Pyspark Song Recommender System

DI HE, JIE GU, SHUNJIA YE, and YUE YIN

Di He, Jie Gu, Shunjia Ye, and Yue Yin. 2021. Pyspark Song Recommender System. 1, 1 (May 2021), 6 pages. 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this final project, we apply big data tools to build and evaluate a collaborative filter based recommender system. The dataset we work on is the Million Song Dataset (MSD), with implicit user feedback. Using Spark's alternating least squares (ALS) method, we learn latent factor representations for users and items, and recommend for users in the test set. Thereafter, we compare our model to single-machine implementation, and the baseline for the extension.

2 DATA

2.1 Data Overview

Data used for the basic recommendation system consists of the train, validation, and test parquet files. Each row in the files consists of user_id (string), count (int), and track_id (string). There are 49,824,519 records for the train, 135,938 records for the validation, and 1,368,430 for the test. Additional data including metadata, features, genre tags, lyric, are also used for the extension.

2.2 Data Preprocessing

Since running the full set of experiments end-to-end on the whole dataset is too time-consuming, we build sub-samples of 1%, 5%, and 25% of the data to save time on debugging and hyperparameter tuning. Although the best hyperparameters from the subsets may not be optimal for the full data, we could find insightful patterns in a rather efficient manner. After finding ranges of hyperparameters that works well in all subsamples, we zoom into those ranges and tune on the full dataset. As only less than 1% training data contains user ids in the validation data, the random subsamples from training set are very likely to miss all the user ids in the validation set, leading to our failure to make any predictions. To address this problem, we combine all training observations that contain unique validation user ids, and random sample from the remaining training data for all three subsamples. Besides, the user ids and track ids from string are encoded into integer to fulfill the format requirement of the ALS model.

3 METHODOLOGY

3.1 Model Implementation

Generally speaking, alternating least squares (ALS) model tries to find latent factors that can be used to predict the expected preference of a user for an item. Since 'count' in our data is an implicit feedback, it is reasonable to choose the implicit ALS model here. In spark ALS model, we set the implicitPrefs=True. The formula of the model is shown as below.

$$\min_{U,V} \sum_{(i,j)\in\Omega} c_{ij}\left(p_{ij} - \langle U_i, V_j\rangle\right)^2 + \lambda\left(\sum_i \|U_i\|^2 + \sum_j \|V_j\|^2\right) \quad , where\ p_{ij} = \begin{cases} 1 & R_{ij} > 0 \\ 0 & R_{ij} = 0 \end{cases} , c_{ij} = 1 + \alpha R_{ij}$$

In our model, R_{ij} represent the counts that user i listens to item j, U_i is a latent factor vector for user i, and V_j is a latent factor vector for item j. When making recommendation for users, items will be recommended based on the preference p_{ij} (from high to low). From the above formula, some parameters need to be chosen by us, such as the dimension of the vectors U,V(rank) and the regularization parameter λ (regParam). Besides, α is our confidence in the implicit feedback 'count'. To solve the optimization problem, we fix U and optimize V, then fix V and optimize U until convergence. Therefore, we also need to determine the max iterations to run (maxIter).

Besides, as there exists quite a few items which some users have no interaction with, we have to deal with the cold start problem. At this stage, the simplest approach for us is to set coldStartStrategy="drop" to let the ALS model drop users when prediction contains NaN values, which guarantees the validity of the evaluation metric.

Authors' address: Di He, dh371@nyu.edu; Jie Gu, jg6617@nyu.edu; Shunjia Ye, sy1880@nyu.edu; Yue Yin, yy1932@nyu.edu.

3.2 Evaluation Metric

During the hyperparameter tuning, Precision at k is chosen as our evaluation metric, which measures how many of the k recommendations are in the set of true relevant documents averaged across all users. In this metric, the order of the recommendations is not taken into account. The formula for our model is as follows. U is the set of N users, D_i is a set of N_i ground truth relevant documents for user i, and R_i is a list of Q_i recommended documents.

$$p(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{k}\sum_{j=0}^{\min(Q_i,k)-1}\mathrm{rel}_{D_i}\left(R_i(j)\right), rel_D(r) = \begin{cases} 1 & \text{if } r \in D \\ 0 & \text{otherwise} \end{cases}$$

Since evaluations are required to be based on predictions of the top 500 items for each user, we set k = 500. However, this may not be the best choice for evaluation purpose. Namely, if a user only has 10 related songs in the validation, but we still let the model make 500 recommendations, then at most 10 recommendations will appear in the relevant documents, leading to a low precision. As it may not be fair to use 500 recommended items for this user to make evaluations, we also discuss some alternative designs in the future work.

3.3 Parameter Tuning

We perform grid search on hyperparameters including rank (latent factor dimension), regParam (regularization), alpha(weight on the implicit feedback) and maxiter (max iterations). To be noted, maxiter is fixed as its default value 10 until we find the best combination of other hyperparameters, since it is more about the optimization instead of the structure of the model. The strategy is to try a wide range of hyperparameters on the 1% and 5% subsamples, zoom in to a smaller range on 25% sub-samples, and then zoom in again on the full dataset. Finally, with the best configuration of the other hyperparameters, we tune the maxiter on the full data. During the whole process, the relationship between rank and regularization needs to be noted. Higher rank indicates more expressiveness, but may cause overfitting. Thus, higher regularization is only desired when the rank is high.

4 RESULT AND ANALYSIS

According to the grid search, the hyperparameter rank leads to the model's higher precision at 500, but longer training time as well. By controlling hyperparameters except for rank, as shown in Figure 1, the precision and time comparisons of different ranks are found. Each plot also includes different sized sub-samples for comparison. From the left plot, it is obvious that as the rank increases, the growth of precision at 500 is sharp at first, then it tends to flatten around rank 100. Overall, rank and precision at 500 seems to follow a logarithmic relationship. On the other hand, the training time increases exponentially as the rank increase. A trade-off on precision and time efficiency is observed for rank. In fact, there is no standard criteria for the selection of an optimal rank in reality. In some cases, companies may be willing to spend considerable training time for marginal precision improvement. However, here we choose the rank 200 as our optimal rank, taking into consideration of the limited time we get, and the already satisfactory result obtained. Another thing needs to be noted here is that the 25% subsample already obtains the similar precision at k to the full training data while the training time of it is much shorter. Thus, it is reasonable to tune hyperparamters on the 25% subsample, choose the best combination, and apply it to the full training data.

We have tried over 400 combinations of hyperparameters for this model, some optimal results for the 25% sub-sample and the full training data are included in the appendix for reference. It is found that when alpha is high, the higher regularization brings better precision, but will make the result worse if it exceeds a certain threshold. Overall, the best parameters we find are 200 for rank, 1 for regularization and 80 for alpha.

Thereafter, we tune the max iteration with other optimized hyperparameters fixed on the full training data. The plots of the precision at 500 and time efficiency as functions of max iteration are shown in Figure 2 below. Keep increasing the max iteration, the precision almost stays the same after the 10th iteration. Based on the linear relationship between training time and max iteration, we choose the max iteration to be 10 though there is a slight increase in precision from max iteration 10 to 15. As a result, our optimal choice of hyperparamters gives us 0.01443 for precision at 500 on testing data, which is reasonable for our project.

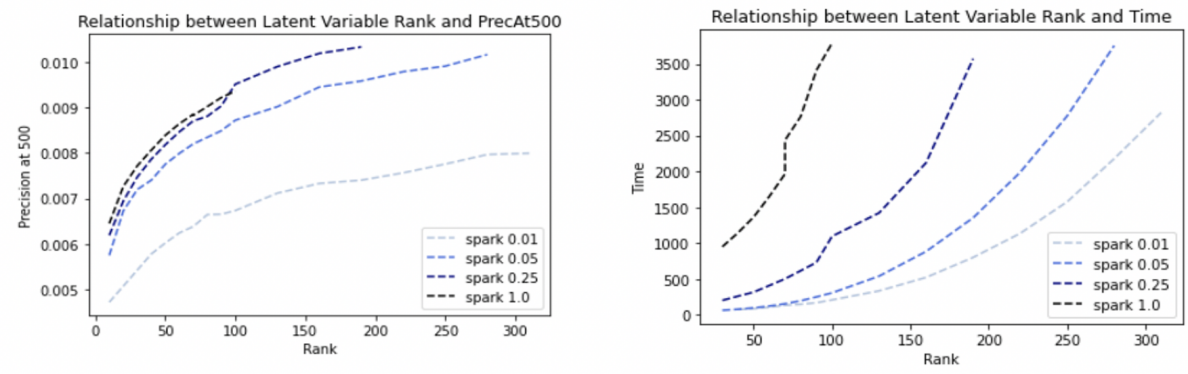

Fig. 1. *Comparison: Precision and Efficiency for Rank*

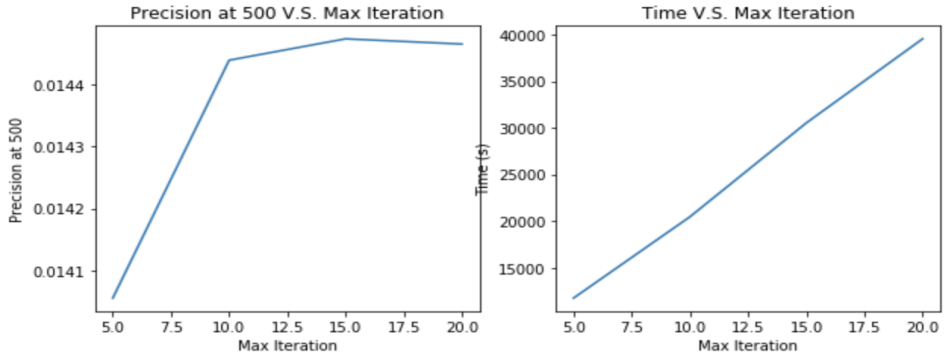

Fig. 2. *Comparison: Precision and Efficiency for Max Iteration*

5 EXTENSIONS

5.1 Single-machine

To compare with the parallel Spark ALS model, we explore a single-machine implementation using Lenskit. Fig. 5 shows their performance comparison controlling all other parameters except for rank.

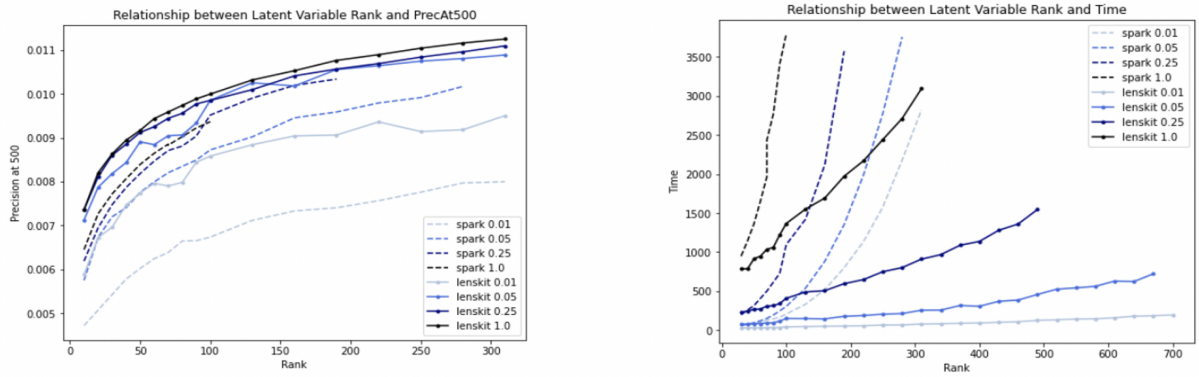

Fig. 3. *Precision and Time Comparison: Spark vs. Lenskit*

As shown on the left graph of Fig. 3, the single-machine recommendations of the 1%, 5%, and 25% subsets generally achieve higher precision at 500 than Spark, over different ranks. This effect diminishes when data size get larger. We also notice the precision of

Lenskit is not so stable as Spark. This may be caused by the different algorithms Lenskit uses to solve for the optimized matrices. Based the Lenskit documentation, it uses conjugate gradient method rather than implicit ALS's LU-decomposition.

Besides, the time efficiency of Lenskit is better than Spark, as shown on the right graph of Fig. 3. Spark's slowness may because the parallelism processing adds significant overhead, but our data is not large enough to overweight this. Further, Lesnkit do purely in-memory in-core processing which consume higher RAM than Spark, but is also faster than disk and network that Spark uses. Although Lenskit enjoys better time efficiency than Spark in our experiments, it is notable that the training time of both implementations increases exponentially as the rank increases, but linearly increases as data size increase.

5.2 Baseline Model

A simple popularity-based user-item bias rating prediction model is implemented with the Lenskit bias algorithm as the baseline of our problem. Since we recommendation for the users, only item bias term is introduced. To increase the generality of the bias model, we tune the damping parameter, the number of mean ratings, to assume a prior for each item. Namely, low-information items is damped towards a mean instead of being permitted to take on extreme values based on few interaction. According to the result on various subsets, higher damping value leads to higher precision at 500 for all sizes, following a logistic trend. Specifically, when damping exceeds 10000, increasing this hyperparameter leads to marginal improvement on the precision score. As the same damping parameter puts higher regularization on small dataset than large ones, we find smaller subsets actually obtain higher precision at the beginning but converge quickly and get lower precision at the end. Overall, even the best precision we could achieve is rather low compared to the ALS model. The optimal ALS model achieving precision at 500 of about 0.015%, is beating this baseline by almost 400%.

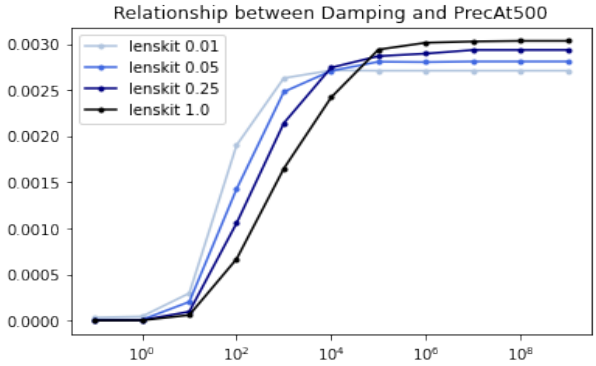

Fig. 4. *Relationship between Damping and PrecisionAt500*

6 CONCLUSION FUTURE WORK

Using alternating least squares (ALS) model, we make 500 recommendations for each user and evaluate the recommendation based on precision at 500 on our music dataset. Our final hyperparameters are rank 200, regularization 1, alpha 80 and max iteration 10, which gives a 0.01443 precision on the testing data.

For the extension, we explored the single-machine, baseline model and visualization. Limited by the paragraph, we only discuss the first two, but the code for the last two, and the Umap is attached for reference. The single-machine implementation has gained rather satisfactory precision and efficiency, especially with small sample sizes. With the baseline model, we are able to get a sense of the prediction power of our ALS model, which beats the baseline by approximately 400%.

For future work, we could explore more sophisticated evaluation metrics, using precision, MAP or NDCG for a smaller recommendation set, to deal with the doubt raised in the evaluation section. One solution we come up with is to set a boundary p_{ij} for recommendation. Namely, our model will no longer recommend items with p_{ij} smaller than certain values. This may reduce the total 500 recommendations and improve the precision at k since those 'irrelevant' items (p_{ij} smaller than the boundary) will not be counted as wrong recommendations anymore. Another simpler method is to evaluate the precision according to the minimum value between the number of true relevant items and the k we choose.

7 TEAM CONTRIBUTIONS

All four team members share the workload fairly throughout this project. The specific contributions are as follows:

Di He: Include but not limited to Spark modeling and result analysis, result visualization, all four extensions, report revising, and support in every part.

Jie Gu: Include but not limited to data preprocessing, hyperparameter tuning, Spark modeling and result analysis, report writing.

Shunjia Ye: Include but not limited to Spark modeling, hyperparameter tuning and visualization extensions.

Yue Yin: Include but not limited to hyperparameter tuning, single-machine and baseline extensions, report writing.

8 APPENDIX

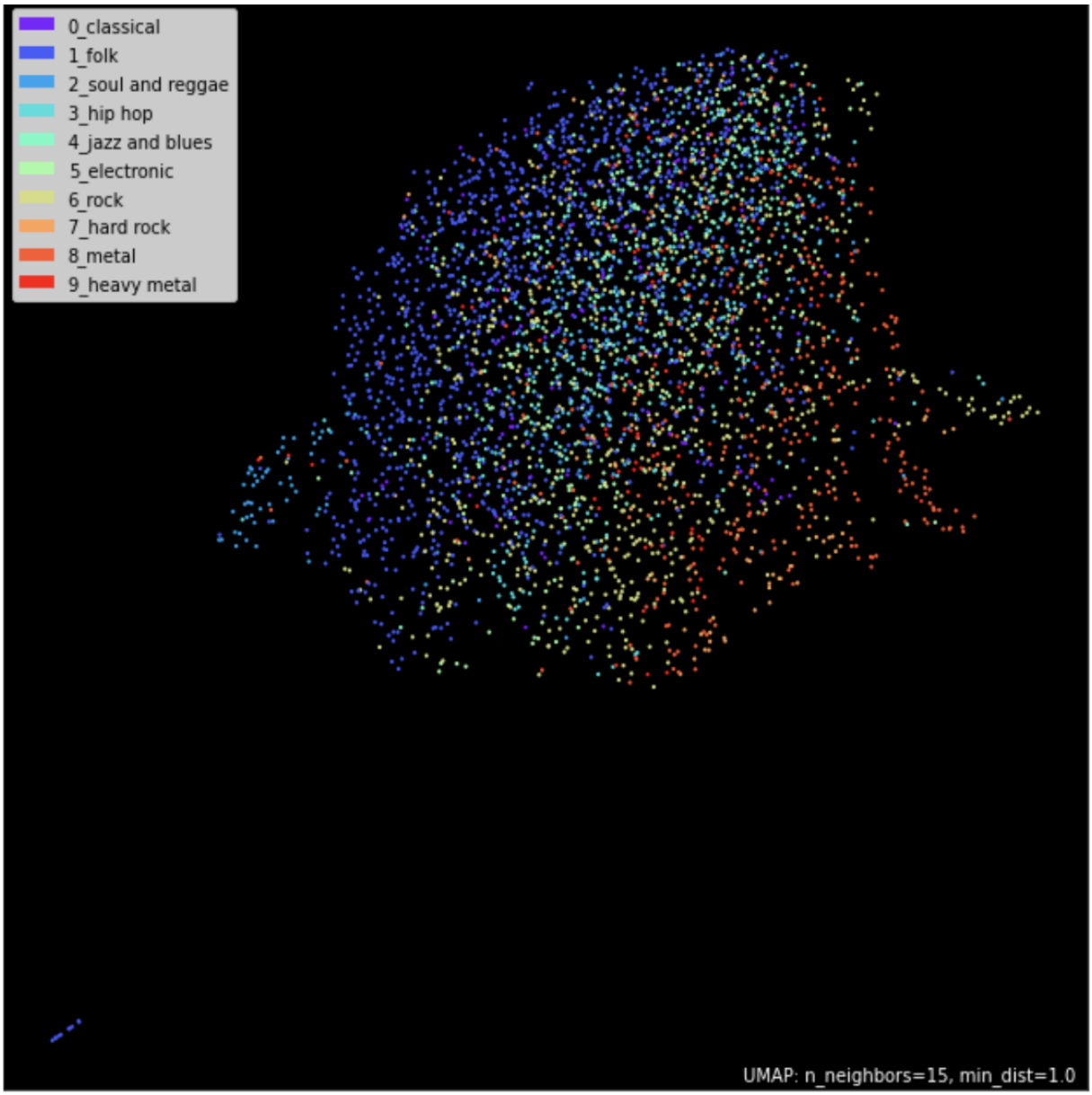

Fig. 5. *Umap*

	rank	regparam	alpha	precision at k
0	200	0.001	80	0.012768
1	200	0.010	80	0.012771
2	200	0.100	80	0.012805
3	200	1.000	80	0.012981
4	5	1.000	80	0.007075
5	10	1.000	80	0.008206
6	50	1.000	80	0.010835
7	100	1.000	80	0.011925
8	250	1.000	80	0.013288
9	300	1.000	80	0.013504
10	80	1.000	1	0.006512
11	80	1.000	10	0.009138
12	80	1.000	50	0.010154
13	80	1.000	100	0.010226

Fig. 6. *table 1: 25% sub-sample*

	rank	regparam	alpha	maxite	precision at k
0	200	1	80	5	0.014055
1	200	1	80	10	0.014438
2	200	1	80	15	0.014472
3	200	1	80	20	0.014464
4	200	10	80	10	0.014002
5	200	20	80	10	0.013149

Fig. 7. *table 1: Full Data*