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:
Zizhuo Tao

Supervisor:
Mingkui Tan

Student ID:
201636824347

Grade:
Junior

December 24, 2018

Recommender System Based on Matrix Decomposition

***Abstract*—This experiment uses matrix decomposition to predict users' unknown ratings in a recommender system. We use a m*k matrix and a k*n matrix to approximate the actual rating matrix which is m*n. We use two algorithms to solve the problem respectively: Alternating Least Square and Stochastic Gradient Descent.**

I. Introduction

Recommender systems are very common nowadays, they apply statistical and knowledge discovery techniques to the problem of making product recommendations. They are becoming more and more popular since they can improve conversion rate, realize the cross-selling and improve customer loyalty. The basic solution to recommend is collaborative filtering. There are memory-based and model-based collaborative filtering. Compared to memory-based CF, model-based CF is a more practical and efficient way. And matrix factorization is the most widely used algorithm in model-based CF.

II. Methods and Theory

Alternating Least Square:

The objective function–loss function is:

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

We define $P = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m]^T \in \mathbb{R}^{m*k}$ and $Q = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n] \in \mathbb{R}^{k*n}$. Moreover, $r_{u,i}$ denotes the actual rating of user u for item i and λ is regularization parameter to avoid overfitting. $n_{\mathbf{p}_u}$ and $n_{\mathbf{q}_i}$ denote the number of total ratings on user u and item i, respectively.

Since we have two variables, which is hard to optimize, we decide to optimize one variable while fixing another.

Firstly, we optimize P while fixing Q. For u-th column in matrix P^T, set $\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = 0$ and we can get:

$$\mathbf{p}_u = \left(QQ^T + \lambda n_{\mathbf{p}_u} I\right)^{-1} Q {R_{u*}}^T$$

R_u*denotes the u-th row of rating matrix R.

Next, we optimize Q while fixing P. For i-th column in matrix Q, set $\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = 0$ and we can get:

$$\mathbf{q}_i = \left(PP^T + \lambda n_{\mathbf{q}_i} I\right)^{-1} P^T R_{*i}$$

We need to repeat the optimization until convergence.

Stochastic Gradient Descent:

The objective function is:

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

Ω denotes the set of observed samples from rating matrix R , λ_p and λ_q are regularization parameters to avoid overfitting.

We randomly select an observed sample $r_{u,i}$ and then we calculate the gradient on this sample:

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

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

$E_{u,i}$ is the prediction error: $r_{u,i} - {\mathbf{p}_u}^T \mathbf{q}_i$

Then we update the feature matrices P and Q with learning rate α:

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

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

We need to repeat above processes until convergence.

Accuracy Measures:

There are two common measures of the accuracy: mean absolute error and root mean square error.

$$MAE = \frac{1}{|\Omega|} \sum_{u,i \in \Omega} |\hat{r}_{u,i} - r_{u,i}|$$

$r_{u,i}$ denotes the actual rating and $\hat{r}_{u,i}$ denotes the prediction. $|\Omega|$ denotes the number of observed set.

$$RMSE = \sqrt{\sum_{u,i \in \Omega} \left(\hat{r}_{u,i} - r_{u,i}\right)^2 / |\Omega|}$$

Comparison Between Two Algorithms:

The time complexity per iteration of ALS is:

$$O(|\Omega|k^2 + (m+n)k^3)$$

The time complexity per iteration of SGD is:

$$O(|\Omega|k)$$

Therefore, SGD has less storage and computational complexity than ALS. As a result, SGD is more scalable to large-scale datasets than ALS. Additionally, since ALS uses matrix computation, it is easier to parallelize and converges faster than SGD.

III. Experiments

A. Dataset

The dataset used in this experiment is MovieLens-100k. It Consists of 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 stored randomly. u1.base / u1.test are train set and validation set respectively, separated from dataset u.data with proportion of 80% and 20%.

B. Implementation

Alternating Least Square:

After getting a training matrix and a validation matrix from the dataset, we need to create a P matrix denoting users and a Q matrix denoting items and initialize them. I personally initialize them with random numbers from 0 to 1. We also need to initialize some parameters like λ, K, and maximal epochs we want to iterate. Then fix the matrix Q, update every column vector in P matrix with the formula mentioned above:

$$\mathbf{p}_u = (QQ^T + \lambda n_{\mathbf{p}_u} I)^{-1} Q {R_{u*}}^T$$

Similarly, we fix the matrix P, update every column vector in Q with the formula:

$$\mathbf{q}_i = (PP^T + \lambda n_{\mathbf{q}_i} I)^{-1} P^T R_{*i}$$

Then we calculate the value of loss function on training set in every iteration. The loss curve on training set is shown in Fig. 1

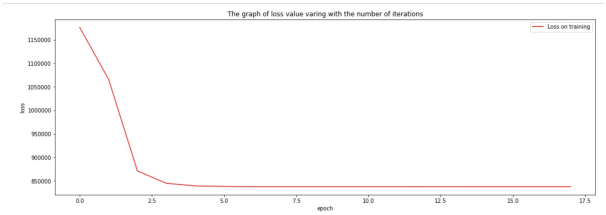


Fig. 1: loss curve on training set

Then we calculate the RMSE on validation set to evaluate the accuracy of prediction. When the values are close enough– difference is smaller than 0.1% of mean error or the number of iteration reaches up to maximum, we stop the iteration.

The curve of $L_{validation}$ is shown in Fig. 2

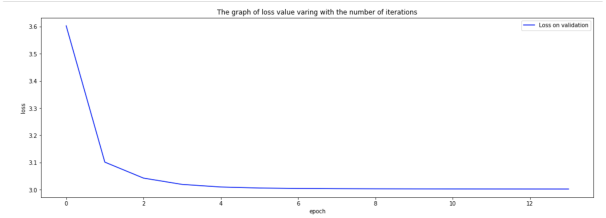


Fig. 2: RMSE curve on validation set

We can see that this algorithm converges very fast. Moreover, the running time is very short, about several seconds per iteration.

Stochastic Gradient Descent:

After getting a training matrix and a validation matrix from the dataset, we need to create a P matrix denoting users and a Q matrix denoting items and initialize them. This experiment initializes them with random numbers from 0 to 1. We also need to initialize some parameters like λ_p , λ_q, K, learning rate α and maximal epochs we want to iterate. Then we randomly select a sample from the training matrix and calculate the gradient on this sample:

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

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

$E_{u,i}$ is the prediction error: $r_{u,i} - {\mathbf{p}_u}^T \mathbf{q}_i$ After that, we update the corresponding row in matrices P and Q with a learning rate α:

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

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

In order to insure it to converge, we can use exponential decay on learning rate α by multiplying 0.5 every five iterations. Then we calculate loss value on training set per iteration. The loss curve of training set is shown in Fig. 3

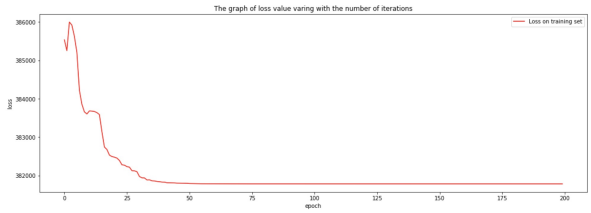
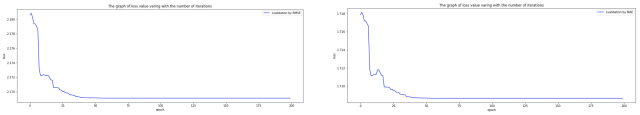


Fig. 3: loss curve on training set

We can choose RMSE and MAE to evaluate the prediction. The RMSE curve of validation set is shown in Fig. 4(a) and MAE curve of validation set is shown in Fig. 4(b)



(a) RMSE (b) MAE

Fig. 4: RMSE and MAE curves on validation set

We can see Stochastic Gradient Descent converges a lot slower than Alternating Least Square. Until 50th iteration, it starts to converge. But if the data becomes much larger, it would behave much better than Alternating Least Square since SGD is more scalable to large datasets.

IV. Conclusion

Alternating Least Square and Stochastic Gradient Descent are two popular algorithms in model-based collaborative filtering. They can both solve the problem of recommender systems well. In general, ALS can parallelize and converges faster. But SGD has less storage complexity and time complexity per iteration. When it comes to large data, SGD becomes more scalable to solve the problem.

These two algorithms are both based on matrix decomposition. ALS optimizes the objective function by setting derivative of every vector in P and Q to zero per iteration. SGD optimizes objective function by updating every vector in P and Q in the gradient descent direction per iteration. In the process of experiment, they can both converge to the minimum. SGD, though converges slower, can behave better than ALS after enough iterations with decaying learning rate.