华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Zhaoyu Zhou, Hongjing Li and Liqin Zhan

Supervisor:
Qingyao Wu

Student ID:
201530613900, 201530081266 and 201530651780

Grade:
Undergraduate

December 28, 2017

Recommender System Based on Matrix Decomposition

***Abstract*—In this experiment, we use Matrix Decomposition to construct a recommendation system with the method of ALS and combine theory with the actual project for learning**

I. Introduction

IN this experiment, we'd like to finish tasks as follows: 1) Explore the construction of recommended system. 2) Understand the principle of matrix decomposition. 3) Construct a recommendation system under small-scale dataset, cultivate engineering ability.

As a result of the experiment, we finished the above tasks and gain more knowledge about recommended system.

II. Methods and Theory

Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data.

Alternating Least Squares Method is a way for Collaborative Filtering and we use it to construct a recommendation system in this experiment. It aims to use two low-dimension matrix X(m*k) and Y(n*k) to approximate the original scoring matrix, that is:

$$R_{m\times n} \approx X_{m\times k} Y^T_{n\times k}$$

And the loss function is:

$$L(X,Y) = \sum_{u,i}(r_{ui} - x_u^T y_i)^2 + \lambda(|x_u|^2 + |y_i|^2)$$

By ALS's method, we first estimate Y using X and estimate X by using Y. After enough number of iterations, we are aiming to reach a convergence point where either the matrices X and Y are no longer changing or the change is quite small. To make the gradient equal zero, we gain:

$$x_u = (Y^T Y + \lambda I)^{-1} Y^T r_u$$

$$y_i = (X^T X + \lambda I)^{-1} X^T r_i$$

When the root-mean-square error is small enough or the max epoch is achieved, ALS is done.

III. Experiments

A. Dataset

We use MovieLens-100k dataset. The u.data consists 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly. U1.base/u1.test are train set and validation set respectively, seperated from dataset u.data with proportion of 80% and 20%. And they are data we use.

B. Implementation

Experiment Step:

1.Read the data set and divide it (use u1.base / u1.test directly). Populate the original scoring matrix R_{n_users,n_items} against the raw data, and fill 0 for null values.

2.Initialize the user factor matrix $P_{n\_users,K}$ and the item (movie) factor matrix $Q_{n\_item,K}$, where K is the number of potential features.

3.Determine the loss function and the hyperparameter learning rate η and the penalty factor λ.

4.Use alternate least squares optimization method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:

4.1.With fixd item factor matrix, find the loss partial derivative of each row (column) of the user factor matrices, ask the partial derivative to be zero and update the user factor matrices.

4.2.With fixd user factor matrix, find the loss partial derivative of each row (column) of the item factor matrices, ask the partial derivative to be zero and update the item

4.3.Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged.

5.Repeat step 4. several times, get a satisfactory user factor matrix P and an item factor matrix Q, Draw a $L_{validation}$ curve with varying iterations.

6.The final score prediction matrix $\hat{R}_{n\_users,n\_items}$ is obtained by multiplying the user factor matrix $P_{n\_users,K}$ and the transpose of the item factor matrix $Q_{n\_item,K}$.

Experiment Code and Results:

```python
# 训练过程，ALS主过程
def train(X, Y, R_train, R_test, loss_train, loss_test, epochs):
    flag = 1
    epoch = 0
    while not is_ok(X, Y, R_train):

        #添加画图数据
        loss_train.append(get_loss(X, Y, R_train))
        loss_test.append(get_loss(X, Y, R_test))
        epochs.append(epoch)
        epoch += 1

        if epoch >= max_epoch: break
        #交替固定X,Y，更新Y,X
        if flag == 1:
            for u in range(m):
                X[u] = np.linalg.inv((np.dot(Y, Y.T) + lamda * I)).dot(Y).dot(R_train[u])
        else :
            for i in range(n):
                Y.T[i] = np.linalg.inv((np.dot(X.T, X) + lamda * I)).dot(X.T).dot(R_train.T[i])
        flag *= -1
```

Fig. 1. Train's code

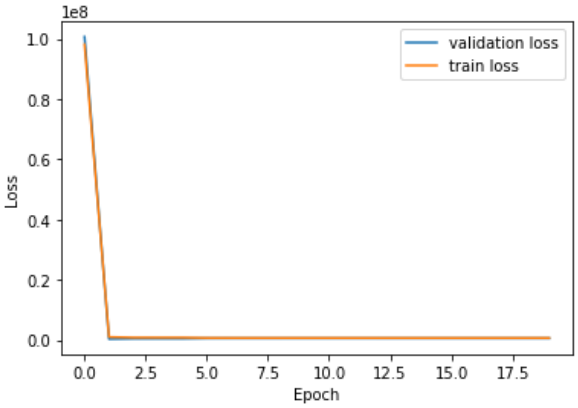

Fig. 2. Loss figure

IV. Conclusion

We find that if K is larger, the performance the given recommendation system will be better. And lambda's effect is subtle because its value must be appropriate. Otherwise, the loss may arise. We guess it may cause some value out of range.

It takes few epochs for loss to converge. But the algorithm is so slow because it requires to inverse a big matrix.