DS-GA 1004 Big Data Final Project: Music Recommendations

Yuchen Dang yd1008@nyu.edu
Tong Li tl2204@nyu.edu
Wenjie Shao ws2237@nyu.edu
Shixuan Song ss14381@nyu.edu

1. Introduction

In our project, we implement a music recommendation system using collaborative filtering [6] on the Million Song Dataset (MSD) [5] using Spark [1] with distributed systems. Moreover, we adopt two extensions to this recommendation system. We start by running this algorithm on a single machine to compare the performance and then move to build a popularity baseline model to measure how much improvement we gain by running the collaborative filter.

2. Dataset Processing

2.1. Downsampling

To avoid overloading the clusters while still maintaining enough information from the full dataset, we downsample the training data in a slightly different manner compared with traditional methods. During tuning, instead of selecting a small fraction of all the interactions, we randomly choose 25 percent of the users that are not present in the validation data and keep all shared users in both training and validation groups. This will preserve all the interactions associated with a specific user. In the final tuning stage where we have narrowed down our range of parameters to choose, we train on the full dataset for validation.

2.2. String Indexing

In the original dataset, both user_id and track_id are formatted as strings. However, it will raise an error in the implementation, since the current API for ALS only supports inputs of user-ids and item-ids in integer values [3]. Hence, we have to index both columns with Pyspark's StringIndexer function. To ensure the correspondence between the original indexes and the transformed integer representations in training, validation, and testing sets, all three datasets are transformed using the same indexer fitted solely on the training data. We apply the similar steps for our two extensions.

3. Implementation

3.1. Collaborative Filtering using PySpark

In this model, we implement the Alternating Least Squares (ALS) function from PySpark. Since we do not directly observe the feedback, we need to use the counts as the implicit feedback schema, which is reflected in the ALS function by setting the parameter implicitPrefs to True. We then fit this model on the training set and save it for reference. For each user in the validation set, we produce 500 recommendations ranked by the score. After converting the predictions to RDD, we adopt the ranking metrics provided by spark to derive Precision at 500, MAP, and

NDCG (500) scores based on the ground truth items in the validation set. Based on the validation performance, we tuned several parameters such as rank, max iteration, alpha, and regularization strength.

3.2. Single Machine CF Implementation with LensKit

To compare the single machine performance with Spark's parallel system, we also implement the ALS model with LensKit on a single machine. Since LensKit does not rank metrics, we write the three metrics from scratch based on Spark's definition. We use the same set of parameters derived from Spark ALS to train and obtain the validation and test results.

3.3. Popularity baseline model with LensKit

Since the ALS model requires a lot of time training and tuning, we also want to develop a popularity baseline model to measure the gain we receive from ALS [2]. We use the LensKit package and apply the Bias algorithm on our training set to generate 500 predictions for users in the validation set. The damping parameter is then tuned based on the validation performance with respect to three metrics.

4. Evaluation

For all three implementations, we measure the performance on three metrics: Precision at 500 (P-500), Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain at 500 (NDCG-500), as defined by Spark's ranking metrics [4]. By checking the recommendations returned by the algorithms, we notice that for PySpark's ALS and LensKit's ALS, the recommendations are generally different for each user and each user gets 500 different items. While for the Popularity-based Bias model, each user gets the same set of predictions. The parameters are chosen based on the highest validation performance. From the preliminary results we have, we notice that the model improves when a larger rank is used. Fig 1 below shows the preliminary results we have before we fine tune the model on the full training set.

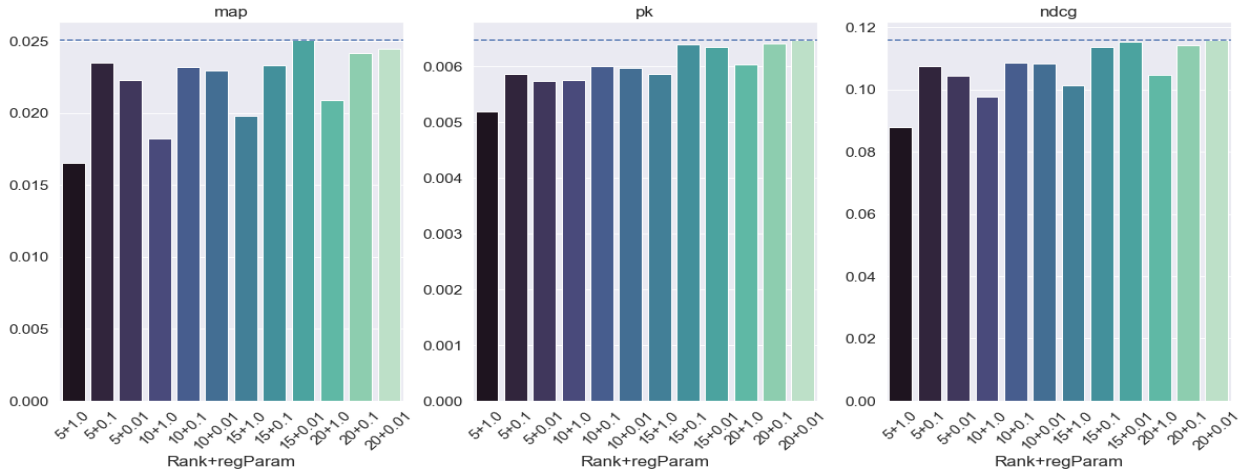

Fig 1. Preliminary Results on PySpark ALS by Rank and Regularization Strength

The parameter settings as well as their corresponding validation and test performance are shown in the table below. (p.s. Peel is not able to return the final test results before the deadline, but we expect the test results to be close to the validation performance. So we provide an estimated result.)

Precision at 500			
Implementation	Parameters	Validation	Test
PySpark ALS	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.01044	Pending Peel Results Estimated ~0.01044
Single Machine	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.01122	0.01121
Popularity Baseline	item_bias = 1M user_bias = 10M	0.00302	0.003124

Mean Average Precision			
Implementation	Parameters	Validation	Test
PySpark ALS	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.0408	Pending Peel Results Estimated ~0.0408
Single Machine	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.0439	0.0435
Popularity Baseline	item_bias = 1M user_bias = 10M	0.0084	0.0119

Normalized Discounted Cumulative Gain at 500			
Implementation	Parameters	Validation	Test
PySpark ALS	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.182	Pending Peel Results Estimated ~0.182
Single Machine	rank = 200, regParam = 0.1, alpha = 1, max iter = 20	0.242	0.2408
Popularity Baseline	item_bias = 1M user_bias = 10M	0.0498	0.0672

When implementing on a single machine, the time taken to train and generate recommendations is much longer than PySpark on peel. However, we notice that it can yield slightly better performance on the three metrics.

The popularity-based baseline model (Bias model) is much quicker to train, while still slow to generate recommendations. Also, this model has the weakest performance. ALS models achieve significantly better results (four to five times better). Thus, the additional time it takes to train the ALS is worth doing.

However, we do notice that even the highest performing ALS models still have very low scores. By looking at the validation set items, we notice that each user may not have 500 items associated with him/her. Thus, Precision at 500 and MAP scores may be an underestimate.

5. Conclusion

For this project, we use the collaborative system to build a music recommender system along with two extensions: (1) single-machine implementation and (2) popularity baseline models. By evaluating the final results, we find that ALS on a single machine achieves the best performance, but it is only slightly better than the parallel system in PySpark while taking much longer time to yield results. We also train a popularity baseline model to measure if ALS has significant improvement. And we conclude that ALS is much better than the baseline model for this task.

6. Contribution

PySpark ALS: Dang (majority of code and tuning), Li (code revision and tuning)
Single Machine ALS: Li
Popularity Baseline: Li, Shao, Song

REFERENCES

[1] Apache Spark - Unified Analytics Engine for Big Data. (n.d.). spark.apache.org/.

[2] Bias. Bias-LensKit 0.12.3 documentation. (n.d.). https://lkpy.readthedocs.io/en/stable/bias.html.

[3] Collaborative Filtering. Collaborative Filtering - Spark 2.4.7 Documentation. (n.d.). https://spark.apache.org/docs/2.4.7/ml-collaborative-filtering.html.

[4] Evaluation Metrics - RDD-based API. Evaluation Metrics - RDD-based API - Spark 2.4.7 Documentation. (n.d.). https://spark.apache.org/docs/2.4.7/mllib-evaluation-metrics.html#ranking-systems.

[5] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[6] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. 2008 Eighth IEEE International Conference on Data Mining. DOI 10.1109/ICDM.2008.22