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: Peizhuo Xie、 Bo Wang、
Wenhui Chen

Supervisor:
Mingkui Tan

Student ID：201530613207、
201530381731、201530611272

Grade:
Undergraduate

December 9, 2017

Recommender System Based on Matrix Decomposition

Abstract—This report tends to illustrate the experiments we have done about construction of recommended system, with respect to understanding and comprehending the core of principle of matrix decomposition.

I. Introduction

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. Just as its name suggests, matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

II. Methods and Theory

In this section, we will introduce theory of matrix factorization and the minimizing methods.

A. Matrix Factorization

Give a rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, with sparse ratings from m users to n items.

Assume rating matrix **R** can be factorized into the multiplication of two low-rank feature matrices $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times n}$.

Objective function: $\mathcal{L}(r_{u,i}, \hat{r}_{u,i}) = (r_{u,i} - \hat{r}_{u,i})^2$

B. ALS Algorithm

The objective function:

$$\mathcal{L} = \sum_{u,i}(r_{u,i} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda(\sum_u n_{\mathbf{p}_u}||\mathbf{p}_u||^2 + \sum_i n_{\mathbf{q}_i}||\mathbf{q}_i||^2)$$

1: Require rating matrix **R**, feature matrices **P, Q** and regularization parameter λ.

2: Optimize **P** while fixing **Q**:

$$\mathbf{p}_u = (\mathbf{q}_i \mathbf{q}_i^\top + \lambda n_{\mathbf{p}_u} I)^{-1} \cdot \mathbf{Q}^\top \cdot \mathbf{R}_{u*}^\top$$

3: Optimize **Q** while fixing **P**:

$$\mathbf{q}_i = (\mathbf{p}_u \mathbf{p}_u^\top + \lambda n_{\mathbf{q}_i} I)^{-1} \cdot \mathbf{P}^\top \cdot \mathbf{R}_{*i}$$

4: Repeat the above processes until convergence.

C. SGD Algorithm

The objective function:

$$\mathcal{L} = \sum_{u,i}(r_{u,i} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda(\sum_u n_{\mathbf{p}_u}||\mathbf{p}_u||^2 + \sum_i n_{\mathbf{q}_i}||\mathbf{q}_i||^2)$$

1: Require feature matrices **P**, **Q**, observed set **Ω**, regularization parameters λ_p, λ_q and learning rate α.

2: Randomly select an observed sample $r_{u,i}$ from observed set **Ω**.

3: Calculate the gradient w.r.t to the objective function:

$$E_{u,i} = r_{u,i} - \mathbf{p}_u^\top \mathbf{q}_i$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = E_{u,i}(-\mathbf{q}_i) + \lambda_p \mathbf{p}_u$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = E_{u,i}(-\mathbf{p}_u) + \lambda_q \mathbf{q}_i$$

4: Update the feature matrices **P** and **Q** with learning rate α and gradient:

$$\mathbf{p}_u = \mathbf{p}_u + \alpha(E_{u,i}\mathbf{q}_i - \lambda_p \mathbf{p}_u)$$

$$\mathbf{q}_i = \mathbf{q}_i + \alpha(E_{u,i}\mathbf{p}_u - \lambda_q \mathbf{q}_i)$$

5: Repeat the above processes until convergence.

III. EXPERIMENT

A. *Data Set*

1.we use Utilizing MovieLens-100k dataset.

2.u.data -- Consisting 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

B. *Implementation*

1) Using alternate least squares optimization(ALS):

The step is as follows:

1. Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix $\boldsymbol{R}_{n_{users},n_{items}}$ against the raw data, and fill 0 for null values.
2. Initialize the user factor matrix $\boldsymbol{P}_{n_{users},K}$ and the item (movie) factor matrix $\boldsymbol{Q}_{n_{item},K}$, where $\boldsymbol{K}$ is the number of potential features.
3. Determine the loss function and the hyperparameter learning rate $\boldsymbol{\eta}$ and the penalty factor $\boldsymbol{\lambda}$.

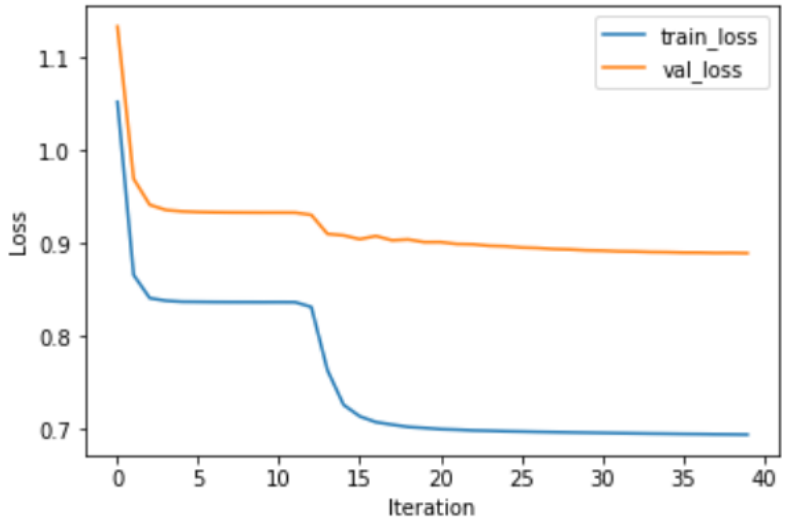
```
def compute_loss():
    global pred, validation_data, train_data
    error = []
    for i in validation_data:
        error.append(i[2] - pred[i[0] - 1][i[1] - 1])
    error = np.array(error)
    val_loss = np.mean(error ** 2)
    error = []
    for i in train_data:
        error.append(i[2] - pred[i[0] - 1][i[1] - 1])
    error = np.array(error)
    train_loss = np.mean(error ** 2)
    return val_loss, train_loss
```

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 fixed 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 fixed 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 $\boldsymbol{L}_{validation}$ on the validation set, comparing with the $\boldsymbol{L}_{validation}$ of the previous iteration to determine if it has converged.
5. Repeat step 4. several times, get a satisfactory user factor matrix $\boldsymbol{P}$ and an item factor matrix $\boldsymbol{Q}$, **Draw a $\boldsymbol{L}_{validation}$ curve with varying iterations**.
6. The final score prediction matrix $\widehat{\boldsymbol{R}}_{n\_users,n\_items}$ is obtained by multiplying the user factor matrix $\boldsymbol{P}_{n\_users,K}$ and the transpose of the item factor matrix $\boldsymbol{Q}_{n\_items,K}$.



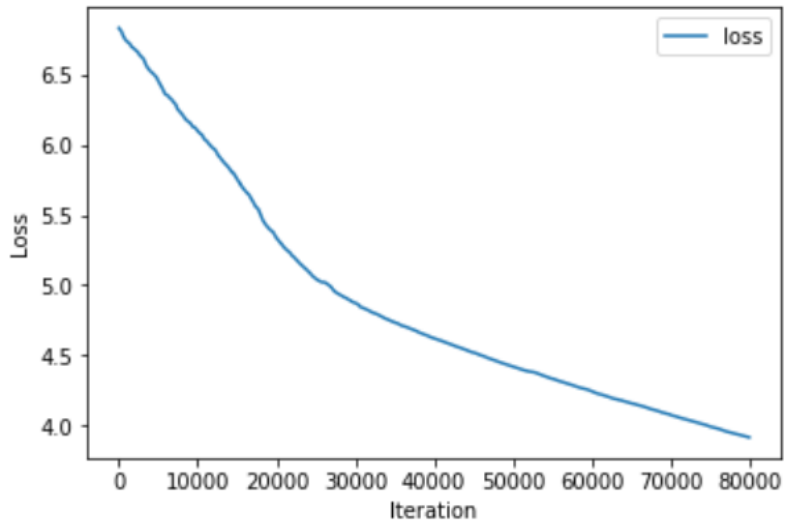
2) Using stochastic gradient descent method(SGD):

The step is as follows:

1. Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix $\boldsymbol{R}_{n_{users},n_{items}}$ against the raw data, and fill 0 for null values.
2. Initialize the user factor matrix $\boldsymbol{P}_{n_{users},K}$ and the item (movie) factor matrix $\boldsymbol{Q}_{n_{item},K}$, where $\boldsymbol{K}$ is the number of potential features.
3. Determine the loss function and the hyperparameter learning rate $\boldsymbol{\eta}$ and the penalty factor $\boldsymbol{\lambda}$.

```
def compute_loss():
    global R, validation_data
    error = []
    for i in validation_data:
        error.append(i[2] - R[i[0] - 1][i[1] - 1])
    error = np.array(error)
    return np.mean(error ** 2)
```

4. Use the stochastic gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:
   4.1 Select a sample from scoring matrix randomly;
   4.2 Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;
   4.3 Use SGD to update the specific row(column) of $\boldsymbol{P}_{n\_users,K}$ and $\boldsymbol{Q}_{n\_item,K}$;
   4.4 Calculate the $\boldsymbol{L}_{validation}$ on the validation set, comparing with the $\boldsymbol{L}_{validation}$ of the previous iteration to determine if it has converged.
5. Repeat step 4. several times, get a satisfactory user factor matrix $\boldsymbol{P}$ and an item factor matrix $\boldsymbol{Q}$, **Draw a $L_{validation}$ curve with varying iterations**.
6. The final score prediction matrix $\widehat{\boldsymbol{R}}_{n\_users,n\_items}$ is obtained by multiplying the user factor matrix $\boldsymbol{P}_{n\_users,K}$ and the transpose of the item factor matrix $\boldsymbol{Q}_{n\_items,K}$.



IV. CONCLUSION

In this experiment, we have tried using ALS and SGD algorithm to minimize the loss function. According to the figure shown above, we can get that ALS converges faster than the SGD, while SGD has less computational complexity than ALS(ALS needs to compute the matrix –vector multiplication).