

	movies									
$y_{ij}$	1	2	3	4	5	6	7	8	9	10
1	2	1		4				5		
2	5	4						1		3
3		3	5		2					
4		4		1	3			5		
5			2			1				4
6	1				5	5		4		
7		2		5						
8	3	3	1		5		2			1
9	3		1			2	3			
10	4		5	1			3			
11		3				3			5	
12	2		1	1						
13		5		2		4	4			
14	1	3	1	5		4	5			
15	1	2		4			5			

Figure 1: Data matrix  $Y$  for a collaborative filtering problem with users and movies.

## Recommender problems and collaborative filtering

Recommender problems are ubiquitous. We are interested recommending books, movies, or other products to users based on what we know about their history, what they do and don't like, and so on. Not surprisingly, recommender systems are in active use in companies such as Amazon.com (books and other products) or Netflix (movies). We will use the Netflix movie recommendation problem as a guiding example to illustrate these type of problems and how we can solve them.

Suppose we have  $n$  users and  $m$  movies. We assume that each user has already rated some movies, i.e., assigned 1-5 stars to movies they have seen. These ratings can be collected into a data matrix  $Y$  as in Figure 1. We will use  $y_{ij} \in \{1, \dots, 5\}$  to denote the rating that user  $i \in \{1, \dots, n\}$  has given to movie  $j \in \{1, \dots, m\}$ . Our task is to complete the data matrix, i.e., predict ratings for all the unobserved entries of the matrix. Recommender problems that can be reduced to matrix (or tensor) completion problems are known as collaborative filtering problems. What makes this problem interesting from a learning point of view is that there are typically many users (e.g., 400,000) and movies (e.g., 17,000) but very few available ratings (e.g., the matrix may be only 1% filled). The key idea behind collaborative filtering (CF) solutions is that we can borrow experience from other similar users to better predict the ratings for any particular user. The implication of this is that we should solve the problem as a matrix problem rather than as multiple independent prediction problems (one for each user).

We will start by solving a bit simpler problem – a single user movie rating problem – that will turn into a useful subroutine for our collaborative filtering algorithm. Suppose we have

translated each movie  $j$  into a feature vector  $\underline{x}_j \in \mathcal{R}^d$ ,  $j \in \{1, \dots, m\}$ . We will discuss later on how to get around specifying such feature vectors. For now, the coordinates of  $\underline{x}_j$  may be thought of as indicating presense/absence of certain actors or actresses, who directed the movie, genre, when the movie was released, whether it is a sequel, and so on. Our task is to take the few observed ratings for any user  $i$  and turn the them into predicted ratings for all the remaining movies. Note that the dimension of the feature vectors serves as a key complexity measure.

**Ordinal regression problem.** Our training set is  $(\underline{x}_j, y_{ij})$ ,  $j \in M_i$ , where  $M_i$  is a subset of  $\{1, \dots, m\}$  indicating the movie ratings that we already have for user  $i$ . The task is then to predict  $y_{ij}$  for  $j \in M \setminus M_i$ . This looks like a simple multi-way classification problem where the labels correspond to the stars that the user assigns to each movie. But it is not a multi-way classification problem. The issue is that the labels we have here, the stars, have structure. Indeed, they correspond to an ordinal scale where  $1 < 2 < \dots < 5$  has meaning. In classification tasks the class labels are just symbols and have no natural ordering. We have to change how we train the predictor so as to respect the ordinal scale. The problem is called an ordinal regression problem.

We can turn the ordinal regression problem into a binary classification problem just as in multi-way classification context but the component classifiers in our case will be mutually constrained. To this end, we will associate any given rating  $y$  with a set of 4 binary labels (always one less than the number of ratings). These binary labels  $s_r$ ,  $r = 1, \dots, 4$ , correspond to binary subtasks such as  $y = 1$  vs  $y > 1$ ,  $y \leq 2$  vs  $y > 2$ , and so on. More formally,  $s_r = -1$  if  $y \leq r$  and 1 otherwise. We will use four linear component classifiers for these binary tasks

$$\hat{s}_r = \text{sign}(\underline{\theta} \cdot \underline{x} - b_r), \quad r = 1, \dots, 4 \quad (1)$$

We have dropped the user index  $i$  to simplify the notation. Note that all of these classifiers have the same  $\underline{\theta}$ . We also require that  $b_1 \leq b_2 \leq \dots \leq b_4$ . These constraints ensure that the outputs of component classifiers are always consistent with one possible rating. In other words, if  $\underline{\theta} \cdot \underline{x} - b_1 < 0$  suggesting that  $y = 1$ , then necessarily  $\underline{\theta} \cdot \underline{x} - b_r < 0$  for  $r = 2, \dots, 4$  as well.

**Perceptron algorithm for ordinal regression.** We can adapt the simple perceptron algorithm to train the ordinal regression model, i.e., find the setting of parameters  $\underline{\theta}$  and  $\underline{b} = [b_1, \dots, b_4]^T$  so that the model agrees with the observations  $\{(\underline{x}_j, y_{ij}), j \in M_i\}$ . To this end, we will first turn each rating  $y_{ij}$ ,  $j \in M_i$ , into a set of binary labels  $\{s_{jr}, r = 1, \dots, 4\}$ , and update the associated component binary classifiers based on mistakes. Since the parameters of the component classifiers are tied, we will have to combine the individual

perceptron updates into block updates, one update per movie. The block updates ensure, for example, that the offset parameters remain ordered as they should. The algorithm is given by<sup>1</sup>

Cycle through the training set  $j \in M_i$ :

Find binary mistakes  $E = \{r : s_{jr}(\underline{\theta} \cdot \underline{x}_j - b_r) \leq 0\}$ ,

Collect together mistake updates:

$$\underline{\theta} \leftarrow \underline{\theta} + \left( \sum_{r \in E} s_{jr} \right) \underline{x}_j \quad (2)$$

$$b_r \leftarrow b_r - s_{jr} \text{ if } r \in E \quad (3)$$

The algorithm has a similar mistake guarantee as the ordinary perceptron algorithm. In other words, let's assume there exists a reference regression method  $\underline{w}^* = (\underline{\theta}^*, \underline{b}^*)$ , where  $\|\underline{w}^*\| = 1$ , such that  $s_{jr}(\underline{\theta}^* \cdot \underline{x}_j - b_r^*) \geq \gamma$  for all  $j \in M_i$  and  $r$ . Then the number of binary mistakes that the above algorithm makes is bounded by  $(k-1)(R^2+1)/\gamma^2$ , where  $R$  is the radius of the ball containing the movie feature vectors  $\underline{x}_j$ ,  $j \in M_i$ .

**SVM for ordinal regression.** We might not be able to fully separate the ratings and it is therefore useful to cast the problem in terms SVMs with slack variables. The formulation is very similar to the standard SVM since we have a collection of binary classifiers, only their parameters are tied. Note that we need to introduce a slack variable for each binary classification constraint. Thus

$$\begin{aligned} \min \quad & \frac{1}{2} \|\underline{\theta}\|^2 + C \sum_{j \in M_i} \sum_{r=1}^4 \xi_{jr} \quad \text{subject to} \\ & s_{jr}(\underline{\theta} \cdot \underline{x}_j - b_r) \geq 1 - \xi_{jr}, \quad \xi_{jr} \geq 0, \quad b_1 \leq b_2 \leq \dots \leq b_4 \end{aligned} \quad (4)$$

It will be helpful to rewrite the optimization problem slightly differently for the purpose of illustrating it in the context of collaborative filtering. Consider fixing  $\underline{\theta}$  and  $\underline{b}$ . What is the optimal value of  $\xi_{jr}$  in this case? It is the smallest non-negative value that satisfies the classification constraint. In other words, it is

$$\xi_{jr}(\underline{\theta}, \underline{b}) = \max\{0, 1 - s_{jr}(\underline{\theta} \cdot \underline{x}_j - b_r)\} = \text{Loss}(s_{jr}(\underline{\theta} \cdot \underline{x}_j - b_r)) \quad (5)$$

---

<sup>1</sup>See Crammer and Singer, "Pranking with Ranking", NIPS 2001.

where the Loss is the Hinge loss. Once we have optimized  $\xi_{jr}$ , there are no more classification constraints (they are all satisfied). So we are left with minimizing

$$\frac{1}{2}\|\underline{\theta}\|^2 + C \sum_{j \in M_i} \sum_{r=1}^4 \xi_{jr}(\underline{\theta}, \underline{b}) = \frac{1}{2}\|\underline{\theta}\|^2 + C \sum_{j \in M_i} \sum_{r=1}^4 \text{Loss}(s_{jr}(\underline{\theta} \cdot \underline{x}_j - b_r)) \quad (6)$$

with respect to  $\underline{\theta}$  and  $\underline{b}$  subject to  $b_1 \leq b_2 \leq \dots \leq b_4$ .

**Alternating minimization algorithm for collaborative filtering.** We are now ready to solve the collaborative filtering problem. We start with some fixed feature vectors for movies  $\underline{x}_j$ ,  $j \in M$ , and solve  $(\underline{\theta}_i, \underline{b}_i)$  for each user (as discussed above) based on their ratings. We will then flip the roles of movies and users and fix  $(\underline{\theta}_i, \underline{b}_i)$ ,  $i = 1, \dots, n$ , treating  $\underline{\theta}_i$  as a feature vector for user  $i$ . As a result, in the second step, we will try to solve for the best movie feature vectors  $\underline{x}_j$ , and so on. Let's see how this works in a bit more detail.

It is perhaps the easiest to understand the alternating algorithm if we arrange the terms of the overall collaborative filtering objective in a matrix form as follows.

$$\begin{array}{cccc} \frac{1}{2}\|\underline{\theta}_1\|^2 + & \frac{1}{2}\|\underline{x}_1\|^2 + & \dots & + \frac{1}{2}\|\underline{x}_m\|^2 \\ (C \sum_{r=1}^4 \text{Loss}(s_{11r}(\underline{\theta}_1 \cdot \underline{x}_1 - b_{1r}))) & \dots & (C \sum_{r=1}^4 \text{Loss}(s_{1mr}(\underline{\theta}_1 \cdot \underline{x}_m - b_{1r}))) & \\ \dots & \dots & \dots & \dots \\ \frac{1}{2}\|\underline{\theta}_n\|^2 + & (C \sum_{r=1}^4 \text{Loss}(s_{n1r}(\underline{\theta}_n \cdot \underline{x}_1 - b_{nr}))) & \dots & (C \sum_{r=1}^4 \text{Loss}(s_{nmr}(\underline{\theta}_n \cdot \underline{x}_m - b_{nr}))) \end{array} \quad (7)$$

where we have assumed for simplicity that the rating matrix is full. This does not change how the parameters are estimated but helps illustrate the algorithm. In practice, only a small fraction of the Loss terms would be present (those corresponding to observations), otherwise the problem is the same. Note that the binary labels  $s_{ijr}$  are user specific as they are based on the user ratings. We have also introduced  $\|\underline{x}_j\|^2/2$  terms symmetrically to  $\|\underline{\theta}_i\|^2/2$ . They will indeed play analogous roles, ensuring large margin solutions for user vectors given movie vectors, and for movie vectors given user vectors.

Now, let's interpret the objective in terms of the alternating minimizing algorithm. If we sum the terms in each row involving  $\underline{\theta}_i$  and  $\underline{b}_i$ , we obtain the SVM objective for user  $i$ , as before:

$$\frac{1}{2}\|\underline{\theta}_i\|^2 + \sum_{j \in M_i} \left( C \sum_{r=1}^4 \text{Loss}(s_{ijr}(\underline{\theta}_i \cdot \underline{x}_j - b_{ir})) \right) \quad (8)$$

If  $\underline{x}_j$  are fixed, we can solve these independently from other users.

Once we have  $(\underline{\theta}_i, \underline{b}_i)$ , we can fix these values and reoptimize the movie features. How to do this? Let's sum the terms in each column involving  $\underline{x}_j$ . This will give us

$$\frac{1}{2} \|\underline{x}_j\|^2 + \sum_{i=1}^n \left( C \sum_{r=1}^4 \text{Loss}(s_{i1r}(\underline{\theta}_i \cdot \underline{x}_j - b_{ir})) \right) \quad (9)$$

This is again an SVM objective, now for  $\underline{x}_j$ , when we interpret  $(\underline{\theta}_i, \underline{b}_i)$  as fixed user features. We can solve these independently for each movie.

Thus, the algorithm alternately minimizes terms in the rows and columns, monotonically minimizing the sum of all the terms – the CF objective. Note that the movie feature vectors that we end up with are those that best help us solve the individual tasks for each user. Each user influences what these feature vectors should be. This is how we “borrow” experience from other users.

There are many other methods developed for collaborative filtering and we may touch upon them again a bit later in the course.