Chapter 9

Time- and Location-Sensitive Recommender Systems

"*Time is the wisest counselor of all.*"–Pericles

9.1 Introduction

In many real scenarios, the buying and rating behaviors of customers are associated with temporal information. For example, the ratings in the Netflix Prize data set are associated with a "*GradeDate*" variable, and it was eventually shown [310] how the temporal component could be used to improve the rating predictions. Similarly, many forms of user activity, such as buying behavior and Web clickstreams, are inherently temporal. In general, recommender systems use the temporal aspects of user activity in two different ways:

1. *Explicit ratings:* In this case, dates are associated with explicit ratings. These dates can be used to improve the accuracy of the prediction process either through the use of forecasting methods, or through periodic and seasonal information (e.g., day of week).

2. *Implicit feedback:* These scenarios correspond to customer *actions*, such as buying items or clicking on Web pages. The historical sequence of user actions is used to make predictions about future actions. The underlying methods often share many similarities with forecasting based on sequential patterns. Such techniques are used frequently in many scenarios such as Web clickstreams or Web log analysis. The techniques can also be used to make predictions about future customer buying behavior.

Generally, it is much harder to use the temporal information in ratings to make recommendations. As we will see later in this chapter, existing temporal models [310] use the temporal

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_9

information in ratings in a limited and carefully calibrated way. On the other hand, the literature on implicit feedback and discrete models is quite rich because it has been explored extensively in the context of Web clickstreams and logs. The latter problem is closely related to that of forecasting of sequential data with categorical attributes. In this case, discrete data mining methods, such as Markovian models and sequential pattern-mining, are very useful. In this chapter, we will study both types of recommenders.

Time can be viewed from a recency and forecasting perspective, or from a contextual (e.g., seasonal) perspective. From a recency perspective, the basic idea is that recent ratings are more important than older ratings. Therefore, various aging and filtering strategies are used to assign greater importance to more recent data. In the contextual perspective, various periodic aspects, such as season or month, may be used.

The latter scenario is closely related to context-aware recommender systems. In context-aware recommender systems [7], an additional variable, such as location or time, is used to refine the recommendation. In standard collaborative filtering with user set U and item set I, the user-item possibilities in $U \times I$ are mapped to ratings. This mapping is learned from the available data. However, the presence of a context C requires us to learn the mapping from the possibilities in $U \times I \times C$ to the ratings. Note that the context C may itself contain multiple attributes such as location, time, weather, season, and so on. These properties could either be dependent or independent of one another. In this particular chapter, we will study the specific case in which the contextual property is a single attribute corresponding to time. When time is viewed as a continuous variable, the recommendations are often created as functions of time. The temporal context can be viewed from a periodic, recency, or modeling point of view. In *periodic* contexts, a specific periodic aspect of the time, such as weekday, time of day, or month is used in order to sharpen the recommendation. For example, it makes more sense for a North American clothing retailer to recommend winter clothing in December rather than in July. Context-aware recommenders are discussed in a generic sense in Chapter 8. However, we have allocated a separate chapter to the time dimension because of the large amount of literature associated with it. Furthermore, many of the temporal methods, such as forecasting-based ratings prediction and discrete sequence-based methods, cannot be easily generalized to other context-sensitive methods and scenarios. Therefore, the temporal aspect of recommender systems needs to be treated separately from context-aware systems, although the connections to context-based methods are highlighted throughout the relevant parts of the chapter.

Time can be treated as a modeling variable by explicitly expressing the predicted ratings as a function of time. The parameters of this function can be learned in a data-driven manner by minimizing the squared error of the predicted ratings with respect to the observed ratings. An example of such a model is *time-SVD++*, which expresses the predicted ratings as a function of temporally parameterized biases and factor matrices. This approach is considered one of the state-of-the-art techniques for temporal prediction. The main advantage of this approach is that it can capture *future* trends, which are not easily captured by recency, decay-based, or periodic models.

Many data domains such as Web clickstreams do not contain explicit ratings, but they contain discrete action sequences. Such data can be viewed as the temporal version of implicit feedback data sets. The methods used in such domains are often quite distinct from those used in the case of ratings. In particular, Markovian models and sequential pattern-mining methods are commonly used. Such methods have been studied extensively in the Web-mining domain because Web logs are widely available for mining purposes. In this chapter, we will also review discrete-sequence mining methods for recommendations in applications such as Web clickstreams.

Like time, location is another commonly used context in recommender systems. With the increasing popularity of GPS-enabled mobile phones, the use of the location context is useful in a variety of scenarios, such as finding movie theaters, restaurants, or other entertainment locations. In some cases, the location context can be combined with time. This chapter will use the location-based scenario as an important example of context-based systems.

This chapter is organized as follows. In section 9.2, we will introduce methodologies for temporal collaborative filtering with ordered ratings. In particular, we will introduce three different types of models. These correspond to recency-based models, periodic models, and more complex parameterized models. An example of the latter is the *time-SVD++* model, which is considered the state-of-the-art for temporal recommendation. The connections of various models with the context-based models of Chapter 8 are also discussed. Section 9.3 discusses how discrete models can be extended to the temporal scenario in cases where the user actions represent discrete selections such as clicks. Markovian models and sequential pattern-mining methods are discussed in this section. Location-aware recommender systems are discussed in section 9.4. A summary is given in section 9.5.

9.2 Temporal Collaborative Filtering

In this section, we will study the use of temporal recommendations with ratings. Temporal information can be used in one of two ways in order to improve the effectiveness of prediction:

1. *Recency-based models:* Some models consider recent ratings more important than older ratings. In these cases, window-based and decay-based models are used for more accurate prediction. The basic idea in all these models is that the recent ratings are given more importance within the collaborative filtering model.

2. *Periodic context-based models:* In periodic context-based models, the specific property of a period, such as the time at the level of specificity of the hour, day, week, month, or season, is used to perform the recommendation. For example, a clothing retailer would make very different recommendations depending upon whether it was summer or winter [567]. Similarly, the movie recommendations during the Christmas week might be very different from those in the week leading to the Oscars [100]. In these methods, time becomes a contextual variable that is exploited in order to make recommendations. These models are closely related to contextual recommender systems, as introduced in Chapter 8.

3. *Models that explicitly use time as an independent variable:* A recent approach, referred to as *time-SVD++*, uses time as an independent variable within the modeling process. Such an approach uses more refined user-specific and item-specific trends to handle local temporal variations and it can also account for intermittent temporal noise in the ratings. Generally, such models are more sophisticated than recency-based models because they include an element of forecasting.

Window-based and decay-based models have the merit that they are simple and easy to implement in a wide variety of settings. On the other hand, they cannot capture the refined temporal characteristics captured by the *time-SVD++* model. Therefore, the latter method is considered the state-of-the-art in temporal collaborative filtering. Nevertheless, recency-based models have the advantage of being easy to implement. Furthermore, a wider variety of models can be generalized to these cases. On the other hand, only a small number of models have been proposed in the second category.

9.2.1 Recency-Based Models

In recency-based models, recent ratings are given greater importance than older ones. The greater importance of recency can be addressed with either decay-based methods or with window-based methods. In decay-based methods, older ratings are given less importance with the use of a decay function. Window-based methods can be viewed as special cases of decay-based methods, in which binary decay functions are used to completely disregard data points that are older than a specific amount of time. In other words, the binary decay function ensures that older ratings are given a weight of 0, whereas recent ratings are given a weight of 1.

9.2.1.1 Decay-Based Methods

In decay-based methods, a time-stamp t_{uj} is associated with each observed rating of user u and item j in the $m \times n$ ratings matrix R. Therefore, the number of observed values of t_{uj} is exactly equal to the number of observed ratings in R. It is assumed that all recommendations should be made at a future time t_f. This future time is also referred to as the *target time*. Then, the weight $w_{uj}(t_f)$ of the rating r_{uj} at target time t_f is defined with the use of a decay function, that penalizes larger distances between t_{uj} and t_f. A decay function, which is commonly used [185], is the exponential function:

$$w_{uj}(t_f) = \exp[-\lambda(t_f - t_{uj})] \tag{9.1}$$

The decay-rate λ is a user-defined parameter that regulates the importance of time. Larger values of λ de-emphasize older ratings to a greater degree. These weights can be used in neighborhood-based methods to regulate the importance of a rating during the prediction phase.

A method proposed in [185] modifies user-based neighborhood methods by changing the final prediction function. The simple approach used in [185] first determines the k-nearest neighbors of each user. The determination of the nearest neighbors (users) is identical to off-the-shelf user-based neighborhood methods. Subsequently, the only difference from other neighborhood-based methods is that the ratings of other users are weighted with $w_{uj}(t_f)$ during the aggregation process. Specifically, Equation 2.4 of Chapter 2 can now be modified as follows for predicting the rating of item j for user u at time t_f as follows:

$$\hat{r}_{uj}(t_f) = \mu_u + \frac{\sum_{v \in P_u(j)} w_{vj}(t_f) \cdot \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} w_{vj}(t_f) \cdot |\text{Sim}(u,v)|} \tag{9.2}$$

Here, $P_u(j)$ represents the k closest users to user u that have specified ratings for item j. Note that the main difference of the aforementioned equation from traditional collaborative filtering is the presence of weights in the prediction function. These weights bias the solutions in favor of more recent trends by discounting stale ratings.

The approach can be easily applied to both user-based and item-based models with a small modification in the final step. In both cases, the final prediction step needs to be augmented with recency-based weights. The optimal value of λ can be learned using cross-validation methods, although such an approach is not discussed in [185].

The work in [186] provides a slightly more refined model in which an item-based neighborhood method is used for collaborative filtering. Aside from weighting each item with item-to-item similarity in the prediction process, a temporal discount factor is also multiplied to the rating of each item within the prediction function. This is, of course, similar to the method used in [185] (and also discussed above). Unlike the work in [185], however, this discount factor is not a simple exponential decay function. Each item is assigned a discount factor by estimating its expected future error and then assigning a weight that is inversely proportional to this error.

Consider a scenario in which the set of peer items of the target item j for which the user u has specified ratings is $Q_j(u)$. The process of determining $Q_j(u)$ is identical to that of item-based neighborhood methods. Then, the discount factor D_{ui} for each item $i \in Q_j(u)$ needs to be determined in order to modify the final prediction function. Note that the discount factor in local to the user u at hand and therefore contains u in the subscript. The prediction of the rating of item j by user u is computed[1] with a discounted version of the item-based prediction function:

$$\hat{r}_{uj} = \frac{\sum_{i \in Q_j(u)} \text{Sim}(i,j) \cdot D_{ui} \cdot r_{ui}}{\sum_{i \in Q_j(u)} |\text{Sim}(i,j)| \cdot D_{ui}} \tag{9.3}$$

How is each discount factor D_{ui} computed? This is achieved by computing the normalized difference in ratings between the user rating r_{ui} on each item $i \in Q_j(u)$, and the average rating O_{ui} of user u on similar items to item i. The similar items to item i are identified using item-to-item similarity computation. The discount factor (weight) $D_{ui} \in (0,1)$ for each user u and item $i \in Q_j(u)$ is computed as a function of these two quantities.

$$D_{ui} = \left(1 - \frac{|O_{ui} - r_{ui}|}{r_{max} - r_{min}}\right)^{\alpha} \tag{9.4}$$

Here, r_{max} and r_{min} are the maximum and minimum values on the scale of ratings. α is a tuning parameter, which can be chosen using cross-validation. The basic idea here is that the difference in the user's rating between that of item i and the average rating of the same user on similar items is a manifestation of the error caused by temporal evolution. Furthermore, different users may have different rates of evolution; therefore, the discount factor is local to the specific user at hand.

The methods in [185, 186] do not incorporate the decay weights and discount factors in the similarity computation, and these weights are used only in the prediction phase. It is, however, also possible to compute the similarity in a weighted way, as discussed in section 6.5.2.1 of Chapter 6. In fact, any of these weighted models may be used, once $w_{ij}(t)$ has been defined. While these weighted models were developed in [67] in the context of ensemble methods such as bagging and boosting, they can easily be adapted to the temporal scenario. Note that the way in which matrix factorization models can be generalized to the weighted case is also shown in section 6.5.2.1. Given this, matrix factorization methods can also be easily generalized to recency-based techniques.

[1]The original work [186] does not use a modulus in the denominator. We have added it in Equation 9.3 because omitting it does not make much sense in the case of negative similarity. Nevertheless, negative similarities in the peer item-group are rare in practical settings because the peers are defined as the most similar items.

9.2.1.2 Window-Based Methods

In window-based methods, ratings that are older than a particular time are pruned from consideration. This approach can be viewed as a special case of *pre-filtering* or *post-filtering* methods in context-based models. Such methods are discussed in a generic sense in Chapter 8. Furthermore, these methods can also be viewed as (discrete) special cases of decay-based methods. There are several ways in which windows can be modeled:

1. If the difference between the target time t_f and the rating time t_{ij} is larger than a particular threshold, then the rating is dropped. The collaborative filtering model is otherwise identical to any of the methods discussed in Chapters 2 and 3. This approach can be viewed as an extreme case of the decay-based model, in which the decay function is binary. It is often suggested [131] that all of the ratings should be used for similarity computation in neighborhood-based methods. Window-based pruning is used only within the prediction function after the similarities have been computed with all the data. Such an approach can sometimes provide better robustness because of the sparsity of the ratings, in which any type of pruning can make similarity computations unstable. Pruning at the time of similarity computation can lead to overfitting.

2. In some cases, it is possible to obtain some insight into the active periods for various items depending on the underlying domain. In such cases, the windows can be set in a domain- and item-specific way. For example, the method in [131] uses not only the most recent ratings, but also the ratings from the same month in the previous years. Therefore, this approach combines window-based models with some periodic information. This method is referred to as *time-periodic biased k-NN* approach.

So far, all the temporal models are based on the time at which an item was *rated*. A somewhat different approach is to associate weights with a different temporal attribute than the rating time. For example, the work in [595] discusses how one might use the production time of a movie to drop it from consideration. A movie, which is too old, might not be relevant for a user looking for more recent movies. Note that such an approach prunes *all* ratings for the item, because the production time is associated with the item and not the user-item combination. Pruning all ratings for an item is equivalent to dropping the item from the ratings matrix. Therefore, such an approach reduces the dimensionality of the data set by effectively removing them from consideration. However, such an approach should be used with caution, and only for items which are inherently time-sensitive on certain well-known traits.

9.2.2 Handling Periodic Context

Periodic context is designed to handle cases in which the time dimension may refer to a specific period in time, such as hour of the day, day of the week, season, or the time intervals in the vicinity of specific periodic events (e.g., Christmas). Such cases are best handled with the use of multidimensional contextual models, as proposed in [6]. These methods are discussed in detail in Chapter 8.

In this case, the target recommendation time defines the context in which the recommendation is made. This context can sometimes play an all-important role in the recommendation process. For example, for a supermarket, the recommendations targeted for the weekend before the Thanksgiving holiday would be very different from those targeted during other times. Several natural ways of handling periodic context are discussed in the following sections.

9.2.2.1 Pre-Filtering and Post-Filtering

There are two types of filtering methods used in the context-based methods that are referred to as *pre-filtering* and *post-filtering*, respectively. These methods are discussed in detail in sections 8.3 and 8.4 of Chapter 8. Here, we provide a brief overview in the context of temporal recommender systems.

In pre-filtering, a significant part of the ratings data are removed that are not relevant to the specific target time (i.e., context) within which the recommendation is being performed or executed. For example, one might use only the ratings from the fortnight before Thanksgiving of each year, in order to construct the models for making recommendations on the weekend before Thanksgiving. A particularly interesting approach along this direction was the use of *contextual microfiles* [61], which segmented the ratings by context. This kind of segmentation effectively filters out the irrelevant ratings from each segment. Some examples of possible segmentations include $\{\text{Morning}, \text{Evening}\}$, $\{\text{WeekDay}, \text{Weekend}\}$, and so on. Within each context, a separate model is constructed for prediction. After filtering, any non-contextual method may be used to make predictions on the pruned data within each segment. The main challenge associated with pre-filtering methods is that the pruned data set is even more sparse than the original data, and therefore the accuracy of the recommendation process is impacted negatively. This is a direct result of overfitting. The success of pre-filtering often depends on the sparsity of the pruned data set. Therefore, the approach cannot be used easily for contexts that are too fine-grained (e.g., day of year). In many cases, hierarchies are used on the periodic context to improve the accuracy of recommendation. For example, consider a scenario where the context is set to 7 AM. Instead of using ratings received between 7 AM and 8 AM, one might use all the ratings received between 6 AM to 9 AM. This will result in the use of a larger number of ratings, and the approach will therefore help to prevent overfitting.

In post-filtering, the recommendations are adjusted based on the context, after a non-contextual method has been used to generate the recommendation on all the data. Therefore, the basic approach of post-filtering uses the following two steps:

1. Generate the recommendations using a conventional collaborative filtering approach on all data, while ignoring the temporal context.

2. Adjust the generated recommendation list with the use of temporal context as a post-processing step. Either the ranks of the recommended list may be adjusted, or the list may be pruned of contextually irrelevant items.

After the recommendation lists have been generated, the ranking is either re-adjusted by weighting with a *contextual relevance weight*, or the items with very low contextual relevance weights are removed. Let $\hat{r}_{uj}$ represent the predicted rating of user u for item j using all the data, before contextual post-filtering has been applied. The resulting ratings (and rankings) are then adjusted with the use of contextual relevance weights $P(u, j, C)$, where C is the context. Therefore, the adjusted rating is given by $\hat{r}_{uj} \cdot P(u, j, C)$.

How is the contextual relevance weight determined? In contrast to pre-filtering methods, which work only with ratings, post-filtering methods often use content properties of items in order to determine the contextual relevance weights. Under the covers, post-filtering methods sometimes incorporate pre-filtering techniques to a minor degree in the process of determining contextual relevance weights. For a given user u for whom a prediction needs to be made, her ratings for the specific period of interest are pre-filtered, and an off-the-shelf recommendation model is constructed on the pre-filtered ratings, to predict her ratings for that specific periodic context C. For example, if movie recommendations are to be made in the context of weekends, the relevance of the user for each movie in the weekend is determined with the use of either a collaborative model or a content-based model on the pre-filtered data. The work in [471] uses a very simple scheme in which the fraction of neighbors of a user who have watched a particular movie in the pre-filtered data is used in order to compute the contextual relevance weight. This relevance weight $P(u, j, C)$ is assumed (or scaled) to be a probability value in $(0, 1)$, where larger values imply greater interest. Then, the predicted rating $\hat{r}_{uj}$ is multiplied with the relevance weight $P(u, j, C)$, or the item is simply removed from the recommended list when $P(u, j, C)$ is very small. These two methods are referred to as *weighting* or *filtering* methods in contextual post-filtering [471]. Post-filtering methods hedge their bets between the robustness of (larger) global data sets and the refined accuracy of the pruned data by making use of both in the recommendation process.

In many cases, the estimation of $P(u, j, C)$ is executed independent of the user u, by using only the content information in the item j. For example, if comedy movies and Steven Spielberg movies are often watched over weekends by all users, the genre/actors/director of the movie can be used as the content, and the label is either *weekends* or *weekdays*. The training data can contain the data over all users and not just user u. A machine learning model then estimates the value of $P(*, j, C)$ by using this training data, where "*" denotes a "don't care." Such an approach is less personalized in the computation of $P(u, j, C)$, but it can handle sparsity more effectively. Note that the final predicted value $\hat{r}_{uj} \cdot P(*, j, C)$ is still personalized to u because of how $\hat{r}_{uj}$ is determined. The specific choice of model used for estimating $P(u, j, C)$ depends on the data set at hand and its sparsity. Readers are advised to refer to Chapter 8 for more details about both methods. In particular, post-filtering methods are discussed in section 8.4.

9.2.2.2 Direct Incorporation of Temporal Context

In pre-filtering and post-filtering methods, the incorporation of context is done either strictly before or strictly after the recommendation process. In both cases, the approach reduces the problem to a 2-dimensional model. However, it is also possible to directly modify existing models such as neighborhood methods in order to incorporate temporal context. In such cases, one works directly with the 3-dimensional representation corresponding to user, item, and context. For example, in the user-based neighborhood scheme, one might modify the distance computation between two users with the use of contextual attributes. If two users give the same rating to an item during weekends, they should be considered more similar than a pair of users that have specified these ratings in different temporal contexts. By using the modified distance computation, the context is automatically incorporated into the recommendation process. One can also modify regression and latent-factor models to incorporate the temporal context directly. These methods apply generally to any context-based scenario (e.g., location), and not just the temporal context. Therefore, this topic is discussed in detail in Chapter 8 on context-based methods. Refer to section 8.5 of that chapter.

9.2.3 Modeling Ratings as a Function of Time

In these methods, the ratings are modeled as a function of time and the parameters of the model are learned in a data-dependent way. A few methods that use time-series models to make predictions are discussed in the bibliographic notes. In this section, we will study the use of *temporal factor models*, which are considered state-of-the-art in this domain.

These methods can intelligently separate long-term trends from transient and noisy trends. Furthermore, the models have a natural element of forecasting built into them. These distinctions are important for making such temporal models robust. Such robustness cannot be achieved with a mere decay-based or filtering approach to the temporal model. In this section, we will study the *time-SVD++* model, on which a lot of subsequent work in the field is based.

9.2.3.1 The Time-SVD++ Model

The *time-SVD++* model can be viewed as a temporal enhancement of the *SVD++* model. Readers are advised to revisit section 3.6.4.6 of Chapter 3, as the discussion in this section relies on that previous passage. We will also briefly discuss the *SVD++* model here; this will provide us with the opportunity to introduce slightly different notations from those in Chapter 3. These notations are relevant to the temporal version of the model.

As in the case of the *SVD++* model, we can assume without loss of generality that we are working with a ratings matrix in which the mean of all ratings in the training data is 0. Note that when the mean (denoted by μ) of all ratings is nonzero, it can be subtracted from all the entries, and the analysis can be performed with this centered matrix in order to predicted correspondingly centered ratings. Later, the mean can be added back to the predicted value of the rating.

Recall that the factor model of section 3.6.4.5, which incorporates bias, expresses the ratings matrix $R = [r_{ij}]_{m \times n}$ in terms of the user biases, the item biases, and the factor matrices. The predicted rating $\hat{r}_{ij}$ is expressed in terms of these variables as follows:

$$\hat{r}_{ij} = o_i + p_j + \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{9.5}$$

Here, o_i is the bias variable for user i, p_j is the bias variable for item j, $U = [u_{is}]_{m \times k}$, and $V = [v_{js}]_{n \times k}$ are factor matrices of rank k. The part $(o_i + p_j)$ does not use any personalized rating data, but it simply relies on global properties of the ratings. Intuitively, the variable o_i indicates the propensity of user i to rate all items highly, whereas the variable p_j denotes the propensity of item j to be rated highly. For example, a generous and optimistic user is likely to have large positive values of o_i, and a box office hit is likely to have large positive values of p_j. This basic bias-based model is further enhanced in section 3.6.4.6 with the notion of implicit feedback variables $Y = [y_{ij}]_{n \times k}$ for each user-item pair. These variables 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 user for the jth latent component (no matter what the actual value of the rating might be). In other words, the jth latent component of any user who has rated item i should be adjusted based on the value of y_{ij}.

Let I_i be the set of items rated by user i. Then, the predicted value of the rating, which includes implicit feedback, can be expressed as follows:

$$\hat{r}_{ij} = o_i + p_j + \sum_{s=1}^{k} \left(u_{is} + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right) \cdot v_{js} \tag{9.6}$$

Note that the term $\sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}}$ on the right-hand side of the aforementioned equation adjusts the sth latent factor u_{is} of user i, based on the implicit feedback. Refer to section 3.6.4.6 of Chapter 3 for a more detailed explanation of this adjustment. Equation 9.6

is identical to Equation 3.21 of Chapter 3, except that we use slightly different[2] notations here in explicitly separating out the bias variables.

The main difference between the *SVD++* model and the *time-SVD++* model is that *some* of the model parameters are assumed to be functions of time in the latter. Specifically, the *time-SVD++* model assumes that the user biases o_i, item biases p_j, and the user factors u_{is} are functions of time. Therefore, these terms will be expressed as $o_i(t)$, $p_j(t)$, and $u_{is}(t)$ to denote the fact that they are functions of time. By using these temporal variables, one now obtains the time-varying predicted value $\hat{r}_{ij}(t)$ of the (i, j)th entry of the ratings matrix at time t as follows:

$$\hat{r}_{ij}(t) = o_i(t) + p_j(t) + \sum_{s=1}^{k} \left(u_{is}(t) + \sum_{h \in I_i} \frac{y_{hs}}{\sqrt{|I_i|}} \right) \cdot v_{js} \tag{9.7}$$

It is noteworthy that the item variables v_{js} and the implicit feedback variables y_{hs} have not been temporally parameterized, and are assumed to stay constant with time. It is possible in principle[3] to temporally parameterize these variables as well, although the *time-SVD++* model chooses a simplified approach in which each temporal parametrization can be justified with intuitive arguments. These intuitions are discussed below, together with the specific form of the temporal parametrization of each of the variables $o_i(t)$, $p_j(t)$, and $u_{is}(t)$, respectively:

1. The intuition for choosing the temporal form of the item bias $p_j(t)$ is that the popularity of an item can vary significantly with time, but it shows a high level of continuity and stability over shorter periods. For example, a box-office hit will have an approximately stable distribution of ratings in the short period after release, but it might be rated very differently after a couple of years have elapsed. Therefore, the time horizon can be split into bins of equal size, and the ratings belonging to a particular bin have the same bias. Smaller bin sizes lead to better granularity but it may also result in overfitting because enough ratings may not be present in each bin. In the original work on Netflix movie ratings [310], a total of 30 bins were used, and each bin represented about 10 consecutive weeks of ratings. The item bias $p_j(t)$ can now be split into a constant part and an offset parameter, which is bin-specific depending on the time t at which item j is rated:
$$p_j(t) = C_j + \text{Offset}_{j,Bin(t)} \tag{9.8}$$
Note that both the constant part C_j and offsets are parameters that need to be learned in a data-driven manner. The optimization problem for this learning process will be discussed later. Note that the value of $p_j(t)$ will be different for different ratings, depending on *when* they were rated. Unlike users, items can be more successfully binned in this way because most items usually have sufficient ratings.

2. A different approach is used to parameterize the user bias $o_i(t)$. The binning approach will not work for users because many users may not have sufficient ratings. Therefore, a functional form may be used to parameterize the user bias, which captures the

[2]In the discussion of section 3.6.4.6, the bias variables are absorbed within the factor matrices U and V by increasing the number of columns in each of the two factor matrices from k to $(k + 2)$. However, in this exposition, we do not absorb the bias variables in the columns of the factor matrices. This is because of the more complex and special way in which bias variables are treated in temporal models. For example, Equation 3.21 of Chapter 3 and Equation 9.6 are identical, but they use somewhat different notations. It is important to keep these notational distinctions in mind to avoid confusion.

[3]The work in [293] uses time-varying item factors.

concept drift of the user over time. Let the mean date of all ratings of user i be denoted by ν_i. Then, the concept drift $\text{dev}_i(t)$ of user i at time t can be computed as a function of t as follows:

$$\text{dev}_i(t) = \text{sign}(t - \nu_i) \cdot |t - \nu_i|^{\beta} \tag{9.9}$$

The parameter β is selected using cross-validation. A typical value of β is around 0.4. In addition, the transient noise at each time t is captured with the parameters ϵ_{it}. Then, the user bias $o_i(t)$ is split into a constant part, a time-dependent part, and transient noise, as follows:

$$o_i(t) = K_i + \alpha_i \cdot \text{dev}_i(t) + \epsilon_{it} \tag{9.10}$$

In practice, the time is often discretized on a day-specific basis. Therefore, ϵ_{it} corresponds to the transient day-specific variability. As in the case of item bias parameters, the parameters K_i, α_i, and ϵ_{it} must be learned in a data-driven manner. The idea here is that the average rating of the user can vary significantly from that at the mean date of rating. The user might be rating most items positively (or negatively) now, but her mean rating might decrease (or increase) in a couple of years. This portion of variability is captured by $\alpha_i \cdot \text{dev}_i(t)$. However, transient mood changes from day to day might lead to sudden and unpredictable spikes or dips in ratings. A user might rate all items poorly when she is having a bad day. Such variations are captured by ϵ_{it}.

3. The user factors $u_{is}(t)$ correspond to the affinity of users towards various concepts. For example, a young user who appreciates action movies today might be interested in documentaries after a few years. As in the case of user biases, the amount of elapsed time is a crucial factor in deciding the amount of drift. Therefore, a similar approach to user biases is used for modeling the temporal change in the user factors:

$$u_{is}(t) = K'_{is} + \alpha'_{is} \cdot \text{dev}_i(t) + \epsilon'_{ist} \tag{9.11}$$

As in the case of user biases, the constant effects, long-term effects, and transient effects are modeled by the three terms. Although we have used similar looking symbols as user biases to emphasize the similarity between the two modeling cases, we have added an apostrophe superscript to each variable to emphasize the fact that the variables in Equations 9.10 and 9.11 are distinct. Note that the same user-specific deviation function $\text{dev}_i(t)$ is used in the two cases although it is possible to use different forms of this function for the two cases.

How does one use the aforementioned model to set up the optimization problem? We assume that the observation time of all the ratings is known. Therefore, for an entry (i, j) whose rating has been observed at time t_{ij}, one must compare the observed value r_{ij} with the predicted value $\hat{r}_{ij}(t_{ij})$ in order to compute the error in prediction. In this case, one needs to minimize the squared error function $[r_{ij} - \hat{r}_{ij}(t_{ij})]^2$ over all the observed ratings in the data. The value of $\hat{r}_{ij}(t_{ij})$ is derived with the help of Equation 9.7. In addition, the squared regularization terms for the various parameters need to be added to the objective function. In other words, if S contains the set of user-item pairs for which ratings are specified in the matrix $R = [r_{ij}]_{m \times n}$, then one must solve the following optimization problem:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} [r_{ij} - \hat{r}_{ij}(t_{ij})]^2 + \lambda \cdot (\text{Regularization Term})$$

The regularization term contains the sum of the squares of all the variables in the model. As in the case of all the factorization models discussed in Chapter 3, a gradient descent approach can be used in order to optimize the objective function J and learn the relevant parameters. The partial derivative of J is computed with respect to each parameter to determine the relevant gradient directions. These learned parameters are then used for prediction. The details of these learning steps are omitted. Readers are referred to the original work [310] for more details. Here, we will discuss how the model can be used once the parameters have been learned.

Using the Model for Prediction

After the parameters of the model have been learned, how can they be used for predictions? For a given user i and item j, one can use Equation 9.7 to determine the predicted rating $\hat{r}_{ij}(t)$ at future time t by substituting the learned values of the parameters. The main problem with doing so is the presence of day-specific parameters, such as ϵ_{it} and ϵ_{ist}. These parameters can only be learned for past days from the historical data, but they cannot be learned for future days. However, these parameters only correspond to transient noise, which cannot, by definition, be learned in a data-driven manner. Therefore, these values are set to 0 for future days under the assumption that a noise-free prediction is being made; accordingly, the learned values of these parameters are not used for prediction. Although these parameters are not used in the final prediction, they are still quite important for the modeling process because they absorb the transient noise and spikes in the ratings. For example, if a user provides very low ratings to all items because she is having a bad day, the presence of these parameters will dampen the effects of this transient noise in the historical data. Therefore, the parameters ϵ_{it} and ϵ_{ist} help in learning the *other* parameters in a more robust way by removing transient spikes and noise. In other words, the day-specific parameters ϵ_{it} and ϵ_{ist} play the role of cleaning the training data in the modeling process.

Practical Issues

An immediate observation is that the aforementioned model has a very large number of parameters as compared to those in Chapter 3. Therefore, it is crucial to have sufficient data, so that overfitting may not remain an issue. This can be a problem for smaller data sets. However, the approach seems to perform quite well [310] for the Netflix data set, which is quite large. Interestingly, the survey in [312] shows that one can obtain reasonably good results on the Netflix Prize data set by dropping the factorization completely and using only the bias terms. The use of only bias terms yielded results that were almost comparable to Netflix's *Cinematch* recommender system. This is because the non-personalized aspect of the ratings (i.e., user-specific and item-specific bias) can explain a very large part of the ratings. These results suggest the importance of incorporating the bias terms in latent factor models, as discussed in section 3.6.4.5 of Chapter 3.

Furthermore, the time-dependent terms in $o_i(t)$ and $u_{is}(t)$ can be modeled using other functional forms such as splines or with the use of periodic trends. These different functional forms can capture different data-specific temporal scenarios. We have restricted our discussion to the simplest possible choices for ease in discussion. A detailed discussion of these alternatives is provided in [312].

Observations

It is noteworthy that user factors vary in the temporal sense but item factors do not. This choice can be intuitively justified. Recall from the discussion in Chapter 3 that the user factors correspond to user affinities towards various concepts, whereas the item factors correspond to item affinities towards various concepts. The basic idea here is that user moods and preferences can change over time, which will be reflected in the changes of their affinities towards the various concepts. On the other hand, the affinity of an item to a concept is inherent to that item, and it can be assumed to be stable over time. Therefore, it is not necessary to increase the complexity of the model by temporally parameterizing the item factors. Unnecessary temporal parametrization increases the complexity of the model and leads to overfitting. Nevertheless, the work in [293] shows how one might use time-varying item factors as well. It is an open question as to whether the use of temporal parametrization of item biases will lead to overall improvement in accuracy over most data sets.

9.3 Discrete Temporal Models

Discrete temporal models are relevant to the case where the underlying data is received as discrete sequences. Such data can be encountered in a variety of application scenarios, most of which are associated with implicit user feedback rather than explicit ratings. Some examples of such application scenarios are as follows:

1. *Web logs and clickstreams:* The user accesses to Web logs can typically be represented as sequential patterns. User patterns can often show predictable access patterns. For example, users will frequently access particular sequences of Web pages. The frequent sequence information can be used to make recommendations [182, 208, 440, 442, 443, 562].

2. *Supermarket transactions:* The customer buying behavior in supermarkets is a form of sequential data. In fact, the problem of sequential pattern-mining was defined [37] to handle this scenario. In fact, because the activity time-stamps are usually available in supermarket data sets, they can be converted into user-specific sequential patterns of buying activity. The *temporal order* is often quite important. For example, it makes sense to recommend a printer cartridge after a user has bought a printer, but not vice versa.

3. *Query recommendations:* Many Web sites record the user queries at their site. The sequence of queries can be used to make recommendations of other more useful queries.

In this section, two types of models will be discussed. The first of these methods is based on Markovian models, whereas the second is based on sequential pattern-mining.

9.3.1 Markovian Models

An interesting Markovian model to predict Web page accesses was proposed in [182]. Although the approach is discussed in the context of Web page accesses, it can be generalized to recommending any type of action, as long as the temporal ordering of the user actions is available. The discussion in this section is based on this work [182].

In Markovian models, the sequential information is encoded in the form of *states*, which are used for predictive purposes. A kth order Markov model defines a state based on the last k actions performed by the user. An *action* is defined in an application-specific way. It might correspond to the user visiting a particular Web page, or it might correspond to the user buying a specific item. The actions are represented by a set of symbols Σ. As the actions are application-specific, the symbol set Σ is application-specific as well. For example, the symbol set Σ could correspond to the indices of the universe of items in an e-commerce application, or it could correspond to the URLs of Web pages in a Web log mining application. We assume that the symbol set Σ contains the symbols $\Sigma = \{\sigma_1 \ldots \sigma_{|\Sigma|}\}$. Therefore, a state $Q = a_1 \ldots a_k$ is defined by sequence of k actions, such that each a_i is drawn from Σ. A state with k actions is drawn from an order-k Markovian model. For example, consider the case in which the symbols in Σ correspond to the act of watching various movies. Furthermore, consider the following state Q:

$$Q = \textit{Julius Caesar}, \textit{Nero}, \textit{Gladiator}$$

This state has three different actions corresponding to the user watching these movies in a particular sequence. Therefore, this state is drawn from an order-3 Markov model. Furthermore, the default assumption in such Markov models is that the movies have been watched *consecutively*. There are a total of $|\Sigma|^k$ possible states in an order-k Markovian model, although many of these states might not occur frequently in a particular data set.

In general, a sequence defines the transitions[4] in a *Markov chain*. In an order-k model, the current state is defined by the last k actions in the Markov chain. Consider a sequence of actions (e.g., Web page accesses), in which t actions $a_1 a_2 \ldots a_t$ have occurred so far in sequence, where $a_i \in \Sigma$. Then, the current state of the order-k Markov model at time t is $a_{t-k+1} a_{t-k+2} \ldots a_t$. The last action in this sequence is a_t, which resulted in a transition from the state $a_{t-k} a_{t-k+1} \ldots a_{t-1}$ to the state $a_{t-k+1} a_{t-k+2} \ldots a_t$. Therefore, the states in a Markov chain are connected by edges, corresponding to transitions. Each edge is annotated with an action drawn from Σ, and a probability of transition. In this particular example, the transition from the state $a_{t-k} a_{t-k+1} \ldots a_{t-1}$ to the state $a_{t-k+1} a_{t-k+2} \ldots a_t$ is associated with the action a_t. As there are $|\Sigma|$ possible transitions out of each of the $|\Sigma|^k$ states, the total number of edges in a complete Markov model of order k is equal to $|\Sigma|^{k+1}$. Any incoming edge of a state $a_{t-k+1} a_{t-k+2} \ldots a_t$ in an order-k Markov chain is always annotated with the last action a_t. The sum of the probabilities of the transitions out of a state is always 1. The probabilities of transitions are learned from the training data (e.g., a sequence of previous Web page accesses). We have shown an order-1 Markov chain drawn on the alphabet $\{A, B, C, D\}$ in Figure 9.1. Note that this Markov chain has 4 states and $4 \times 4 = 16$ edges. The sequence of actions $AABCBCA$ corresponds to the following path of states in the Markov chain:

$$A \Rightarrow A \Rightarrow B \Rightarrow C \Rightarrow B \Rightarrow C \Rightarrow A$$

Note that a Markov model of order-2 would contain $4^2 = 16$ states and $4^3 = 64$ edges. This is already too large to neatly show in a diagram like Figure 9.1. The corresponding sequence of transitions for the action sequence $AABCBCA$ is given by the following:

$$AA \Rightarrow AB \Rightarrow BC \Rightarrow CB \Rightarrow BC \Rightarrow CA$$

Consider a situation, where we have trained a Markov model of order k, and we need to make a prediction for the next action after the sequence $a_1 \ldots a_t$. Then, for each action $\sigma_i \in \Sigma$, we need to estimate the value of the action σ_i, given the current state of the last k actions. In

[4]Refer to the bibliographic notes for background on Markov chains.

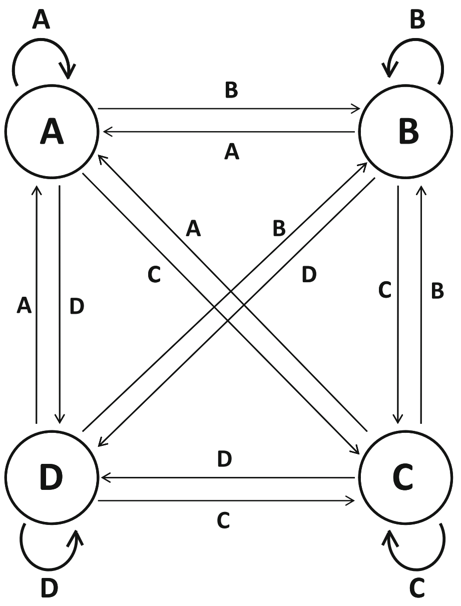

Figure 9.1: An order-1 Markovian model

other words, we need to estimate the probability $P(a_{t+1} = \sigma_i | a_{t-k+1} a_{t-k+2} \ldots a_t)$ for each $\sigma_i \in \Sigma$. The top-r actions with the largest probabilities can be returned as the predictions. Note that the probabilities $P(a_{t+1} = \sigma_i | a_{t-k+1} a_{t-k+2} \ldots a_t)$ need to be estimated from the training data. This suggests the following simple approach for training and prediction on a Markov model of order k.

1. **(Training phase)** Let $\mathcal{S}$ be the set of $|\Sigma|^k$ possible sequences of length k. For each possible sequence (state) $S \in \mathcal{S}$, use the training data to learn the $|\Sigma|$ probabilities $P(\sigma_i|S)$ corresponding to each candidate action $\sigma_i \in \Sigma$. Note that a total of $|\Sigma|^{k+1}$ probabilities need to be learned, which is also equal to the number of edges in an order-k Markov model. Each of the learned probabilities corresponds to the transition probability of an edge in the Markov model.

2. **(Prediction phase)** For a current sequence of user actions, determine the relevant state S_t in the Markov chain using the last k actions of the user. Report the top-r actions in Σ with the largest values of $P(\sigma_i|S_t)$ as the recommendations.

The Markovian approach relies on the *short memory assumption* of typical user action sequences. The idea is that the user actions depend only on the set of k immediately preceding actions. While this assumption may not be completely true in practice, it often approximates many real-world scenarios.

It remains to be explained how the probabilities are estimated from a given training data set. This can be achieved by extracting all of the k-sequences from the training database and determining the fraction of times that each action in σ_i occurs after this sequence. This estimate is determined as the relevant probability. Consider a sequence S, which is one of the $|\Sigma|^k$ possible sequences. If this sequence occurs $F(S)$ times in the training data, and the sequence S is followed by the action σ_i for a total of $f(S, \sigma_i) \leq F(S)$ times in the data, then the probability $P(\sigma_i|S)$ is estimated as follows:

$$P(\sigma_i|S) = \frac{f(S, \sigma_i)}{F(S)} \tag{9.12}$$

Note that the training data may contain either one long sequence or multiple sequences. In either case, the frequencies $f(S, \sigma_i)$ and $F(S)$ count the repeated occurrences within a single sequence as multiple occurrences.

This estimation may sometimes be difficult when the value of $F(S)$ is small. In fact, when the value of $F(S)$ is 0, the estimated probability becomes indeterminate. In order to address this problem, we use Laplacian smoothing. A Laplacian smoothing parameter α is used to modify the aforementioned estimation as follows:

$$P(\sigma_i|S) = \frac{f(S, \sigma_i) + \alpha}{F(S) + |\Sigma| \cdot \alpha} \tag{9.13}$$

Typically, the value of α is set to a small quantity. Note that when the value of $F(S)$ is 0, each action is estimated to the probability value of $1/|\Sigma|$. This is quite reasonable when we do not have sufficient data about the specific action after a particular sequence. The notion of Laplacian smoothing serves a similar function to that of regularization by avoiding overfitting from limited training data. In practice, states with zero frequency are not represented at all in the Markov model. This means that some of the states are missing, and it may not be possible to find a matching state for a particular test sequence. Such test instances are said to be *uncovered* by the Markov model. How are such states handled?

The work in [182] builds all Markov models up to maximum order l, and then uses the highest-order model that covers the test instance. In other words, if all models up to order 3 are constructed, the approach first tries to find a matching state in the order-3 Markov model. If such a state is found, then it is used for prediction. Otherwise, the Markov model of order 2 is tested, and then the model of order 1 is tested. For most training data sets of reasonable sizes, all possible $|\Sigma|$ states are present in the Markov model of order 1, and therefore it serves as a default model for difficult cases in which matching models of higher order are not found. If needed, a default catch-all prediction corresponding to the most frequent action can be returned if no matching state is found.

9.3.1.1 Selective Markov Models

One of the problems with the approach outlined in the previous section is that the number of possible states may be too large, and most of them may not even be present in a particular training data set. The large number of states also makes it expensive to train the model, necessitating the estimation of as many as $|\Sigma|^{k+1}$ possible probabilities for an order-k Markov model. For larger values of k, it may be impractical to train such a model. Furthermore, many of the states with little presence in the training data may be unreliable for training purposes.

The main idea in [182] is to propose the notion of *selective* Markov models, in which many of the irrelevant states are pruned up front during model construction. This pruning may be accomplished in several ways:

1. *Support-pruned Markov model:* The *support* of a state (or k-sequence) is the frequency of its presence in the training data. The basic assumption is that states with low support are unreliable in terms of their predictive power on unseen test data.

In particular, the estimated probabilities of states with low support might be unreliable because of overfitting. Support pruning can drastically reduce the number of states in the models of higher orders. The support threshold is defined as an absolute frequency (rather than as a fraction), and is defined as the same value across models of all orders. Higher-order models have lower support values, and are therefore more likely to have their states pruned. This approach greatly reduces the state-space complexity of the model because the number of possible states increases exponentially with the order of the model.

2. *Confidence-pruned Markov model:* The confidence-pruned Markov model tends to favor states in which the highest probability of an outgoing edge from a state is as large as possible. Note that if all the transition probabilities on the edges exiting a state are similar, one cannot confidently claim that any of the actions in Σ is significantly more likely than the others. In the other extreme case, if one of the edges exiting a state has a probability of almost 1 and the others are almost 0, then one can confidently predict the next action at this state. Such states are more useful. How can one determine the appropriate confidence thresholds for pruning?

   This approach computes the $100 \cdot (1-\alpha)$ confidence interval around the most probable action, and then determines if the second-highest probability lies in this interval. Consider a candidate state for pruning, which has raw frequency n in the training data. Let p_1 and p_2 be the transition probabilities of the first- and second-highest probable edges exiting that state. We are already assured that $p_2 \leq p_1$ because p_1 is probability of the most probable edge. Let $z_{\alpha/2}$ be the absolute value of the Z-number matching the upper $(\alpha/2)$-percentage point of the standard normal distribution. Then, in order for the state to be pruned, the following condition must hold:

$$p_2 \geq p_1 - z_{\alpha/2}\sqrt{\frac{p_1(1-p_1)}{n}} \tag{9.14}$$

   Note that $\sqrt{\frac{p_1(1-p_1)}{n}}$ represents the standard deviation of the average of n i.i.d. Bernoulli variables, each of which have probability p_1 of success. The level of pruning is controlled by the confidence threshold α.

3. *Error-pruned Markov model:* In the error-pruned Markov model, a validation set is held out from the training data and is not used for building the Markov model. This validation set is used to test the accuracy of the model. The accuracy specific to each state is computed with the validation set. For each higher-order state, its immediate lower-order prediction alternatives are determined. For example, for an order-4 state $a_1a_2a_3a_4$, the same action sequence can be predicted with the lower-order states $a_2a_3a_4$, a_3a_4, and a_4. If the error rate of a higher-order state is greater than that of any of its lower-order alternatives, then it is pruned. This process is recursively applied to states of all orders, starting from the higher to the lower, until no more states can be pruned. States of order 1 are always retained in order to maximize coverage.

   Although the aforementioned approach compares the errors of higher- and lower-order states, it does not use the same validation examples to compare the accuracy of a pair of states. A second approach for error pruning that uses the same set of validation examples for comparing the error of two states is as follows. First, all the validation examples that can be predicted with a higher-order state are determined. Then the same validation examples are tested with respect to the lower-order states. If the error with the use of the higher-order state is greater than that of any of the lower-order states *on the same validation examples*, then the higher-order state is pruned. This approach is applied recursively to all states of lower order, except for states of order 1.

These alternatives were experimentally tested in [182]. It was shown that all forms of pruning provided some advantage, although the greatest advantage was obtained with the use of error-pruned models. There was little statistical difference between the support-pruned and confidence-pruned Markov models.

9.3.1.2 Other Markovian Alternatives

In the Markovian models of this section, a *contiguous* sequence of actions is used to predict the next action. Furthermore, the states are fully visible and can be directly explained in terms of the last k user actions. A more sophisticated alternative is the use of *Hidden* Markov Models (HMMs) in which the states are hidden. In such cases, non-contiguous subsequences can be used to make predictions. The HMM approach is beyond the scope of this book; refer to the bibliographic notes.

9.3.2 Sequential Pattern Mining

Sequential pattern-mining was originally proposed as a method for mining patterns from sequences of supermarket data. Sequential patterns can be used to create rule-based predictive models for temporal sequences. Such methods can be considered the temporal analog of rule-based methods discussed in section 3.3 of Chapter 3. First, we define the notions of *subsequences* and *frequent subsequences.*

Definition 9.3.1 (Subsequence) *A sequence of symbols $a_1 a_2 \ldots a_k$ is said to be a subsequence of the sequence $b_1 b_2 \ldots b_n$, if we can find k elements $b_{i_1} \ldots b_{i_k}$, such that $i_1 < i_2 < \ldots < i_k$, and $a_r = b_{i_r}$.*

In the original definition of sequential pattern mining [37], the elements are themselves allowed to be sets, and the condition $a_r = b_{i_r}$ is replaced with the condition $a_r \subseteq b_{i_r}$. However, in most recommender applications, this complex definition is not necessary, and we can work with sequences of individual symbols. Therefore, we will use this simplified definition in this chapter. It is noteworthy that the definition of a subsequence allows gaps in the matching. The allowance of such gaps is useful in accounting for the noise in sequences.

In sequential pattern-mining methods, the goal is to determine the frequently occurring subsequences in the data at a support level s. The frequency is defined with respect to a database $\mathcal{D}$ of multiple sequences.

Definition 9.3.2 (Frequent Subsequences) *A subsequence $a_1 \ldots a_k$ is said to be a frequent subsequence with respect to a database $\mathcal{D}$ of sequences at minimum support s, if it is a subsequence of at least a fraction s of the sequences in the data.*

Note that the support s is always a fraction by definition. One can also define the confidence of a rule in sequential pattern mining. Traditionally, the notion of confidence is defined only for non-temporal association rules, but one can also extend the definition to sequential pattern mining in various ways.

Definition 9.3.3 (Confidence) *The confidence of a rule $a_1 \ldots a_k \Rightarrow a_{k+1}$ is equal to the conditional probability that $a_1 \ldots a_{k+1}$ is a sequence in the database, given that $a_1 \ldots a_k$ is a sequence. In other words, if $f(S)$ denotes the support of sequence S, then the confidence of the rule $a_1 \ldots a_k \Rightarrow a_{k+1}$ is defined as follows:*

$$\mathit{Confidence}(a_1 \ldots a_k \Rightarrow a_{k+1}) = \frac{f(a_1 \ldots a_{k+1})}{f(a_1 \ldots a_k)}$$

Note that the definition of confidence in sequential rule mining is adapted from association rule mining. The notion of confidence can be defined in other ways depending on the application at hand. For example, one can impose the constraint that a_{k+1} immediately follows a_k without a gap in some applications.

The definitions of support and confidence can be used to define sequential pattern-based rules.

Definition 9.3.4 (Sequential Pattern-Based Rule) *A rule $a_1 \ldots a_k \Rightarrow a_{k+1}$ is said to be valid at minimum support s and minimum confidence c, if both the following conditions are satisfied:*

1. *The support of $a_1 \ldots a_{k+1}$ is at least s.*
2. *The confidence of $a_1 \ldots a_k \Rightarrow a_{k+1}$ is at least c.*

Algorithms for determining frequent sequential patterns are discussed in [23]. After the sequential patterns have been determined, one can also determine the rules at the desired level of minimum support and confidence. The training phase in sequential pattern-mining methods finds all the rules at the specified level of minimum support and confidence. After the rules have been determined, the following approach is used for the prediction of the relevant ranked list of items (e.g., clicks in a Web clickstream) for a current test sequence T:

1. Identify all the matching rules for the test sequence T.
2. Rank the items in the consequents of the matching rules in decreasing order of confidence. Heuristic methods can be used to aggregate the predictions when multiple rules contain the same item in the consequent.

In some cases, it may be desirable to restrict the gaps between successive elements. For example, when the sequences are very long, it generally becomes more desirable to impose gap constraints on the sequences during the training and prediction process. Depending on the specific application at hand, many variations of this basic approach may be used. These variations are as follows:

1. Maximum gap constraints may be imposed during the process of finding the frequent sequences. In other words, the matching process may allow the maximum gap between a pair of adjacent sequences to be at most δ. Alternatively, one can impose a maximum constraint on the time difference between the first and last elements of the sequence. Such constraints can be handled by *constrained sequential pattern-mining* methods, and they are particularly important when the individual sequences in the database are very long. A discussion of constrained sequential pattern-mining methods may be found in [22].

2. The entire test sequence T may not be necessary for prediction. Rather, only the most recent window from the test sequence of a pre-defined size may be used. The windowing approach is necessary when the lengths of the individual sequences are long.

The most appropriate variation of this methodology depends on the specific application at hand. The bibliographic notes contain pointers to various recommender systems that use sequential pattern mining. Many of these systems have been developed in the context of Web clickstreams. Sequential-pattern-mining methods have the advantage that one can use numerous off-the-shelf tools for efficiently finding the patterns in large databases.

9.4 Location-Aware Recommender Systems

Location-aware recommender systems can be viewed as special cases of context-aware recommender systems, in which the context is defined by location. Location can influence the recommendation process in a wide variety of ways, of which the following two ways are particularly common:

1. The global geographical location of a user can have a significant influence on her preferences in terms of taste, culture, clothing, eating habits, and so on. For example, an analysis [343] of the MovieLens data set shows that the top genre preference of users from Wisconsin is *War*, whereas the top genre preference of users from Florida is *Fantasy.* Similar results are also shown on the Foursquare data set. This property is referred to as *preference locality.* In this case, the locality is inherently associated with the user, but not with the item. Therefore, in this case, the users are spatial, whereas the items are not.

2. Mobile users often want to discover restaurants or leisure places in the vicinity of their current location. In this case, the recommended items are inherently spatial. This property is referred to as *travel locality.* For example, an analysis [343] of the Foursquare data set shows that 45% of the users travel 10 miles or less, and 75% of the users travel 50 miles or less to visit a restaurant in their locality. In these applications, the location is inherently associated with the item (e.g., restaurant). Although users might specify their *current* location, this transient property is specified only during querying, and it is not inherently associated with the ratings specified by the user. Therefore, in this case, items are spatial, whereas users are not.

3. It is possible to imagine scenarios in which both users and items are spatial. For example, a traveling user might set up a profile, which indicates their permanent address. At the same time, they may record their ratings for spatial items such as restaurants. For example, consider two users from New Orleans and Boston, respectively, who are spending a vacation in Hawaii. These tourists might specify their ratings for restaurants in Hawaii. In this case, both users and items are spatial because their choices of their restaurants will be affected by their place of origin. At the same time, travel locality preferences will also play a role in their choice of restaurant, when the users query from specific locations in Hawaii during their vacation.

Location-aware recommender systems can be treated as special cases of context-sensitive methods. One can use the multidimensional techniques discussed in the previous sections in order to handle context within the recommender systems. This is especially true for the notion of preference locality, in which the multidimensional model of [6] may be used by treating location as a context, associating a hierarchical taxonomy of grid regions with the spatial location, and then reducing the problem to a traditional collaborative filtering application within one of the hierarchical regions of the grid. In fact, the *Location Aware Recommender System (LARS)* [343] does use a similar reduction-based approach for handling preference locality. However, the approach in [343] is much more sophisticated than a straightforward application of the multidimensional methodology of [6]. To represent the hierarchical taxonomy of grid regions, it uses a multidimensional indexing structure. This indexing structure can support incremental addition of ratings, and can therefore work well in settings that require scalability. Furthermore, the same work also proposes methods for handling travel locality and for combining travel and preference locality.

9.4.1 Preference Locality

As discussed earlier, the notion of preference locality shares a number of characteristics of the reduction-based multidimensional model of recommender systems [6]. For example, consider the example of the MovieLens data set, where the user locations are available in addition to ratings information. For a user in California, we might use only the ratings entered by other users of California in order to provide the recommendations for that user. This approach is equivalent to extracting a slice of the $User \times Item \times Location$ data cube, by fixing the location to California. Then a 2-dimensional recommender system can be used on this slice. This is a direct application of a reduction-based system [6].

Of course, such an approach is rather crude because the locality information may be available to a greater degree of granularity. For example, one might have the address of each user at hand. Users in southern California might show different preferences from those in northern California. On the other hand, for a small state or locality, enough rating data might not be available to make a robust recommendation. Therefore, one might need to combine the data from multiple adjacent regions. How can one meaningfully address such trade-offs?

The *LARS* approach [343] divides the entire spatial region in hierarchical fashion using a pyramid-tree or quad-tree [53, 202]. Note that this approach partitions the data *space*, rather than the data *points*, in order to ensure that every point in the space is included in one of the partitions. This ensures that new test locations can be effectively handled in the querying process, even if they are not represented in the data. The pyramid-tree decomposes the space into H levels. For any level $h \in \{0 \ldots H-1\}$, the space is partitioned into 4^h grid cells. The top-level at $h = 0$ contains only one cell, and it contains the entire data space. For example, consider the case where the top level of the model contains the region corresponding to the entire United States. Then, the next level divides the United States into four regions, with a separate model for each. The next level divides each of these regions into four more regions, and so on. Each grid cell contains a collaborative filtering model *only* for the region of the data space bounded by the corresponding rectangle. Therefore, the top-level grid cell contains a traditional (non-localized) collaborative filtering model containing all of the ratings. An example of the hierarchical partitioning of the pyramid-tree is illustrated in Figure 9.2. In the figure, the cell-identifier is denoted by CID, and the table entry to the left of it contains the pointer to the relevant collaborative filtering model for that cell. This data structure is maintained dynamically, so that ratings can be inserted or deleted from the system. One challenge in the dynamic update process is that it is sometimes not possible to maintain the models for a subset of the cells because of dynamic merges or splits of the cells during the updates. Note that the models for these new cells need to be recreated from scratch if cells merge or split during updates. This can sometimes be computationally expensive. However, if the tree is built up front without dynamic updates, then it is possible to maintain the models for all the entries. Therefore, the approach is straightforward when only static data is considered. The approach can also be extended to dynamic updates with some modifications. Readers are referred to [343] for details of the dynamic update process.

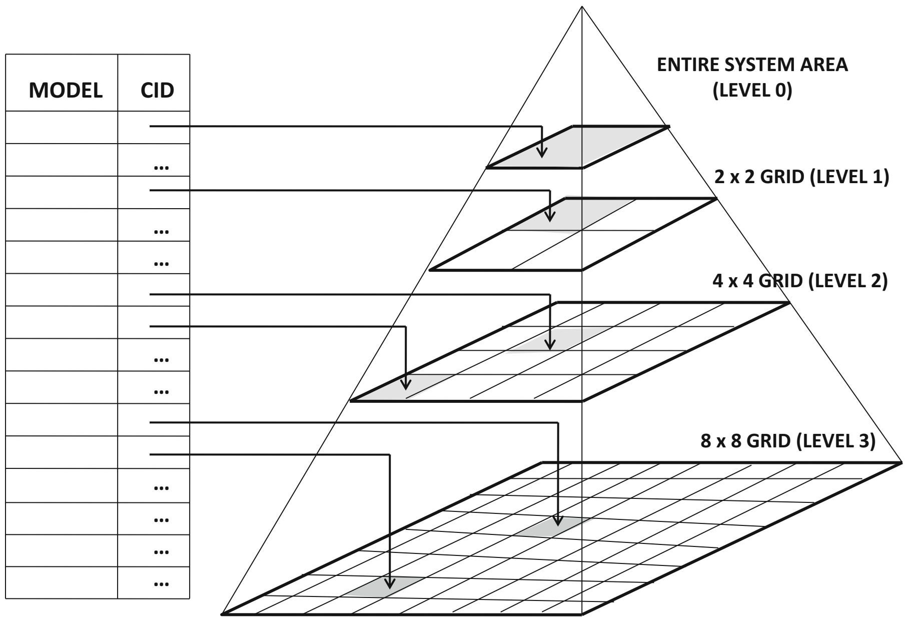

Figure 9.2: The pyramid-tree for location-aware query processing [343]

The query processing approach uses this pyramid data structure. In order to recommend items for a given user, the *LARS* approach determines the lowest level cell, which is maintained in the pyramid structure. The localized collaborative filtering model at this level is used to predict the rating. An item-based (neighborhood) collaborative filtering technique is used to perform the recommendations. Note that any conventional collaborative filtering model can be used in principle. The model does need to be incrementally updated as new ratings come in. Therefore, it is important to choose a base model that is amenable to incremental updates.

The approach is also able to support *continuous* queries where the location of a user changes with time. Note that the rate of change of the location of the user is highly application-specific. For cases in which the user-location corresponds to their address, the rate of change is very slow. However, it is possible to envision other definitions of location in which the change occurs faster over time. However, preference locality usually does not change very rapidly as a rule. In continuous queries, an initial recommendation is made as discussed above. Then, the system waits for the user location to change sufficiently, so that it crosses a cell boundary. When a cell-boundary is crossed, the lowest level cell is again used to updated the recommendation. Therefore, the last reported answer can be incrementally updated over time.

Finally, it is also possible for the users to optionally specify the geographic level of granularity at which their recommendation process should be executed. Instead of using the lowest maintained grid-cell, one might work with a user-specified level in the pyramid tree. For example, by specifying a level of 0, only the root node can be used. This results in a traditional collaborative filtering model, which does not use locality at all. This approach allows the user to specifying varying levels of geographic resolution in her queries.

9.4.2 Travel Locality

In this case, the locations are associated with items and not the users. For example, in a restaurant recommender system, the locations are associated with the restaurants. However, the users might specify their *current* location in a particular query. Clearly, it is desirable to provide responses that are close to the specified location in the query. This is achieved in *LARS* with the notion of *travel penalty.* The distance $\Delta(i, j)$ between the query location of user i and the location of item j is computed. The rating $\hat{r}_{ij}$ of user i for item j is first predicted with a traditional collaborative filtering model on the entire data. Then, the predicted rating is penalized with a function $F(\cdot)$ of $\Delta(i, j)$. The adjusted rating $\hat{r}_{ij}^{\Delta}$ is computed as follows:

$$\hat{r}_{ij}^{\Delta} = \hat{r}_{ij} - F(\Delta(i, j)) \tag{9.15}$$

Here, $F(\cdot)$ is a non-decreasing function of the distance $\Delta(i, j)$, so that the penalty is normalized to the rating scale. The exact nature of the penalty function $F(\cdot)$ is heuristic in nature. The approach in [343] uses straightforward normalization of the travel distance to the rating scale in order to define the function. If desired, it can even be assumed to be a specific function (e.g., linear function) of the distance, and the coefficients of this function can be optimally chosen with cross-validation. The choice of optimal function is an interesting research problem that can be explored in future work, because it directly affects the accuracy of the system. It is likely that the optimal choice of function is specific to the data set at hand.

9.4.3 Combined Preference and Travel Locality

It is possible to have situations in which the locations are associated with both users and items. For example, a tourist with a primary address in New Orleans might have a different restaurant preference as compared to one with a primary address in Boston, when the two of them visit Hawaii for a vacation. At the same time, the recommender system should also take into account their *transient* query location within Hawaii during the recommendation process. In this case, the methods associated with preference locality and travel locality can be combined. First, the pyramid-tree structure is used, based on the primary user location, in order to predict the ratings. Then, the transient query location is used in conjunction with the aforementioned travel penalty. The top-ranked items are then returned to the user.

9.5 Summary

Many types of time- and location-aware systems fall under the category of context-aware recommender systems. The notion of time can greatly enhance the effectiveness of recommender systems. Temporally sensitive recommender systems can use recency-based methods, context-based methods, or they may use time as a modeling variable. One of the most well-known methods of the last type is the *time-SVD++* model, which proposes a latent factor model for recommendations. Recommendation methods have also been proposed for data that are expressed as discrete sequences. For example, Web clickstreams or supermarket data contain discrete sequences of activity. These scenarios often arise in the context of implicit feedback data sets. A variety of discrete sequential methods are used to perform recommendations in such cases. Discrete Markovian models and sequential pattern-mining methods are used to perform recommendations in such cases.

Location-aware recommender systems are special cases of context-aware systems, in which the spatial location provides the context in which the recommendations are made. In location-based systems, the location could be associated with the user, the item, or both. These different forms of context lead to distinctly different methods for performing the recommendations.

9.6 Bibliographic Notes

Temporal recommendations belong to the class of context-aware recommendations, which are discussed in a generic sense in Chapter 8. A recent survey on time-aware recommender systems may be found in [130]. Some of the earliest time-weighted and decay-based collaborative filtering models are discussed in [185, 186]. A variety of decay functions are tested in [635]. The work in [249] also incorporates the time-similarity between ratings into the computation. Methods based on time-windows are proposed in [230] in which ratings from inactive intervals are essentially pruned. The work in [595] performs movie recommendations by pruning according to the year of *production*. Such an approach reduces the dimensionality of the data set because it drops a subset of the items as opposed to a method that only prunes older ratings.

Methods for extending neighborhood models with evolutionary models are discussed in [366]. Another technique that uses adaptive neighborhoods for temporal collaborative filtering is discussed in [333]. This work also showed that many of the existing recommendation algorithms did not seem to work very well on the Netflix Prize data set when using past ratings in order to predict future ratings. Time-sensitive methods for location-based recommendation are discussed in [655]. Methods for performing temporal recommendations with the use of random walk methods are discussed in [639]. An interesting class of algorithms, related to temporal collaborative filtering, is the multi-arm bandit class of algorithms in which the recommender trades off exploration vs exploitation in the recommendation space [92, 348]. This methodology is also closely related to active learning, which is discussed in Chapter 13.

A generic method for performing time-aware recommendation is to treat the time information as a discrete contextual value to create a multidimensional representation [6, 7]. Subsequent work [626] specifically addressed the temporal context within this framework. Various forms of context were tested in [61] for performing music recommendations. Although some forms of context, such as “morning” and “evening,” were shown to improve the recommendation, the greatest improvement was shown using a meaningless split such as “odd hours” and “even hours.” This might be the result of data-specific characteristics, and therefore further research is needed to understand these effects.

Realistic methods for evaluating temporal recommender systems are discussed in [335]. A recent survey [130] points to the significant importance of evaluation methods in disambiguating the contradictory findings of the recent results and also proposes a number of evaluation metrics for temporal recommender systems. The combination of multiple variables such as *season*, *timeOfDay*, and *dayOfWeek* was discussed in [337]. Other ways of combining temporal dimensions in a more complex way are discussed in [231, 471]. The work in [100] studies the use of periodic context in movie recommendations. For example, the movie recommendations during Christmas week are very different from those in the week leading to the Oscars. The use of contextual methods for improved recommendation of seasonal products is discussed in [567]. A temporal regression approach was used in this work. The *Context-Aware Movie Recommendation Challenge (CAMRA)* [515] was the platform on which the work in [100] was tested. This challenge studied contexts of various types and not just the temporal context. A contextual method [131] evaluates the effect of various time dimensions, including the hour of the day, day of the week, and the date of the rating. The work in [458] used support vector machines to model various types of context, such as time, weather, and company.

The use of time-series models been used in the context of ratings has been investigated in multiple studies [136, 435]. In these methods, a time series of user ratings is used to predict the current user interests. Time-series methods have also been used in cases of implicit feedback, where explicit ratings are not available. For example, the work in [684] encodes the Web logs as time series, and time-series techniques are used for the purpose of forecasting. The work in [266] builds several time-unaware models for different time buckets and then uses a blending approach to combine the predictions of these models. The use of factorization models in temporal recommendations was first proposed in [310]. Similar models have also been applied to the music recommendation scenario [304]. The work in [310] does not differentiate the item factors based on time. A more refined model is proposed in [293] in which differentiated item factors are learned based on the rating time-stamps. Subsequently, many matrix and tensor factorization methods were also proposed for contextual recommendation in which time is treated as a discrete contextual value [212, 294, 332, 495, 496]. These methods can be viewed as a generic implementation of the multidimensional contextual model of [7].

Discrete methods are common in the context of the Web domain, where the personalization needs to be performed with Web clickstreams [109]. A primer on finite Markov chains is provided in [296]. The sequential pattern-mining problem was defined in the context of supermarket data [37]. An overview of common algorithms for sequential pattern mining can be found in [22, 23]. In order to use these methods on Web logs, a significant amount of data preparation is required [169]. Discrete Markovian methods for predicting Web page accesses are discussed in [182]. The required background in Markov chains may be found in [265]. Sequential pattern-mining methods for predicting accesses in Web logs are discussed in [208, 440, 442, 443, 562]. The use of long repeating subsequences to predict Web page accesses is discussed in [479]. The use of path profiles for predicting Web page requests is discussed in [532]. An evaluation of various pattern-mining approaches for next-request prediction is found in [218]. A detailed discussion on Hidden Markov Models may be found in [319], and a simplified discussion on applications to data mining may be found in [22].

A significant amount of recent work has focussed on location-aware recommender systems [64, 108, 343, 447, 464, 645, 649]. Much of this work has been motivated by hardware enhancements in mobile phone technology and GPS-enabled phones. As a result, the field of mobile recommender systems [504] has gained increasing prominence. One of the earliest works [54] proposes methods to use the data from GPS-enabled mobile phones to predict user movement across various locations. Context-aware media recommendations for smart phones are discussed in [654]. A mobile advertisement recommender system with the use of collaborative filtering is proposed in [40]. Numerous tourist guide applications, such as INTRIGUE [52], GUIDE [156], MyMap [177], SPETA [213], MobiDENK [318], COMPASS [611], Archeoguide [618], and LISTEN [685], have been proposed in the literature. Some of the location-based recommender systems [633, 649] use hybrid systems in order to perform context-aware recommendations. The work by Bohnert *et al.* [89] uses the sequence of patterns in user visits to various locations to predict the next location.

It was also explored how a hybrid content-based model that captured user interests could impact the overall effectiveness of the recommender system. The addition of content provided only limited improvements. The work in [649] discussed how to handle cold-start in location-aware recommender systems by combining content and collaborative systems, and also incorporating community endorsement.

9.7 Exercises

1. Design a method for performing collaborative filtering with the Bayes model, while incorporating time decay. Refer to Chapter 3 for collaborative filtering with the Bayes algorithm.

2. Design a latent factor model that incorporates time-decay in the factorization process.

3. Implement the *time-SVD++* algorithm.

4. Suppose that you want to design an order-k Markov model on a set of actions Σ, so that $|\Sigma| = n$. Furthermore, we are assured that there is never any repetition of an action in a window of size $(k + 1)$. What is the maximum number of states and transition edges in such a model, assuming that we do not keep any states or edges with probability 0?

5. Implement a sequential pattern mining algorithm for making temporal recommendations. You have wide leeway in making appropriate design choices for your algorithm.

6. Suppose you have a large log containing sequences of actions from various users. The discussion in the chapter shows how one might perform recommendations with *item*-based rules. Show how to design an analogous approach with user-based rules. How well do you think such an approach will work in practice?

7. Discuss why an R-Tree might not be as suitable as the pyramid tree for the preference-locality technique of collaborative filtering.