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:
Xue Lei, Chengneng Jin and Deiwei Su

Supervisor:
Mingkui Tan

Student ID:
201530611876, 201530611838 and
201530612736

Grade:
Undergraduate

December 24, 2017

Recommender System Based on Matrix Decomposition

***Abstract*—Collective filtering is an effective approach for recommender systems in predicting user preferences. A major approach for collective filtering is matrix factorization, which is based on the assumption that**

I. Introduction

COLLABORATIVE filtering is a subset of algorithms that exploit other users and items along with their ratings(selection, purchase information could be also used) and target user history to recommend an item that target user does not have ratings for. Fundamental assumption behind this approach is that other users preference over the items could be used recommending an item to the user who did not see the item or purchase before. CF differs itself from content-based methods in the sense that user or the item itself does not play a role in recommendation but rather how and which users rated a particular item. (Preference of users is shared through a group users who selected, purchased or rate items similarly). Model based CF is developed using different data mining, machine learning algorithms to predict users' rating of unrated items. There are many model-based CF algorithms. Bayesian networks, clustering models, latent semantic models such as singular value decomposition, probabilistic latent semantic analysis, multiple multiplicative factor, latent Dirichlet allocation and Markov decision process based models. Through this approach, dimensionality reduction methods are mostly being used as complementary technique to improve robustness and accuracy of memory-based approach. In this sense, methods like singular value decomposition, principle component analysis, known as latent factor models, compress user-item matrix into a low-dimensional representation in terms of latent factors. 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. A reduced presentation could be utilized for either user-based or item-based neighborhood algorithms that are presented in the previous section. There are several advantages with this paradigm. 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. Specifically, in this paper, 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. The rest of the paper is organized as follows: in section 2, we first introduce matrix factorization based CF, then we explain ALS and SGD in details. In section 3, we introduce our experimental details in implementing these two algorithms and further analysis the experimental results. We conclude our paper in Section 4.

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 $\mathbf{R}$ as the product of two matrices:

$$\mathbf{R} \approx \mathbf{PQ}$$

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

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

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

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

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

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

In what follow, we will present two popular algorithm for matrix factorization (ALS), Alternating Least Square and Stochastic Gradient Descent (SGD).

B. Alternating Least Squares Optimization

A popular approach for this is matrix factorization, where we fix a relatively small number k(e.g. $k \approx 10$), and summarize each user u with a k dimensional vector x_u, and each item i with a k dimensional vector y_i. These vectors are referred to as factors. Then, to predict user us rating for item i, we simply predict $r_{ui} \approx x_u^T y_i$. This can be put in matrix form: Let $x_1, \ldots, x_n \in R^k$ be the factors for the users, and $y1, \ldots$

. , $y_m \in R^k$ the factors for the items. The $k \times n$ user matrix X, and the $k \times m$ item matrix Y are then defined by:

$$X = \begin{bmatrix} | & & | \\ x_1 & ... & x_n \\ | & & | \end{bmatrix}, Y = \begin{bmatrix} | & & | \\ y_1 & ... & y_n \\ | & & | \end{bmatrix}$$

Our goal is then to estimate the complete ratings matrix $R \approx X^T Y$. We can formulate this problem as an optimization problem in which we aim to minimize an objective function and find optimal X and Y. In particular, we aim to minimize the least squares error of the observed ratings(and regularize):

$$\min_{X,Y} \sum_{r_{ui} observed} (r_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

Notice that this objective is non-convex (because of the $x_u^T y_i$ term); in fact its NP-hard to optimize. Gradient descent can be used as an approximate approach here, however it turns out to be slow and costs lots of iterations. Note however, that if we fix the set of variables X and treat them as constants, then the objective is a convex function of Y and vice versa. ALS will therefore be to fix Y and optimize X, then fix X and optimize Y , and repeat until convergence.

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 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 matrices matrices P (of size $|U| \times |K|$) and Q (of size $|D| \times |K|$) such that their product approximates $|R|$:

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

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 their vectors:

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

Now, we have to find a way to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is the first initialize 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. Usually we will choose a small value for α, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

A question might have come to your mind by now: if we find two matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{P} \times \mathbf{Q}$ approximates $\mathbf{R}$, isn't that our predictions of all the unseen ratings will be zeros? In fact, we are not really trying to come up with $\mathbf{P}$ and $\mathbf{Q}$ such that we can reproduce $\mathbf{R}$ exactly. Instead, we will only try to minimize the errors of the observed user-item pairs. In other words, if we let T be a set of tuples, each of which is in the form of (u_i, d_j, r_{ij}), such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimize every e_{ij} for $(u_i, d_j, r_{ij}) \in T$. (In other words, T is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learned.

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.

$$E = \sum_{(u_i,d_j,r_{ij}) \in T} e_{ij} = \sum_{(u_i,d_j,r_{ij}) \in T} (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

III. Experiments

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

a) Using 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 $\hat{R}_{n\_unsers,n\_items}$ is obtained by multiplying the user factor matrix $P_{n\_users,K}$ and the transpose of the item factor matrix $Q_{n\_item,K}$.

b) Using SGD: Read the data set and divide it. Populate the original scoring matrix R_{n_unsers,n_items} against the raw data, and fill 0 for null values. Initialize the user factor matrix $P_{n\_users,K}$ and the item (movie) factor matrix $Q_{n\_items,K}$, 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 $P_{n\_users,K}$ and $Q_{n\_items,K}$; 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 $\hat{R}_{n\_users,n\_items}$ is obtained by multiplying the user factor matrix $P_{n\_users,K}$ and the transpose of the item factor matrix $Q_{n\_items,K}$.

C. Discussion

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

As for initialization of the weight of each samples,we assigned random value to the initial matrix.

The parameters we choose in this experiment is in Table I.

The experiment results are shown in Fig. 1

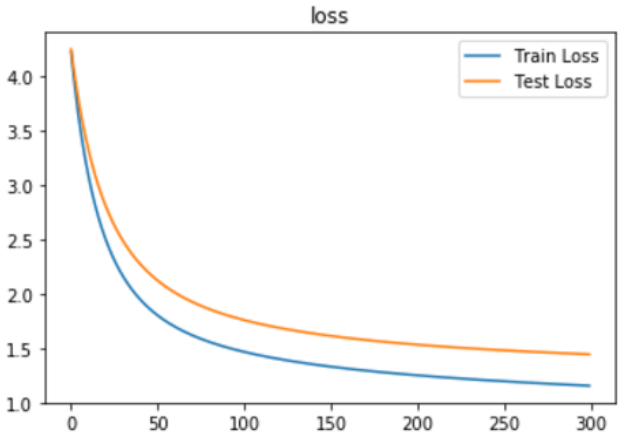

Fig. 1. The loss of matrix factorization algorithm.

TABLE I
MODEL PARAMETERS

Iteration Time	$steps = 300$
Number of Implicit Factor	$K = 20$
Learning Rate	$\alpha = 0.00001$
Regularization Item	$\beta = 0.02$
Number of Implicit Factor	$K = 20$

IV. CONCLUSION

In this paper, we focus on matrix factorization approach for recommender system. We explore two algorithms ALS and SGD to implement matrix factorization. We conduct experiments on real life dataset and analysis the results. We gain a better understanding of the matrix factorization algorithms and application scenarios.