3.6 Latent Factor Models

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

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

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

3.6.1 Geometric Intuition for Latent Factor Models

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

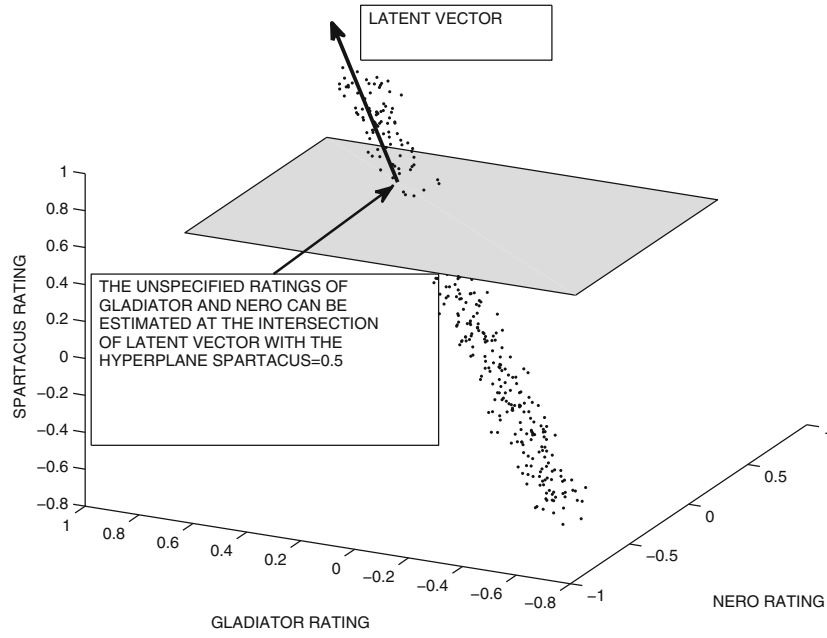


Figure 3.6: Leveraging correlation-based redundancies in missing data estimation for a user whose only specified rating is a value of 0.5 for the movie *Spartacus*

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

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

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

3.6.2 Low-Rank Intuition for Latent Factor Models

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

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

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

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

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

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

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

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

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

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

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

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

3.6.3 Basic Matrix Factorization Principles

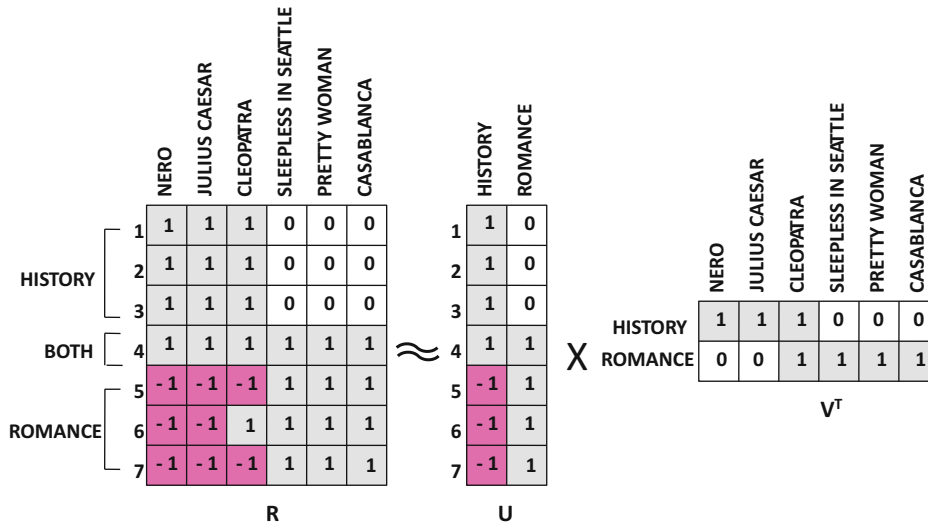
In the basic matrix factorization model, the $m \times n$ ratings matrix R is approximately factorized into an $m \times k$ matrix U and an $n \times k$ matrix V, as follows:

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

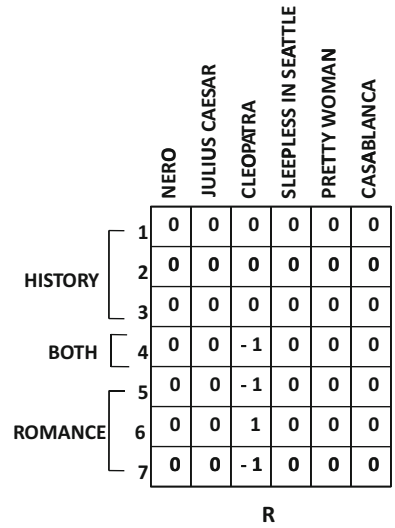
Each column of U (or V) is referred to as a latent *vector* or latent *component*, whereas each row of U (or V) is referred to as a latent *factor*. The ith row $\overline{u_i}$ of U is referred to as a *user factor*, and it contains k entries corresponding to the affinity of user i towards the k concepts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

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

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



(a) Example of rank-2 matrix factorization



(b) Residual matrix

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

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

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

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

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

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

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

3.6.4 Unconstrained Matrix Factorization

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 3.8: Gradient descent

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

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

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

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

a sparse data structure. Subsequently, the updates can be computed as follows:

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

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

3.6.4.1 Stochastic Gradient Descent

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

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

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

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

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

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

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

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

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

Figure 3.9: Stochastic gradient descent

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

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

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

3.6.4.2 Regularization

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.6.4.3 Incremental Latent Component Training

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

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

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

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

Figure 3.10: Component-wise implementation of stochastic gradient descent

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

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

3.6.4.4 Alternating Least Squares and Coordinate Descent

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

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

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

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

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

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

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

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

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

3.6.4.5 Incorporating User and Item Biases

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

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

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

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

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

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

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

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

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

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

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

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

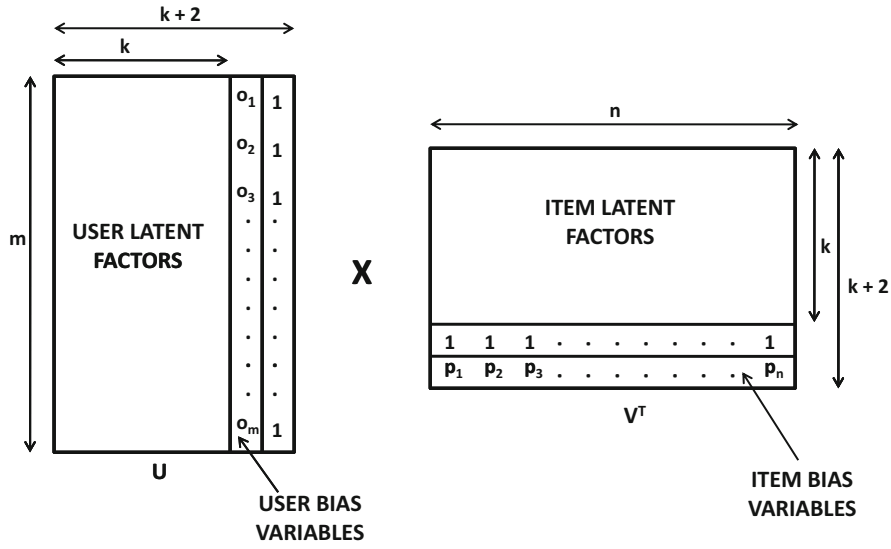


Figure 3.11: Incorporating user and item biases in the latent factor model

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

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \ldots k+2\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}) \quad \forall q \in \{1 \ldots k+2\}$$
Reset perturbed entries in $(k+2)$th column of U and $(k+1)$th column of V to 1s

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

A natural question that arises is why this formulation should perform better than unconstrained matrix factorization. The addition of constraints on the last two columns of the factor matrices should only reduce the global solution quality, because one is now optimizing over a smaller space of solutions. However, in many cases, adding such constraints biases the solution while reducing overfitting. In other words, the addition of such intuitive constraints can often improve the generalizability of the learning algorithm to *unseen* entries, even though the error over the *specified* entries may be higher. This is particularly helpful when the number of observed ratings for a user or for an item is small [473]. Bias variables add a component to the ratings that are global to either the users or the items. Such global

properties are useful when limited data is available. As a specific example, consider the case in which a user has provided ratings for only a small number (1 or 2) items. In such cases, many recommendation algorithms, such as neighborhood-based methods, will not give reliable predictions for the user. On the other hand, the (non-personalized) predictions of the item bias variables will be able to give reasonable predictions. After all, if a particular movie is a box-office hit on a global basis, then the relevant user is also more likely to appreciate it. The bias variables will also reflect this fact and incorporate it into the learning algorithm.

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

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

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

3.6.4.6 Incorporating Implicit Feedback

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

subject to:

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

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

Last two columns of Y contain only 0s

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

variables as constraints on the factorization process.

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

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

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

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

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

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

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

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

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

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

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