

Recommender problems

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.

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

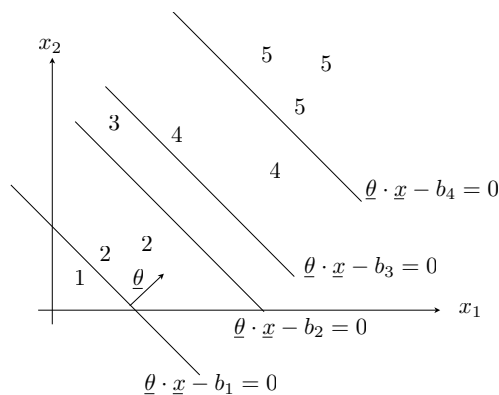


Figure 1: An ordinal regression model where $b_1 \leq b_2 \leq \dots \leq b_4$. The numbers in the figure correspond to points (movie feature vectors) with the corresponding ratings.

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. Figure 1 illustrates the regression model.

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 by¹

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:

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

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

¹See Crammer and Singer, “Pranking with Ranking”, NIPS 2001.

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 ξ_{jr} in this case? It is the smallest non-negative value that satisfies the classification constraint. In other words, it is

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

where the Loss is the Hinge loss. Once we have optimized ξ_{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 $b_1 \leq b_2 \leq \dots \leq b_4$.