Recommendation System on ListenBrainz Dataset

Spring 2023 DS-GA1004 Final Project

Ivy Cui
Center for Data Science
New York University
mc9432@nyu.edu

Yuqing Cui
Center for Data Science
New York University
yc6285@nyu.edu

1 Introduction

The growing ubiquity of digital music platforms has led to a vast amount of data being generated, presenting the opportunity to develop data-driven recommendation systems to enhance user experience. In this report, we explored different models including the popularity baseline model and two popular recommendation algorithms: Alternating Least Squares (ALS) and LightFM. We examined the performance and efficiency of these models in recommending music based on ListenBrainz Dataset. The goal is to identify the model that provides the best balance between recommendation quality and computational efficiency. The modeling process includes model training, hyperparameter tuning, performance evaluation, and assessment of model scalability.

2 Data Preprocessing

We split the provided train set into train and validation sets with weights 0.7 and 0.3. The idea is to group by user_id and split on recording_msid for each user, ensuring both train and validation sets have same set of user_id. For user_id that only appears once in the interactions table, we cannot split them. If we keep them in both train and validation, it will cause data leakage. We counted the total number of distinct users and number of users with only one observation. The latter only accounts for 0.72% of all distinct users, so we can safely filter them out. For the rest of data, we grouped them by user_id and randomly assigned 70% to train set, 30% to validation set.

After splitting, we performed two steps of sanity check to ensure the split is done properly.

1. The number of distinct user_id should be same in train and validation.
2. total number of rows in train : total number of rows in validation ≈ 70:30

Note that we did not join the interactions table with other tables. Though recording_mbid is unique, some music does not have it. Recording_msid does not uniquely identify one music, but it is good enough to work with.

Table 1 Summary Statistics for Train and Validation

Total num of distinct user_id	7909
Num of user_id with one appearance	57
Num of distinct user_id after filtering	7852
Num of distinct user_id in train	7852
Num of distinct user_id in validation	7852
Total num of rows in train	125622859
Total num of rows in validation	53843207
Train / (Train + Validation)	69.99%

3 Popularity Baseline

The popularity baseline model recommends music that is most popular based on the number of times being listened to and the number of users listening to it. This recommendation is for all users without any personalization.

Using the train set, we created a score by weighing the number of occurrences and the number of distinct user_id for each unique recording_msid. We selected the top 100 as recommendations for all users.

$$score = \beta \times num\ of\ occurence + (1-\beta) \times num\ of\ distinct\ user\_id$$
$$\beta \in [0,1]$$

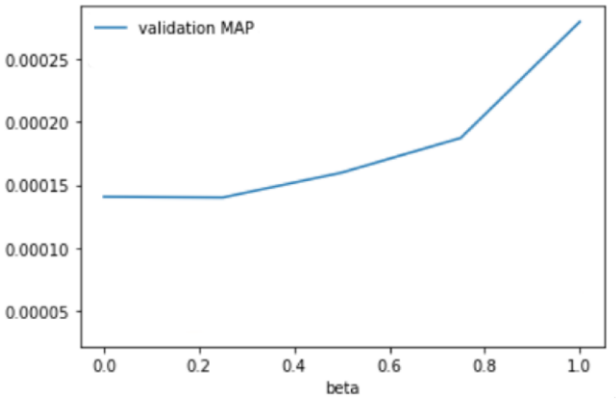
For the validation set, we grouped the data by user_id, and for each user, selected the top 100 most listened recording_msid based on occurrence as "truth". We created a map that converts recording_msid to double. The same recording_msid will be mapped to the same double. This can facilitate performance evaluation. We used Mean Average Precision (MAP) as the evaluation metric. The performance on validation set is shown below. Popularity solely based on the number of distinct users ($\beta = 1$) performs the best. Using this parameter, we evaluated the test set and achieved a test MAP of 6.32527e-05.

Table 2 Popularity Baseline Result

Beta	Validation MAP	Test MAP
0	0.000140545	N/A
0.25	0.000139939	N/A
0.50	0.000159766	N/A
0.75	0.000187298	N/A

1	**0.000279743**	**6.32527e-05**

Figure 1 Baseline Validation MAP



4 Data Filtering

To implement ALS and LightFM, we constructed a sparse matrix, where each row represents a user and each column represents music. The value ("rating") is the number of times each user listened to that music.

The first step of filtering involved removing rows with ratings below or equal to 4. The second step focused on items. We identified music that has been listened to by fewer than 5 users and excluded them.

From recommendation point of view, filtering removes low-rated items and those with limited user exposure. It can help enhance the relevance of the recommendation results. From computation point of view, filtering can significantly reduce the number of columns of the sparse matrix. The original dataset contains more than 20 million unique recording_msid, which consumes too much computation resource. After filtering, it is reduced to around 0.1 million, speeding up the computation.

5 Alternating Least Squares (ALS)

5.1 Model

The recommendation model used for this part is based on the Alternating Least Squares (ALS) method, an algorithm well-suited for collaborative filtering on large-scale implicit feedback datasets. ALS decomposes the user-item interaction matrix into lower-dimensional user and item factor matrices, which are learned alternately by fixing the other. The advantage of ALS over traditional matrix factorization methods lies in its scalability and parallelizability, which make it an ideal choice for big data applications.

5.2 Hyperparameter Tuning

Rank: This parameter determines the number of latent factors in the model, i.e., the dimensionality of the user and item factor matrices. We have tested the model with rank values of 10, 50, and 100.

Regularization Parameter (regParam): This parameter helps to prevent overfitting by constraining the magnitude of the model parameters. We have experimented with regParam values of 0.0001 and 0.01.

Implicit Feedback Parameter (alpha): This parameter controls the confidence in the observed user-item interactions. Higher alpha values indicate a stronger preference for observed interactions. We have used alpha values of 0.1, 0.5, and 1.

Each combination of these hyperparameters was used to train the ALS model, and the model performance was evaluated using Mean Average Precision (MAP) on the validation set.

Table 3 Alternating Least Squares Result

Rank	RegParam	Alpha	Validation MAP	Test MAP
10	0.0001	0.1	0.0156153527	N/A
10	0.0001	0.5	0.0156067829	N/A
10	0.0001	1	0.01561691709	N/A
10	0.01	0.1	0.01452286367	N/A
10	0.01	0.5	0.01452434019	N/A
10	0.01	1	0.01452382040	N/A
50	0.0001	0.1	0.01606887177	N/A
50	**0.0001**	**0.5**	**0.0160824253**	**0.0069804**
50	0.0001	1	0.01607954292	N/A
50	0.01	0.1	0.01540973645	N/A
50	0.01	0.5	0.01540689809	N/A
50	0.01	1	0.01540963319	N/A
100	0.0001	0.1	0.01568871527	N/A
100	0.0001	0.5	0.01568650232	N/A
100	0.0001	1	0.01568092358	N/A

5.3 Test Performance

The best performance on the validation set is achieved with a rank of 50, regParam of 0.0001, and alpha of 0.5, yielding a MAP score of 0.01608242534747942. When evaluated on the test set with these optimal hyperparameters, the model achieved a MAP score of 0.006980472856014383.

5.4 Model Efficiency

To assess the scalability of the ALS model, we evaluated the training time for different fractions of the total training data (25%, 50%, 75%, 100%). A fraction of 0.25 took longer than larger datasets might be due to the randomness of data. 25% subset of the dataset may contain a higher degree of variability or complexity compared to the 50% subset. But in general, the result suggests that the model scales up with the amount of data, which is an expected behavior for the ALS algorithm.

Table 4 Time Efficiency

Fraction of Data	Time (Seconds)
0.25	76.45

0.50	62.49
0.75	77.65
1.00	87.26

6 LightFM

6.1 Model

Compared to ALS using Spark which runs on distributed clusters, LighFM is a single machine implementation of recommendation algorithms. We are interested to see how accuracy and time differ.

6.2 Hyperparameter Tuning

No_components: This parameter determines the dimensionality of the feature latent embeddings. Increasing the value of this parameter will result in a higher-dimensional space and a more expressive model, but the complexity of the model and the computational resources required will also increase. We started with small values [10, 20, 30] and increased to [40, 50, 100] afterward.

Item_alpha: This parameter is the L2 penalty on item features. Setting this parameter too large will increase computation time. We experimented with small to intermediate values [0.0001, 0.001, 0.1].

User_alpha: This parameter is the L2 penalty on user features. Similarly, we experimented with small to intermediate values [0.0001, 0.01, 0.1].

We started by creating combinations of each item_alpha, user_alpha and no_components of 10, 20, 30 to train the LightFM model. The model performance was evaluated using Mean Average Precision (MAP) on the validation set. Based on the evaluation results, it became apparent that user_alpha and item_alpha of 0.0001 performed better than other values (see Table 5). So for the second part of the hyperparameter tuning, we fixed user_alpha and item_alpha and only increased the no_components to 40, 50, 100 to see if larger no_components give better results (Table 6).

Table 5 LightFM Result Part 1

no_components	item_alpha	user_alpha	Validation MAP
10	**0.0001**	**0.0001**	**0.0016640678**
10	0.0001	0.01	8.415384e-06
10	0.0001	0.1	9.482830e-06
10	0.01	0.0001	1.244517e-05
10	0.01	0.01	0.0003130987
10	0.01	0.1	7.574769e-06
10	0.1	0.0001	7.026318e-06
10	0.1	0.01	7.978077e-06
10	0.1	0.1	7.615795e-06
20	**0.0001**	**0.0001**	**0.0020178261**
20	0.0001	0.01	8.064729e-06
20	0.0001	0.1	9.866661e-06
20	0.01	0.0001	8.614182e-06
20	0.01	0.01	0.0001677566
20	0.01	0.1	6.656676e-06
20	0.1	0.0001	5.860273e-06
20	0.1	0.01	8.830651e-06
20	0.1	0.1	8.766760e-06
30	**0.0001**	**0.0001**	**0.0022209100**
30	0.0001	0.01	1.261631e-05
30	0.0001	0.1	1.017758e-05
30	0.01	0.0001	1.004042e-05
30	0.01	0.01	0.0001337667
30	0.01	0.1	1.007353e-05
30	0.1	0.0001	6.937612e-06
30	0.1	0.01	1.017886e-05
30	0.1	0.1	6.375648e-06

Table 6 LightFM Result Part 2 (no_components)

no_components	item_alpha	user_alpha	Validation MAP
10	0.0001	0.0001	0.0016640678
20	0.0001	0.0001	0.0020178261
30	0.0001	0.0001	0.0022209100
40	0.0001	0.0001	0.00233751165
50	0.0001	0.0001	0.00239770184
100	**0.0001**	**0.0001**	**0.00259791654**

6.3 Test Performance

The best performance on the validation set is achieved with item_alpha and user_alpha of 0.0001, yielding a MAP score of 0.00259791654. When evaluated on the test set with these optimal hyperparameters, the model achieved a MAP score of 0.00146392768.

6.4 Model Efficiency

To assess the scalability of LightFM model, we evaluated the training time of best model for different fractions of the total training data (25%, 50%, 75%, 100%). The modeling time progressively increases as a larger portion of the data is used (see Table 7). This is due to the increased computational complexity associated with larger datasets.

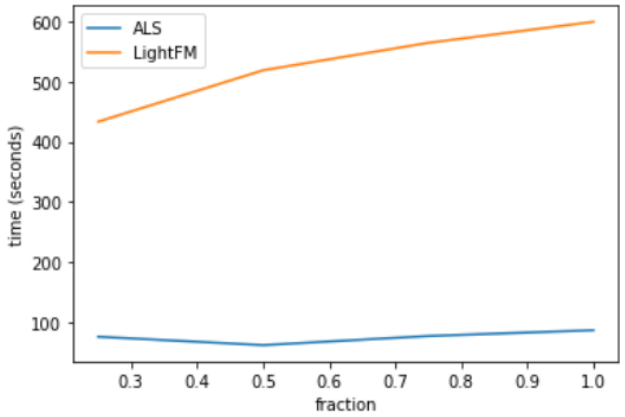
Table 7 Time Efficiency

Fraction of Data	Time (Seconds)
0.25	433.54
0.50	518.98
0.75	564.47
1.00	599.40

As the below figure shows, We compared LightFM with ALS. As expected, ALS exhibited significantly faster modeling times compared to LightFM, due to the distributed computing capabilities of Spark ALS in contrast to the single-machine

implementation of LightFM. Additionally, treating time as a function of data fraction, the slope of LightFM is steeper than that of ALS, meaning for the same level of data increase, ALS generalizes more efficiently. Taking into consideration single model efficiency, scalability, and MAP, ALS appears to be a better choice.

Figure 2 ALS and LightFM Time Comparison



7 Conclusion

In this project, we compared the performance and efficiency of one popularity baseline and two recommendation algorithms, ALS and LightFM, on a music recommendation task. We found that all models can provide meaningful recommendations based on implicit user feedback. However, when compared to the popularity baseline, ALS and LightFM exhibited a significantly higher Mean Average Precision (MAP) , indicating their effectiveness of providing personalized recommendations. ALS outperformed LightFM in terms of both recommendation quality (measured by MAP) and computational efficiency. Specifically, Spark ALS's distributed computing capabilities enable it to handle larger datasets more efficiently compared to single machine LightFM. Moreover, ALS showed better scalability, making it a more suitable choice for big data applications. Overall, our findings highlight the importance of considering both recommendation quality and computational resources when selecting a recommendation algorithm.

8 Team Contributions

Ivy Cui: Data Preprocessing, Alternating Least Squares (ALS), Report

Yuqing Cui: Data Preprocessing, Popularity Baseline Model, LightFM, Report

9 Project Files

GitHub Link: Final-project-group-40
Data Preprocessing and Filtering:

- data_split.py
- preprocess.py
- train_process.py

Popularity Baseline:

- popularity.py
- popularity_test.py

ALS:

- model_train.py
- model_train_v1.py
- als.py
- als_evaluate.py
- als_time.py

LightFM:

- model_lightfm.py
- model_lightfm_tuneN.py
- lightfm_evaluate.py
- lightfm_evaluate_N.py
- lightfm_time.py

Plot:

- plot.ipynb