DSGA 1004: Final Project- Recommender System

Weidong Liu wl2841@nyu.edu
Haohang Yan hy2664@nyu.edu
Group 150: https://github.com/nyu-big-data/final-project-150/tree/main

1 Introduction

In this project, we delved into the potential of collaborative filtering recommender systems using the ListenBrainz dataset. Our approach began with the development of a popularity-based model, followed by the implementation of an Alternating Least Squares (ALS) model to provide users with personalized song recommendations. Upon completion, we evaluated the efficiency of the ALS model, which was implemented in Spark running on a cluster, and compared it to a local recommender system that we developed using LightFM, with respect to dataset size. This comparison offered valuable insights into the performance of these recommender systems.

2 Data Preprocessing and partition

We first merged interaction data and track data. During the merging process, we combined recording_mbid and recording_msid into a new column named recording_id. The rule followed was to prioritize recording_mbid by assigning it to recording_id, and using recording_msid only when recording_mbid was absent. Subsequently, we filtered out inactive users who had fewer than 10 interactions. Finally, we randomly split the interaction data into an 80% training set and a 20% validation set.

3 Popularity Baseline Model

3.1 Implementation and hyperparameter tuning

We implemented our popularity model using PySpark, basing it on the song listen count. We first grouped by recording_id and calculated the song's listening count as song listen_count, then we grouped by (recording_id, listener) to get the distinct number of listeners for each song as listeners_count. In the popularity function, we utilized the formula listen_count/(listeners_count+beta), where beta acted as the damping factor. We iterated through beta with values of 0, 100, 1000, and 10000. As for our metrics, we employed Mean Average Precision at 100 (MAP@100) and Normalized Discounted Cumulative Gain at 100 (NDCG@100) to compare the top 100 songs we selected with the songs users listened to in both training and validation sets.

The following results were obtained from the hyperparameter tuning of the Baseline popularity model:

Beta Value	Train_MAP @100	Train_NDCG @100	Validation_MAP @100	Validation_NDCG @100
0	0.00297899257	0.0210717199	0.001256435	0.011223922
100	0.00297899257	0.0210717199	0.001256434	0.011202258
1000	0.00297974354	0.0210723594	0.001256690	0.011202493
10000	0.00297974354	0.0210723593	0.001256639	0.011202497

3.2 Evaluation

Our analysis determined that the optimal beta value for our model was 100. Upon applying this model to our test data, we obtained a MAP@100 result of 0.001034369 and an NDCG@100 result of 0.010402445.

4 Latent Factor Model：Alternating Least Sqaures(ALS)

4.1 Implementation and hyperparameter tuning

We built this als model to learn latent factor representation for listeners and songs using Pyspark library, We tuned our hyperparameters on the rank of the latent factor(rank) choosing from 10, 50, 100 the implicit feedback parameter(alpha) 1,10, 100 from and the regularization parameter(reg) from 1, 0.1, 0.01. In the ALS model setting, we use user_id as userCol, recording_id as itemCol, count as ratingCol. We have MaxIter to be 10 and then set implicitPrefs as True since we only have listen count information instead of rating. Here is the model performance on our validation set(rounded to 4 decimal places)：

map@100 score on validation	regularization=1	reg=0.1	reg=0.01	reg=0.001
alpha=1, rank=10	0.0174	0.0175	0.0199	0.0180
alpha=1, rank=50	0.0180	0.0182	0.0203	0.0187
alpha=1, rank=100	0.0187	0.0189	0.0235	0.0195
alpha=10, rank=10	0.0160	0.0169	0.0178	0.0172

alpha=10, rank=50	0.0172	0.0186	0.0189	0.0187
alpha=10, rank=100	0.0179	0.0190	0.0202	0.0190
alpha=100, rank=10	0.0155	0.0159	0.0166	0.0161
alpha=100, rank=50	0.0160	0.0169	0.0171	0.0171
alpha=100, rank=100	0.0160	0.0171	0.0184	0.0178

4.2 Evaluation

We determined that the optimal hyperparameters for our model were alpha=1, rank=100, and regularization=0.01. Subsequently, we applied the ALS, trained with these parameters, to our test data, yielding a map@100 result of 0.029590278578.

5 Extension: Comparing ALS and single-machine model

5.1 LightFM model implementation procedure

We carried out a single-machine implementation of the LightFM model using our original dataset. LightFM, a Python library specifically tailored for building recommendation systems, offers a unique collaborative filtering and content-based method, creating a hybrid approach to recommendations.

To investigate the relationship between the model fitting time and the dataset size, we began by downsampling our dataset to a fraction of 10% and then progressively increased the sample size. This allowed us to construct an interaction matrix between users and items, which was subsequently passed to the LightFM model.

In the final stage of our analysis, we measured the model fitting time and precision for different dataset sizes (with a fraction of 1 indicating that we utilized the entire training dataset).

5.2 Comparison with ALS

The results from the LightFM model(with epoches=50) were then compared to those obtained from the ALS model(), providing us with a comprehensive overview of the performance of both models. The

Dataset_size	Training time	Training_precision @100	Validation_precision @100	Test_precision @100
LightFM				
Frac = 10%	572.779s	0.042051195761	0.005202312915	0.004219519502

Frac = 50%	2671.153s	0.034138407384	0.006851195752	0.007002852910
Frac = 100%	5525.829s	0.021929104262	0.013129235788	0.010951005736
ALS				
Frac = 10%	2210.127s	0.288742001929	0.005871238904	0.005194105849
Frac = 50%	4061.019s	0.184919057398	0.015980012847	0.019482159039
Frac = 100%	5919.856s	0.178471948565	0.023519523302	0.029590278578

5.2.1 Efficiency

The table shows that LightFM runs much faster than spark's parallel ALS model especially when the dataset is small enough. For example, with a fraction of 10% of the data set, the ALS model takes roughly 4 times longer to train than the LightFM model (2210.127s vs 572.779s). However, when the dataset is becoming larger and larger, the model training process seems to accelerate faster. When we use the whole dataset, the model fitting time for ALS is just slightly larger than that of LightFM model(5919.856s vs 5525.829s).

5.2.2 Accuracy(Precision@100)

When evaluating prediction precision, the ALS model appears to have an edge over LightFM, especially as the size of the dataset increases. For example, with a fraction of 10% of the data set, the test precision of the ALS model is higher than that of LightFM (0.005 vs 0.004) and this trend continues as the data set size increases. When we use the whole dataset, the test precisions are 0.029 and 0.010, the ALS one is much larger.
In conclusion, while the LightFM model is more efficient for small datasets, the ALS model generally provides more accuracy and might be faster when the dataset is larger.

6 Contributions

Weidong Liu(wl2841): Preprocessed data, refined the baseline model, constructed and enhanced the LightFM model, and handled the related documentation.
Haohang Yan(hy2664): Constructed the baseline model, constructed and enhanced the ALS model, and handled the related documentation.