Recommendation System of ListenBrainz

DSGA 1004 Big Data Final Project: https://github.com/nyu-big-data/final-project-group-34

Yuhao Liu [†]
Center of Data Science
New York University
United State
yl5308@nyu.edu

Qingwei Meng
Center of Data Science
New York University
United State
qm351@nyu.edu

ABSTRACT

Recommendation systems have gained immense popularity in the current digital age as they can predict user preferences and offer relevant items to users. This essay involved constructing recommendation system using Spark's alternating least squares (ALS) method. The ListenBrainz dataset is employed in this project to build a basic recommender system that uses Spark's ALS method to acquire latent factor representations for users and items. The interaction data is partitioned into training and validation samples, and a popularity baseline model is implemented before moving on to the latent factor model. The hyper-parameters are fine-tuned to optimize performance on the validation set, and Spark's ranking metrics are implemented to assess accuracy on the validation and test data. Then we compare Spark's parallel ALS model to LightFM to measure both efficiency and resulting accuracy.

CCS CONCEPTS

• Recommendation System • Latent factor model • Spark ALS Method

KEYWORDS

Recommendation System, Spark, ALS, Latent factor model, Baseline model, LightFM

Basic recommender system

The first part discusses the process of building a recommendation system using ALS method, all are available at the URL https://github.com/nyu-big-data/final-project-group-34. The first step involves partitioning the interaction data (both small and larger) into training and validation samples, build the popularity baseline model, and tune the hyper-parameters, the damping factors, to optimize the system's performance on the validation set.

1. Data Preprocessing

Before we start to build the recommendation model, the first thing we do is to preprocess the data:

1. Only keep users who have listened to at least 10 songs to prevent outlier problems.
2. Split the interaction training dataset into training and validation set with 80/20 split.
3. Join the interaction dataset with the track dataset to get the track information for each interaction.
4. Use the "recording_mbid" column as the track id. If "recording_mbid" is missing, impute it with the value in the "recording_msid" column.
5. Drop unnecessary columns (“timestamps”, “artist_name”, “track_name”) from the dataset.

We just keep the column “recording_mbid” and “user_id” and use it to calculate the popularity ratings and build our popularity baseline model.

2. Popularity Baseline Model

Then we start to build the popularity baseline model for the small and large data with the following steps:

1. To calculate the popularity score of each track, we use the formula we learned in class.

$$P[i] \leftarrow \left(\sum_u R[u,i]\right) / \left(|R[:,i]| + \beta\right)$$

Figure 1: Popularity calculation formula

2. Next, we calculate the popularity score for all tracks in both the small training and validation sets.
3. We calculate the mean average precision (MAP) at 100 and NDCG@100 using the popularity scores from the small training set as our predictions and those from the small validation set as the ground truth., with the **pyspark.mllib.evaluation.RankingMetrics** library.
4. We tune the damping factor hyper-parameter and determine that a beta value to maximize the result of MAP@100 and NDCG@100. We try to ensure that our baseline model works properly for the small dataset.
5. Then we repeat the same process for our large train and validation dataset. We get beta values more than 1620 yields

good performance (MAP@100>0.0001 and NDCG@100>0.001) for the popularity baseline model on the validation set.

6. When beta=10000, we achieve the best result for our baseline model:
   a) **Validation dataset**: MAP@100=0.00025
      NDCG@100=0.003

   b) **Test dataset**: MAP@100=0.00019488
      NDCG@100=0.00094784.

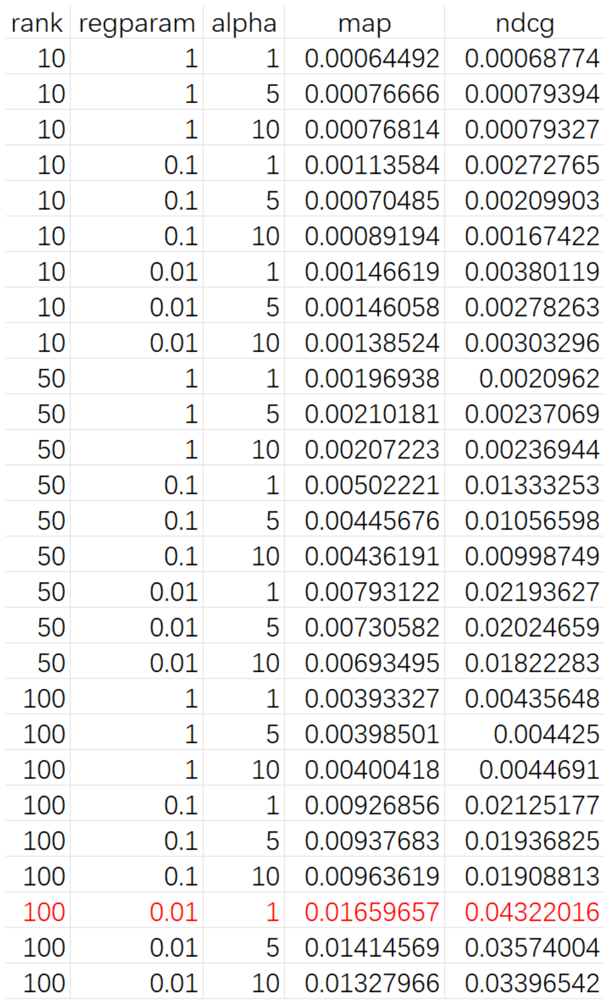
3. Latent Factor Model and evaluation

In order to optimize the performance of our recommendation model, we leverage Spark's alternating least squares (ALS) method to learn latent factor representations for users and items. Prior to applying the ALS function, we convert the "mbid" field into integers by ranking them. This enables us to incorporate the "mbid" into the ALS function correctly. To build our model, we utilize the pyspark.ml.recommendation module.

Due to limited resources on the cluster, we attempt various methods to subsample our extensive training dataset, such as repartition and reduce maxIter parameter in ALS function. However, despite our efforts, we are still unable to obtain the answer with the large training set and limited cluster resources. Consequently, we only run the ALS model on our smaller training and validation datasets. The ALS model takes approximately 50 minutes to run.

To evaluate the model's effectiveness, we compare the top 100 recommendations for each user using recommendForUserSubset() against the ground truth results. We fine-tune the hyperparameters, such as the rank of the latent factors, implicit feedback parameter (alpha), and regularization parameter, to achieve optimal outcomes on the validation set. We choose the rank from 10 to 200, alpha from 1 to 10, regularization parameter from 0.01 to1. We set the maxIter to 3 to reduce the running time of ALS model. The results of the validation set are displayed in Figure 2.

From the table, we observe that the ALS model performs best when the rank is set to 100, the regression parameter is 0.01, and alpha is 1. We employ these hyperparameters to evaluate our test dataset. The test dataset yields a MAP@100 of 0.01887425 and an NDCG@100 of 0.05337379. There is a significant improvement compared to the popularity baseline model, indicating that the ALS model outperforms the baseline model.



rank	regparam	alpha	map	ndcg
10	1	1	0.00064492	0.00068774
10	1	5	0.00076666	0.00079394
10	1	10	0.00076814	0.00079327
10	0.1	1	0.00113584	0.00272765
10	0.1	5	0.00070485	0.00209903
10	0.1	10	0.00089194	0.00167422
10	0.01	1	0.00146619	0.00380119
10	0.01	5	0.00146058	0.00278263
10	0.01	10	0.00138524	0.00303296
50	1	1	0.00196938	0.0020962
50	1	5	0.00210181	0.00237069
50	1	10	0.00207223	0.00236944
50	0.1	1	0.00502221	0.01333253
50	0.1	5	0.00445676	0.01056598
50	0.1	10	0.00436191	0.00998749
50	0.01	1	0.00793122	0.02193627
50	0.01	5	0.00730582	0.02024659
50	0.01	10	0.00693495	0.01822283
100	1	1	0.00393327	0.00435648
100	1	5	0.00398501	0.004425
100	1	10	0.00400418	0.0044691
100	0.1	1	0.00926856	0.02125177
100	0.1	5	0.00937683	0.01936825
100	0.1	10	0.00963619	0.01908813
100	0.01	1	0.01659657	0.04322016
100	0.01	5	0.01414569	0.03574004
100	0.01	10	0.01327966	0.03396542

Figure 2: ALS model tuning hyper-parameters on validation set.

4. Extension: LightFM on Single Machine

Then we start to compare Spark's parallel ALS model to a single-machine implementation, LightFM, considering both efficiency and resulting accuracy. "LightFM is a Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback, including efficient implementation of BPR and WARP ranking losses. It's easy to use, fast (via multithreaded model estimation), and produces high quality results. It also makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalize to new items (via item features) and to new users (via user features)." However, the current LightFM only supports NumPy and sparse libraries, which is not an ideal tool to analyze recommender system within the big data context.

Due to LightFM's single machine architecture and limited resources, we have only used a reasonable sample (10 million rows) of small datasets to run LightFM. The running time for LightFM is 251.9705786705017 seconds for the sample dataset. The precision at k for LightFM is 0.010029764845967293. The running time for ALS model is 50 minutes and the precision at k is 0.003. The time taken for LightFM model is much less than ALS model and the precision at k value is also larger for LightFM model.

5. Contributions

Yuhao Liu: Baseline model, ALS model, Report
Qingwei Meng: Baseline model, Extension

REFERENCES

[1] NYU Big Data, Final Project Group 34. 2023. GitHub Repository: Final Project Group 34. GitHub. https://github.com/nyu-big-data/final-project-group-34. (Accessed May 16, 2023).

[2] Apache Spark. (n.d.). Ranking Metrics - Apache Spark 3.0.1 Documentation. https://spark.apache.org/docs/3.0.1/mllib-evaluation-metrics.html#ranking-systems (Accessed May 16, 2023).

[3] Spark Collaborative Filtering. https://spark.apache.org/docs/3.0.1/ml-collaborative-filtering.html. (Accessed May 16, 2023).

[4] LightFM. https://github.com/lyst/lightfm. (Accessed May 16, 2023).