1 Matrix Factorization

1.1 Setup

Let me suppose that we have, for instance, movie rating data of N_u users and N_i items. We use u and i to identify users and items, respectively. The rating data is represented by a $N_u \times N_i$ matrix R, where (u, i) component R_{ui} is the rating of an item i by a user u. Usually, we do not know all the components of this matrix; only some of them are known. If we can predict unknown components using the known rating data, we can efficiently recommend a movie to each user.

1.2 Matrix factorization

Matrix factorization is a method to realize such prediction. We decompose R into two matrices U and V as

$$R = U^T V, \tag{1.1}$$

where U is a $K \times N_u$ matrix characterizing each user and V is a $K \times N_i$ matrix characterizing each item. K is an arbitrary positive integer. This decomposition reads $R_{ui} = u_u^T \cdot v_i$, where we write $U = (u_1, u_2, \ldots, u_{N_u})$ and $V = (v_1, v_2, \ldots, v_{N_i})$. We can interpret K-dimensional vector u_u as user u personality, and v_i as item i characteristic. This is similar to the singular value decomposition, so it would be easy to perform such decomposition if we knew all the component of R. The rating data here, however, has a number of unknown components, and hence we need another strategy to do it.

1.3 Factorization algorithm

We write the existing data-set by D such that $(i, u) \in D$ is the pair whose rating is already known. We define the error function by

$$E \equiv \frac{1}{|D|} \sum_{(u,i)\in D} (R_{ui} - \hat{R}_{ui})^2, \tag{1.2}$$

where R is a given rating matrix, $\hat{R}$ is our model prediction, and $|D|$ is the size of D. The sum is taken over the known pair of (u, i). Once we tune $\hat{R}$ to minimize the error function, $\hat{R}$ is expected to predict unknown components of the rating matrix. With the decomposition (1.1) in mind, we model $\hat{R}$ as

$$\hat{R}_{ui} = \mu + b_u^{(U)} + b_i^{(I)} + u_u^T \cdot v_i, \tag{1.3}$$

where $\mu = (\sum_{(u,i)\in D} R_{ui})/(\sum_{(u,i)\in D} 1)$ is the total average, $b^{(U)}$ the user bias vector, and $b^{(I)}$ the item bias vector. $\hat{R}$ is a function of $b^{(U)}$, $b^{(I)}$, U and V. These bias vectors are important because, for instance, some users tend to give higher score than others, or some movies may have better quality. After introducing the biases, the term $u_u^T \cdot v_i$ is expected to measure the goodness of matching between a certain user and an item. In actual computation, there are a lot of free parameters which leads to overfitting the data. To avoid it, the regularization terms are added to the squared error as

$$\tilde{E} \equiv \frac{1}{|D|} \sum_{(u,i)\in D} (R_{ui} - \hat{R}_{ui})^2 + \frac{\beta}{2} \left(\|b^{(U)}\|^2 + \|b^{(I)}\|^2 + \|U\|_F^2 + \|V\|_F^2 \right), \tag{1.4}$$

where $\|U\|_F^2 = \sum_{ij} U_{ij}^2$. The regularization terms keep the model parameters small. We minimize $\tilde{E}$ with respect to the bias vectors and matrices U and V.

1.4 Stochastic gradient descent

For minimization, we adopt stochastic gradient descent (SGD). It is easy to implement and has several good properties compared to the naive gradient descent method. In particular, it brings randomness which prevent being trapped at local minima. SGD does not use the full error function (1.4) but use only the (u, i) sector

$$\tilde{E}_{ui} = (R_{ui} - \hat{R}_{ui})^2 + \frac{\beta}{2}\left(\|b^{(U)}\|^2 + \|b^{(I)}\|^2 + \|U\|_F^2 + \|V\|_F^2\right). \tag{1.5}$$

Note that $\tilde{E} = (\sum_{(i,u)\in D} \tilde{E}_{ui})/|D|$. For notational simplicity, let $\vec{w}$ be the collection of all the model parameters. At each minimization step, we first choose a known pair (u, i) at random, and take derivatives of $\tilde{E_{ui}}$. Then we shift model parameters with a efficiency paramter $\alpha > 0$ as

$$w_a \longrightarrow w_a - \alpha \frac{\partial \tilde{E}_{ui}}{\partial w_a}. \tag{1.6}$$

We obtain $\hat{R}$ by iterating this many times until it converges to a certain value. Technically, we iterate this procedure over many epochs. At each epoch, we randomly choose a pair $(u, i) \in D$ one by one, and perform shift (1.6). We use each pair in D only once, so an epoch ends after $|D|$ iteration. Then we iterate this over many epochs until convergence.

2 Improvements

- We have three parameters, K, α and β, which are fixed by hand. They are constant for all learning process, and tuned by just trial and error. If we fail to choose appropriate values, we will encounter over-learning or SGD will not converge. In particular, α determines the learning rate; if α is too small, it takes a lot of time to arrive at convergence, but if α is too large, the error function never approaches a minimum. Therefore, we better tune α in each SGD step. I tried very naive way of updating learning rate as

$$\alpha \longrightarrow \alpha/t, \tag{2.1}$$

  where t is the number of epoch, but it does not improve the convergence much. We may try more sophisticated methods.

- The matrix factorization has the *cold start problem*. At the very beginning of the system, we have few rating data and learning does not go well. To deal with this, we may collect explicit data; we can ask users what kind of movies they like or personal information (age, gender, nationality...) may be useful. Another way is to use implicit data such as users browsing history. We can model such information and implement it in U as $u_u \longrightarrow u_u + x_u$.

References