DS GA-1004 Term Project: Million Song Dataset Recommender System

ZHIFAN NAN, New York University, USA
LONG CHEN, New York University, USA
YUHAN GAO, New York University, USA
YUHAN LIU, New York University, USA

CCS Concepts: • **Information systems** → **Personalization**; • **Human-centered computing** → *Visualization toolkits.*

Additional Key Words and Phrases: recommender system, collaborative filtering, fast search, visualization

1 INTRODUCTION

The objective of this project is to build a recommender system (recSys) based on the Million Song Dataset (MSD) located on HDFS. The interaction data consists of implicit feedback: *count*, which indicates how many times a user listened to a specific track. For the baseline model, we applied Spark's alternating least squares (ALS) algorithm, which is built for large-scale collaborative filtering problems and good at solving scalability and sparseness of the data. We then fine tuned several hyperparameters including rank, regularization, and alpha using grid search to obtain the model with the best performance on the validation set. The model performance is measured by two major ranking metrics, the Mean average precision (MAP) and Precision at K. For the extensions, considering the massive amount of tracks in our dataset, we implemented Annoy (Approximate Nearest Neighbors) to accelerate the searching during query time. Also, to better understand the latent factors learned by the model, we visualized the track representations together with their genres with the help of t-SNE, which provides interesting and insightful findings and further validate our model's quality.

2 BASELINE MODEL

In this section, we introduce our approaches as well as findings to develop a baseline model for the recSys.

2.1 Dataset and Preprocessing

The project used the Million Song Dataset [2]. Specifically, we worked on the Taste Profile Subset, which contains music listening activity for each user as a triplet of (user_id, track_id, play_count), that is, each entry represents the count of plays of one music trach by one user.

Preprocessing. To prepare our dataset for our recSys model, we convert user_id and track_id columns to numeric. For this purpose, StringIndexer in Spark (pyspark.ml.feature) allows us to transform the textual data to numeric form while keeping categorical. The indexers for both features are separately trained using our training corpus and then applied to training, validation and test corpus, then saved these processed datasets to HDFS for training and evaluation purpose.

2.2 Model

We deployed collaborative filtering (CF) model for our recommendation task [3]. An assumption of CF is that users sustain their behavior and will make similar decisions in the future. Therefore, only the past activities of users are required and no features of the contents are needed, which perfectly suits our dataset.

For implementation, the Alternating Least Square (ALS) model in Spark (spark.ml.recommendation) is used. ALS [4] employs matrix factorization technique which decomposes the dataset into the product of two lower dimensional matrices, one user matrix where for each row as user, the columns represent the latent factors; and the other one item matrix (track matrix in our project) where for each column as track, the rows represent latent factors. Intuitively, ALS investigates the latent factors for users and tracks to find factor weights that minimize the loss of ratings (in this case the count of plays). The ALS model in Spark library has capability of running in parallel.

Rank \ Alpha	5	10	15
20	0.02227	0.01548	0.01548
30	0.0254	0.01856	0.01527
40	0.02793	0.02258	0.01928
50	0.02646	0.02333	0.0208

(a) regParam=0.01

Rank \ Alpha	5	10	15
20	0.03001	0.02670	0.02376
30	0.03144	0.02971	0.02422
40	0.03225	0.03140	0.02800
50	0.03369	0.03265	0.02812

(b) regParam=0.1

Rank \ Alpha	5	10	15
20	0.02880	0.02902	0.02967
30	0.03045	0.03076	0.03238
40	0.03213	0.03256	0.03260
50	0.03344	0.03423	**0.03458**

(c) regParam=1

Rank \ Alpha	5	10	15
20	0.014	0.0185	0.0201
30	0.0142	0.0196	0.0229
40	0.0146	0.0212	0.0245
50	0.0146	0.0218	0.0257

(d) regParam=10

Table 1. MAP score of hyper-parameter tuning. A setting of rank=50, regParam=1, and alpha=15 has best validation MAP of 0.03458.

2.3 Evaluation Metrics

We implemented ranking metrics provided by Spark library to evaluate the performance of our model. More specifically, we selected Mean Average Precision (MAP) and Precision at K as the two metrics for our hyperparameter tuning and performance evaluation. Given the magnitude of the datasets and the computation capacity, we chose $K = 500$ to evaluate our model.

- **Mean Average Precision**: It's the average of Average Precision (AP) across all users, and AP will reward the system not only having more relevant items in the recommendation list but also front-loading those correct ones in the list. [1]
- **Precision at K**: It's a measurement of what percentage of the top K recommended items are in the collection of true relevant items for a user, and taking the average across all users, while it does not take the order into consideration.

2.4 Hyper-parameter Tuning

We applied grid search technique to search for the best hyper-parameters and evaluate the model using validation set. For the baseline ALS model, we choose three hyper-parameters which we believe will have significant effects on the model performance: rank, regParam, and alpha. As ALS attempts to estimate the ratings matrix R as the product of two matrices and iteratively minimize the mean square error, rank is arguably the important hyper-parameter as it is the dimension of latent factors in the model, representing the complexity of the distributed representation of the user and item matrices. Higher rank might lead to better results, but at the same time, need more memory and computation time and probably causing overfitting to certain extent. Second, regParam is the regularization parameter limiting the magnitude of latent factor representations, which prevents the model from overfitting. Lastly, we selected alpha as the final hyper-parameter to tune because it could potentially improves the popularity baseline by controlling the confidence that the model put on users' play count observations. Here we list the values we tried for each hyper-parameter:

- rank: [20, 30, 40, 50]
- regParam: [0.01, 0.1, 1, 10]
- alpha: [5, 10, 15]

For evaluation, we prioritize MAP over Precision at k since MAP provides more interpretability and takes ordering into consideration that are more in line with the objective of the project. The full result of hyper-parameter tuning is shown in Table 1, with the set of best performing hyper-parameters rank=50, regParam=1, and alpha=15. The best validation MAP is 0.034575 and the precision at 500 is 0.00906.

2.5 Testing Performance

Testing our optimal model on the test set, the testing MAP is 0.03477, and the precision at 500 is 0.00904.

[1] https://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html

3 EXTENSIONS

3.1 Fast Search with ANNOY

Ouerying for the most relevant items for a user is a time-consuming process considering the huge amount of tracks in our dataset. Using a brute-force search will take $\mathcal{O}(n)$ by computing the dot product between the user factor and each item factor, which would be impractical and inefficient in lots of real time applications. **Annoy**(Approximate Nearest Neighbors) is a library that can efficiently search for nearest neighbors in high-dimensional space, and it allows us to do queries in $\mathcal{O}(\log n)$ with the help of spatial data structure. We adopted this library to accelerate the retrieval process required for making recommendation to users, and it greatly reduces the query time without sacrificing much accuracy.

Annoy utilizes binary trees to recursively partition the space into smaller subspaces, where each node represents a hyperplane that was randomly chosen to split a particular subspace and its child nodes defines the two sides of this hyperplane. The intuition behind this algorithm is that if two points are close in the original space, it's unlikely that any hyperplane can separate them, and they will very likely end up in the same leaf node of the tree. Therefore, to search the nearest neighbors for any given point, we can traverse the binary tree from the root down to the leaf node, which is essentially a process of iteratively asking the question of which side of the hyperplace to go, and finally the leaf node contains the nearest neighbors for that point. It also uses tricks like priority queue and building a forest of trees to improve the accuracy and recall of the search algorithm.[2]

In order to work smoothly with Annoy, we need to first export the representations learned from our optimal model (rank=50, reg=1, alpha=15) from HDFS to our local machine and use Annoy's python API [3] to build a forest of trees based on all the learned item factors. During querying time, given a user factor, Annoy will traverse all the trees and find the union set of the neighboring items for that user factor, and it will only compute the dot product between the user factor and each item factor within that candidate set. Finally, it will sort the items based on the dot product results and recommend the top500 items in order from the candidate set.

To compare the performance between Annoy and Brute-force search, we recorded their evaluation time that includes making recommendation for all users in the validation set and computing the ranking metrics. We also tried to alter the number of trees in the forest to understand their impact on the searching performance. As shown in Table 2, the efficiency gain from Annoy is quite impressive, while at the same time sacrificing some accuracy compared to the brute-force method. We can see as the number of trees grows, the MAP increases and so does the evaluation time, which is the kind of tradeoff that needs to be taken into consideration by any decision maker. In our case, by setting the number of tree to 300, we can save the evaluation time was by almost 2/3 with only 0.006 drop on MAP and 0.0001 drop on Precision@500. Recommendation systems with such magnitude of efficiency, together with the comparative accuracy, provides users a more delightful and engaged experience and ultimately derive higher profits for the company.

# of trees	MAP	Prec@500	Evaluation Time (s)
50	0.01767	0.00500	185.40
100	0.02180	0.00629	224.75
150	0.02361	0.00695	256.94
200	0.02651	0.00751	286.64
250	0.02771	0.00782	319.48
300	0.02856	0.00800	344.65
Brute-force	0.03457	0.00906	932.19

Table 2. ANNOY vs Brute-force

3.2 Genre Exploration

To better understand the latent factors learned by our model, we did an item-based visualization with the help of MSD allmusic Genre Dataset (MAGD)[4] provided by the Information Management and Preservation Lab at the Department of Software Technology and Interactive Systems at Vienna University of Technology. This dataset contains a curated set of genre labels for most of the

[2]https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html
[3]https://github.com/spotify/annoy
[4]http://www.ifs.tuwien.ac.at/mir/msd/partitions/msd-topMAGD-genreAssignment.cls

tracks in our MSD, and these labels can be easily merged to our data through the *track id* field. To make our visualization more clear and intuitive, we carefully selected 10 genres by excluding genres with huge amount of tracks like Pop/Rock and extremly small genres like Avant-garde and Holiday. Here we list the 10 genres we included:

- 'Blues', 'Country','Folk', 'Latin', 'Metal', 'Punk', 'Rap', 'Reggae', 'RnB', 'World'

The visualization technique we used is **T-SNE** (t-distributed Stochastic Neighbor Embedding) [7], which is a powerful tool for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional space. More specifically, if two datapoints are close in the original high-dimensional space, they will be also shown by T-SNE as nearby points, otherwise they will locate distant from each other.

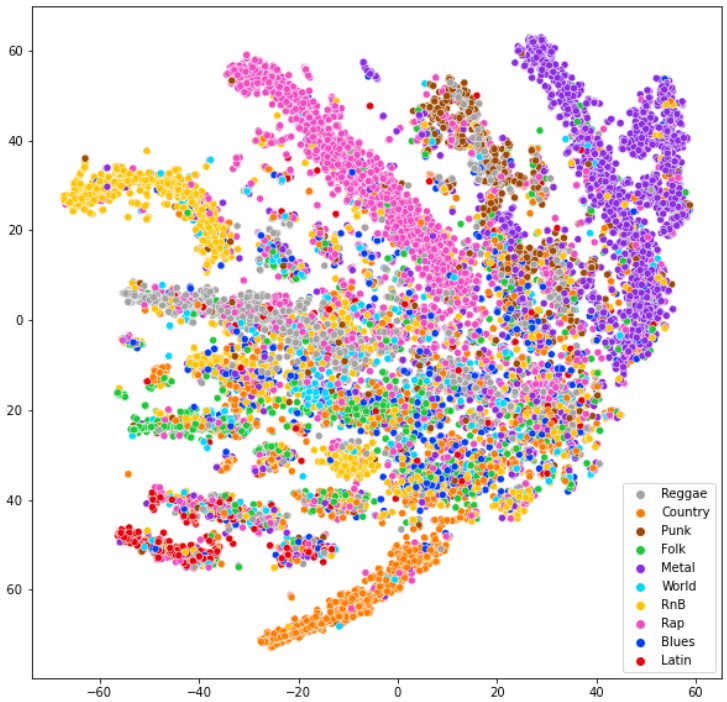

Fig. 1. genre clustering

We visualized the latent item factors learned by our optimal model (rank=50, reg=1, alpha=15) together with their genres, which is shown in figure 1. It explicitly reveals very informative and interesting patterns. For example, we can see that Reggae and R&B are two clusters that are very close to each other, and it turned out that Reggae was actually developed from Jamaican R&B, which is largely based on American R&B. Therefore, our model accurately captures this intimate relationship between these two genres. Other adjacent pairs include punk and metal, folk and country, as well as rap and R&B. All of them are well-known closely related genres, and our model learned such patterns to a great extent.

4 TEAM CONTRIBUTIONS

- **Long Chen**: Baseline Model, Extensions, Final Report
- **Zhifan Nan**: Hyper-Parameter Tuning, Extensions, Final Report
- **Yuhan Gao**: Baseline Model, Hyper-Parameter Tuning, Final Report
- **Yuhan Liu**: Hyper-Parameter Tuning, Extensions, Final Report

REFERENCES

[1] Fabio Aiolli. 2013. A Preliminary Study on a Recommender System for the Million Songs Dataset Challenge.. In *IIR*. Citeseer, 73–83.

[2] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. 2011. The million song dataset. (2011).

[3] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. 2000. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. 241–250.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[5] Colin Larkin. [n.d.]. Ska. ([n. d.]).

[6] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. 2012. The million song dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web*. 909–916.

[7] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).