South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Zhifan Xu, Haotian Huang and Kai Song

Supervisor:
Qingyao Wu

Student ID:
201530613269,201530611708,201530612729

Grade:
Undergraduate

December 28, 2017

Recommender System Based on Matrix Decomposition

Abstract*—*To explore the construction of recommended system., we used matrix decomposition by gradient decent to construct a recommendation system under small-scale dataset, cultivate engineering ability.

Ⅰ. INTRODUCTION

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item.

Recommendation systems typically produce a list of recommendations in one of two ways – through collaborative filtering or through content-based filtering. Collaborative filtering approaches build a model from a user's past behavior as well as similar decisions made by other users. This model is then used to predict items that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. These approaches are often combined.

Ⅱ. METHODS AND THEORY

A. Alternating Least Square(ALS)

A common problem faced by internet companies is that of recommending new products to users in personalized settings (e.g. Amazon's product recommender system, and Net ix movie recommendations). This can be formulated as a learning problem in which we are given the ratings that users have given certain items and are tasked with predicting their ratings for the rest of the items. Formally, if there are n users and m items, we are given an n * m matrix R in which the (u; i) the entry is rui {the rating for item i by user u. Matrix R has many missing entries indicating unobserved ratings, and our task is to estimate these unobserved ratings.

Our goal is then to estimate the complete ratings matrix R _ XT 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:

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

Notice that this objective is non-convex ; in fact it's 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 _x the set of variables X and treat them as constants, then the objective is a convex function of Y and vice versa. Our approach will 1 therefore be to _x Y and optimize X, then fix X and optimize Y , and repeat until convergence. This approach is known as **ALS(Alternating Least Squares)**. For our objective function, the alternating least squares algorithm is as follows:

Algorithm 1 ALS for Matrix Completion

```
Initialize $X, Y$
repeat
  for $u = 1 \dots n$ do
      $x_u = \Big(\sum_{r_{ui} \in r_{u*}} y_i y_i^{\mathsf{T}} + \lambda I_k\Big)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} y_i$    (2)
  end for
  for $i = 1 \dots m$ do
      $y_i = \Big(\sum_{r_{ui} \in r_{*i}} x_u x_u^{\mathsf{T}} + \lambda I_k\Big)^{-1} \sum_{r_{ui} \in r_{*i}} r_{ui} x_u$    (3)
  end for
until convergence
```

For a single machine we can analyze the computational cost of this algorithm. Updating each xu will cost O(nuk2 +k3) where nu is the number of items rated by user u, and similarly updating each yi will cost O(nik2 + k3) where ni is the number of users that have rated item i.

Once we've computed the matrices X and Y , there are several ways to do prediction. The first is to do what was discussed before, which is to simply predict rui = xTu yi for each user u and item i. This approach will cost O(nmk) if we'd like to estimate every user-item pair. However, this approach is prohibitively expensive for most real-world datasets. A second (and more holistic) approach is to use the xu and yi as features in another learning algorithm, incorporating these features with others that are relevant to the prediction task.

B. SGD in Recommender Systems

SGD could be used to perform matrix factorization[3], a collaborative filtering technique used in recommender systems. The idea is as follow: from ratings, we can make a sparse matrix, called rating matrix and indicated as R, where rows of this are users and columns are items. Matrix factorization founds two matrix, each one with dimensionality less than R, whose dot product approximate the rating matrix, by using only non-empty entries in the computation. This two matrices, that we indicate as P and Q, are respectively representations of users features and items features.

This could be described as a minimization problem, where we want to minimize the error between the rating matrix and the approximation matrix. The objective function of the problem that we could use is the regularized square loss, expressed as:

$$L(P,Q) = \sum_{(i,j)\in P,Q} (r_{ij} - p_i q_j)^2 + \lambda(\|P\|_F^2 + \|Q\|_F^2)$$

where ||·||F is the Frobenius norm and 0 is a regularization factor to avoid overfitting[4]. Because this is a minimization problem, we can apply SGD to solve it. This is performed by firstly computing the error between a training point in the rating matrix and its approximation:

$$e_{ij} = r_{ij} - p_i q_j$$

and then using this value to update matrices P and Q:

$$p_i = p_i + \mu(e_{ij} q_j - \lambda p_i)$$

$$q_j = q_j + \mu(e_{ij} p_i - \lambda q_j)$$

This operations are performed for each training point in the rating matrix for each epoch. Implement a parallel version of this algorithm is not an easy operation. Indeed, each update of P and Q influences subsequent updates.

Provide a lock-free approach to implement SGD in a parallel environment. Parallel thread will randomly select elements from the rating matrix, 3 and will compute stochastic updates on them. Due to the fact that R is highly sparse, the probability that two independent threads will select two elements that share the same row or column is very small. Then, we can have a very small error introduction into the computation.

III.EXPERIMENTS

A. Data set

This experiment consist 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. The u1.base / u1.test are train set and validation set respectively, separated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base / u1.test to u5.base / u5.test.

B. Implementation

The steps are:

(1)Read the data set and divide it. Populate the original scoring matrix R_{n_users,n_items} against the raw data, and fill 0 for null values.

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

(3)Determine the loss function and hyperparameter learning rate η and the penalty factor λ .

(4)Use the stochastic gradient descent 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;

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 $Q_{n\_item,K}$;

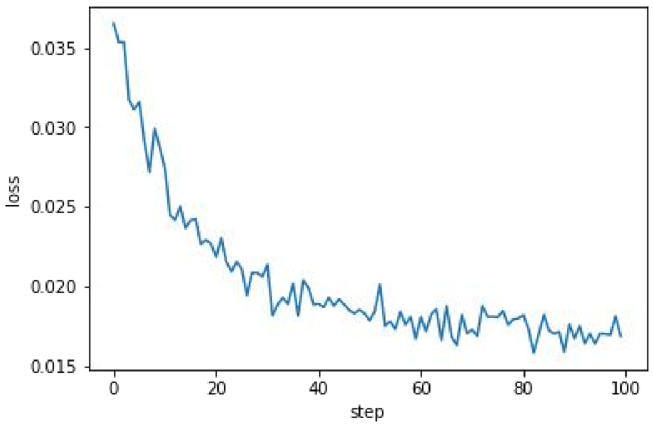
4.4 Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged.

(5)Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix , Draw a $L_{validation}$ curve with varying iterations.

(6)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 $Q_{n\_item,K}$.

C. Results and analysis:

In this experiment we used the SGD algorithm. The hyper-parameter K we used is 10. The loss curve of this algorithm as Fig.1, And we can see that the convergence speed in SGD is a little slow and the fluctuation of the curve is larger.



IV. CONCLUSION

we have a deeper understanding of the SGD algorithm and the basic idea of recommendation system after this experiment, implementing it in python. What's more, writing the code and analyzing the results make us know that recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. Finally, teamwork makes us work more efficiently and cushily.