Recommendation System Model

The code implements a simple matrix-factorization-based recommender using gradient descent. Given a set of ratings (from one star to five stars) from users on many movies they have seen, we implement a personalized rating predictor for a given user on unseen movies. This can be seen as a function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathrm{R}$

Data representation We have m users and n items, and a rating given by a user on a movie. We can represent this information as a form of matrix, namely rating matrix M. The rows of M represent users, while columns do movies, then the size of matrix will be $m \times n$. Each cell of the matrix may contain a rating on a movie by a user. In $M_{15,47}$, for example, it may contain a rating on the item 47 by user 15. If he gave 4 stars, $M_{15,47} = 4$.
However, as it is almost impossible for everyone to watch large portion of movies in the market, this rating matrix should be very sparse in nature. Typically, only 1% of the cells in the rating matrix are observed in average. All other 99% are missing values, which means the corresponding user did not see (or just did not provide the rating for) the corresponding movie.
Our goal with the rating predictor is estimating those missing values, reflecting the user's preference learned from available ratings.

Model Our approach for this problem is matrix factorization. Specifically, we assume that the rating matrix M is a low-rank matrix. Intuitively, this reflects our assumption that there is only a small number of factors (e.g, genre, director, main actor/actress, released year, etc.) that determine like or dislike. Let's define r as the number of factors. Then, we learn a user profile $U \in \mathbb{R}^{m \times r}$ and an item profile $V \in \mathbb{R}^{n \times r}$. (Recall that m and n are the number of users and items, respectively.) We want to approximate a rating by an inner product of two length r vectors, one representing user profile and the other item profile. Mathematically, a rating by user u on movie i is approximated by

$$M_{u,i} \approx \sum_{k=1}^{r} U_{u,k} V i, k \tag{1}$$

We want to fit each element of U and V by minimizing squared reconstruction error over all training data points. That is, the objective function we minimize is given by

$$E(U, V) = \sum_{(u,i) \in M} (M_{u,i} - U_u^\top V_i)^2 = \sum_{(u,i) \in M} (M_{u,i} - \sum_{k=1}^{r} U_{u,k} V_{i,k})^2 + \lambda \sum_{u,k} U_{u,k}^2 + \sum_{i,k} V_{i,k}^2 \tag{2}$$

where U_u is the u^{th} row of U and V_i is the i^{th} row of V. We observe that this looks very similar to the linear regression (except for the regularization part). Recall that we minimize in linear regression:

$$E(\theta) = \sum_{i=1}^{m} (Y^i - \theta^\top x^i)^2 = \sum_{i=1}^{m} (Y^i - \sum_{k=1}^{r} \theta_k x_k^i)^2 \tag{3}$$

where m is the number of training data points. Let's compare (2) and (3). $M_{u,i}$ in (2) corresponds to Y_i in (3), in that both are the observed labels. $U_u^\top V_i$ in (2) corresponds to $\theta^\top x^i$ in (3), in that both are our estimation with our model.

The only difference is that both U and V are the parameters to be learned in (2), while only θ is learned in (3). This is where we personalize our estimation: with linear regression, we apply the same θ to any input x^i, but with matrix factorization, a different profile U_u are applied depending on who is the user u. As U and V are interrelated in (2), there is no closed form solution, unlike linear regression case. Thus, we need to use gradient descent with a regularization term to penalie large values in U and V:

$$U_{v,k} \leftarrow U_{v,k} - \mu \frac{\partial E(U,V)}{\partial U_{v,k}} \qquad V_{j,k} \leftarrow V_{j,k} - \mu \frac{\partial E(U,V)}{V_{j,k}} \tag{4}$$

where μ is a hyper-parameter deciding the update rate. It would be straightforward to take partial derivatives of $E(U,V)$ in (2) with respect to each element $U_{v,k}$ and $V_{j,k}$. Then, we update each element of U and V using the gradient descent formula in (4).