None-negative Matrix Factorization on Sparse Matrix in the Field of Recommender System

Shunyu Yao

June 16, 2022

Abstract

None-negative Matrix Factorization is a method that can decompose a matrix into two or more matrices which can be multiplied together to obtain the original matrix. Specifically, in the field of recommender system, we split the sparse rating matrix into two matrices, denoted as V = w * h. As a latent factor model, the equation has two matrices with semantic meaning. The first matrix, w, represents the weight for users' preferences, while the second matrix, h, represents the scores each item has on those features, when multiply, it generate a matrix that indicate all users' estimated rating on every movie. After I done my research on previous models, I developed my NMF model with the feature of ignore zeroes, initialize specially according to the input and has the potential to train with multi-thread, and then I classified users into different groups and trained my model on them respectively, and plot the distribution of different groups of users as well as specific users.

Contents

1 Previous model vs. my NMF model

1.1 Sklearn, TorchNMF and Surprise

Model	RMSE loss	Speed
Sklearn	2.2300	13.86s
Surprise	0.9359	37.22s
TorchNMF	2.8691	2.47s

Figure 1: Compareation between previous models

I chose MovieLens100k as my dataset and tested several libraries on it. MovieLens100k[1] is a stable benchmark dataset, which has 100,000 ratings from 1000 users on 1700 movies, released 4/1998. I trained Sklearn model, TorchNMF model and Surprise model on this dataset, and use the root mean square error (RMSE) loss as the criterion: $RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\textbf{ Predicted }_i - \textbf{ Actual }_i)^2}{N}}$

As is shown in the graph above, Sklearn[4] and TorchNMF[3] both have a very large loss, because both model cannot ignore those zeroes in the input matrix. In the MovieLens100k dataset, the sparse ratio is 5%, which means every user interact with 5% of movies in average. And these two models cannot ignore those misleading zeroes, and treated them as if users hate 95% of movies and rated them zero point.

As for Surprise[5], it is the abbreviation for Simple Python Recommendation System Engine which is a famous library built for this kind of stuff. It can fit the original matrix very well for it can ignore those zeroes, nevertheless, it does not run as fast as Sklearn and TorchNMF, and can utilize only one core of the CPU. So, in the following part of my paper, I will compare it with my model.

1.2 my model

In my model, there are several techniques I used to improve the quality of recommendation.

The first is sparse matrix. I use sparse matrix[2] from science python to store the original matrix, this special data structure can accelerate the process of accessing entries and calculating. In stead of a two dimensional list, it can compressed the sparse matrix according to rows or columns for different occasion. It can largely reduce the memory my program take, and can access entries faster.

The second is zeroes-mask. I focus on those none zero entries, and calculate gradient on them, to make sure my model can understand the task.

I use special initialization techniques. Because the ratings range from 1 to 5 points, so I initialized the two matrices w and h, so after multiply the matrix V should be full of 3. So, it will convergent faster.

1.2.1 Objective Function

Denote RM as the actual rating matrix, w as the weight matrix, h as the feature matrix, E is the set of all non-zero entries, k is the length of w and the height of h, which is a hyper parameter, for each [u][i] in E, the relation is: $\mathrm{RM}[u][i] = \sum_{j=1}^{k} w[u][i]h[j][i]$

My objective function is: $F = \sum_{v,i \in E}(RM[u][i] - ERM[u][i]))^2$,

and the gradient for each none-zero entry is:

$$\begin{aligned}\frac{\partial F}{\partial w[u][o]} &= \frac{\partial (RM[u][i] - ERM[u][i])^2}{\partial w[u][i]} \\ &= 2\ \mathrm{Error[u][i]}\ * h[j][i]\end{aligned}$$

1.2.2 RMSE loss

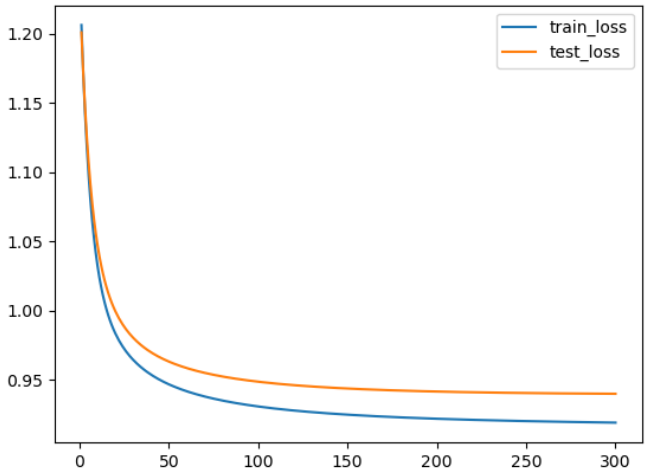

Figure 2: RMSE loss of my model during training

After 300 iterations, it reached 0.939 on the MovieLens100k datasets. As it is shown in the picture, although no regularization was added in the objective function, the test loss is still declining after 300 iterations, it is safe to say my model does not over fitting at 300 epochs, still a regularization should be include in order to reach a better result.

1.2.3 Versus surprise

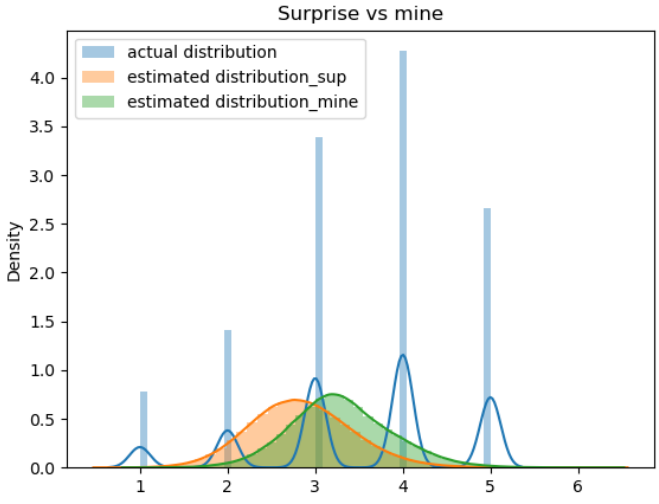

Figure 3: Distribution of actual ratings, Suprise, and mine

Because a lower loss on the test sets does not necessarily lead to a better recommendation, I examed the distribution of all users on on movies.

As it is shown in the graph, the peak of my model is a little bit over 3 and the peak of surprise library is a little bit lower than 3.

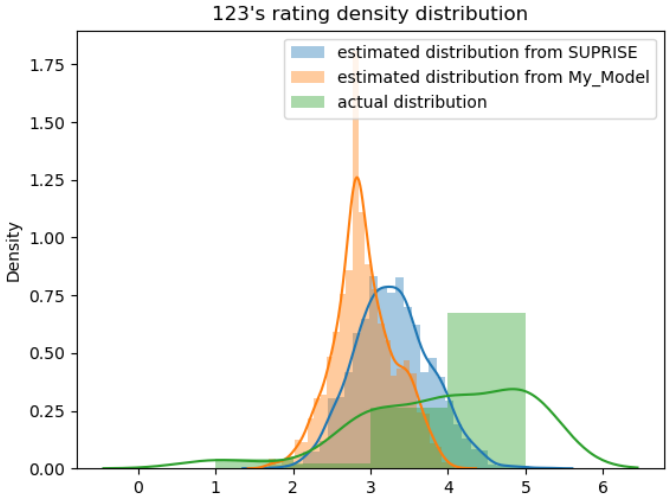

Figure 4: Distribution of user number 123

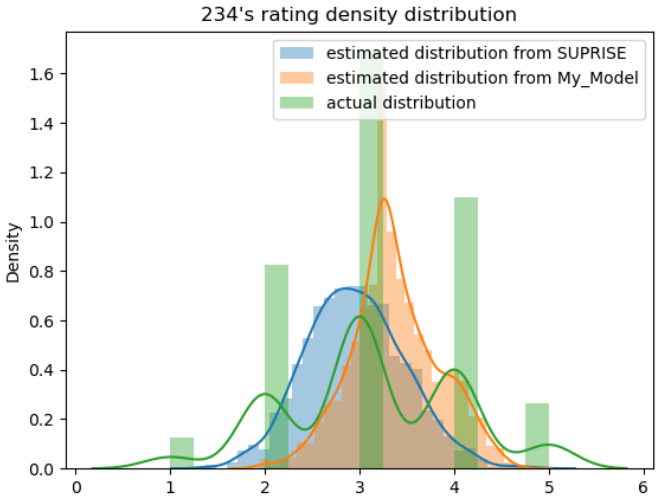

Figure 5: Distribution of user number 234

As I examed the distribution of random user, it turns out that my model indeed tend to give high ratings than the surprise model, which tend to give negative scores.

2 Classify users into groups

Because user varies from one another, it makes even harder to capture all users' preferences with one model. So, I classify users into different groups according their preferences and train my model on them separately.

2.1 K means

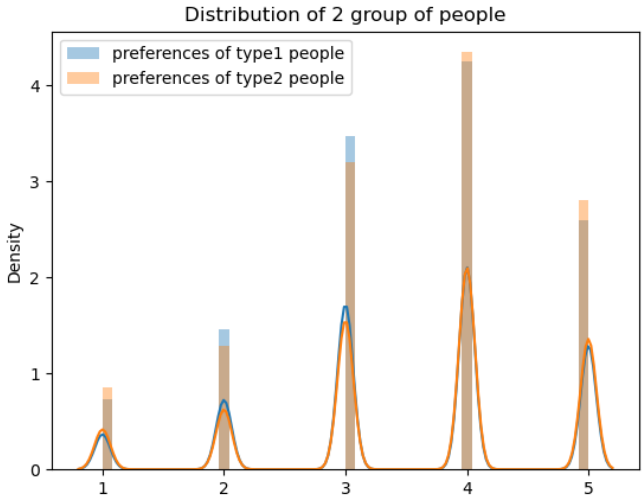

Figure 6: Distribution of 2 group of people

K means is an unsupervised learning algorithm, which can classify users into different groups. I will explain how it works briefly. First step, randomly select K centroids from the all points. Second step, for all points, calculate the distance between it and each centroids. Third step, for every point, find the nearest centroid, and label this point to this class. Fourth step, re-calculate centroids according to the classification. And repeat those four steps until the centroids do not change. I use K means algorithm on the MovieLens100k dataset and classify users into two group.

2.2 train my model on two groups of users

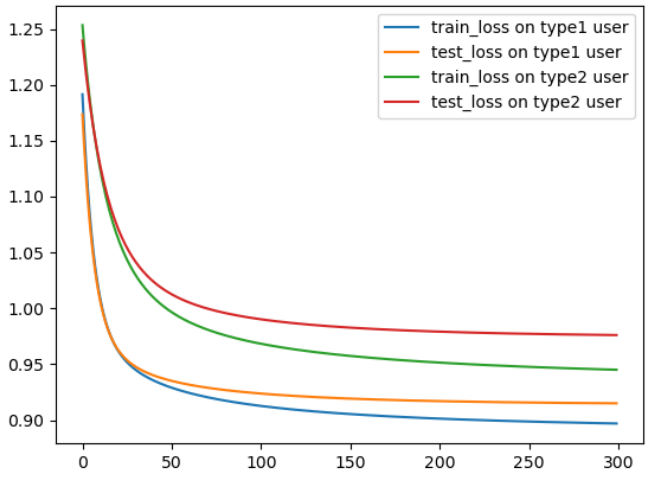

Figure 7: RMSE loss of models trained on group1 and group2

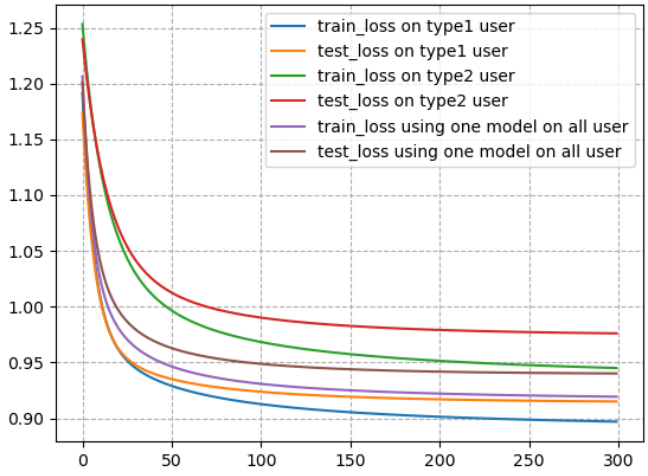

Figure 8: Difference between two models and one model

There are 943 users in total, in type1 there are 694 strict users who tend to give more 3 and 2, and in type2 there are 294 generous users who tend to give more 4 and 5. After training 300 epochs, model can predict type1 users' behavior more accurately than the beseline, which is train my model on the whole dataset. On the other hand, can not capture type2 users' preferences that good, compared to baseline. But the number of type1 user is larger than the number of type2 user. So, the general out come will be better, still, more work should be done on type2 user in order to improve their experiences.

3 next step

Due to some unexpected situation, I have to carry out the whole project on my own, including all the coding, all the paper work and presentation. I was indeed fully engaged in the PBL, but still, it is a pity that some features I was going to implemented that I have not done, some flaws I was going to fix remains unsolved.

The first is confidence, I have not come up with a function to assess how good my model is, besides using loss functions, like RMSE or MAE.

The second thing is multi-process. I actually have implement multi-thread version of my model, but python was invented in 1990s, by then, a CPU only have one core, so it has a GIL(global interpreter lock) to make sure the several threads in program would not mess up a variable. But nowadays, a CPU usually has 4 or more cores, multi-thread in python will not accelerate training. So, introducing multi-process would be more reasonable.

The last thing is regularization. As I mentioned in 1.2.2 RMSE loss, I did not add a regularization in the objective function, and the model is not yet over fitting, still, add a regularization and re-train my model would be better. An Euclidean norm or an Absolute-value norm is enough.

Here is the link for my github repository of all my working during PBL, you can find all the code and explanation to my work. If I have time, I will finish all the features I mentioned in the 3.next setp, and upload my work to this repository, you can access it by clicking the hyper reference – https://github.com/Jas000n/NMF_sparse

References

[1] https://grouplens.org/datasets/movielens/100k/

[2] https://scipy.github.io/devdocs/reference/sparse.htmlmodule-scipy.sparse

[3] https://pypi.org/project/torchnmf/

[4] http://surpriselib.com/

[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html