EPFL Machine Learning CS-433 - Project II Recommender System

WANG Qimin
School of Architecture, Civil and Environmental Engineering
Email: qimin.wang@epfl.ch

CUI Mingbo
School of Microengineering Robotics
Email: mingbo.cui@epfl.ch

HUANG Ruibin
School of Computer and Communication Sciences
Email: ruibin.huang@epfl.ch

***Abstract*—This paper applied matrix factorization (MF) on the rating matrix to construct the recommender system. The latent features of users and items were updated based on stochastic gradient descent (SGD), and the user-item rating matrix is reconstructed using the inner product of the latent features. Then the regularizers and the biases of users' and items' features are introduced into the model respectively, for improving the model accuracy and robustness. To tune the hyperparameters in the recommender system, grid search have been conducted, concerning the latent feature size (K), the learning rate (γ) and the regularizers (λ_u, λ_i). Some improvement in initializing the latent features was made and the result was analyzed. In the end, some metrics to sophisticate the recommender system was proposed for future studies.**

INTRODUCTION

Companies using recommender systems focus on increasing sales as a result of providing personalized suggestions and enhancing customer experience. Recommendations typically speed up searches and make it easier for users to access content they're interested in. Meanwhile, companies are able to gain and retain customers by sending out emails with links to new offers that meet the recipients interests, or suggestions of films and TV shows that suit their profiles.

Recommender systems mainly use two types of information:

1) Characteristic information. This includes information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
2) User-item interactions. This mainly focuses on ratings, likes, purchases, etc.

In the project, we devote ourselves to building a recommender system using the user-item rating matrix. Comparing to models based on the first type of information, this system can recommend products in which the user has not shown an interest before.

The paper is organized as follows. Firstly, the statistics of the dataset is introduced. Then we present the implementation of models of the regularized matrix factorization and the biased matrix factorization. Comprehensive parameter tuning processes are realized with grid search, which is followed by the results analysis and model selection. In the end, we have concluded the process of building the recommender system and proposed some new evaluation metrics to determine the order of a recommendations list for future research.

TABLE 1
DATASET DESCRIPTION

Attribute	Sample value	Meaning
Id	r44_c1	user id and movie id
Prediction	3	rating

TABLE 2
STATISTICS OF THE USER-ITEM RATINGS MATRIX

number of non-zero ratings	1,176,952
number of users	10,000
number of movies	1,000
average rating	3.86
median rating	4
matrix sparsity	88.23%

I. DATASET DESCRIPTION

The given dataset is composed of around 1.2 million valid non-zero ratings. In **data_train.csv**, each record contains 2 attributes, **Id** and **Prediction**, as shown in Table1. **Id** consists of a row id (user id) and a column id (movie id), e.g. r44_c1, which means that the movie 1 got a rating form the user 44. **Prediction** is the ratings corresponding to **Id** ranging from 1 to 5 by integer. The user-item ratings matrix X was constructed by **Id**. To be consistent with the courses, X were transposed for further analysis and was now a $D \times N$ matrix where D is the number of movies and N is the number of users.

In this user-item ratings matrix, only 11.76% data are non-zero, as shown in Table 2. Therefore **scipy** library was used for storing the sparse matrix and accelerating computation.

The number of ratings per movie and per user is shown in Fig.1. 50% users have rated over 100 items and over 80% items have ratings. So it is reasonable to take into consideration all the users and movies in the dataset. The data was split into a training set and a test set. In the training set, there were 90% data to guarantee all the items id and user id were included.

As we can see from Fig.2, the dominant ratings range from 3 to 5. One possible reason is that most users are tended to give a higher rating. Therefore, we considered building a biased matrix factorization model. The details will be discussed later.

TABLE 3
GRID SEARCH RESULTS OF THE REGULARIZED MF MODEL

λ_u \ λ_i	K=10				K=20				K=50			
	0.001	0.01	0.1	1	0.001	0.01	0.1	1	0.001	0.01	0.1	1
0.001	1.011	1.004	0.987	0.999	1.030	1.001	1.000	1.017	1.020	0.986	1.001	1.018
0.01	1.011	1.003	0.982	1.010	1.027	0.998	1.000	1.017	1.015	0.985	1.001	1.017
0.1	1.002	0.986	1.003	1.050	1.006	0.983	1.004	1.050	0.990	0.984	1.004	1.050
1	1.007	1.007	1.048	1.416	1.005	1.007	1.048	1.416	1.006	1.007	1.048	1.416

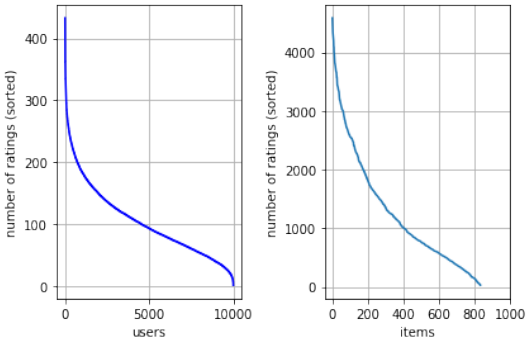

Fig. 1. Number of ratings per movie and user

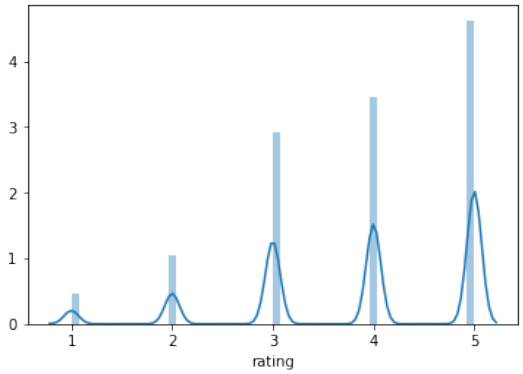

Fig. 2. Ratings distribution

II. MODEL OF REGULARIZED MATRIX FACTORIZATION

A. Model establishing

By using matrix factorization, characteristics of users and items are mapped into the K-dimensional joint latent feature space, and the user-item rating matrix is modeled as inner products in the latent factor space. Every item would be associated with the vector $w_i \in \mathbb{R}^K, i = 1, 2, \cdots, D$ and every user would also be associated with the vector $z_u \in \mathbb{R}^K, u = 1, 2, \cdots, N$.

For a given item i, the elements of w_i measures the extent to which the item possesses those factors, positive or negative. For a given user u, the elements of z_u measure the extent of interest the user has in items that are high on the corresponding factors[1]. The resulting dot product, $w_i^T \cdot z_u$, represents the interaction between user u and item i, namely the overall interest of a certain user in a specific items characteristics. This approximates the rating of user u to item i, which is denoted by $x_{i,u}$. The final goal of matrix factorization is to find matrix W and Z, which will enable the model to make the best approximation:

$$X \approx WZ^T \tag{1}$$

To explain every entry in X, namely $x_{i,u}$ with the inner product of the generated movie feature vector and the user feature vector, we have to find the best feature vector first. Minimizing the squared error between predictions and training data is the most common way:

$$\min_{W,Z} \mathbb{L}(W, Z) := \frac{1}{2|\Omega|} \sum_{(i,u)\in\Omega} [x_{i,u} - (WZ^T)_{i,u}]^2, \tag{2}$$

where Ω is the set of pairs (i, u) with known ratings.

As aforementioned, K is the number of latent features. If K is very big , it will give rise to overfitting. To avoid overfitting, we added the regularized item to find a trade-off:

$$\min_{W,Z} \mathbb{L}(W, Z) := \frac{1}{2|\Omega|} \sum_{(i,u)\in\Omega} [x_{i,u} - (WZ^T)_{i,u}]^2 + \lambda_i ||W||_F^2 + \lambda_u ||Z||_F^2, \tag{3}$$

where λ_i and λ_u both serve as regularized terms.

To minimize Equation 3, we used the Stochastic Gradient Descent (SGD). Differs from other applications using SGD, we could not initialize the latent feature matrix W and Z with zeros. Because multiplication of two matrices filled with zeros will always be a matrix with zeros, which will not enable us to learn anything. Given that, We initialize W and Z with i.i.d. (independent and identically distributed) random variables, whose value uniformly distributes between 0 and 1.

To get the gradient of W, we fix Z and to get the gradient of Z, we fix W . The latent features are updated as

$$\begin{aligned} w_i &= w_i + \gamma \cdot (e_{i,u} \cdot z_u - \lambda_i \cdot w_i) \\ z_u &= z_u + \gamma \cdot (e_{i,u} \cdot w_i - \lambda_u \cdot z_u) \end{aligned}, \tag{4}$$

where $e_{i,u} = x_{i,u} - [WZ^T]_{i,u}$.

B. Grid Search for Hyperparameters

We tried to use the library of Surprise[2] to grid search hyperparameters with 5-folds cross-validation, the optimized model had pretty good performance on the test dataset with RMSE (root mean square error) less than 0.9, but performed not well when we submitted the prediction on crowdAI, in which prediction the RMSE was 1.055. Therefore, we reasoned that there was somehow overfitting even that we had

added regularizers to the model. This is also the motivation that we did the grid search by ourselves. Given that our limited computational resource, we removed the cross-validation in the grid search of hyperparameters.

As aforementioned, we have mainly four hyperparameters in this model, that is, the learning rate γ , the latent feature size K, the regularizers λ_u and λ_i. To have an optimized model performance, we have to find the best combinations of all the hyperparameters. The grid search is a basic but useful approach. To reduce the complexity of grid search, we set the proper $\gamma = 0.025$ according to our experience, then we took a three nested for-loops to search for λ_i, λ_u, and K. Note that λ_i and λ_u were searched in the log-scale because we are more interested in the magnitude rather than the exact values for such type of hyperparameters. Please refer to our codes for the details about parameters tuning.

Indicated by Table 3, the optimal combinations of hyperparameters is that $K = 20$, $\lambda_u = 0.1$, $\lambda_i = 0.01$.

However, a better tuning way is to tune γ again by fixing the optimized λ_i, λ_u, and K and to repeat this process until all parameters converges under some threshold, as shown in Fig.3. Due to the limit of time and computation resources, this idea was not realized.

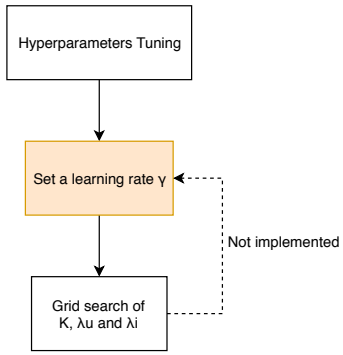

Fig. 3. The process of tuning hyperparameters

III. Model of Biased Matrix Factorization

To better model the differences from users to users and from items to items, we want to identify the portion of these values which the biases could explain. The biases are expressed as

$$b_{i,u} = \mu + b_i + b_u \tag{5}$$

The purpose of adding bias terms is to eliminate the effects of difference from users or movies: if there is a user who likes to rate all movies 1 star higher than average ratings, in order to compare his/her ratings with others, we should remove 1 star to all of his/her ratings.

The bias $b_{i,u}$, involved in $x_{i,u}$, accounts the effects of user u and movie i. μ denotes the overall average of the whole non-zero ratings; b_u and b_i represent the average deviations of ratings of user u and item i. To make this more clear, let us quote the famous example here again[1]: suppose that you want a first-order estimate for user Joes rating of the movie Titanic. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average, namely $b_i = 0.5$. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average, namely $b_u = -0.3$. Thus, the estimate for Titanics rating by Joe would be 3.9 stars ($\mu + b_i + b_u = 3.7 + 0.5 - 0.3$). So by adding bias, we could get the final evaluation of $\hat{x_{i,u}}$ as follows:

$$\hat{x_{i,u}} = \mu + b_i + b_u + w_i^T \cdot z_u. \tag{6}$$

It is worth noting that the global average and items/users bias, computed from training set should also be used on the testing set and the prediction set since by theory we have assumed all the data are from the same distribution and only the training set is known.

The item-user interaction matrix of the biased MF is composed of 4 parts: global average, items bias, users bias, and the inner product of latent features. This decomposition could enable every part to explain different signal respectively. We minimize the mean squared error again to learn the feature vector after we added the bias

$$\begin{aligned}\min_{W,Z} \mathbb{L}(W,Z) := \frac{1}{2|\Omega|} \sum_{(i,u)\in\Omega} [x_{i,u} - (WZ^T)_{i,u}]^2 \\ +\lambda_w(||W||_F^2 + b_i^2) + \lambda_z(||Z||_F^2 + b_u^2)\end{aligned}. \tag{7}$$

Table 4 shows the result of grid search for the hyperparameters of the biased MF. The optimized parameters set is that $K = 50$, $\lambda_u = 0.1$, $\lambda_i = 0.01$.

IV. Model Selection and Final Submission

A. Model Initialization

For the final submission, we reviewed our codes and found some improvement points in initializing the latent features matrix. Line 11 and Line 12 in the original function **init_MF** in **SGD_helpers.py** was as follows.

```
def init_MF(train, num_features):                          6
    """init the parameter for matrix                       7
        factorization."""
                                                           8
    num_item, num_user = train.get_shape()                 9
                                                           10
    user_features = np.random.rand(                        11
        num_features, num_user)
    item_features = np.random.rand(                        12
        num_features, num_item)
```

we have the expectation of $x_{i,u}$ in the ratings matrix X as

$$\begin{aligned}E(x_{i,u}) &= E\left(w_i \cdot z_u^T\right) \\ &= E\left(\sum_{k=1}^{K} w_{i,k} z_{u,k}\right) \\ &= K \cdot E(w_{i,k})\, E(z_{u,k}) \\ &= \frac{K}{4},\end{aligned} \tag{8}$$

TABLE 4
GRID SEARCH RESULTS OF THE BIASED MF MODEL

λ_u \ λ_i	K=20				K=50				K=100			
	0.001	0.01	0.1	1	0.001	0.01	0.1	1	0.001	0.01	0.1	1
0.001	NAN	NAN	1.033	0.987	NAN	1.201	1.082	0.989	1.317	1.249	1.089	0.989
0.01	1.084	1.056	0.996	0.998	1.205	1.126	1.004	0.997	1.257	1.141	1.004	0.998
0.1	1.037	0.985	0.997	1.003	1.063	0.984	0.996	1.003	1.070	0.985	0.995	1.003
1	0.995	1.003	1.003	1.003	0.994	1.003	1.003	1.003	0.994	1.003	1.003	1.003

and the variation of $x_{i,u}$ in the ratings matrix X as

$$\begin{aligned} Var\left(x_{i,u}\right) &= Var\left(w_i \cdot z_u^T\right) \\ &= Var\left(\sum_{k=1}^{K} w_{i,k} z_{u,k}\right) \\ &= K^2 \cdot Var\left(w_{i,k}\right) Var\left(z_{u,k}\right) \\ &= \frac{K^2}{144}, \end{aligned} \tag{9}$$

given that the initial latent features matrix W and Z and every element inside them obey i.i.d. uniform distribution in $[0, 1]$.

As the number of latent features K increase, so does the mean rating value and its variation from the inner product; However, the rating has an upper bound, but K has not. If so, the constant learning rate γ in our model is not globally optimized. The online RMSE score using that code is 1.045.

In a later version of our codes, we introduced the number of latent features to control the expansion of the rating matrix. Line 11 and Line 12 has been revised as follows:

```
11      user_features = np.random.rand(
            num_features, num_user)/num_features
12      item_features = np.random.rand(
            num_features, num_item)/num_features
```

Therefore, the initial value of ratings got constraint to the real values, which could be crucial for the limited training like 50 epochs. The online RMSE score using this code is 1.025.

B. With or without bias

Indicated by Table 3 and Table 4, the recommender system using bias MF should have a better performance on the prediction set. However, crowdAI, the results submission platform reported higher RMSE of 1.073 using bias MF, which is worse than the performance of model using regularized MF, which has an RMSE of 1.045 (before changing model initialization). This might result from the fact that we did not utilize cross validation due to the limited computational resources.

Eventually, after correcting model initialization with the number of latent features, our online prediction performance increased to 1.025 and the results have been validated on MacOS and Windows.

V. CONCLUSION

We have implemented different models and approaches to achieve the best result in this project, but we have also given up for some choices. For example, to have a faster optimization process, we have also explored the alternating least-squares (ALS). However, the ALS model we tried performed worse than the one trained by SGD. In the end, we still use SGD to train our model. The model by the bias MF should give better performance theoretically, but it failed to convince us with the worse results on crowdAI, so finally we chose to use the model of regularized MF.

One of the biggest challenges we had in the project was the limited computational resource available on our own computer. Given that, we had o change the process of grid search by fixing the learning rate (γ).

VI. FUTURE WORK

To put our recommender system into practical use, a better metric for evaluation is required. RMSE only tells us how accurate our recommendations are but it does not focus on the order of recommendations, i.e. they do not focus on which items (movies) to recommend first and what follows after that. We need some other metric that also considers the order of the items recommended. For instance, by using Mean Reciprocal Rank (MRR), the list of recommendations can be evaluated as

$$MRR = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{R_i} \tag{10}$$

where R_i denotes the rank of item i in a n-item recommendations list to a user.

Suppose we have recommended 5 items in order (Item A to Item E) to a user, but the user only liked Item E. So the reciprocal rank of Item E will be $\frac{1}{5}$ and the mean reciprocal rank for the given recommendations list is $\frac{1}{25}$. The larger the MRR, the better the recommendations.

Up to now, we have learned how to build a basic recommender system directly based on the user-item ratings matrix. Although matrix factorization cannot provide recommendations for new items if there are no user ratings upon which to base a prediction, it will take some time before the item has received enough ratings in order to make accurate recommendations. A system that combines content-based filtering and matrix factorization could potentially take advantage of both the representation of the content as well as the similarities among users, which remains as our future work.

REFERENCES

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[2] Surprise library. http://surpriselib.com/.