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:
Zhuoman Liu and Feng Chen and
Congcong Zhan

Supervisor:
Mingkui Tan

Student ID:
201530612415 and 201530611159 and
201530613573

Grade:
Undergraduate

January 6, 2018

Recommender System Based on Matrix Factorization

***Abstract*—This article describes the matrix factorization to solve the recommender system problem.**

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. 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.

a) : Utilizing MovieLens-100k dataset. 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.Then we construct a recommendation system under the dataset.

b) : In terms of evaluation, we will compute the loss of validation set, and draw a $\mathbf{L}_{validation}$ curve with varying iterations.

II. Methods and Theory

In recommender system, a user-item rating matrix is normally represented for the preference of users for items. Matrix factorization is a common and useful method to solve the difficulty of recommendation. The system firstly provides a rating matrix $\mathbf{R} \in \mathbb{R}^{m\times n}$, with sparse ratings from m users to n items. Then we assume that, rating matrix $\mathbf{R}$ can be factorized into a multiplication of two low-rank feature matrices $\mathbf{P} \in \mathbb{R}^{m\times k}$ and $\mathbf{Q} \in \mathbb{R}^{k\times n}$. A simple example of matrix factorization is shown as Fig.1.

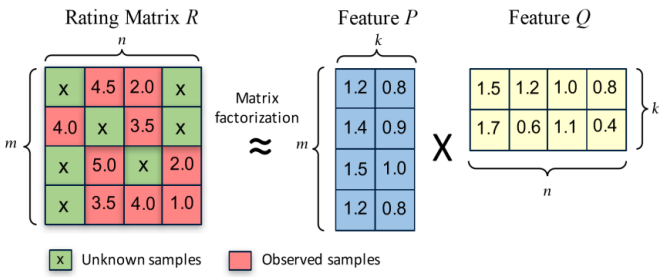

Fig. 1: An example of matrix factorization.

From the matrix factorization, we can obtain an objective function using squared error loss:

$$\mathcal{L}(r_{u,i}, \hat{r}_{u,i}) = (r_{u,i} - \hat{r}_{u,i})^2 \tag{1}$$

where $r_{u,i}$ donates the actual rating of user u for item i and $\hat{r}_{u,i}$ denotes the prediction. Then minimizing the objective function (Eq.2) with regularization using SGD (stochastic gradient descent),

$$\mathcal{L} = \sum_{u,i\in\Omega} (r_{u,i} - \mathbf{p}_u^T\mathbf{q}_i)^2 + \lambda_p\|\mathbf{p}_u\|^2 + \lambda_q\|\mathbf{q}_i\|^2 \tag{2}$$

where $\mathbf{P}$=$[\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m]^T \in \mathbb{R}^{m\times k}$, $\mathbf{Q}$=$[\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_m]^T \in \mathbb{R}^{k\times n}$, Ω denotes the set of observed samples from rating matrix $\mathbf{R}$ and λ_p, λ_q are regularization parameters to avoid overfitting. With the definition of prediction error $E_{u,i}$=$r_{u,i} - \mathbf{p}_u^T\mathbf{q}_i$, the gradient of the object function is:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{p}_u} = E_{u,i}(-\mathbf{q}_i) + \lambda_p\mathbf{p}_u \tag{3}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{q}_i} = E_{u,i}(-\mathbf{p}_u) + \lambda_q\mathbf{q}_i \tag{4}$$

$$\mathbf{p}_u = \mathbf{p}_u + \alpha(E_{u,i}\mathbf{q}_i - \lambda\mathbf{p}_u) \tag{5}$$

$$\mathbf{q}_i = \mathbf{q}_i + \alpha(E_{u,i}\mathbf{p}_u - \lambda\mathbf{q}_i) \tag{6}$$

Then updating the feature matrices $\mathbf{P}$ and $\mathbf{Q}$ with learning rate α (Eq.5,6) and repeating optimization until convergence.

III. Experiments

A. Dataset

1.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.

3.u1.base / u1.test are train set and validation set respectively, seperated from dataset u.data with proportion of 80% and 20%.

B. Implementation

1) Methon: Stochastic gradient descent method(SGD).

2) Steps: 1.Read the data set and 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.

```
R = np.zeros((N,M)
R[user][movie] = rating
```

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

```
K = 10
P = np.random.rand(N,K)
Q = np.random.rand(M,K)
```

3.Determine the loss function and the hyperparameter learning rate α and the penalty factor β.

```
alpha=0.05
beta=0.01
```

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: Select a sample from scoring matrix randomly;

```
i = random.randint(0, len(R)-1)]
j = random.randint(0, len(R[i])-1)
```

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 $P_{n_users,K}$ and $Q_{n_item,K}$;

```
eij = R[i][j] - np.dot(P[i,:],Q[:,j])
for k in range(K):
 P[i][k]=P[i][k]+alpha*(2*eij*Q[k][j]-beta*P[i][k])
 Q[k][j]=Q[k][j]+alpha*(2*eij*P[i][k]-beta*Q[k][j])
```

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

```
for i in range(len(R_test)):
 for j in range(len(R_test[i])):
  if R_test[i][j] > 0:
    e=e+pow(R_test[i][j]-np.dot(P[i,:],Q[:,j]),2)
    for k in range(K):
      e=e+(beta/2)*(pow(P[i][k],2)+pow(Q[k][j],2))
```

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 $R/hat_{n_users,n_items}$ is obtained by multiplying the user factor matrix $P_{n_users,K}$ and the transpose of the item factor matrix $Q_{n_item,K}$.

```
R_new = np.dot(nP, nQ.T)
```

3) Result:: The hyperparameter in Table I and the loss curve with varying iterations. Fig. 2. It can be seen from the figure that, though the final result is not well converged to a certain value, the magnitude of the change has been very small, so it has no effect.

TABLE I: Parameters

number of potential features	$k = 10$
learning rate	$\alpha = 0.05$
penalty factor	$\eta = 0.01$
epoch	$epoch = 5000$

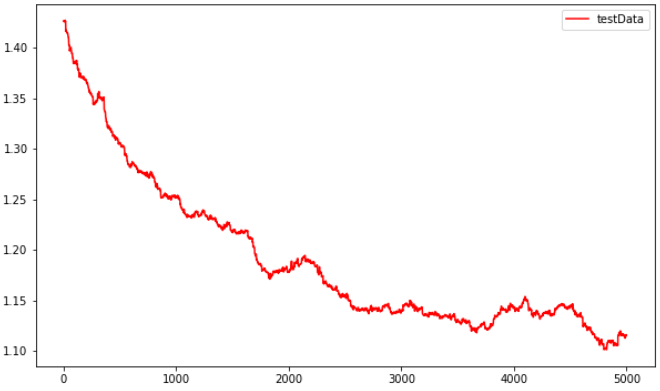

Fig. 2: The average Loss

IV. Conclusion

In this experiments, we understand the principle of matrix decomposition and the construction of recommended system. It is easy to program for matrix decomposition for the recommended method itself, with low complexity, good predictive power, and scalability.