Machine Learning for Building Movie Recommendation System

Huajian Qiu, Zhantao Deng, Yinan Zhang
Name ”yn” on the leaderboard, Final RMSE = 1.02
EPFL, Switzerland

***Abstract*—This project attempts to predict missing ratings given a sparse rating matrix. We try to solve this problem with various algorithms and build our final model based on Matrix Factorization ALS.**

I. Introduction

For a movie platform, an accurate and flexible recommendation system is very important. Our aim is to recommend suitable movies to users according to their preferences. Since there are a huge number of available movies, good personalized recommendations can make initial choices for customers. Matrix factorization model outperform classic nearest-neighbor methods[1]. Thus, We mainly focus on two matrix factorization approaches: alternating least squares and stochastic gradient descent. In addition, we have tried other collaborative filtering algorithms, such as KNN, matrix factorization SVD, co-clustering, etc from supriselib. Blending different models is a good way to eliminate potential bias of a single model, which will be discussed in detail later.

The remainder of this report is structured as follows. In the second part, we discuss some basic methods that are employed in this project for reproducibility. In the third part, we compare different models and provide rationale for choosing certain parameters. In the fourth part, homogeneous or heterogeneous models are blended in order to do more accurate predictions. Finally, we generate top N movie recommendations for each user and give our opinions about how to make further improvements.

II. Basic methods

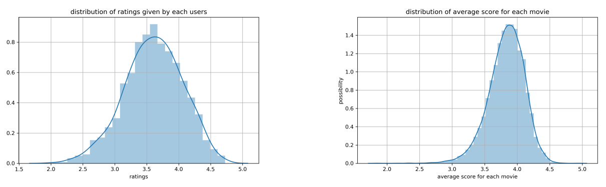
Matrix factorization is an effective and popular means of building recommendation system. User-item interactions are modeled as inner products in a joint latent factor space[1]. The following cost function needs to be minimized:

$$\min_{\mathbf{W},\mathbf{Z}} \quad \frac{1}{2}\sum_{(d,n)\in\Omega}[x_{dn}-(\mathbf{W}\mathbf{Z}^T)_{dn}]^2+\lambda_w\|\mathbf{W}_{Frob}\|^2+ \lambda_z\|\mathbf{Z}_{Frob}\|^2 \tag{1}$$

A. Data Processing

We turn original data into a $D \times N$ sparse matrix, where each row represents a movie and each column represents a user. The sparsity of this rating matrix is 11.77%.

Figure 1 shows how the distribution of the user mean and the item mean. The distribution is close to normal distribution. Both of user mean and item mean can serve as a solid baseline.



(a) The distribution of mean ratings given by each user

(b) The distribution of mean ratings for each item

Figure 1: The distribution of mean ratings

Besides, initialization method of matrix **W** and **Z** has an impact on the result. For alternating least squares algorithm, initial element-wise values of two matrices are assigned by normal distribution random numbers. For stochastic gradient descent algorithm, we generate two random matrices and assign the average rating for each item to the first column of initial item feature.

B. RMSE Calculation

The predictions given by our models are float numbers, which is not the case in reality and affect RMSE calculation results. Integer prediction is given by:

$$\max\{\min\{\lfloor \hat{r}_{ui}+0.5\rfloor,1\},5\} \tag{2}$$

$\hat{r}_{dn}$ is the estimated rating of user d for item n. We only calculate and compare RMSE after rating predictions are transferred to integers. When generating the final submission,nintegers from 1 to 5 are served as predictions.

C. Cross Validation

The implementation of cross validation differs from what we have seen in the class. Because we need to control that the sampling ratio is equal for each column(representing each user). The following schema guarantees that the ratings given by each user is fully represented on our training and testing set.

Rather than equally split the matrix into K parts at one time, we adopt an alternative method to generate K-fold training and testing set. Firstly, we make a copy of the original ratings matrix, and set the sampling ratio as $\frac{1}{K}$. Randomly choosing $\frac{1}{K}$ of nonzero ratings of each user as the first testing set, then we set the selected ratings as zero and treating it as training set. For the second time, we set the sampling ratio as $\frac{1}{K-1}$ and randomly choose $\frac{1}{K-1}\%$ nonzero ratings (attention: entries that have been selected in the last time are set to zero) per user as the second testing set, the original matrix minus the second testings matrix is the second training matrix. After K times of random selections ,we get K-fold cross validation sets. The sampling ratio is calculated by $\frac{1}{K-i}, i = 0, 1, \ldots, K-1$. This process is shown in Figure2.

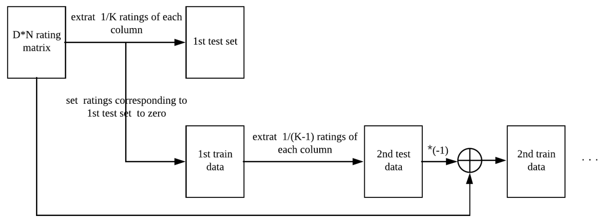

D*N rating matrix
extrat 1/K ratings of each column
1st test set
set ratings corresponding to 1st test set to zero
1st train data
extrat 1/(K-1) ratings of each column
2nd test data
*(-1)
2nd train data
. . .

Figure 2: Schematic diagram of the process of generating crossing validation sets

Initially, 5-fold cross validation was used to estimate the test error when searching for the best parameters. If test errors of all five test sets are similar and small, then it suggests that the associated parameters are reliable. In this way, we make full use of the available ratings. However, it takes more than half an hour to run only one set of parameters, for example, one combination of K, λ_w, λ_z for the Matrix Factorization SGD algorithm. Moreover, it is found that test errors of different test sets are very close to each other. Because we are dealing with a large amount of data , e.g. 117246 test ratings if splitting 10% as test data, it makes good sense to estimate true RMSE using the test set without doing cross-validation. So, we simply split 90% as train data and 10% as test data to search for optimal parameters.

D. *Baselines*

There are three baselines: global mean, user mean and item mean. The first one use the mean value of all ratings to fill in the missing values. The second one and the third one use the mean rating for each user or each item to fill in the corresponding missing values. A valid model should be superior to all three baselines. As shown in TableI predictions generated by user mean has the smallest RMSE, while global mean has the worst performance.

III. Models

For each model, it is important to search for optimal parameters. TableI shows the best version of each model and chosen parameters.

A. *Matrix Factorization SGD*

Stochastic gradient descent is an ideal approach because of less computational complexity. For each nonzero element x_{dn} of the train set, the corresponding feature vectors of item d and user n will be updated. The number of iterations is set to be 50 to ensure that the algorithm has converged. However, it is too ambitious to look for the global minimum and we often end up with a local minimum without information about degree of suboptimality.

We attempted looking for the best parameters for SGD adopting 5-fold cross validation. Initially, we set a fixed λ_w and a fixed λ_z in order to search for the best k value. Then we fix another two parameters and redo this process. Step by step, we will reach the optimal value in theory. However, doing cross validation cost lots of time. SGD perform worse than ALS if assigned the same parameters. So we discard this way when building our final model.

B. *Matrix Factorization ALS*

It is found that alternating least squares is better than stochastic gradient descent when assigning the same set of parameters. But it has more computational complexity and consequently needs more calculation time. Assuming either the item feature or the user feature is known, we use the known one to update the unknown one. A more strict convergence condition will more likely lead to a smaller RMSE result. The total number of iterations is set to be 50, and it is not claimed that the algorithm has converged and jump out of the loop until the difference in RMSE is less than 10^{-5}. λ_w and λ_z are set to be 0.02 and 0.2 respectively, the default numbers of surpriselib. The number of features k is tuned from 1 to 30.

Figure3 shows that as the number of latent features become larger, the train error become smaller but the test error reach its minimum when k equals 8 and then starts to go up. We underfit our data when k is small and overfit when k is large.

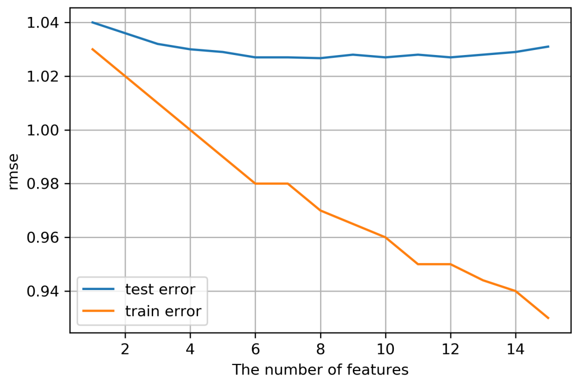


Figure 3: Train error vs Test error, k from 1 to 15

C. Other models

We employ some models in surpriselib, such as KNN Baseline, matrix factorization SVD and co-clutering. KNN give predictions based on a similarity measure. SVD algorithm considers bias for each user and each item, compared to the matrix factorization method mentioned above. Co-clustering algorithm assigns clusters to users and items and make predictions.

# Method	RMSE
Global Mean	1.131
Item Mean	1.120
User Mean	1.071
Matrix Factorization (ALS)	1.027
Matrix Factorization (SGD)	1.043
KNN Baselines	1.0148
KNNwithmeans	1.027
Slopeone	1.004
SVD	1.033
Co-clustering	1.0183
Homogeneous Blending	1.02

Table I: Comparison of various models

[1]

IV. Blending

A. Homogeneous Blending

Each model cannot make perfect predictions and has inevitable bias. Instead of relying on one single model, we combine rating predictions generated by the same ALS algorithm but with different parameters. Here, we also combine user mean baseline and item mean baseline. Linear regression is employed between predicted ratings and true ratings on the test set. It is an effective way to improve the reliability of the model. The RMSE after blending homogeneous models on the test set is , less than the best single model.

B. heterogeneous Blending

It is natural to consider blending heterogeneous models and test if this way can further reduce RMSE. Because these models are built based on distinct principles, they may be able to complement each other. It should be noticed that models other than ALS and SGD are generated by surpriselib, so while blending these models, linear regression must be done on the same test set.

But we find that the models generated from surpriselib perform even worser than user mean baseline (notice that RMSE of surpriselib models in TableI are not calculated by integral ratings). So we use homogeneous models to build our final model.

[1]k refers to the number of latent features, only meaningful for ALS and SGD. The RMSE is obtained from local test set. Note that surpriselib do not generate RMSE of integer ratings. The difference between two is about 0.1.

V. Final CrowdAI Model

Although hybrid model can reduce RMSE and improve accuracy, there seem to be stagnation in this improvement. We have tried to incorporate sixty models, including SGD, ALS, baselines, KNN, SVD, etc. Although there are a slight improvement in RMSE on the local test data, the result given by CrowdAI does not change. We are skeptical that this minor change results from the fact that linear regression is done on the local test data. When ensuring the same performance, we want to build our final model as simply as possible. We blend 13 models in total: 11 from ALS, the other two are user mean baseline and item mean baseline respectively. λ_w equals 0.02, λ_z equals 0.2. The number of features range from 3 to 13. We get the final RMSE 1.02 on the leaderboard.

VI. Top N recommendations

For a real recommendation system, the acquisition of a rating matrix is just the initial step. We have tried a simple application: generating top 5 movies for each user as described in Figure4. We recommend movies that are assumed to get best ratings from a certain user. In this way, we can get some feedback from the customer and update our rating matrix. Movies that have been watched will not be presented in the next time.

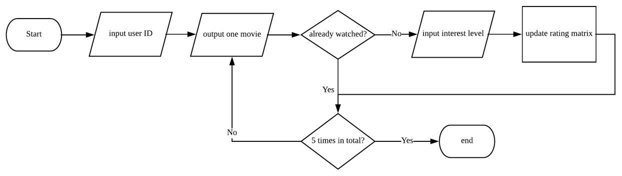


Figure 4: The flow chart of generating top 5 movies to one user

VII. Conclusion

Considering the prediction accuracy of a single model, alternating least squares has the best performance. Blending different models can lower the RMSE to some extent, but the best single model decides how well it can be. If possible, a finer grid search for parameters should apply to each model. We wanted to assign a different regularization parameter to each item and each user, but it needs huge amount of computational resources. Additionally, customers change their taste as time goes by. A more complex dynamic model should be set up to track the time-varying need of customers for practical application.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[2] N. Hug, “Surprise, a Python library for recommender systems,” http://surpriselib.com.

[3] V. Faramond, “Recommender system challenge for machine learning course,” https://github.com/vfaramond/ML-Project2.