Final Report

Team member: Dian Zhang (dz584), Yichao Shen (ys3197)

Implementation:

The original training data set contains approximately 1 million users. The size is too large for training and not efficient for us to analyze the performance. Therefore, we focused on the users who merely appear in validation and test set, which are exactly the 110,000 users with partial history in training data. We used SQL query to filter the rows containing user_id who also show up in validation and test set, which was stored in parquet format as training sample file. For the column user_id and item_id, we converted them into numbers by using string indexer function, which can improve storage and loading efficiency to a large extent. After finishing all steps described above, we got the final training and validation dataset.

For the ALS model, we built a hypermeter pool consisting of rank, regularization parameter and alpha. Rank is the number of presumed latent factors that the ratings are determined by, and it represents the complexity of the ALS model. We believe if we slowly increase the size of rank, it would lead to higher accuracy. Since we had sufficient time, the range of rank we tuned for the baseline model was from 9 to 700. Regularization parameter is used to avoid overfitting of the model. It usually smaller than 1 so we tuned it from 0.2 to 1. Alpha is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations. Due to the time limit, we just tried the range of 0.3, 0.5, 1 for most iterations.

After specifying the pool for hyperparameter, we made several loops for the training. Within each loop, we built a model with a group of hyperparameter. Also, for faster training speed and turnaround time, we have decreased the max number of iterations from default 10 to 5. After training, we used training model to predict first 500 items recommended for all users appeared in validation set and compared with true labels. To evaluate the accuracy of recommendation, Mean Average Precision (MAP) in ranking metrics module was used as standard. Lastly, the process would find the best model with highest MAP and stored it for the final baseline model.

Evaluation Results:

The process of tuning was really an interesting part. First, from few test runs at the beginning, we noticed that the final MAPs we got were usually very small, below 10^{-4}. This was probably due to the nature of our model and MAP calculation: we gave the top 500 items for each user in our recommendation, while the precision was how many of the actual label items, in this case, tracks, being listened to by the user. In the validation dataset, most users would only listen to about 15 more tracks, which means even all actual label items were in the prediction set, the majority of the prediction items were still false positives. Thus, the low accuracy in evaluation was expected.

For the tuning of rank, we started with 5 and increased it slightly at first, yet the effect was barely noticeable. Thus, we increased rank to 500 to 1000. Keeping the other parameters the same, the model accuracy indeed improved significantly as rank increases, from about 4×10^{-5} when Rank = 10 to 1.5×10^{-4} when Rank = 500, and continued to become more accurate when Rank reached 1000. However, the more latent factors the model considers, the more computation resources it requires from the server, and the longer it takes to train the model. When Rank = 700, the model usually took around 3 hours to finish training. Thus, even though we might achieve better accuracy with an even higher Rank, we would focus on the 500 to 700 range as a tradeoff for time efficiency.

If focusing on regularized parameter, we found out that value around 0.5 could help the accuracy of model reach the plateau. Too small value as 0.2 or too big value as 0.9 just decreased the accuracy a little bit. Alpha was found out to be not very useful in the whole process of hyperparameter tuning. We even got the same result for groups of different alphas but same everything else. Therefore, we did not pay much attention to tune the value of alpha.

Extension:

We chose to do the “Alternative model formulations”. As this extension requires parameter tuning based on the transformed training data, to not making the iterations complicated and time consuming, we used the best range of rank and regularized parameter we summarized for the baseline model.

The basic for this part was that we filter or transform the column of count by certain rules to change the behavior of the ALS model. Because counts, which serve as the ratings on items in this project, are implicit feedbacks from the users, they are usually sparse and represent weak signal. Transforming counts would give the model a better idea on how to interpret the rating and give better recommendations.

Firstly, we emphasized the importance of items with higher count. These tracks are assumed to represent the users’ real interest as they have been listened to repeatedly. In addition, some counts of items are too small, like 1, do not play a crucial part in the prediction but even increase the complexity of model. Thus, we first decided to drop all rows whose count is lower than 2 and see whether it can help improve the performance of model. Surprisingly, we figured out that the MAP increased about 4 times than the original setting while we kept all hyperparameters the same.

Similar rationale was applied for the log compression of the count column, but instead we emphasized the importance of items with lower count, or simply speaking we emphasized more on whether the tracks were listened or not by the users. We assumed that the higher counts have marginal effect on representing users’ real interest; for example, a user might not have huge preference difference to tracks being listened 5 times or 10 times, yet the difference is huge between once and twice. This transformation gives only a mild increase in MAD score.

We think the difference in result MAD of the two transformation agrees with the common sense: if a user listen to one track repeatedly, it definitely means the user prefers the genre or artist more than the others.

Result for test dataset:

Finally we used each best model to fit the test data and get the MAP as follows:

Baseline model: 0.000169

Drop_low model: 0.000713

Log_compression model: 0.00023

We can see that for each best model, the MAP for the test data is close to what we got in the validation data but a little lower. The best accuracy among these three models happened in the dropping low-count model with about 0.000713, which means that we can dig deeper in this direction if we continually study the case.

Appendix:

			MAP		
Rank	RegParam	Alpha	Original Count	Log Compression	Remove Low Count
5	0.2	0.1	0.00003		
5	0.2	0.3	0.00003		
5	0.2	0.5	0.00003		
5	0.6	0.1	0.00003		
5	0.6	0.3	0.00003		
5	0.6	0.5	0.00003		
5	0.7	0.5	0.00003		
9	0.2	0.1	0.00003		
9	0.2	0.3	0.00003		
9	0.2	0.5	0.00003		
9	0.6	0.1	0.00004		
9	0.6	0.2	0.00004		
9	0.6	0.5	0.00004		
9	0.7	0.5	0.00003		
10	0.1	1	0.00004		
10	0.1	1.5	0.00004		
10	0.5	1	0.00003		
10	0.5	1.5	0.00003		
12	0.2	0.1	0.00004		
12	0.2	0.3	0.00004		
12	0.2	0.5	0.00004		
12	0.6	0.1	0.00003		
12	0.6	0.3	0.00003		
12	0.6	0.5	0.00003		
12	0.7	0.5	0.00003		
20	0.1	1	0.00004		
20	0.1	1.5	0.00004		
20	0.5	1	0.00004		
20	0.5	1.5	0.00004		
30	0.1	1	0.00006		
30	0.1	1.5	0.00006		
30	0.5	1	0.00005		
30	0.5	1.5	0.00005		
50	0.1	1	0.00007		
50	0.1	1.5	0.00007		
50	0.5	1	0.00006		
50	0.5	1.5	0.00006		
70	0.7	0.5	0.00008		

70	0.7	1	0.00008		
70	0.9	0.5	0.00008		
70	0.9	1	0.00008		
150	0.7	0.5	0.00012		
150	0.7	1	0.00012		
150	0.9	0.5	0.00010		
150	0.9	1	0.00010		
300	0.7	0.5	0.00016		
300	0.7	1	0.00016		
300	0.9	0.5	0.00014		
300	0.9	1	0.00014		
500	0.7	1	0.00018	0.00026	Remove count <= 1: 0.00063 Remove count <= 2: 0.00106
600	0.7	1	0.00018	0.00027	Remove count <= 1: 0.00066 Remove count <= 2: 0.00114
700	0.7	1	0.00019	0.00026	Remove count <= 1: 0.00069 Remove count <= 2: 0.00118
1000	0.7	1	0.00020		
1000	0.9	1	0.00017		

Distribution of work:

Dian Zhang:

1. code for baseline model and extension of log compression
2. half of hyperparameter tuning work for all models
3. evaluation and appendix part of final report

Yichao Shen:

1. code for baseline model and extension of dropping low count
2. half of hyperparameter tuning work for all models
3. implementation and final result part of final report