DSGA 1004 Group 32 - Final Project Report

Yaozhong Huang, Ning Yang, Shanshan Gong
{yh2563, ny675, sg5507}@nyu.edu

https://github.com/nyu-big-data/final-project-group-32

New York University

May 2023

1 Introduction

In this project, we developed a music recommendation system leveraging the ListenBrainz dataset, which is a large-scale music listening dataset of millions of records. We employed big data technologies, particularly Pyspark, to train our models efficiently. With the development of the model, we experimented with different approaches to understand the difference between each model. We started from a popularity-baseline method and then implemented the alternating least squares(ALS) model and its several extensions.

2 Dataset and Preprocessing

2.1 Dataset

Our analysis is based on the ListenBrainz dataset, which contains around 600 million records. There are three types of data introduced in the dataset, including interaction, track, and user data. We mainly focused on the interaction data. The interaction data is composed of three attributes: *user_id*, *recording_msid*, and *timestamps*. The *user_id* is a unique numeric identifier allotted to each distinct user. The *recording_msid* serves as the unique identifier for each song, comprised of a combination of alphanumeric characters. This attribute is pivotal in tracking the specific songs that users interact with. Lastly, the *timestamps* attribute is a record of the listening time in seconds.

2.2 Data preprocessing

Besides the train-validation split, we also need to derive the average utility for each song for the baseline popularity-based model. To do this, we counted the interaction events between users and songs for training and validation sets. To be specific, we grouped the data by *user_id* and *recording_msid* and used the count function to obtain the play count for each interaction. In addition, we only kept the users with more than 10 listening records.

Moreover, since the data type of the original *recording_msid* is *string*, we have to convert it to *integer* type to fit into Spark's alternating least squares (ALS) model. First of all, We select all distinct *recording_msid* from the training, validation and test sets to ensure that we create a unified set of recording identifiers across all datasets. Then, we generate a unique *recording_id* for each distinct *recording_msid* using the `monotonically_increasing_id` function. Finally, we added the corresponding *recording_id* as a new column to the original train, validation, and test datasets separately based on the *recording_msid* column. At last, we saved the processed training and validation data as Parquet files for use in the later stages of the project.

2.3 Train/validation splits

Before training our recommendation model, we partitioned the train data into training and validation sets for the training purpose. To ensure that the training set includes the full user list and to avoid the cold start problem, we applied a train-validation split to each *user_id*. We first extracted the distinct *user_id* from the dataset and created a dictionary containing fractions representing the 80% partition for each *user_id*. In this way, we can partition all the interactions into train and validation set by *user_id*. Then, we use stratified sampling on the interactions dataset using the `sampleBy` function, which provided us with the training set. Finally, we applied the subtract function to obtain the validation set by removing the training data from the original interactions data.

3 Model and Evaluation

We totally created four models: popularity baseline model, ALS model, LightFM model, and ALS model with spatial data structure (Annoy) (the last two models are introduced in the extension section). In the evaluation phase, we used Mean Average Precision (MAP) as our primary metric. MAP is a measure of how many of the recommended recordings are in the set of true relevant recordings, where the order of the recommendations is taken into account. We utilized the built-in ranking metrics provided by Spark and evaluated our model based on predictions of the top 100 items for each user (MAP(100)).

3.1 Popularity baseline

Before delving into more complex recommendation models, we established a basic popularity-based model as a baseline. This approach recommends the most popular items (in this case, the most listened recordings) to all users. We took the top 100 songs from our training data and recommended these songs to all the users in the validation and test set. The calculation of popularity is based on the equation:

$$popularity = \frac{\text{count(listens)}}{\text{count(listener)} + \text{beta}}$$

In the validation and test sets, for each user, we fetched all their listened recordings because we only care whether the user listened to that recording or not instead of the number of listens.

3.1.1 Damping factor

The damping factor is a technique used to enhance recommendation systems by diminishing the influence of items with fewer listened users. It adjusts the utility of each item by adding a damping factor to the denominator. Consequently, all the items have their average utility decreases, whereas items with fewer listened users experience a larger decrease. This results in a more balanced distribution and increases the stability of the recommendations.

We experimented with various damping factors [0, 100, 1000, 10000] and we found that Mean Average Precision (MAP) becomes larger as beta is increased until beta is 10000. The optimal MAP is about 0.000206 for the test set. Details are shown in Table 1.

3.2 Latent Factor Model

To enhance personalization in our recommendations, we employed a Latent Factor Model, specifically utilizing the Alternating Least Squares (ALS) approach. ALS is a matrix factorization technique that decomposes the user-item interaction matrix into lower-dimensional user and item matrices.

3.2.1 Hyperparameter

Next, we applied the ALS model, adjusting variables: rank, regParam, coldStartStrategy, and nonnegative. We set coldStartStrategy to "drop" to avoid NaN evaluation metrics, and set nonnegative to "True" to only factorize the matrix into non-negative factors. What's more, we tried different combinations of ranks ([50, 100, 200]) and regParam ([0.001, 0.005, 0.01]). We discovered that the best performance on the validation set was achieved with a rank of 200 and a regularization parameter of 0.001. Results are shown in Table 2.

4 Extensions

4.1 Extension1: Comparison to single-machine implementations

First, we compared Spark's parallel ALS model to a single-machine implementation, LightFM. LightFM is a Python implementation of recommendation algorithms that is designed for single-machine operations, which contrasts with Apache Spark's parallel computing design. It allows the incorporation of both music item and user information into the traditional matrix factorization algorithms. Noting LightFM's advantage in sparse data, we intend to investigate the differences in efficiency and accuracy between LightFM and the ALS algorithms. As illustrated in Table 3, we performed hyperparameter tuning and optimized it with 'warp' loss, 0.04 learning rate, and 40 latent factors (nc-components), which is then used in the comparison results shown in Table 3.

4.2 Extension2: Fast search with spatial data structure

To improve the efficiency of our recommendation process, we use a spatial data structure to implement accelerated search at query time. Specifically, we used the Annoy library for approximated nearest neighbors search. By indexing the item vectors in a tree structure, Annoy allows for much quicker similarity computations compared to brute force methods. This efficiency gain is particularly valuable in scenarios with a large number of items like our dataset. The comparison of MAP and efficiency between different models is shown in Table 4.

5 Conclusion

During this project, we primarily investigated different recommendation models: baseline popularity model, popularity model with damping factors, latent factor model using alternating least squares (ALS), single-machine implementation of latent factor model (LightFM), and ALS with spatial data structure (Annoy).

Comparing all the models, we could conclude that: 1. For the popularity model, adding a damping factor would improve the model performance. 2. ALS performed better than the baseline popularity model, which provided personalized recommendations for each user. 3. Single-machine implementation of the ALS model using LightFM was more efficient and accurate than the ALS model. 4. Spatial data structure like partition trees accelerated search at query time and improved the precision compared to the original ALS model.

Contributions

Yaozhong Huang: data prepossessing and train/validation splits, fast search with Annoy, documentation for models and their evaluations

Ning Yang: popularity baseline model, ALS model, fast search with Annoy, and corresponding evaluations

Shanshan Gong: LightFM model, ALS model, and corresponding evaluations

References

Listenbrainz. ListenBrainz. (n.d.). `https://listenbrainz.org/`.

Evaluation metrics - RDD-based API. Evaluation Metrics - RDD-based API - Spark 3.0.1 Documentation. (n.d.). `https://spark.apache.org/docs/3.0.1/mllib-evaluation-metrics.html#ranking-systems`.

Lyst. (n.d.). Lyst/lightfm: A python implementation of lightfm, a hybrid recommendation algorithm. GitHub. `https://github.com/lyst/lightfm`.

Spotify. (n.d.). Spotify/Annoy: Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk. GitHub. `https://github.com/spotify/annoy`.

Appendix

Table 1: MAP Results for Baseline Popularity Model

Beta	**Training MAP(100)**	**Validation MAP(100)**	**Test MAP(100)**
0	6.34e-07	2.64e-06	8.57e-05
100	6.46e-06	1.27e-05	0.000120
1000	3.01e-05	6.39e-05	0.000143
10000	5.04e-05	8.28e-05	0.000206

Table 2: MAP Results for ALS Model

Rank	**Reg**	**Training MAP(100)**	**Validation MAP(100)**	**Test MAP(100)**
50	0.001	-	0.000702	-
50	0.005	-	0.000690	-
100	0.001	-	0.001421	-
100	0.005	-	0.001160	-
200	0.001	0.009950	0.002135	0.005964
200	0.005	-	0.002098	-

Table 3: MAP Results for LightFM

Loss Functions	**Learning Rate**	**Dimensionality (no components)**	**Validation MAP(100)**	**Test MAP(100)**
warp	0.04	10	0.003658	0.002906
warp	0.1	10	0.002766	0.003101
warp	0.04	40	0.001421	0.007549
bpr	0.04	10	0.001545	0.002994
bpr	0.1	10	0.003996	0.003490
bpr	0.1	40	0.007434	0.007101

Table 4: Extension Results

Model	**Best Test MAP(100)**	**Fitting Time (in seconds)**
Baseline	0.000206	652.4
ALS	0.005964	4201.7
LightFM	0.007549	10800
Annoy	0.008909	2789.9