Recommendation System Model Report

DS-GA 1004 Big Data Project

Yimin Chen
Center of Data Science
New York University
New York
yc3294@nyu.edu

Yuan Tian
Tandon of Engineering
New York University
New York
yt2093@nyu.edu

Jiaxuan Lan
Center of Data Science
New York University
New York
jl13072@nyu.edu

1) Introduction

In this project, we first explore the dataset and calculate the distributions of the dataset as a reference for the data splitting and the adjustment of parameters in the future. For model selection, we mainly build the baseline model, pipeline Alternating Least Squares (ALS) model, and the Lightfm model. By comparing different metrics and parameters, the differences between the ALS model and the single-machine Lightfm model are compared. Finally, we performed a dimensional reduction visualization for exploring with T-SNE and PCA. (Github repository is at the end)

2) Data Overview

2.1) DataSets

The datasets that we utilize are from MovieLens. datasets are divided into two categories: a large dataset and a small dataset. The small dataset (size: 1MB) has 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The large dataset (size: 265MB) has 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. (Includes tag genome data with 14 million relevance scores across 1,100 tags.) For the baseline model, we focus on userId, movieId, and rating (which is a numerical value from 0 to 5) and filter out the data whose value ranks in the top 100.

2.2) Data Exploration

After statistical analysis of the dataset, we find that both small and large datasets are imbalanced datasets. But a large dataset is more stable than a small dataset. A small dataset has a jagged distribution, while a large dataset only has a larger amount of data on some specific userIds. (Figure1)

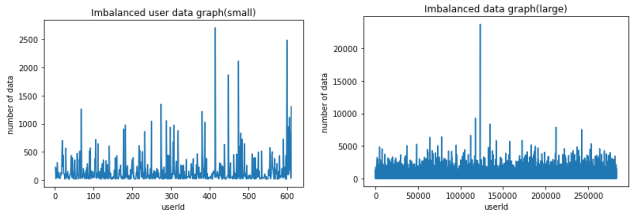

Figure 1: Small dataset distribution (Left) and large dataset distribution (Right)

2.3) Data Proprecession and Splitting

The following operations apply to both small dataset and large dataset. The main library we use is *pyspark.sql*. For the construction of the model, since our dataset (especially the small dataset) is imbalanced and the volume of the large dataset is gigantic, we need to consider the efficiency and imbalance. For the issue of efficiency, we first eliminate the data that has a smaller impact factor on the recommendation system (Remove users that have less than 50 rated movies and Remove movies that have less than 50 ratings) and optimize the data structure (we use the cluster operation is performed by *parquet* structure). Since computational resources are limited, the dataset will be downsampled by 5% of the whole dataset (which can solve the problem of imbalance to a certain extent) and use the rdd.flatMap method to divide the dataset into 3:1:1 ratios training set, validation set, and test set. Finally, we filter validation and test set to the training set by half and check whether the movieId exists in the training set.

2.4) Baseline and Results

We trained a popularity baseline model by calculating the top 100 most popular movies as the recommended results for each user. Comparing the top 100 rating movies for each user with recommended results, we use MAP, Precision@100 and NDCG ranking metrics to evaluate the model. The results show in the below Table 1. From the table we can observe that the baseline of both val dataset and test dataset performs poorer as the dataset becomes larger on all of our three metrics. We consider that the probability of predicting the correct outcome decreases due to a larger base.

	MAP	Precision	NDCG
small val dataset	0.000984	0.01666	0.022527
small test dataset	0.000678	0.02	0.024928
large val dataset	1.61e-06	5.59e-05	5.40e-05
large test dataset	7.61e-07	1.825e-05	1.826e-05

Tabel 1: Baseline model with three metrics under different datasets and dataset sizes

3) ALS Model Implementation

3.1) ALS Introduction

Assuming that there is a matrix composed of n users and m items, entry (u, i)th represents the rating of ith item by uth user. The basic idea of the ALS model is to decompose the sparse matrix and evaluate the value of the missing entries. ALS factorizes the utility matrix into the product of two matrices U and V. U and V are then estimated by the least-square method. We mainly use the ALS model from *pyspark.ml* library for model fitting.

3.2) Metrics Evaluation

In the evaluation process, our idea is to use multiple metrics to evaluate the model, so that different metrics can make up for each other's shortcomings and make the evaluation more objective and accurate. We take 1/3 of the weight of each metric, and finally add them together into an aggregated metric for evaluation. We mainly use three metrics here: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG) and Precision@k. These three metrics are all from *RankingMetrics* of *pyspark.mllib* library For easy understanding, the following interpretation of metrics:

1. **Precision@k** : Precision at k is the proportion of recommended items in the top-k set that are relevant.
2. **MAP**: Mean Average Precision is the number of recommended items that are relevant and average across all users.
3. **NDCG** : Normalized Discounted Cumulative Gain is that the model ranks the candidate items according to the similarity between the user and the candidate items, and returns the k most similar items as the recommendation result. Keeping the recommendation order of the model, label each item with its score in the original dataset, and compute the DCG. Then, the marked scores are rearranged from large to small, and the DCG is calculated again (iDCG). NDCG is obtained from DCG divides by the iDCG.

3.3) Fine-tuning Model

Before processing, we downsampling 5% of the datasets and take precision at k = 100 as our metrics. The grid search is our method to obtain the result of fine-tuning. There are three parameters we have to check out : *max iteration, rank* and *regs*. For convergence, we set *max iteration* to be 30, *regs* to be [1e-6, 1e-5, 1e-4, 1e-3, 1e-2], and *ranks* to be [5,10, 20, 50]. Finally, we fix precision, NDCG, and MAP at k = 100 as our metrics.

3.4) Model Performance (see figures in Appendix A)

By analyzing Figures in Appendix A, for a small dataset, it can be clearly found that regs = 0.001 is a threshold. When regs is greater than 0.001, the three metrics are not very sensitive to the changes in ranks, and the relationship between ranks and the three metrics is not correlated. When regs are lower than

0.001, for each regs, the higher the ranks, the worse the metrics. On the contrary, compared with a larger dataset, for the same regs parameter, the larger the ranks, the better the metrics. For the same ranks, the larger the regs, the better the metrics. It can be found that for small dataset, penalty and ranks cannot significantly improve the effect of the model. Besides, for larger dataset, regs and ranks show a positive correlation for the three metrics, however it should be noted that gigantic regs and ranks are easy to cause the model to overfit. The best parameters, reg and rank, for the ALS model are 1e-5 and 5 for the small dataset, 1e-2 and 50 for the large dataset respectively obtained by using the training and validation set. Also, by applying these parameters on the test set, we got the following results shown in Table 2:

	MAP	Precision	NDCG
small test dataset	0.02592	0.11	0.1304
large test dataset	0.004566	0.03288	0.03671

Tabel 2: ASL model with three metrics tested on the test dataset using the best parameter obtained from the training and validation set.

4) Single-Machine Lightfm Model (running on google colab)

4.1) Lightfm Introduction and implementation

Lightfm is a python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback, which can also incorporate both item and user metadata into the matrix factorization. The main library we use is *lightfm* library.

1. Downsampling 5% of the datasets. For building the input interaction matrix, we use *lightfm.data* to create an object and fit our training data.
2. We filter out the rating (>=2) as a true label after the establishment of an interaction matrix for efficiency.
3. The parameter finding method we use is grid search for set regs as [1e-6, 1e-5, 1e-4, 1e-3, 1e-2] and ranks as [5, 10, 20, 50].
4. Parameter interpretation: the lightfm model in our project contains four parameters: no_components*, item_alpha,user_alpha, loss.* We set *no_components* as *rank, item_alpha* and *user_alpha* as *reg,* and *loss* as 'WARP' (Weighted Approximate Rank Pairwise Loss).
5. Finally, we treat precision at k = 100 as metrics and collect the results of parameters and precision itself.

4.2) Comparison between ALS and Lightfm (see Figures in Appendix B)

In this part, we fix rank for each Figures and change the regs in order to see the relationship between time/precision, dataset size and models (ALS and lightfm).

Figures 10-13 describe the relationship between time, dataset size, and models. For the same model, larger dataset size will increase the time. For the same dataset size, lightfm runs faster than ALS.

Figures 14-17 describe the relationship between time, dataset size, and models. For ALS models, smaller dataset will increase the precision. For lightfm models, larger dataset will increase the precision. For the same dataset size, lightfm performs better than ALS.

In conclusion, for the time part, lightfm performs better than ALS on both dataset sizes. For the precision part, ALS performs worse and lightfm performs better when dataset size increases. Also regs = 0.001 is a threshold for ALS with large dataset size. Besides, as rank increases, the performance of both models tends to slightly raise.

5) Exploration for T-SNE and PCA (running on google colab, see Figures for small data in Appendix C)

We extended our project to make the qualitative error analysis by visualizing the learned item representation via dimensionality reduction techniques (T-SNE and PCA). To be more specific, we exploited the item latent factors from the optimal models(ALS) with rank=50, maximum iteration=30 and regularization

parameter=1e-02 on a 5% subsample dataset. The latent item factors obtained from the best-performing model contain 10 latent features. We calculated the averaged rating for each movieId, rounded them to ensure that there won't be a lot of different variable types to plot and appended the rounded averaged rating to the pre obtained item factor matrix. For both T-SNE and PCA, we standardized the data for data preprocessing.

5.1) T-SNE of Latent Factor

T-SNE, namely the T-distributed stochastic neighbor embedding, is a popular dimensionality reduction algorithm that arises from probability theory. In Figure 2, we can see that there is a quite obvious trend of the different rounded ratings. Movies with rating of zero and ones mostly appear in the left upper region, while those with rating of four and five mostly appear in the lower right region.

Figure 2: T-SNE visualization with rounded average rating on large dataset

5.2) PCA of Latent Factor

Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, using the first 2 principal components. Generated from PCA, the first principal component contains approximately 11.92% information and the second principal component contains approximately 6.67% information. From Figure 3, we can see that there are explicit patterns in the scatter plot with respect to the first and second principal component. Reasons may be that average rating is important in the ALS latent representation of movies, ALS is good enough to learn the movie representation feature, or PCA can better keep the feature information after dimension reduction.

Figure 3: PCA visualization with rounded average rating on large dataset

6) Github Repository

https://github.com/nyu-big-data/final-project-group_5

7) Contributions

All team members worked together on constructing the code, creating graphs and writing the report. We contributed evenly to the project.

Appendix A

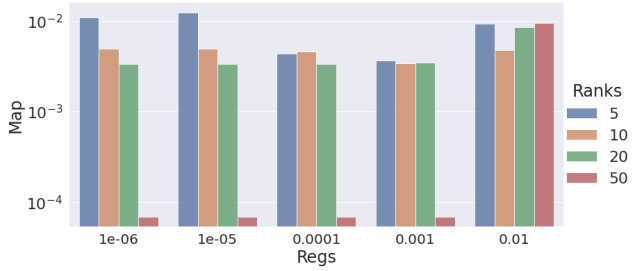

Figure 4: Different MAP categorize by regs and ranks on small dataset

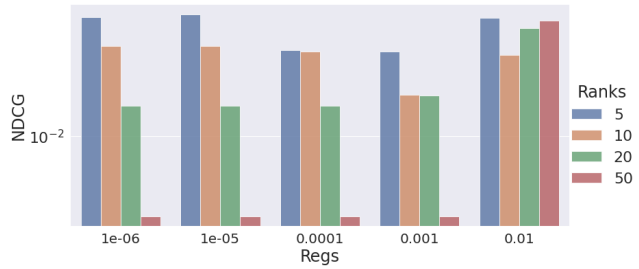

Figure 5: Different NDCG categorize by regs and ranks on small dataset

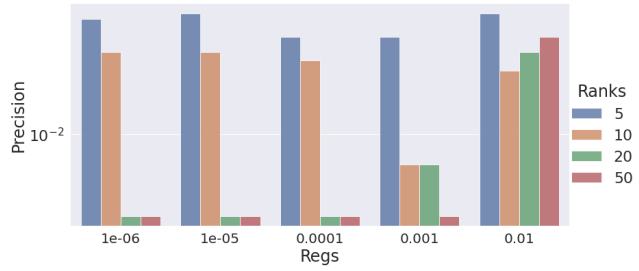

Figure 6: Different Precision categorized by regs and ranks on the small dataset.

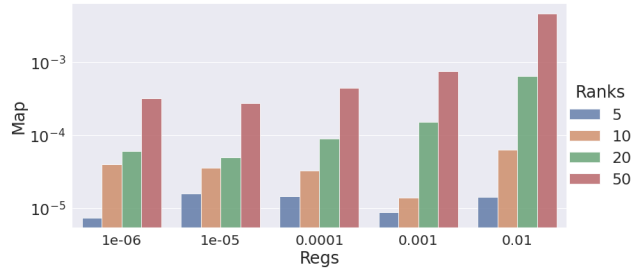

Figure 7: Different MAP categorized by regs and ranks on the large dataset.

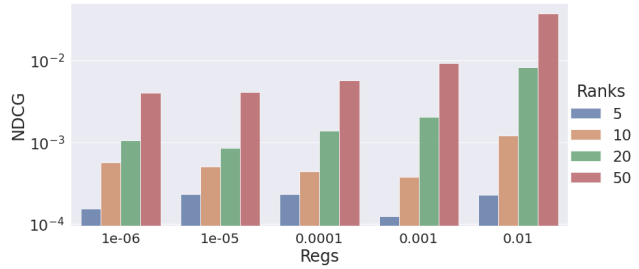

Figure 8: Different NDCG categorized by regs and ranks on large dataset.

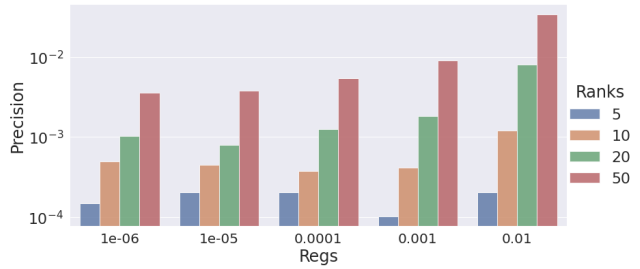

Figure 9: Different Precision categorize by regs and ranks on the large dataset

Appendix B

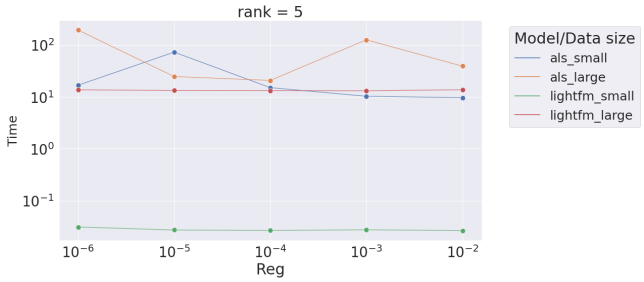

Figure 10: Different time under different regs and same rank=5 with various combination of model and data size

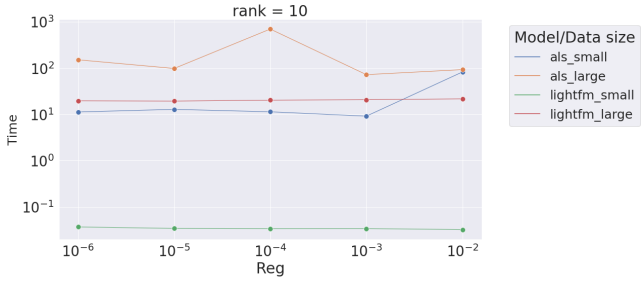

Figure 11: Different time under different regs and same rank=10 with various combination of model and data size

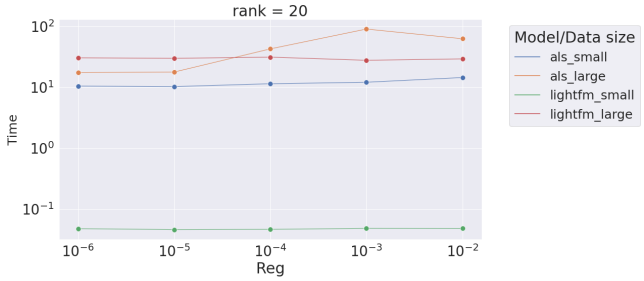

Figure 12: Different time under different regs and same rank=20 with various combination of model and data size

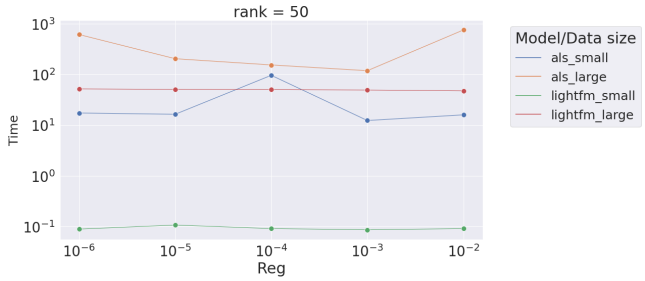

Figure 13: Different time under different regs and same rank=50 with various combination of model and data size

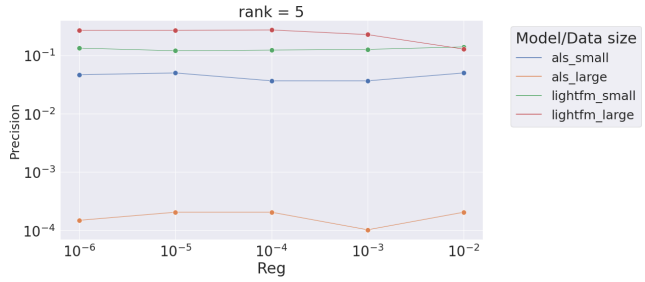

Figure 14: Different precision under different regs and same rank=5 with various combination of model and data size

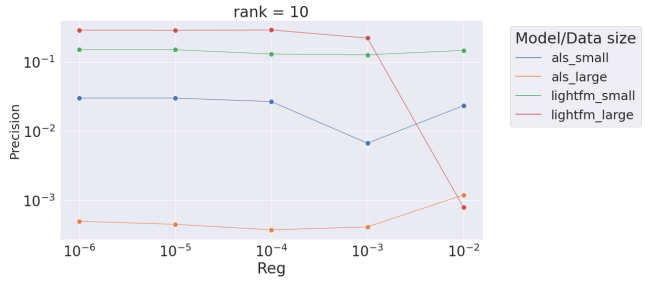

Figure 15: Different precision under different regs and same rank=10 with various combination of model and data size

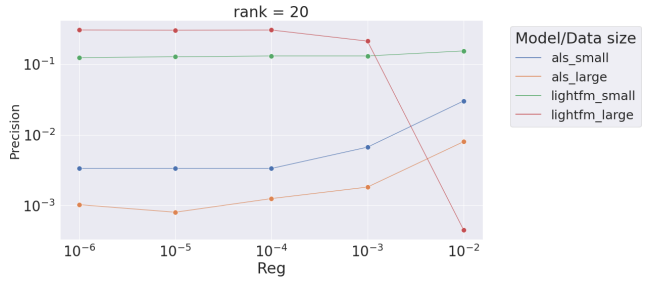

Figure 16: Different precision under different regs and same rank=20 with various combination of model and data size

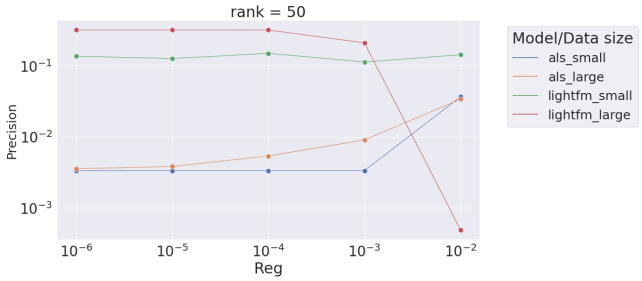

Figure 17: Different precision under different regs and same rank=50 with various combination of model and data size

Appendix C

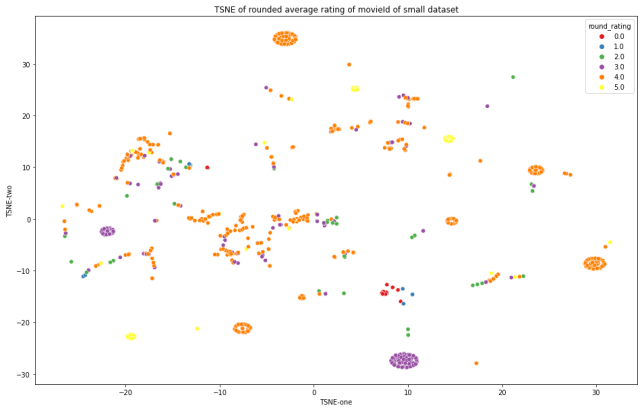

Figure 18: T-SNE visualization with rounded average rating on small dataset

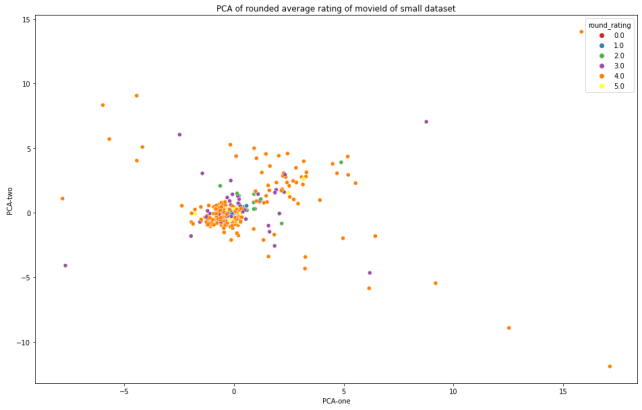

Figure 19: PCA visualization with rounded average rating on small dataset