DS-GA 1004 Big Data Final Report

Team: Rui Jiang(rj1407), Zhengyuan Ding(zd415), Ziyu Lei(zl2350)
Code: https://github.com/nyu-big-data/final-project-dingjianglei

1 Implementation of Baseline

The implementation process mainly consists of three parts: training data downsampling, transformation of features from string to numeric indices and ALS model building.

1.1 Downsampling

Due to limited resources, the training set is downsampled to prototype our model. The subset includes the last 110K rows from the training set to make sure that we can do evaluation on both validation and test set. In addition, we also randomly sampled about 2% of the training set without replacement. As a result, our training subset has 1103251 instances, which is about 2.2% of the original data set. (Note: all the “training set”s mentioned below in the report refers to this training subset.)

1.2 StringIndexer

To transform the two columns user_id and track_id from strings to numerics, we built two string indexers respectively. Then the training set was fitted to the indexers and transformed. The same process was conducted on the validation/test set as well using the indexers trained by the training set. In particular, user IDs that don’t appear in the training set are skipped by the indexers in the transformation.

1.3 ALS Model

Two parameters we specifically set here for our ALS model, one is the “implicitPrefs” and the other is the “coldStartStrategy”. Since the available dataset only includes “count” which measures implicit feedback derived from users’ listening behavior, we set the parameter “implicitPrefs” of the Pyspark ALS to be “True”.

As for the cold-start strategy parameter, according to the Pyspark documentation, is the strategy to deal with the new users at the prediction time. We set it to “drop”, meaning we will drop the unseen users in the validation/test set when predicting.

2 Evaluation

2.1 Evaluation Metric: MAP

In terms of evaluation metrics, we choose mean average precision(MAP) to measure the performance of our model. Since our data only involves implicit feedbacks, we do not have a reliable feedback regarding which items are disliked by each user. The absence of a listening behavior may due to multiple reasons(not simply because a user does not like it). Thus, metrics like RMSE or MSE are not appropriate in this case, as they are designed for measuring explicit feedbacks(e.g.rating prediction) and require data of user interactions on disliked items. Metrics like MAP on the other hand not only measures how accurate are the predictions on the unseen data, but also accounts for the ordering of recommendations. Therefore, we sticked with MAP@500 in the whole evaluation process.

The formula for MAP is as follows:

$$MAP = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{|D_i|} \sum_{j=0}^{Q-1} \frac{rel_{D_i}(R_i(j))}{j+1}$$

Basically, it measures how many of the recommended items are in the set of true relevant items. Therefore, for each user we generated both the top 500 tracks recommended by our model and the ground truth relevant tracks to calculate a relevance score given by MAP.

2.2 Hyper-parameters Tuning

We have three hyperparameters for our ALS model to tune: rank, alpha and the regularization parameter. We used the validation set to tune the hyper-parameters and the results are shown in the table below.

Regularization Tuning

maxIter	rank	regParam	alpha	MAP
5	10	0.01	1	0.0090711
5	10	0.05	1	0.0091000
5	10	0.1	1	0.0090568
5	10	0.3	1	0.0083365
5	10	1	1	0.0067571

(Table 1)

Rank Tuning

maxIter	rank	regParam	alpha	MAP
5	10	0.05	1	0.0091000
5	20	0.05	1	0.0082838
5	50	0.05	1	0.0075866
5	100	0.05	1	0.0062395

(Table 2)

First we tuned the regularization parameter and kept the others unchanged. The results in Table 1 show that MAP is optimized when “regParam” = 0.05. Then in Table 2 we tuned rank from 10 to 100 and found the MAP decreased. Thus we chose the best rank to be 10.

Alpha Tuning

maxIter	rank	regParam	alpha	MAP
5	10	0.05	1	0.0090711
5	10	0.05	3	0.0084354
5	10	0.05	5	0.0083411
5	10	0.05	10	0.0073587

(Table 3)

Max Iterations Tuning

maxIter	rank	regParam	alpha	MAP
5	10	0.05	1	0.0090711
10	10	0.05	1	0.0106603

(Table 4)

When tuning alpha in Table 3, MAP also decreased as alpha increased. The best MAP was achieved when alpha is 1. Finally, we increased the number of max iterations to 10 and got a better performance with MAP at around 0.01067.

2.3 Best Model
The performance may be better if we train our model on larger dataset, but due to the limited resources on dumbo we end up getting our best model with validation MAP at around 0.01067. Its hyperparameters are: rank = 10, regParam= 0.05, alpha = 1 and maxIter = 10. When we test it on the test set, it achieves MAP at around 0.01269.

3 Extensions
3.1 Alternative model formulations
The baseline ALS model we developed in Spark is based on implicit feedback, which is because we only have access to “count” (representing the strength in observations of user actions). Compared to the explicit feedbacks indicating “preference”, implicit feedbacks indicate “confidence”. Having implicit data is more common in real life although it might be more noisy and not as apparent as explicit feedbacks. Therefore, in addition to merely using the original “count” data, it’s worthwhile to try different transformations on it.

We first examined the distribution of original count values in our training set: it is extremely skewed with 59.2% of our data’s count equals to 1, 74.3% of our data having count<=2, and maximum value 784. Thus the very first approach we tried is to do log compression using pyspark.sql.functions.log on count, which reduced imbalance in our dataset to some extent. Another approach we applied is to drop low count values(since the dataset would be rather small if we drop instances with count<=2, we decided to drop the instances having count=1). After applying those approaches to our training data respectively, we tuned the hyperparameters using the same grid search approach.

Compared to the results in our baseline model, we found that both the two approaches did not improve MAP somehow. With maxIter=10, the best MAP we achieved after tuning hyperparameters was improved to 0.0142 using “drop”, which may due to the reduced noise in “count”. However, log compression does not have clear impact on improving model’s performance, the reason of which we guess may be the strength of implicit feedback may be reduced after taking log on “count”. The more best results are shown as below:

Approach	Rank	RegParam	Alpha	MAP@500_validation	MAP@500_test
Log Compression	10	0.01	1	0.0103	0.0093
Drop Low Count	10	0.01	1	0.0125	0.0142

(Table 5)

3.2 Fast search
Our goal is to accelerate query time when generating recommendation lists for the users. We chose to use the Non-Metric Space Library (NMSLIB), which is an efficient cross-platform similarity

search library and a toolkit for evaluation of similarity search methods[3]. Moreover, it can be 10 times faster by using Hierarchical Navigable Small World Graph (HNSW) and also have high recall to retain accuracy. Thus, we can use this toolkit to find the top 500 similar items for every user with the highest score, which is the inner product of every user and all item factors.

Firstly, we exported a model with parameters, maxIter=10, rank=10, regParam=0.01, alpha=0.1 from Spark to our local machine and got two sets of latent factors, user Factors and item Factors. Then we combined all of the user Factors and item Factors respectively. Next, we ran the brute-force search once to calculate the inner product of each user with item Factors and sorted in descending order to pick top 500. After that we initialized the NMSLIB, specified the cosine similarity space, and created an index for item Factors. Finally, we ran it several times with different query parameter settings. The results of query time are as follows.

	Query Time /s	Index Time /s	efSearch	efConstruction	M
Brute-force	5826.07	\	\	\	\
NMSLIB	64.65	33.9	100	100	20
NMSLIB	69.51	64.63	100	300	20
NMSLIB	162.63	54.49	800	300	20
NMSLIB	193.20	80.78	800	500	20

(Table 6)

From the table we can see that NMSLIB can significantly accelerate the query time, which is 90 times faster than brute-force search. And it is worth to take some time to create index for item factors.

4 Contributions

- Downsampling: Ziyu Lei
- Baseline: Zhengyuan Ding, Rui Jiang
- Tuning: Zhengyuan Ding
- Extension 1 - *Alternative model formulations:* Rui Jiang
- Extension 2 - *Fast search*: Ziyu Lei

5 References

[1] Hu, Y., Koren, Y., & Volinsky, C. (2008, December). Collaborative Filtering for Implicit Feedback Datasets. In *ICDM*(Vol. 8, pp. 263-272).
[2] Pyspark Evaluation Metrics Documentation
https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html#ranking-systems
[3] Non-Metric Space Library https://github.com/nmslib/nmslib