The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:

钟恩俊 王泽众 诸俊浩

Supervisor:
Mingkui Tan

Student ID：

201530613818

201530612598

201530613962

Grade:

Undergraduate

December 9, 2017

Recommender System Based on Matrix Decomposition

Abstract—**Collective filtering is an effective technique used by recommender systems in predicting user preferences. There are two different forms of Collective filtering system, which is memory-base system and model-base system. As for model-base system, matrix decomposition is an effective algorithm. In this experiment , we have a exploration in 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. 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. One advantage of using this approach is that instead of having a high dimensional matrix containing abundant number of missing values we will be dealing with a much smaller matrix in lower-dimensional space. It handles the sparsity of the original matrix better than memory based ones. Also comparing similarity on the resulting matrix is much more scalable especially in dealing with large sparse datasets. In this experiment, we will focus on matrix factorization methods which is based on the low-rank assumption of user- item matrix. Further, we also explore two algorithms that implement the matrix factorization, alternating least square optimization and stochastic gradient descent.

II. Methods and Theory

a)Matrix Factorization

Matrix factorization (MF) is one of the most often applied techniques for CF problems. The idea behind MF techniques is very simple. Suppose we want to approximate the matrix R as the product of two matrices:

$$R \approx PQ$$

where P is an N × K and Q is a K × M matrix. This factorization gives a low dimensional numerical representation of both users and items. Note, that Q and P typically contain real numbers, even when R contains only integers. In the case of the given problem, the unknown ratings of R cannot be represented by zero. For this case, the approximation task can be defined as follows. Let P_{uk} denote the elements of $P \in R_{N \times K}$, and q_{ki} the elements of $Q \in R_{K\times M}$. Let P_u^T denote the transpose of the u-th row of P, and Q_i the i-th column of Q. Then:

$$\hat{r}_{ui} = \sum_{k-1}^{K} p_{uk} q_{ki} = p_u^T q_i$$

$$e_{ui} = r_{ui} - \hat{r}_{ui} for(u,i) \in \tau$$

$$SSE = \sum_{(u,i)\in\tau} e_{ui}^2 = \sum_{(u,i)\in\tau} (r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki})^2$$

$$RMSE = \sqrt{\frac{SSE}{|\tau|}}$$

$$(\mathbf{P}^*, \mathbf{Q}^*) = \arg\max_{(\mathbf{P},\mathbf{Q})} SSE = \arg\min_{(\mathbf{P},\mathbf{Q})} RMSE$$

b)Alternating Least Squares Optimization

We have users u for items ii matrix as in the following:

$$Q_{ui} = \begin{cases} r & \text{if user u rate item i} \\ 0 & \text{if user u did not rate item i} \end{cases}$$

where r is what rating values can be. If we have mm users and n items, then we want to learn a matrix of factors which represent movies. That is, the factor vector for each movie and that would be how we represent the movie in the feature space. Note that, we do not have any knowledge of the category of the movie at this point. We also want to learn a factor vector for each user in a similar way how we represent the movie. Factor matrix for movies $Y \in \mathbb{R}^{fxn}$ and factor matrix(each movie is a column vector) for users $X \in \mathbb{R}^{mxf}$(each user is a row vector). However, we have two unknown variables. Therefore, we will adopt an alternating least squares approach with regularization. By doing so, 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. However, there is a small problem in the data. We have neither user full data nor full items data, this is also why we are trying to build the recommendation engine in the first place. Therefore, we may want to penalize the movies that do not have ratings in the update rule. By doing so, we will depend on only the movies that have ratings from the users and do not make any assumption around the movies that are not rated in the recommendation. Let's call this weight matrix w_{ui} as such:

$$w_{ui} = \begin{cases} 0 & \text{if } q_{ui} = 0 \\ 1 & \text{else} \end{cases}$$

Then, cost functions that we are trying to minimize is in the following:

$$J(x_u) = (q_u - x_u Y) W_u (q_u - x_u Y)^T + \lambda x_u x_u^T$$

$$J(y_i) = (q_i - X y_i) W_i (q_i - X y_i)^T + \lambda y_i y_i^T$$

Note that we need regularization terms in order to avoid the overfitting the data. Ideally, regularization parameters need to be tuned using cross-validation in the dataset for algorithm to generalize better. In this post, I will use the whole dataset. Solutions for factor vectors are given as follows:

$$x_u = (YW_uY^T + \lambda I)^{-1} YW_u q_u$$

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

where $W_u \in \mathbb{R}^{nxn}$and $W_u \in \mathbb{R}^{mxm}$ diagonal matrices. The algorithm is pretty much of it. In the regulaization, we may want to incorporate both factor matrices in the update rules as well if we want to be more restrictive. That may generalize better, though.

c) Stochastic Gradient Descent

Having discussed the intuition behind matrix factorization, we can now go on to work on the mathematics. Firstly, we have a set U of users, and a set D of items. Let $\mathbf{R}$ of size $|U| \times |D|$be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrics matrices $\mathbf{P}$(a $|U| \times K$matrix) and $\mathbf{Q}$(a $|D| \times K$ matrix) such that their product approximates $\mathbf{R}$:

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

In this way, each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item $d_j$by u_i, we can calculate the dot product of the two vectors corresponding to $u_i$and d_j:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Now, we have to find a way to obtain $\mathbf{P}$and $\mathbf{Q}$. One way to approach this problem is the first intialize the two matrices with some values, calculate how different their product is to $\mathbf{M}$, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference.

The difference here, usually called the error between the estimated rating and the real rating, can be calculated by the following equation for each user-item pair:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating.

To minimize the error, we have to know in which direction we have to modify the values of p_{ik} and q_{kj}. In other words, we need to know the gradient at the current values, and therefore we differentiate the above equation with respect to these two variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij} q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij} p_{ik}$$

Having obtained the gradient, we can now formulate the update rules for both $p_{ik}$and q_{kj}:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Here, α is a constant whose value determines the rate of approaching the minimum. Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process.

III. Experiments

In this experiment, we use the SGD method to optimize the loss function.

a). Dataset

We conduct experiments on MovieLen-100k dataset, which 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.

b) Implementation of SGD

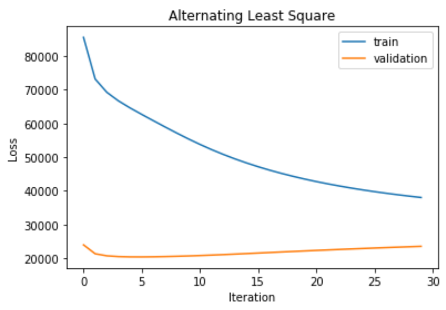
Read the data set and divide it. Populate the original scoring matrix $R_{nusers,nitems}$ against the raw data, and fill 0 for null values. Initialize the user factor matrix $P_{nusers,K}$ and the item (movie) factor matrix $Q_{nitem,K}$, where K is the number of potential features. Determine the loss function and hyper parameter learning rate η and the penalty factor λ. Use the stochastic gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix: Select a sample from scoring matrix randomly; Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix; Use SGD to update the specific row(column) of $P_{nusers,K}$ and $Q_{nitem,K}$; Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged. 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. The final score prediction matrix $R_{nusers,nitems}$ is obtained by multiplying the user factor matrix $P_{nusers,K}$ and the transpose of the item factor matrix $Q_{nitem,K}$

c). Implementation of ALS

Read the data set and divide it. Populate the original scoring matrix $R_{nusers,nitems}$ against the raw data, and fill 0 for null values. Initialize the user factor matrix $P_{nusers,K}$ and the item (movie) factor matrix $Q_{nitem,K}$, where K is the number of potential features. Determine the loss function and the hyper parameter learning rate η and the penalty factor λ. Use alternate least squares optimization method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix: 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. 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 Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged. Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix , Draw a $L_{validation}$ curve with varying iterations. The final score prediction matrix $R_{nusers,nitems}$ is obtained by multiplying the user factor matrix $P_{nusers,K}$ and the transpose of the item factor matrix $Q_{nitem,K}$.

c).Result

```
Turns 0  train loss is 85496.6242637  test loss is
24014.4757976
Turns 1  train loss is 73147.6652743  test loss is
21380.6890835
Turns 2  train loss is 69224.9236782  test loss is
20767.5280798
Turns 3  train loss is 66673.0214631  test loss is
20535.3190549
Turns 4  train loss is 64588.7735577  test loss is
20452.2226044
Turns 5  train loss is 62670.8208447  test loss is
20443.818174
Turns 6  train loss is 60805.4358206  test loss is
20478.5709515
Turns 7  train loss is 58964.302964  test loss is
20541.4619326
Turns 8  train loss is 57162.1270181  test loss is
20625.5937711
Turns 9  train loss is 55427.8373396  test loss is
20728.0597845
Turns 10  train loss is 53785.9595723  test loss is
20847.0734586
Turns 11  train loss is 52250.1097478  test loss is
20980.3811326
Turns 12  train loss is 50824.3848566  test loss is
21124.9838685
Turns 13  train loss is 49506.8292935  test loss is
21277.5644602
Turns 14  train loss is 48292.2128375  test loss is
21434.9978624
Turns 15  train loss is 47173.7528894  test loss is
21594.6658457
Turns 16  train loss is 46144.096901  test loss is
21754.5567767
Turns 17  train loss is 45195.8670315  test loss is
21913.2261324
Turns 18  train loss is 44321.9551883  test loss is
22069.6951491
Turns 19  train loss is 43515.6753771  test loss is
22223.3381434
Turns 20  train loss is 42770.8335613  test loss is
22373.7832121
Turns 21  train loss is 42081.7496075  test loss is
22520.8342872
Turns 22  train loss is 41443.2516801  test loss is
22664.4140161
Turns 23  train loss is 40850.6552264  test loss is
22804.5236888
Turns 24  train loss is 40299.7337544  test loss is
22941.2158813
Turns 25  train loss is 39786.6856044  test loss is
23074.5760541
Turns 26  train loss is 39308.0991327  test loss is
23204.7102143
Turns 27  train loss is 38860.9177168  test loss is
23331.7365813
Turns 28  train loss is 38442.4054521  test loss is
23455.7798655
Turns 29  train loss is 38050.1141439  test loss is
23576.9672678
```



IV. CONCLUSION

We can see that the training loss decrease with the iterations, but the test loss begin to increase after several iterations. That might be overfittings. We can solve this problem by reduce k.