Factorization-Based Data Modeling
Practical Work 2

CHEN Hang, FAN Zheng

December 2018

1 Matrix Factorization with Stochastic Gradient Descent

1.1 Complete the stochastic gradient algorithm

For completing the stochastic gradient algorithm, first, we need to load the data matrix, which corresponds to the rating of movies given by users.
As the matrix is sparse with 4.3% non-zero entries, we store this matrix in COO format to reduce the memory. And we define a "mask" matrix to denote if a particular entry of data matrix is observed or not.
Since the amount of data in the matrix is too large, we randomly permute 1000 data in the data matrix to update the matrices W and H in each iteration by using the formulas below:

$$\mathbf{W} \leftarrow \mathbf{W} + \eta(\mathbf{X} - \mathbf{WH})\mathbf{H}^\top$$

$$\mathbf{H} \leftarrow \mathbf{H} + \eta\mathbf{W}^\top(\mathbf{X} - \mathbf{WH})$$

According to the formulas, we can have the code for each iteration:

```
grad_w = (cur_x-cur_xhat) * H[:,cur_j].T
grad_h = (W[cur_i,:].T) * (cur_x-cur_xhat)
W[cur_i,:] = W[cur_i,:] + eta * grad_w
H[:,cur_j] = H[:,cur_j] + eta * grad_h
```

As for the cur_x and cur_xhat, cur_x is a value of 1000 samples and cur_xhat is a value obtained by multiplying the corresponding rows and columns of the last iteration results of W and H.

```
cur_x = xsp.data[data_index[i]]
cur_xhat = np.dot(W[cur_i,:],(H[:,cur_j]))
```

xsp is the data matrix stored in COO format.

1.2 Compute the root-mean-squared-error

At the end of each iteration, we compute the roor-mean-squared-error given as follows:

$$RMSE = \sqrt{\frac{\| \mathbf{M} \odot (\mathbf{X} - \mathbf{WH}) \|_F^2}{N}}$$

So for each iteration, we have the code :

```
tmp = np.multiply(M, (xsp0.toarray()- np.dot(W,H)))
rmse_sgd[t] = np.sqrt(sum(sum(tmp**2))/N)
```

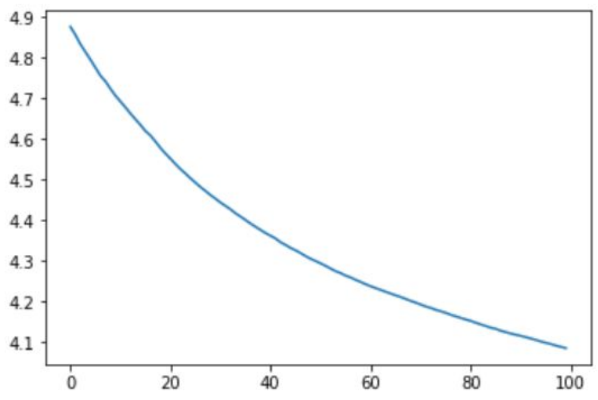

Figure 1: roor-mean-squared-error

1.3 Play with the algorithm parameters

1.3.1 Step-size

$$\eta = 0.03$$

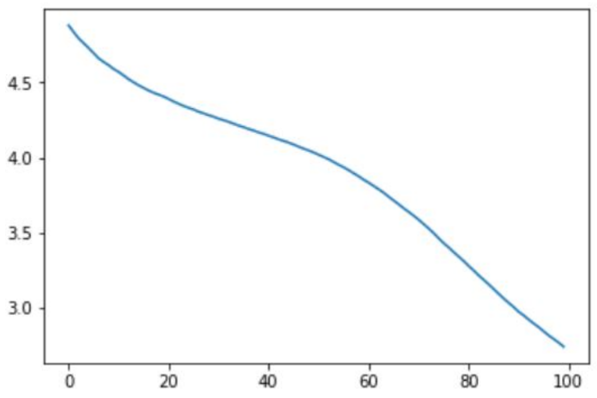

Figure 2: roor-mean-squared-error

$$\eta = 0.08$$

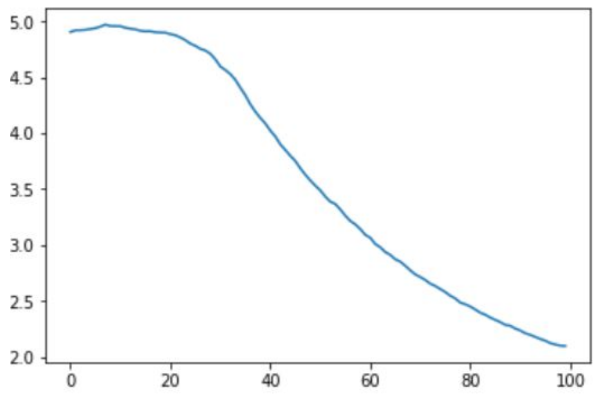

Figure 3: roor-mean-squared-error

η controls the speed of the gradient. When η is small, the objective function needs more times of iteration to be convergent. When η is large, the objective function can converge to a smaller value.

1.3.2 Batch-size

$$batchSize = 500$$

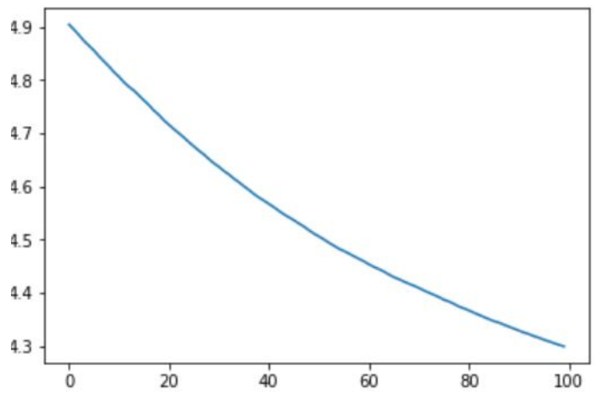

Figure 4: roor-mean-squared-error

$$batchSize = 1500$$

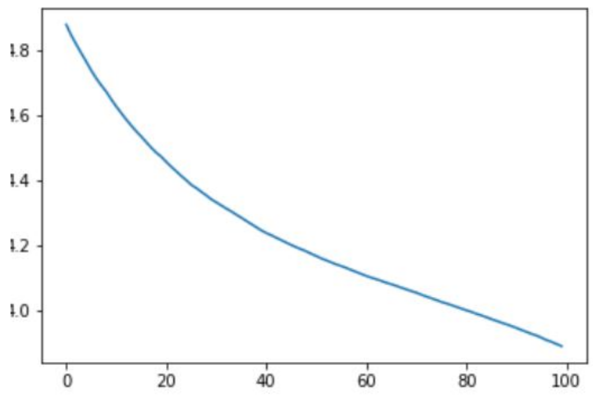

Figure 5: roor-mean-squared-error

When batch size is larger, the objective function can converge to a smaller value, because the algorithm can sample more data values to learn two matrices W and H more correctly.

1.3.3 The rank of factorization

$$K = 5$$

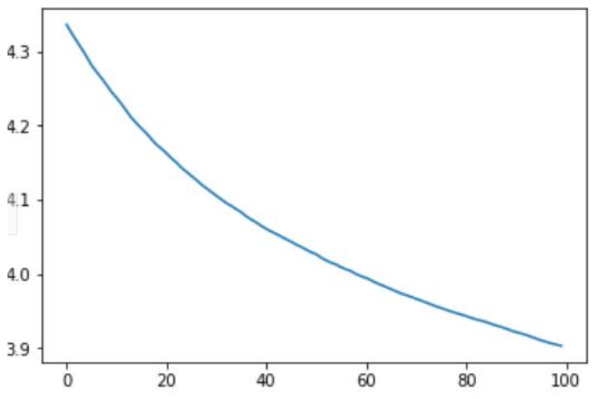

Figure 6: roor-mean-squared-error

$K = 15$

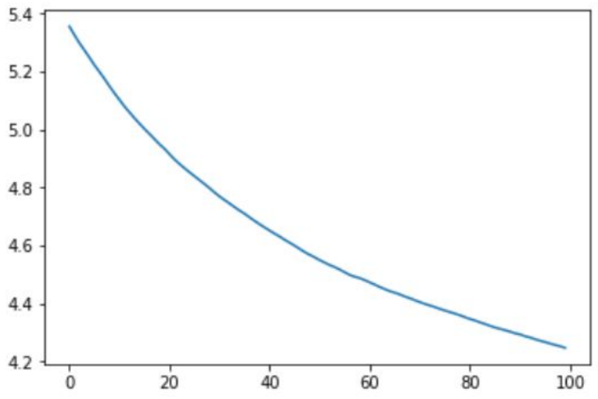

Figure 7: roor-mean-squared-error

When the rank of factorization K is larger, the error of each iteration is larger than the smaller ranks'.

1.3.4 Interact of step-size and batch-size

η controls the speed of gradient, and $batchSize$ controls the size of samples. If η is larger, it means that the speed of gradient id=s higher, and if $batchSize$ is larger, it means that the objective function can converge to a smaller value but also means that the convergence needs more times of iteration.
In conclusion, when $batchSize$ is larger, we can make η larger, so that while we get a small error, the speed of convergence is also guaranteed.

2 Movie Recommendation

2.1 Recommend a movie for a given user

After estimating W and H, for a given user_index = 11, we can use W and H to predict the ratings of each movies given by this user. Then, according to the ratings, we can recommend a movie with the highest rating for this user.

To complete the prediction, we have the code below:

```
for i in range(I):
    predict_xhat.append( W[i,:].dot(H[:, user_index]))
movie_index = predict_xhat.index(max(predict_xhat))
```