4 Recommender Problems

Which movie will you watch today? There are so many choices that you may have to consult a few friends for recommendations. Chances are, however, that your friends are struggling with the exact same question. In some sense we are truly lucky to have access to vast collections of movies, books, products, or daily news articles. But this "big data" comes with big challenges in terms of finding what we would really like. It is no longer possible to weed through the choices ourselves or even with the help of a few friends. We need someone a bit more "knowledgeable" to narrow down the choices for us at this scale. Such knowledgeable automated friends are called recommender systems and they already mediate much of our access to information. Whether you watch movies through Netflix or purchase products from Amazon, for example, you will be guided by recommender systems. We will use the Netflix movie recommendation problem as a running example in this chapter to illustrate how these systems actually work.

In order to recommend additional content, modern systems make use of little feedback from the user in the form of what they have liked (or disliked) in the past. There are two complementary ways to formalize this problem – *content based recommendation* and *collaborative filtering*. We will take a look at each of them individually but, ideally, they would be used in combination with each other. The key difference between them is the type of information that the machine learning algorithm is relying on. In content based recommendations, we first represent each movie in terms of a feature vector, just like before in typical classification or regression problems. For example, we may ask whether the movie is a comedy, whether Jim Carey is the lead actor, and so on, compiling all such information into a potentially high dimensional feature vector, one vector for each movie. A small number of movies that the user has seen and rated then constitutes the training set of examples and responses. We should be able to learn from this training set and be able to predict ratings for all the remaining movies that the user has not seen (the test set). Clearly, the way we construct the feature vectors will influence how good the recommendations will be. In collaborative filtering methods, on the other hand, movies are represented by how other users have rated them, dispensing entirely with any explicit features about the movies. The patterns of ratings by others do carry a lot of information. For example, if we can only find the "friend" that likes similar things, we can just borrow their ratings for other movies as well. More sophisticated ways of "borrowing" from others lies at the heart of collaborative filtering methods.

4.1 Content based recommendations

The problem of content based recommendations based on explicit movie features is very reminiscent of regression and classification problems. We will nevertheless recap the method here as it will be used as a subroutine later on for collaborative filtering. So, a bit more formally, the large set of movies in our database, m of them in total, are represented by vectors $x^{(1)}, \ldots, x^{(m)}$, where $x^{(i)} \in \mathbb{R}^d$. How the features are extracted from the movies, including the overall dimension d, may have substantial impact on our ability to predict user preferences. Good features are those that partition the movies into sets that people might conceivably assign different preferences to. For example, genre might be a good feature but the first letter of the director's last name would not be. It matters, in addition, whether the information in the features appears in a form that the regression method can make use of it. While properly engineering and tailoring the features for the method (or the other way around) is important, we will assume that the vectors are given and reasonably useful for linear predictions.

Do you rate the movies you see? Our user has rated at least some of them. Let $y^{(i)}$ be a star rating (1-5 scale) for movie i represented by feature vector $x^{(i)}$. The ratings are typically integer valued, i.e., $1, 2, \ldots, 5$ stars, but we will treat them as real numbers for simplicity. The information we have about each user (say user a) is then just the training set of rated movies $S_a = \{(x^{(i)}, y^{(i)}), i \in D_a\}$, where D_a is the index set of movies user a has explicitly rated so far. Our method takes the typically small training set (a few tens of rated movies) and turns it into predicted ratings for all the remaining movies. The movies with the highest predicted ratings could then be presented to the user as possible movies to watch. Of course, this is a very idealized interaction with the user. In practice, for example, we would need to enforce some diversity in the proposed set (preferring action movies does not mean that those are the only ones you ever watch).

We will try to solve the rating problem as a linear regression problem. For each movie $x^{(i)}$, we will predict a real valued rating $\hat{y}^{(i)} = \theta \cdot x^{(i)} = \sum_{j=1}^{d} \theta_j x_j^{(i)}$ where θ_j represents the adjustable "weight" that we place on the j^{th} feature coordinate. Note that, unlike in a typical regression formulation, we have omitted the offset parameter θ_0 for simplicity of exposition.

We estimate parameters θ by minimizing the squared error between the observed and predicted ratings on the training set while at the same time trying to keep the parameters small (regularization). More formally, we minimize

$$\begin{aligned} J(\theta) &= \sum_{i \in D_a} \big(y^{(i)} - \theta \cdot x^{(i)}\big)^2/2 + \frac{\lambda}{2}\|\theta\|^2 && (54) \\ &= \sum_{i \in D_a} \big(y^{(i)} - \sum_{j=1}^{d} \theta_j x_j^{(i)}\big)^2/2 + \frac{\lambda}{2}\|\theta\|^2 && (55) \end{aligned}$$

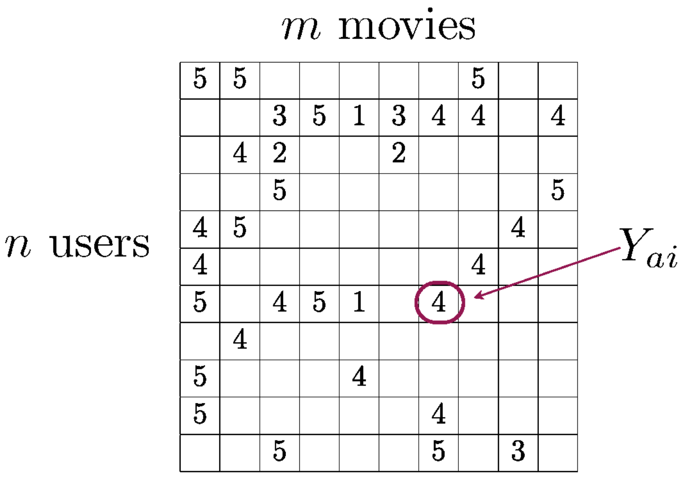
with respect to the vector of parameters θ. There are many ways to solve this optimization problem as we have already seen. We could use a simple stochastic gradient descent method or obtain the solution in closed-form. In the latter case, we set the derivative of the objective $J(\theta)$ with respect to each parameter θ_k to zero, and obtain d linear equations that constrain the parameters

$$\begin{aligned} \frac{d}{d\theta_k} J(\theta) &= -\sum_{i \in D_a} \big(y^{(i)} - \sum_{j=1}^{d} \theta_j x_j^{(i)}\big) x_k^{(i)} + \lambda \theta_k && (56) \\ &= -\sum_{i \in D_a} y^{(i)} x_k^{(i)} + \sum_{j=1}^{d} \theta_j \sum_{i \in D_a} x_j^{(i)} x_k^{(i)} + \lambda \theta_k && (57) \\ &= -\sum_{i \in D_a} y^{(i)} x_k^{(i)} + \sum_{j=1}^{d} \sum_{i \in D_a} x_k^{(i)} x_j^{(i)} \theta_j + \lambda \theta_k && (58) \end{aligned}$$

where $k = 1, \ldots, d$. It is easier to solve these in a matrix form. To this end, define a vector $b = \sum_{i \in D_a} y^{(i)} x^{(i)}$ and matrix $A = (\sum_{i \in D_a} x^{(i)} {x^{(i)}}^T) + \lambda I$ so that the linear equations above can be written as $-b_k + \sum_{j=1}^{d} A_{kj}\theta_j = 0$ for $k = 1, \ldots, d$. The resulting matrix form $-b + A\theta = 0$ gives $\hat{\theta} = A^{-1}b$. Recall that A is guaranteed to be invertible provided that $\lambda > 0$[4]

How well will this method work? There are several factors that affect this. The accuracy will clearly depend on the quality of features we add, i.e., whether the features represent natural distinctions along which human preferences vary. E.g., genre, director, lead actor/actress, and so on, are likely to be all good features to add. The more features we add, the more intricate preferences we can model by adjusting the parameters. However, creating and using high dimensional feature vectors for movies also means that we must have more ratings available during training in order to be able to distinguish good regression methods from a large set of other possibilities. It is not atypical to have just a few tens of rated movies. This is a problem, but it could be alleviated by "borrowing" data from other users. We will do that next with collaborative filtering. Another subtle issue is that there may be features about the movies that are not even possible to extract without considering other people's reactions to the content. For example, someone may strongly prefer movies with happy endings. While it is hard to automatically extract such a feature, it may be easy to recognize that your preferences align with such a person. Collaborative filtering tries to relate users based on their ratings and has a chance to leverage it.

4.2 Collaborative Filtering



5	5						5		
		3	5	1	3	4	4		4
	4	2			2				
		5							5
4	5							4	
4							4		
5		4	5	1		4			
	4								
5				4					
5						4			
		5				5		3	

Figure 9: Sparse data matrix Y for a collaborative filtering problem with users and movies. Empty spaces represent so far unseen (unrated) movies. The goal is to fill in these missing ratings.

How could we recommend movies to you if we have no information about the movies

[4] $(\sum_{i \in D_a} x^{(i)} {x^{(i)}}^T)$ is positive semi-definite by construction and λI is positive definite. Thus all of A's eigenvalues are strictly positive and the matrix is invertible.

beyond their arbitrary ids? Content based recommendations (as discussed above) are indeed doomed to failure in this setting. For example, if you like movie 243, what could we say about your preference for movie 4053? Not much. No features, no basis for prediction beyond your average rating, i.e., whether you tend to give high or low ratings overall. But we can solve this problem quite well if we step up and consider all the users at once rather than individually. This is known as collaborative filtering. The key underlying idea is that we can somehow leverage experience of other users. There are, of course, many ways to achieve this. For example, it should be easy to find other users who align well with you based on a small number of movies that you have rated. The more users we have, the easier this will be. Some of them will have rated also other movies that you have not. We can then use those ratings as predictions for you, driven by your similarity to them as users. As a collaborative filtering method, this is known as nearest neighbor prediction. Alternatively, we can try to learn explicit feature vectors for movies guided by people's responses. Movies that users rate similarly would end up with similar feature vectors in this approach. Once we have such movie feature vectors, it would suffice to predict ratings separately for each user, just as we did in content based recommendations. The key difference is that now the feature vectors are "behaviorally" driven and encode only features that impact ratings. In a matrix factorization approach, the recommendation problem is solved iteratively by alternating between solving linear regression for movie features, when user representation is fixed, and solving it for users, when movie representation is fixed.

Let's make the problem a bit more formal. We have n users and m movies along with ratings for some of the movies as shown in Figure 9. Both n and m are typically quite large. For example, in the Netflix challenge problem given to the research community there were over $400,000$ users and over $17,000$ movies. Only about one percent of the matrix entries had known ratings so Figure 9 is a bit misleading (the matrix would look much emptier in reality). We will use Y_{ai} to denote the rating that user $a \in \{1, \ldots, n\}$ has given to movie $i \in \{1, \ldots, m\}$. For clarity, we adopt different letters for indexing users $(a, b, c, \ldots)$ and movies $(i, j, k, \ldots)$. The rating Y_{ai} could be in $\{1, \ldots, 5\}$ (5 star rating) as in Figure 9, or $Y_{ai} \in \{-2, -1, 0, 1, 2\}$ if we subtract 3 from all the ratings, reducing the need for the offset term. We omit any further consideration of the rating scale, however, and instead simply treat the matrix entries as real numbers, i.e., $Y_{ai} \in \mathcal{R}$.

4.2.1 Nearest-neighbor prediction

We can understand the nearest neighbor method by focusing on the problem of predicting a single entry Y_{ai}. In other words, we assume that user a has not yet rated movie i and we must somehow fill in this value. It is important that there are at least some other users who have seen movie i and provided a rating for it. Indeed, collaborative filtering is powerless if faced with an entirely new movie no-one has seen.

In the nearest neighbor approach, we gauge how similar user a is to those users who have already seen movie i, and combine their ratings. Note that the set of users whose experience we "borrow" for predicting Y_{ai} changes based on the target movie i as well as user a. Now, to develop this idea further, we need to quantitatively measure how similar any two users are. A popular way to do this is via sample correlation where we focus on the subset of movies that both users have seen, and ask how these ratings co-vary (with normalization). Let $CR(a,b)$ denote the set of movies a and b have both rated. $CR(a,b)$ does not include movie i or other movies that a has yet to see. Following this notation, the sample correlation is defined as

$$\mathrm{sim}(a,b) = \mathrm{corr}(a,b) = \frac{\sum_{j \in CR(a,b)} (Y_{aj} - \tilde{Y}_a)(Y_{bj} - \tilde{Y}_b)}{\sqrt{\sum_{j \in CR(a,b)} (Y_{aj} - \tilde{Y}_a)^2} \sqrt{\sum_{j \in CR(a,b)} (Y_{bj} - \tilde{Y}_b)^2}} \tag{59}$$

where the mean rating $\tilde{Y}_a = (1/|CR(a,b)|) \sum_{j \in CR(a,b)} Y_{aj}$ is evaluated based on the movies the two users a and b have in common. It would potentially change when comparing a and c, for example. This subtlety is not visible in the notation as it would become cluttered otherwise. Note that $\mathrm{sim}(a,b) \in [-1,1]$ where -1 means that users are dissimilar, while value 1 indicates that their ratings vary identically (up to an overall scale) around the mean. So, for example, let's say user a rates movies j and k as 5 and 3, respectively, while user b assigns 4 and 2 to the same movies. If these are the only movies they have in common, $CR(a,b) = \{j,k\}$ and $\mathrm{sim}(a,b) = 1$ since the overall mean rating for the common movies does not matter in the comparison.

Once we have a measure of similarity between any two users, we can exploit it to predict Y_{ai}. Clearly, it would be a good idea to emphasize users who are highly similar to a while downweighting others. But we must also restrict these assessments to the subset of users who have actually seen i. To this end, let $KNN(a,i)$ be the top K most similar users to a who have also rated i. K here is an adjustable parameter (integer) we can set to improve the method. For now, let's fix it to a reasonable value such as 10 so as to build some statistical support (including more data from others) while at the same time avoiding users who are no longer reasonably similar. Now, the actual formula for predicting Y_{ai} is composed of two parts. The first part is simply the average rating $\bar{Y}_a$ computed from all the available ratings for a. This would be our prediction in the absence of any other users. The second part is a weighted sum of correction terms originating from other similar users. More formally,

$$\hat{Y}_{ai} = \bar{Y}_a + \frac{\sum_{b \in KNN(a,i)} \mathrm{sim}(a,b)(Y_{bi} - \bar{Y}_b)}{\sum_{b \in KNN(a,i)} |\mathrm{sim}(a,b)|} \tag{60}$$

Let's disentangle the terms a bit. $(Y_{bi} - \bar{Y}_b)$ measures how much user b's rating for i deviates from their overall mean $\bar{Y}_b$. Note that, in contrast to $\tilde{Y}_b$ appearing in the similarity computation, $\bar{Y}_b$ is based on all the ratings from b. We deal with deviations from the mean as corrections so as to be indifferent towards varying rating styles of users (some users tend to give high ratings to every movie while others are more negative). This is taken out by considering deviations from the mean. Now, the correction terms in the formula are weighted by similarity to a and normalized. The normalization is necessary to ensure that we effectively select (in a weighted manner) whose correction we adopt. For example, suppose $KNN(a,i)$ includes only one user and a did in fact rate i. So, $KNN(a,i) = \{a\}$ and Y_{ai} is available. In this case, our formula would give

$$\hat{Y}_{ai} = \bar{Y}_a + \frac{\mathrm{sim}(a,a)(Y_{ai} - \bar{Y}_a)}{|\mathrm{sim}(a,a)|} = \bar{Y}_a + \frac{1\,(Y_{ai} - \bar{Y}_a)}{|1|} = Y_{ai} \tag{61}$$

as it clearly should.

The nearest neighbor approach to collaborative filtering has many strengths. It is conceptually simple, relatively easy to implement, typically involves few adjustable parameters (here only K), yet offers many avenues for improvement (e.g., the notion of similarity). It works quite well for users who have stereotypical preferences in the sense that there are groups of others who are very close as evidenced by ratings. But the method degenerates for users with mixed interests relative to others, i.e., when there is no other with the same pattern across all the movies even though strong similarities hold for subsets. For example, a person who dislikes clowns (coulrophobic) might have a pattern of rating very similar to a group of others except for movies that involve clowns. The simple nearest neighbor method would not recognize that this subset of movies for this user should be treated differently, continuing to insist on overall comparisons and borrowing ratings from non-coulrophobic users. Since all of us are idiosyncratic in many ways, the method faces an inherent performance limitation. The matrix factorization method discussed next avoids this problem in part by decoupling the modeling of how preferences vary across users from how exactly to orient each user in this space.

4.2.2 Matrix factorization

We can think of collaborative filtering as a matrix problem rather than a problem involving individual users. We are given values for a small subset of matrix entries, say $\{Y_{ai}, (a,i) \in D\}$, where D is an index set of observations, and the goal is to fill in the remaining entries in the matrix. This is the setting portrayed in Figure 9. Put another way, our objective is to uncover the full underlying rating matrix Y based on observing (possibly with noise) only some of its entries. Thus the object for learning is a matrix rather a typical vector of parameters. We use X to denote the matrix we entertain and learn to keep it separate from the underlying target matrix Y (known only via observations). As in any learning problem, it will be quite necessary to constrain the set of possible matrices we can learn; otherwise observing just a few entries would in no way help determine remaining values. It is through this process of "squeezing" the set of possible matrices X that gives us a chance to predict well and, as a byproduct, learn useful feature vectors for movies (as well as users).

It is instructive to see first how things fail in the absence of any constraints on X. The learning problem here is a standard regression problem. We minimize the squared error between observations and predictions while keeping the parameters (here matrix entries) small. Formally, we minimize

$$\sum_{ai \in D} (Y_{ai} - X_{ai})^2/2 + \frac{\lambda}{2} \sum_{ai} X_{ai}^2 \tag{62}$$

with respect to the full matrix X. What is the solution $\hat{X}$? Note that there's nothing that ties the entries of X together in the objective so they can be solved independently of each other. Entries $X_{ai}, (a,i) \in D$ are guided by both observed values Y_{ai} and the regularization term X_{ai}^2. In contrast, $X_{ai}, (a,i) \notin D$ only see the regularization term and will therefore be set to zero. In summary,

$$\hat{X}_{ai} = \left\{ \begin{array}{ll} \frac{1}{1+\lambda} Y_{ai}, & (a,i) \in D \\ 0, & (a,i) \notin D \end{array} \right. \tag{63}$$

$$\begin{bmatrix} 5 & 7 \\ 10 & 14 \\ 30 & 42 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} \times \begin{bmatrix} 5 & 7 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \\ 3 \end{bmatrix} \times \begin{bmatrix} 10 & 14 \end{bmatrix}$$

Figure 10: An example rank-1 matrix and its two possible factorizations. Note that the factorizations differ only by an overall scaling of its component vectors.

It's not exactly useful to predict all zeros for unseen movies regardless of the observations. In order to remedy the situation, we will have to introduce fewer adjustable parameters than there are entries in the matrix.

Low-rank factorization The typical way to control the effective number of parameters in a matrix is to the control its rank, i.e., how it can be written as a product of two smaller matrices. Let's commence with the simplest (most constrained) setting, where X has rank one. In this case, by definition, it must be possible to write it as an outer product of two vectors that we call U ($n \times 1$ vector) and V ($m \times 1$ vector). We will adopt capital letters for these vectors as they will be later generalized to matrices for ranks greater than one. Now, if X has rank 1, it can be written as $X = UV^T$ for some vectors U and V. Note that U and V are not unique. We can multiply U by some non-zero constant β and V by the inverse $1/\beta$ and get back the same matrix: $X = UV^T = (\beta U)(V/\beta)^T$. The factorization of a rank 1 matrix into two vectors is illustrated in Figure 10.

We will use $X = UV^T$ as the set of matrices to fit to the data where vectors $U = [u^{(1)}, \ldots, u^{(n)}]^T$ and $V = [v^{(1)}, \ldots, v^{(m)}]^T$ are adjustable parameters. Here $u^{(a)}$, $a = 1, \ldots, n$ are just scalars, as are $v^{(i)}$, $i = 1, \ldots, m$. Notationally, we are again gearing up to generalizing $u^{(a)}$ and $v^{(i)}$ to vectors but we are not there yet. Once we have U and V, we would use $X_{ai} = [UV^T]_{ai} = u^{(a)}v^{(i)}$ (simple product of scalars) to predict the rating that user a assigns to movie i. This is quite restrictive. Indeed, the set of matrices $X = UV^T$ obtained by varying U and V has only $n + m - 1$ degrees of freedom (we loose one degree of freedom to the overall scaling exchange between U and V that leaves matrix X intact). Since the number of parameters is substantially smaller than nm in the full matrix, we have a chance to actually fit these parameters based on the observed entries, and obtain an approximation to the underlying rating matrix.

Let's understand a bit further what rank 1 matrices can or cannot do. We can interpret the scalars $v^{(i)}$, $i = 1, \ldots, m$, specifying V as scalar features associated with movies. For example, $v^{(i)}$ may represent a measure of popularity of movie i. In this view $u^{(a)}$ is the regression coefficient associated with the feature, controlling how much user a likes or dislikes popular movies. But the user cannot vary this preference from one movie to another. Instead, all that they can do is to specify an overall scalar $u^{(a)}$ that measures the degree to which their movie preferences are aligned with $[v^{(1)}, \ldots, v^{(m)}]$. All users gauge their preferences relative to the same $[v^{(1)}, \ldots, v^{(m)}]$, varying only the overall scaling of this vector. Restrictive indeed.

We can generalize the idea to rank k matrices. In this case, we can still write $X = UV^T$ but now U and V are matrices rather than vectors. Specifically, if X has rank at most k then $U = [u^{(1)}, \ldots, u^{(n)}]^T$ is an $n \times d$ matrix and $V = [v^{(1)}, \ldots, v^{(m)}]^T$ is $m \times d$, where $d \leq \min\{n, m\}$. We characterize each movie in terms of k features, i.e., as a vector $v^{(i)} \in \mathbb{R}^d$, while each user is represented by a vector of regression coefficients (features)

$u^{(a)} \in \mathbb{R}^d$. Both of these vectors must be learned from observations. The predicted rating is $X_{ai} = [UV^T]_{ai} = u^{(a)} \cdot v^{(i)} = \sum_{j=1}^d u_j^{(a)} v_j^{(i)}$ (dot product between vectors) which is high when the movie (as a vector) aligns well with the user (as a vector). The set of rank k matrices $X = UV^T$ is reasonably expressive already for small values of k. The number of degrees of freedom (independent parameters) is $nd + md - d^2$ where the scaling degree of freedom discussed in the context of rank 1 matrices now appears as any invertible $d \times d$ matrix B since $UV^T = (UB)(VB^{-T})^T$. Clearly, there are many possible Us and Vs that lead to the same predictions $X = UV^T$. Suppose $n > m$ and $k = m$. Then the number of adjustable parameters in UV^T is $nm + m^2 - m^2 = nm$, the same as in the full matrix. Indeed, when $k = \min\{n, m\}$ the factorization $X = UV^T$ imposes no constraints on X at all (the matrix has full rank). In collaborative filtering only small values of k are relevant. This is known as the low rank assumption.

Learning Low-Rank Factorizations The difficulty in estimating U and V is that neither matrix is known ahead of time. A good feature vector for a movie depends on how users would react to them (as vectors), and vice versa. Both types of vectors must be learned from observed ratings. In other words, our goal is to find $X = UV^T$ for small k such that $Y_{ai} \approx X_{ai} = [UV^T]_{ai} = u^{(a)} \cdot v^{(i)}, \quad (a, i) \in D$. The estimation problem can be formulated again as a least squares regression problem with regularization. Formally, we minimize

$$\begin{aligned} J(U,V) &= \sum_{(a,i)\in D} (Y_{ai} - [UV^T]_{ai})^2/2 + \frac{\lambda}{2}\sum_{a=1}^{n}\sum_{j=1}^{k} U_{aj}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{j=1}^{k} V_{ij}^2 && (64) \\ &= \sum_{(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\sum_{a=1}^{n} \|u^{(a)}\|^2 + \frac{\lambda}{2}\sum_{i=1}^{m} \|v^{(i)}\|^2 && (65) \end{aligned}$$

with respect to matrices U and V, or, equivalently, with respect to feature vectors $u^{(a)}$, $a = 1, \ldots, n$, and $v^{(i)}$, $i = 1, \ldots, m$. The smaller rank k we choose, the fewer adjustable parameters we have. The rank k and regularization parameter λ together constrain the resulting predictions $\hat{X} = \hat{U}\hat{V}^T$.

Is there a simple algorithm for minimizing $J(U, V)$? Yes, we can solve it in an alternating fashion. If the movie feature vectors $v^{(i)}$, $i = 1, \ldots, m$ were given to us, we could minimize $J(U, V)$ (solve the recommendation problem) separately for each user, just as before, finding parameters $u^{(a)}$ independently from other users. Indeed, the only part of $J(U, V)$ that depends on the user vector $u^{(a)}$ is

$$\sum_{i:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|u^{(a)}\|^2 \qquad (66)$$

where $\{i : (a, i) \in D\}$ is the set of movies that user a has rated. This is a standard least squares regression problem (without offset) for solving $u^{(a)}$ when $v^{(i)}$, $i = 1, \ldots, m$ are fixed. Note that other users do influence how $u^{(a)}$ is set but this influence goes through the movie feature vectors.

Once we have estimated $u^{(a)}$ for all the users $a = 1, \ldots, n$, we can instead fix these vectors, and solve for the movie vectors. The objective $J(U, V)$ is entirely symmetric in terms of $u^{(a)}$ and $v^{(i)}$, so solving $v^{(i)}$ also reduces to a regression problem. Indeed, the only part of $J(U,V)$ that depends on $v^{(i)}$ is

$$\sum_{a:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|v^{(i)}\|^2 \tag{67}$$

where $\{a : (a,i) \in D\}$ is the set of users who have rated movie i. This is again a regression problem without offset. We know how to solve it since $u^{(a)}$, $a = 1, \ldots, n$ are fixed. Note that the movie feature vector $v^{(i)}$ is adjusted to help predict ratings for all the users who rated the movie. This is how they are tailored to user patterns of ratings.

Taken together, we have an alternating minimization algorithm for finding the latent feature vectors, fixing one set and solving for the other. All we need in addition is a starting point. The complete algorithm is given by

(0) Initialize the movie feature vectors $v^{(1)}, \ldots, v^{(m)}$ (e.g., randomly)

(1) Fix $v^{(1)}, \ldots, v^{(m)}$ and separately solve for each $u^{(a)}$, $a = 1, \ldots, n$ by minimizing

$$\sum_{i:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|u^{(a)}\|^2 \tag{68}$$

(2) Fix $u^{(1)}, \ldots, u^{(n)}$ and separately solve for each $v^{(i)}$, $i = 1, \ldots, m$, by minimizing

$$\sum_{a:(a,i)\in D} (Y_{ai} - u^{(a)} \cdot v^{(i)})^2/2 + \frac{\lambda}{2}\|v^{(i)}\|^2 \tag{69}$$

Each minimization step in our alternating algorithm utilizes old parameter values for the other set that is kept fixed. It is therefore necessary to iterate over steps (1) and (2) in order to arrive at a good solution. The resulting values for U and V can depend quite a bit on the initial setting of the movie vectors. For example, if we initialize $v^{(i)} = 0$ (vector), $i = 1, \ldots, m$, then step (1) of the algorithm produces user vectors that are also all zero. This is because $u^{(a)} \cdot v^{(i)} = 0$ regardless of $u^{(a)}$. In the absence of any guidance from the error terms, the regularization term drives the solution to zero. The same would happen in step (2) to the new movie vectors since user vectors are now zero, and so on. Can you figure out what would happen if we initialized the movie vectors to be all the same but non-zero? (left as an exercise). The issue with initialization here is that while each step, (1) or (2), offers a unique solution, the overall minimization problem is not jointly convex (bowl shaped) with respect to (U,V). There are locally optimal solutions. The selection of where we end up is based on the initialization (the rest of the algorithm is deterministic). It is therefore a good practice to run the algorithm a few times with randomly initialized movie vectors and either select the best one (the one that achieves the lowest value of $J(U,V)$) or combine the solutions from different runs. A theoretically better algorithm would use the fact that there are many Us and Vs that result in the same $X = UV^T$ and therefore cast the problem directly in terms of X. However, for computational reasons, fully representing X for realistic problems is infeasible. The alternating algorithm we have presented is often sufficient and widely used in practice.

Alternating minimization example For concreteness, let's see how the alternating minimization algorithm works when $k = 1$, i.e., when we are looking for a rank-1 solution. Assume that the observed ratings are given in a 2×3 matrix Y (2 users, 3 movies)

$$Y = \begin{bmatrix} 5 & ? & 7 \\ 1 & 2 & ? \end{bmatrix} \tag{70}$$

where the question marks indicate missing ratings. Our goal is to find U and V such that $X = UV^T$ closely approximates the observed ratings in Y. We start by initializing the movie features $V = [v^{(1)}, v^{(2)}, v^{(3)}]^T$ where $v^{(i)}$, $i = 1, 2, 3$, are scalars since $d = 1$. In other words, V is just a 3×1 vector which we set as $[2, 7, 8]^T$. Given this initialization, our predicted rating matrix $X = UV^T$, as a function of $U = [u^{(1)}, u^{(2)}]^T$, where $u^{(a)}$, $a = 1, 2$, are scalars, becomes

$$UV^T = \begin{bmatrix} 2u^{(1)} & 7u^{(1)} & 8u^{(1)} \\ 2u^{(2)} & 7u^{(2)} & 8u^{(2)} \end{bmatrix} \tag{71}$$

We are interested in finding $u^{(1)}$ and $u^{(2)}$ that best approximates the ratings (step (1) of the algorithm). For instance, for user 1, the observed ratings 5 and 7 are compared against predictions $2u^{(1)}$ and $8u^{(1)}$. The combined loss and regularizer for this user in step (1) of the algorithm is

$$J_1(u^{(1)}) = \frac{(5 - 2u^{(1)})^2}{2} + \frac{(7 - 8u^{(1)})^2}{2} + \frac{\lambda}{2}(u^{(1)})^2 \tag{72}$$

To minimize this loss, we differentiate it with respect to $u^{(1)}$ and equate it to zero.

$$\frac{dJ_1(u^{(1)})}{du^{(1)}} = -66 + (68 + \lambda)u^{(1)} = 0 \tag{73}$$

resulting in $u^{(1)} = \frac{66}{\lambda+68}$. We can similarly find $u^{(2)} = \frac{16}{\lambda+53}$.

If we set $\lambda = 1$, then the current estimate of U is $[66/69, 16/54]^T$. We will next estimate V based on this value of U. Now, writing $X = UV^T$ as a function of $V = [v^{(1)}, v^{(2)}, v^{(3)}]^T$, we get

$$UV^T = \begin{bmatrix} \frac{66}{69}v^{(1)} & \frac{66}{69}v^{(2)} & \frac{66}{69}v^{(3)} \\ \frac{16}{54}v^{(1)} & \frac{16}{54}v^{(2)} & \frac{16}{54}v^{(3)} \end{bmatrix} \tag{74}$$

As before, in step (2) of the algorithm, we separately solve for $v^{(1)}$, $v^{(2)}$, and $v^{(3)}$. The combined loss and regularizer for the first movie is now

$$\frac{(5 - \frac{66}{69}v^{(1)})^2}{2} + \frac{(1 - \frac{16}{54}v^{(1)})^2}{2} + \frac{\lambda}{2}(v^{(1)})^2 \tag{75}$$

We would again differentiate this objective with respect to $v^{(1)}$ and equate it to zero to solve for the new updated value for $v^{(1)}$. The remaining $v^{(2)}$ and $v^{(3)}$ are obtained analogously.

4.3 Exercises

Exercise (4.1) Suppose we are trying to predict how a user would rate songs. Each song x is represented by a feature vector $\phi(x)$ with coordinates that pertain to acoustic properties of the song. The ratings are binary valued $y \in \{0, 1\}$ ("need earplugs" or "more like this"). Given n already rated songs, we can use regularized linear regression to predict the binary ratings. The training criterion is

$$J(\theta) = \frac{\lambda}{2}\|\theta\|^2 + \frac{1}{n}\sum_{t=1}^{n}(y^{(t)} - \theta \cdot \phi(x^{(t)}))^2/2 \tag{76}$$

(a) If $\hat{\theta}$ is the optimal setting of the parameters with respect to the above criterion, which of the following conditions must be true:

A ; $\lambda\hat{\theta} - \frac{1}{n}\sum_{t=1}^{n}(y^{(t)} - \hat{\theta} \cdot \phi(x^{(t)}))\phi(x^{(t)}) = 0$

B ; $J(\hat{\theta}) \geq J(\theta)$, for all $\theta \in \mathcal{R}^d$

C ; If we increase λ, the resulting $\|\hat{\theta}\|$ will decrease

D ; If we add features to $\phi(x)$ (whatever they may be), the resulting squared training error will NOT increase

(b) Once we have the estimated parameters $\hat{\theta}$, we must decide how to predict ratings for new songs. Write down an expression that shows which label we should choose for a new song x

(c) What will happen to the predicted ratings if we increase the regularization parameter λ during training?

Exercise (4.2) In this question, we apply low-rank factorization algorithm to the matrix of observations Y:

$$Y = \begin{bmatrix} 5 & ? & 7 \\ ? & 2 & ? \\ ? & 1 & 4 \\ 4 & ? & ? \\ ? & 3 & 6 \end{bmatrix} \tag{77}$$

Assume that $k = 1$ and $\lambda = 1$. After one iteration of the algorithm, we obtain the following values for U and V: $U = [6, 2, 3, 3, 5]^T$, and $V = [4, 1, 5]^T$.

1. evaluate matrix X of predicted ranks.
2. evaluate the squared error and the regularization terms for the current estimate X.
3. evaluate the new estimate of $u^{(1)}$ if Vs are kept fixed.

Exercise (4.3) Suppose we try to estimate a simple rank-1 model $X_{ai} = u_a v_i$ where $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ are just scalar parameters associated with users u_a and items v_i, respectively. Please select which (if any) of the following 2x2 matrices cannot be reproduced by the rank-1 model.