst flexibility in specifying domain knowledge about important pairs of interacting blocks of variables. We can use this indicator to generalize Equation 8.9 as follows:

$$\hat{y}(\overline{x}) = g + \sum_{i=1}^{p} b_i x_i + \sum_{i=1}^{p} \sum_{j=i}^{p} (\overline{v_i} \cdot \overline{v_j}) \delta_{ij} x_i x_j \tag{8.13}$$

Unlike Equation 8.9, this equation allows x_i to interact with itself when δ_{ii} is nonzero. This can be useful in some versions of polynomial regression, when x_i is real. This model can be used to *exactly* simulate SVD++ by defining m user-indicator variables, n item-indicator variables, and an additional set of n implicit feedback variables associated with items. Therefore, there are two sets of item variables corresponding to explicit and implicit feedback, respectively. For implicit feedback variables, the feature values are nonzero only for items in the set I_u rated by the relevant user u. These nonzero values are all set to $1/\sqrt{I_u}$. The value of δ_{ij} is set to 1 only for interactions between users and explicit feedback (item) variables, and between implicit and explicit feedback (item) variables. With this definition, one can easily show that Equation 8.13 is *exactly* SVD++.

The solution methodology is almost the same as that of factorization machines. The update step of Equation 8.10 can be used for stochastic gradient-descent. The only difference is that the partial derivative of the predicted variable with respect to each model parameter θ needs to be modified:

$$\frac{\partial \hat{y}(\overline{x})}{\partial \theta} = \begin{cases} 1 & \text{if } \theta \text{ is } g \\ x_i & \text{if } \theta \text{ is } b_i \\ x_i \sum_{j=1}^{p} \delta_{ij} \cdot v_{js} \cdot x_j & \text{if } \theta \text{ is } v_{is} \end{cases} \tag{8.14}$$

A recent method, referred to as *SVDFeature* [151], can also be shown to be a special case of this setting by defining δ_{ij} appropriately. *SVDFeature* and factorization machines were the top two finishers in a network recommendation task at the KDD Cup contest (2012) [715].

8.5.2.3 Other Applications of Latent Parametrization

Factorization machines impose a low-rank structure on a large parameter space in order to reduce overfitting. A rarely noticed fact is that this general principle had been used earlier in the completely different context of *conditionally factored* Restricted Boltzmann Machines (RBMs) for collaborative filtering [519]. The basic idea is that the number of weights between two successive layers of a neural network can be represented as a matrix $W = [w_{ij}]$ (see Figure 8.5). The size of the matrix can be rather large in the collaborative filtering setting because the size of the input layer scales with the number of items and the size of the hidden layer can be of the order of several hundred units. The size of W is defined by the product of these two values. A large parameter space will inevitably lead to overfitting. Therefore, the work in [519] assumes that the matrix $W = UV^T$ is the product of two low-rank matrices U and V^T. Instead of learning W, the approach then learns the parameters of U and V. It has been shown in [519] that this type of low-rank reduction of the parameter space has significant advantages both in terms of accuracy and running time. These results show that a natural approach for effectively handling large and matrix-structured parameter spaces is by imposing a low-rank structure on them. Although the approach in [519] has been designed for conventional collaborative filtering, it can be

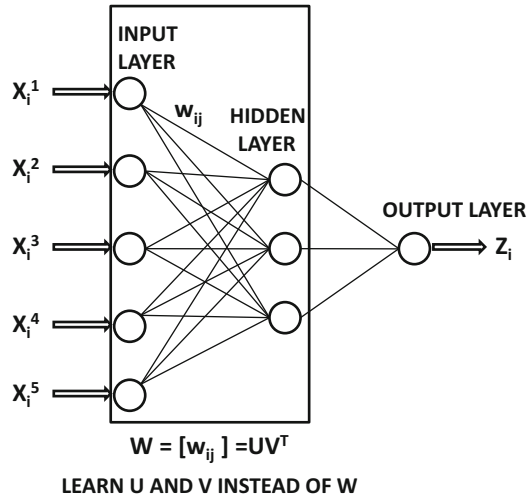

Figure 8.5: Low-rank parametrization of neural networks to avoid overfitting

easily extended to context-sensitive scenarios by adding appropriate input nodes for context features. An untapped potential of this approach is to leverage the low-rank parametrization in deep-learning methods for collaborative filtering. Deep neural networks with many layers can benefit [516] from the low-rank decomposition of the matrices corresponding to the weights between successive layers of neurons. This can be particularly useful because of the pervasive problem of overfitting in deep-learning methods.

8.5.3 Content-Based Models

A variety of machine learning models, such as support vector machines and linear regression, are used in conjunction with context-sensitive recommender systems. These methods can be viewed as generalizations of content-based models because they use the attributes associated with users, items, and context. Recall that content-based models use only the attributes associated with the items. However, in this case, we make the more general assumption that attributes are associated with any of the dimensions.

In most of these cases, the users or items are represented as vectors in feature space, and the ratings correspond to the dependent or class variable. One of the earliest methods, which used support vector machines, was proposed in [458]. In this case, a restaurant recommender system is proposed, in which additional contextual dimensions, such as weather, companion, and time, are used. In this case, each item-context combination is represented as a feature vector that is either liked or disliked. A support vector machine is constructed that separates the liked items from the disliked items. A previously unseen item-context combination will be recommended as liked if it falls on the liked side of the support-vector separation. This model can be viewed as a direct generalization of content-centric models because a separate model is built for each user, and the prediction for the model is specific to each user. Furthermore, the attributes of the user are not used in this model. However, it is possible, in principle, to use a global model which is constructed for prediction over all users.

An example of such a model is discussed in [50] where a single linear regression model is constructed to predict the ratings of any user-item combination. In this case, the attributes of the user are also used. For ease in discussion, we assume that the features are treated

as the frequencies of discrete keywords, although one can also use numeric attributes after discretization. Note that the features of users and items are contained in the user or item profile.

First, we describe a simple linear regression model that does not use any contextual information. Later, we will show how to extend this model with the use of context. Consider the following linear regression model, which estimates the rating r_{ij} as a linear function of the user features, item features, and Kronecker cross-product features:

$$\hat{r}_{ij} = \overline{W_1} \cdot \overline{y_i} + \overline{W_2} \cdot \overline{z_j} + \overline{W_3} \cdot (\overline{y_i} \otimes \overline{z_j}) \tag{8.15}$$

Here, $\overline{W_1}$, $\overline{W_2}$, and $\overline{W_3}$ are linear regression coefficient *vectors* of the appropriate length. For example, the length of the coefficient vector $\overline{W_1}$ is the same as that of the feature space representing all the different $\overline{y_i}$. Furthermore, $\overline{y_i}$ corresponds to the feature variable vector of user i (e,g., gender or race), $\overline{z_j}$ corresponds to the feature variable vector of item j (e.g., movie genre and production studio), and $(\overline{y_i} \otimes \overline{z_j})$ corresponds to the entries in the Kronecker product between the feature vectors of user i and item j. The Kronecker product is defined by all possible cross-product combinations between the feature values of user i and item j. In the aforementioned example, combinations correspond to various possibilities for gender-genre, race-genre, gender-studio, and race-studio. For a particular user-item instance, the relevant combinations might be male-comedy, Caucasian-comedy, male-Sony, and Caucasian-Sony. The values of these combinations are set to 1, and those of all other possible combinations (e.g., female-comedy) are set to 0. In this case, all feature values are binary, although it is also possible to work with arbitrary frequencies by multiplying the frequencies of the corresponding value-pairs. For example, if the keyword "golf" has a frequency of 2 in the user profile and the keyword "cart" has a frequency of 3 in the item profile, the corresponding frequency of the keyword pair is $2 \times 3 = 6$. The basic idea here is to represent a rating of a user-item combination (i, j) in terms of the features of user i, the features of item j, and the interaction effects among them.

The observed ratings are used as the training data in order to create the model and learn the coefficient vectors $\overline{W_1}$, $\overline{W_2}$ and $\overline{W_3}$. The interaction coefficients tell us how various combinations of user-item features affect the model. The coefficients of the user features and the item features tell us about the specific biases of the user and the item at hand. The work in [50] uses a Markov Chain Monte Carlo (MCMC) method in order to estimate these coefficients and learn the model from the observed ratings. This approach is a generalization of linear regression models for content-based methods, which create *user-specific* models with *item* features (cf. section 4.4.5 of Chapter 4). Here, the model is constructed over all users and items, and the features are also extracted from the user-item combination, which is being rated. Therefore, this approach is richer than an off-the-shelf content-based model.

The approach can be generalized easily to the contextual scenario by introducing additional feature variables for the contextual dimensions [7, 607]. As a specific example, consider the case where time is used as a contextual variable, and the feature variables associated with the kth possible value of the time dimension are denoted by the vector $\overline{v_k}$. The features associated with time might correspond to various descriptions, such as the time of the day, whether it is a weekday, the season, and so on. Since we have three dimensions, the ratings r_{ijk} are now subscripted by three different values. Here i denotes the index of the user, j denotes the index of the item, and k denotes the index of the time dimension. Then, the predicted value $\hat{r}_{ijk}$ of the rating can be computed as a linear function of the feature variables and interaction variables as follows:

$$\hat{r}_{ijk} = \overline{W_1} \cdot \overline{y_i} + \overline{W_2} \cdot \overline{z_j} + \overline{W_3} \cdot \overline{v_k} + \overline{W_4} \cdot (\overline{y_i} \otimes \overline{z_j}) + \overline{W_5} \cdot (\overline{z_j} \otimes \overline{v_k}) + \overline{W_6} \cdot (\overline{y_i} \otimes \overline{v_k}) + \overline{W_7} \cdot (\overline{y_i} \otimes \overline{z_j} \otimes \overline{v_k}) \tag{8.16}$$

To reduce the number of coefficients, it is possible to set the third-order coefficients $\overline{W_7}$ to 0. Such a model is in a similar form as second-order factorization machines, although interactions only occur between attributes from different dimensions (e.g., users and items). This is similar to the model of Equation 8.13. A similar gradient-descent approach can be used.

This generic approach can, in fact, be used in conjunction with *any* off-the-shelf machine learning model, and not just regression-based models. The overall approach is as follows:

1. Generate a multi-dimensional data record $\overline{X}_{ijk}$ for each observed rating r_{ijk}, where the class label of the record is the value r_{ijk}.
2. Generate the frequencies of the discrete features corresponding to $\overline{y_i}$, $\overline{z_j}$, $\overline{v_k}$, $\overline{y_i} \otimes \overline{z_j}$, $\overline{z_j} \otimes \overline{v_k}$, and $\overline{y_i} \otimes \overline{v_k}$. Let these frequencies represent the feature vector for $\overline{X}_{ijk}$.
3. Use the pairs $(\overline{X}_{ijk}, r_{ijk})$ in conjunction with any off-the-shelf supervised learning algorithm to build a model $\mathcal{M}$.
4. For any entry (i_1, j_1, k_1) in the ratings cube for which the value is unknown, extract the feature representation $\overline{X}_{i_1 j_1 k_1}$ using the aforementioned approach, and apply the machine learning model $\mathcal{M}$ to predict the value of the rating.

As the dimensionality of the context-sensitive system increases, the model becomes more likely to overfit. Furthermore, the scalability of the system is adversely affected. This can sometimes be a significant challenge in such systems and is, of course, a general drawback of contextual modeling methods, which directly try to work with the w-dimensional ratings matrix rather than reducing it to a 2-dimensional problem with pre- or post-filtering methods. Nevertheless, if sufficient ratings data are available, then direct contextual modeling is likely to provide the most robust results. Such methods are likely to become increasingly relevant in the "big-data" era.

8.6 Summary

Various types of context, such as location, time, and social information, have a significant influence on the recommendation process. The multidimensional model is used frequently to create a general framework for various types of context-aware recommendations. There are three primary ways in which context-aware recommendations are performed. In pre-filtering, the problem is reduced to a 2-dimensional collaborative filtering problem by filtering the w-dimensional cube to a 2-dimensional ratings matrix *before* applying the collaborative filtering algorithm. In post-filtering, the context is ignored during the first phase of collaborative filtering. Subsequently, the results are adjusted with the use of a predictive model that regulates the relative importance of context. Finally, a recent approach is to incorporate the context directly into the model by treating it as a w-dimensional prediction problem. Generalizations of matrix factorization and linear regression models have been proposed in this setting. This approach is computationally intensive, but it is a generic approach with the best potential when a large amount of data is available.

8.7 Bibliographic Notes

Some of the earliest work on context-aware recommendation systems was performed in the context of mobile applications [2, 3], such as creating a mobile context-aware tour guide. An early survey of context-aware computing research for mobile systems may be found in [147]. A recent survey on context-based recommender systems may be found in [7]. Context-sensitive systems have been used in a wide variety of domains, such as news recommendation [134], Web search [336], tourist recommendations [2, 3], and database querying [39]. A survey of context-aware recommender systems, which use technology-enhanced learning technologies, may be found in [612].

The notion of multidimensional recommender systems was proposed in [6]. An interesting discussion is also found in [466]. A query language, referred to as *Recommendation Query Language (RQL)* [9], was proposed for context-based systems. Another recent query language in the context of personalizing recommendations is *REQUEST* [10].

The use of pre-filtering methods has a rich history in recommender systems. The reduction-based approach of [6] is one of the seminal techniques for pre-filtering. Many subsequent methods based on this broad methodology have been developed. The work in [62] uses the notion of item splitting, in which a single item is split into several fictitious items corresponding to various contexts. The work in [61] develops the notion of micro-profiles, each of which is relevant to a specific context. Specifically, a different model is constructed for the user in different contexts. This approach was used for time-sensitive recommender systems, and it is discussed in section 9.2.2.1 of Chapter 9. The basic idea in [61] is similar to the reduction-based approach described in this chapter. A mobile advertisement recommender system, which uses pre-filtering, is discussed in [40]. The application of the approach in an online retailing application is discussed in [374]. A comparison of pre- and post-filtering methods in context-sensitive systems is provided in [471]. Results on the accuracy and diversity of context-sensitive systems are provided in [470]. The use of neighborhood-based methods for context-sensitive recommendations is discussed in [7, 8]. Many matrix and tensor factorization methods have been proposed for contextual recommendations in which time is treated as a discrete contextual value [212, 294, 332, 495, 496]. The notion of *factorization machines* [493] has found significant popularity in these settings. Factorization machines can be viewed as generalizations of large classes of latent factor models, and they have found increasing popularity for context-sensitive recommendation applications. Alternating least-squares methods for factorization machines are discussed in [496]. A related model, referred to as *SVDFeature*, is proposed in [151].

The support vector machine method for model construction was proposed in [458]. The work in [63] proposes a family of matrix-factorization methods for context-aware recommendation, although the approach discussed in this book is more general than the methods discussed in this family. Scalable algorithms for building context-sensitive recommender systems are discussed in [607].

A major issue is the selection of appropriate attributes for contextual methods. A discussion of how appropriate attributes may be selected for contextual methods is provided in [188]. The use of latent contextual information as a possible representation is discussed in [47].

8.8 Exercises

1. Discuss how you might decide whether pre-filtering, post-filtering, or contextual modeling is most appropriate for a particular data set.

2. Discuss how you might use hybrid recommender systems to combine the power of pre-filtering, post-filtering, and contextual modeling. Propose as many schemes as you can imagine. How would you decide which of these to use?

3. Implement the pre-filtering algorithm with a single contextual attribute. Use an item-based (neighborhood) collaborative filtering as the base method.

4. Suppose that you have three contextual attributes (say, location, time, and companion), each of which has its own taxonomy. Your system is designed to recommend items for a given user in the context of location, time, and companion. For a given context at the lowest level of the hierarchy, you might have the sparsity problem because only a modest number (say, 500) of the observed ratings might be available in which the three contexts are fixed to the queried values. This can cause overfitting in a pre-filtering method if only 500 ratings are used for the training process. You decide that you will use a more general level of the taxonomy from each of the three contexts, in order to extract the relevant segments and increase the amount of training data. Describe how to determine *the specific level of the* taxonomy to use for each contextual attribute. Once you have extracted the taxonomy level for each context, describe the collaborative filtering algorithm.

5. Consider the w-dimensional matrix factorization discussed in section 8.5.2, in which the prediction function is of the following form:

$$\hat{r}_{i_1 \ldots i_w} = \sum_{a<b\leq w} [U_a (U_b)^T]_{i_a i_b}$$

   (a) Let S be the set of all the w-dimensional coordinates of the specified entries in the w-dimensional data cube. Show that the objective function for optimization, with regularization, is of the following form:

$$J = \sum_{(i_1 \ldots i_w) \in S} (r_{i_1 \ldots i_w} - \sum_{a<b\leq w} [U_a (U_b)^T]_{i_a i_b})^2 + \lambda \sum_{a=1}^{w} ||U_a||^2$$

   (b) How would you use this objective function to derive the gradient descent method?

   (c) Let $e_{i_1 \ldots i_w} = r_{i_1 \ldots i_w} - \hat{r}_{i_1 \ldots i_w}$ represent the prediction error of an entry $(i_1 \ldots i_w)$ at an intermediate stage of the gradient descent updates. Show that the gradient-descent update equations to each U_a $(1 \leq a \leq w)$ are of the following form for each specified entry $(i_1 \ldots i_w) \in S$:

$$[U_a]_{i_a q} \Leftarrow [U_a]_{i_a q} + \alpha(e_{i_1 \ldots i_w} \cdot \sum_{b \neq a} [U_b]_{i_b q}) - \lambda \cdot [U_a]_{i_a q}) \quad \forall q \in \{1 \ldots k\}$$