DS-GA 1004 Music Recommender System

Ningyuan Huang (nh1724), Zacharie Martin (zm922), Amy Lei (al4064)

I. Introduction

We implement a collaborative filtering recommender system using the Alternating Least Squares (ALS) algorithm to learn a low-rank matrix factorization of a large data matrix R into matrices U and V such that $R \approx UV^T$. The data matrix R contains the count data for each track by user, i.e., entry $R_{i,j}$ is the number of times user i listened to track j. The learned user factors are stored in the matrix U so that the *i-th* row of U, u_i^T, is the vector of latent factors for user i. Similarly, the vector v_j is the *j-th* column of the matrix V^T. The rank of U, V is a parameter in the ALS algorithm; setting `rank = 10` implies $u_i^T, v_j \in \mathbb{R}^{10}$. [3]

Our data contains implicit user feedback (counts), as opposed to explicit user feedback (ratings). Therefore, we use the implicit ALS algorithm, which solves the following non-convex optimization problem:

$$_{U,V} \sum_{i,j} c_{i,j}(p_{i,j} - u_i^T v_j)^2 + \lambda(\sum_i \|x_i\|^2 + \sum_j \|v_j\|^2)$$

where the preference p is mapped to:

$$p_{i,j} = \begin{cases} 1 & \text{if } r_{i,j} > 0 \\ 0 & \text{if } r_{i,j} = 0 \end{cases}$$

Preference is a binary indication of whether there is an user-item interaction. The confidence c is defined as:

$$c_{i,j} = 1 + \alpha r_{i,j}$$

The confidence accounts for the weight per interaction, where α is a scaling factor. It adds 1 to ensure there is a minimal confidence even the user never plays the track. This is the approach taken by Hu et al. '08. [1] Looking at the optimization problem closely, we observe that the algorithm will try to push $u_i^T v_j \rightarrow 1$ on sample $p_{i,j}$ if $c_{i,j}$ has a relatively high value. This is necessary for recommending items to users. When implementing the implicit ALS algorithm, we perform hyper-parameter tuning on its rank, regularization (λ) and confidence scale (α) parameters.

II. Implementation

A. Key Dataset Overview

Table I
Dataset Overview

Dataset	# Records	# Unique Users	# Unique Tracks
cf_train	49,824,519	1,129,318	385,371
cf_validation	135,938	10,000	50,074
cf_test	1,368,430	100,000	159,717
metadata	1,000,000	NA	1,000,000

B. Preparation of Training Data

Table I indicates that there are approximately 350X more records in the training data than in the validation data. We attempt fitting the initial model on the full training data, but the High Performance Computing (HPC) job crashes due to the dataset's large size. Therefore, we downsample the training data to select only the records corresponding to the users that are in `cf_validation` and `cf_test`. This downsampling ensures that the user-item interactions in the validation and test data are accounted for when we perform hyper-parameter tuning and predict on the `cf_test`. The downsampled training data `data_final` contains 27,010,946 records, 110,000 unique users, and 163,206 unique tracks.

C. Indexers and Model Pipeline

Since the ALS model can only handle numeric data, we index all users using training data and all tracks using `metadata`. It is critical to use metadata as the indexer because it contains information for all tracks, which allows us to avoid the cold-start

issue when fitting the model to `cf_validation` and `cf_test`. We implement a pipeline with track_indexer, user_indexer and ALS model using the Spark `ml.recommendation.ALS` module.

III. Evaluation Results

A. Metrics

We use Mean Average Precision (MAP) to evaluate the top 500 tracks recommended for each user in hyper-parameter tuning using `cf_validation` and to test model performance using `cf_test`. For each user, given the ground truth labels $D_i = \{d_1, ..., d_N\}$ and the recommendations $R_i = \{r_1, ..., r_Q\}$ sorted in descending order, the relevance score is calculated as:

$$rel_D(r_i) = \begin{cases} 1 & \text{if } r_i \in D \\ 0 & \text{otherwise} \end{cases}$$

For M users, MAP is defined as:

$$MAP = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{|D_i|} \sum_{j=1}^{Q} \frac{rel_{D_i}(r_j)}{j}$$

where $Q = 500$ for the top 500 recommended items for each user.

B. Baseline Model

We implement the baseline ALS model using `data_final` and perform hyper-parameter tuning for model selection. Note that all the input datasets are transformed by the track indexer fitted from `metadata`. Results are shown in **Table II**. The best model has rank = 50, λ = 0.05, and α = 1. We fit this model on `cf_test` and obtain MAP = 0.0481 (**Table VI**). The MAP score for `data_final` is similar to that for `cf_test`, indicating that our model does not overfit the training data and generalizes well to the test data.

IV. Extensions

A. Alternative Model Formulations

1) Log Compression
To account for records with count = 1, we add a pseudo-count to each count value before applying a log transformation. We perform hyper-parameter tuning on the log-transformed data and evaluate each model using `cf_validation`. Results are shown in **Table III**. We fit the best model on `cf_test` and obtain a higher MAP than that obtained from fitting the baseline model on `cf_test` (**Table VI**). We conclude that log compression improves model performance.

Table II
Baseline Model: Hyper-Parameter Tuning Sorted by Decreasing MAP

Rank	λ	α	MAP
50	0.05	1	0.04815
20	0.05	1	0.03914
50	0.05	15	0.03892
10	0.05	1	0.03171
20	0.05	15	0.02999
10	0.05	15	0.02334

Table III
Log Compression Model: Hyper-Parameter Tuning Sorted by Decreasing MAP

Rank	λ	α	MAP
50	0.05	1	0.05074
50	0.05	15	0.04591
20	0.05	1	0.04279
20	0.05	15	0.03827
10	0.05	1	0.03599
10	0.05	15	0.02928

2) Dropping Low Counts
We attempt to drop records with low count values in the range [1,3]. **Table IV** shows the number of records that remain in `data_final` after dropping records with count values in the specified range. Almost half of the records have count = 1. Due to the significantly long run time for calculating the MAP metric, we only filter records with count > 1 to train the model. Hyper-parameter tuning results are shown in **Table V**. We fit the best model on `cf_test` and obtain a lower MAP than that obtained from fitting the baseline model to `cf_test` (**Table VI**). We conclude that model performance does not improve when records with count = 1 are dropped from `data_final`.

Table IV
Dropping Low Counts: Records

Count	# Records
1	11,431,547
2	7,415,071
3	5,611,102

Table V
Dropping Low Counts: Hyper-Parameter Tuning Sorted by Decreasing MAP

Rank	λ	α	MAP
50	0.05	1	0.04030
20	0.05	1	0.03392
10	0.05	1	0.02899
50	0.05	15	0.02770
20	0.05	15	0.02050
10	0.05	15	0.01917

Table VI
Evaluation Using Best Trained Model on `cf_test`

Model	Rank	λ	α	MAP
Baseline	50	0.05	1	0.04812
Log Compression	50	0.05	1	0.05081
Dropping Low Counts	50	0.05	1	0.04084

B. Exploration

For our second extension, we develop a set of visualizations to depict the clustering of the tracks and users in the learned factor space using additional features from the metadata. We use the T-SNE algorithm to produce these visualizations. [2] We vary the perplexity parameter to take values in the range $\{5, \ldots, 50\}$. After analyzing all the supplementary data, the most interesting features are chosen from `artist_hotttness`, `artist_familiarity` in `metadata` and `tag` in `tags`. Our first approach finds meaningful clusters in the learned user factors. We take the best `rank=50` model, apply the t-SNE algorithm to map our data to two dimensions, and then color the data points based on values from the metadata. **Figure 1** is a plot of the low-dimensional representation of our user latent factors. Red indicates a higher average `artist_hotttness` and blue indicates a lower average `artist_hotttness`. We observe some interesting structure: users who prefer "hot" tracks

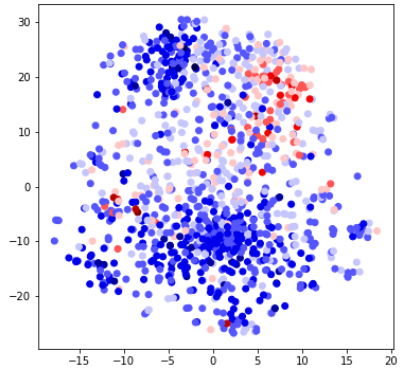

Figure 1. Low dimensional representation of learned user factors colored by mean artist_hotness.

appear to cluster in the top right portion of the plot. Users with a lower mean `artist_hotttness` cluster below the y-axis.

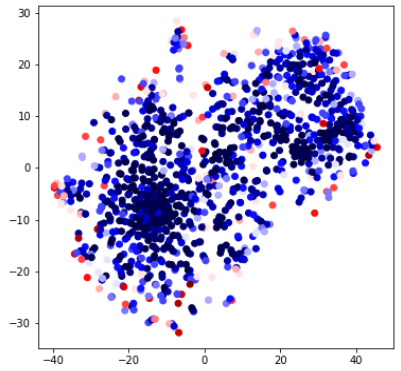

Figure 2. Low dimensional representation of learned user factors colored by sum of counts.

Similarly, we color users by the sum of counts across tracks `data_final` in **Figure 2**. We see that "low count" users organize into one or two main clusters while "high count" users are at the periphery of these clusters.

We attempt the same process to visualize a

low-dimensional representation of items with colors corresponding to `artist_familiarity` and `artist_hottttness`.

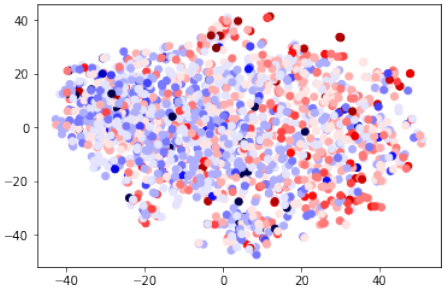

Figure 3. Low-dimensional representation of learned item factors colored by artist_hotness.

The most interesting visualization of the tracks is at `perplexity = 25` for the `artist_hotness` feature as depicted in **Figure 3**. Each data point is a track. Each track has an associated artist "hotness" score. The cone shape seems to contain tracks with lower `artist_hotttness` in the head. Tracks with higher `artist_hotttness` seem more prevalent in the tail.

V. Discussion

Our recommender system model is limited to the constraint of running jobs on the HPC cluster. It can take up to two hours to evaluate each model on `cf_validation` in hyper-parameter during due to the MAP calculation. If not for this constraint, we would conduct a more extensive hyper-parameter grid search. Note that MAP is a better metric for evaluating an ALS model with implicit user feedback, as opposed to RMSE, which is more suitable for evaluating an ALS model with explicit user feedback.

VI. Contributions

- Ningyuan: baseline model, hyper-parameter tuning, extension: alternative model formulations, additional file exploration
- Zacharie: hyper-parameter tuning, extension: exploration
- Amy: preparation of training data, hyper-parameter tuning, extension: alternative model formulations

References

[1] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". In: *2008 Eighth IEEE International Conference on Data Mining* (2008). DOI: 10.1109/ICDM.2008.22.

[2] L.J.P. van der Maaten and G.E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: *Journal of Machine Learning Research* (2008), pp. 2579–2605. DOI: 10.1109/ICDM.2008.22.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *IEEE Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/MC.2009.263.