DSGA 1004 Final Project: Music Recommendation Using Implicit Feedback

Aaron Kreiner ak6817, Shaan Khosla sk8520, Max Needle mn1931, Ted Yap tyy231

1 Introduction

Recommender systems are based on the simple concept of predicting the utility of an item to a user. The system, therefore, recommends the items of highest predicted utility to help the user find the best item match. Our project uses Alternating Least Squares to suggest songs based on a latent factor model. This model breaks down the user and item interaction matrix into a product of two matrices: U, V. U represents the user factor matrix and V represents the item factor matrix. These two matrices contain an embedding for users and items within vectors.

We used the Million Song Dataset [1] and user interaction data from the Million Song Dataset Challenge. Within the given dataset, the training sample contains roughly one million users and an incomplete history of eleven thousand users. The test and validation sets contain the remainder of the histories for the eleven thousand users. Each row of the data represents the interaction of the item and a user. The User ID column and Track ID column match to the user string and track string respectively. The count variable measures the number of times a user listened to a specific track. This count variable is considered implicit feedback as it measures a relative confidence threshold for a given user. In other words, higher counts can be thought to relate to higher preference.

2 Model Implementation

The model implementation process begins with pre-processing the data. In practice, we use the parallel computing ability of PySpark to build and process our model. Because the User ID column and Track ID column are given to us as strings, we use string indexing to convert them to integers. After pre-processing the data, we use Alternating Least Squares (ALS) to make the item predictions. [2] This algorithm is an iteration-based approach that supplies the user and item embedding matrices. Given the notation in the introduction, we fix U, V at the first iteration. In subsequent iterations, one of the matrices becomes fixed and the other we calculate using least squares estimation. In the next step, the current factor matrix is fixed to optimize the other matrix. This iterative procedure is then repeated until convergence. This convergence is measured directly from the objective function of ALS. Before giving the objective function, it is helpful to define some variables. c_{ij} is linearly correlated with implicit feedback values by constant alpha, where alpha is the scaling parameter on the count data. Define R_{ij} to be the count of User_i and Track_j. Finally, let λ be the regularization hyperparameter. The objective function, therefore, can be constructed as follows:

$$\min_{U,V} \sum_{(i,j)} c_{ij}(p_{ij} - \langle U_i, V_i \rangle)^2 + \lambda(\sum_i ||U_i||^2 + \sum_j ||V_j||^2)$$

$$\text{where } p_{ij} = \begin{cases} 1, & R_{ij} > 0 \\ 0, & R_{ij} = 0 \end{cases} \text{ and } c_{ij} = 1 + \alpha R_{ij}$$

Using the ALS procedure, we fit a model on the training set and make predictions on the validation set. For testing purposes, we ensured that our model ran correctly on $1\%, 5\%,$ and 25% before running it

on the full dataset. Within the model, there are two hyperparamaters that we controlled for. First, we tuned for rank, or the dimension of the latent factors. We tested ranks of $[5, 10, 15]$. We also calibrated a regularization hyperparameter to control for overfitting. We used values of $[0.1, 1, 10]$ for λ. To evaluate each hyperparameter combination, we used the following metrics on the top 500 predicted items: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG), and Precision at 500. The mathematical and conceptual definition of these will be described in the next section. Due to its ubiquitous use in ALS modeling, we selected MAP as the key metric to pick the final model from the test hyperparameter tuning runs. In other words, we picked the model with the highest MAP on the validation set, and ran this configuration on the test set. The results of these configurations will be reported in the results section.

3 Evaluation Metrics

As our primary metric, we evaluated the MAP. Conceptually, MAP measures the quantity of the recommended documents that are in the set of true relevant documents. In this calculation, the order of the recommended documents matters and there are higher penalties associated with highly relevant documents. We can also define this mathematically. Let each user be represented as follows: $U = [u_0, u_1, ..., u_{M-1}]$, each ground truth document as $[d_0, d_1, .., d_{N-1}]$, and a list of Q ordered documents sorted by relevance: $R_i = [r_0, r_1, ..., r_{Q-1}]$. The mean average precision is:

$$\text{MAP} = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{|D_i|}\sum_{j=0}^{Q-1}\frac{\text{rel}_{Di}(R_i(j))}{j+1} \text{ where: } \text{Rel}_D(r) = \begin{cases} 1, & r \in D \\ 0, & \text{else} \end{cases}$$

As a secondary metric, we evaluate precision at k, where $k = 500$. This measures how many of the predicted items are relevant, regardless of the ordering of the recommended items. This means we measure model performance based on the top 500 recommended items. This can be expressed mathematically below:

$$\text{Precision at k} = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{k}\sum_{j=0}^{\min(|D|,k)-1}\text{rel}_{D_i}(R_i(j))$$

Our final metric is Normalized Discounted Cumulative Gain at k where $k = 500$. To calculate this metric, we take the first 500 documents, and measure how many of the recommended documents are in the set of true relevant documents across all users. Differing from precision at k, NDCG at k takes into account the ordering of the recommended documents. The formula for this metric is expressed below:

$$\text{NDCG}(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{\text{IDCG}(D_i,k)}\sum_{j=0}^{n-1}\frac{\text{rel}_{D_i}(R_I(j))}{\log(j+2)}$$

$$\text{Where: } n = \min(\max(Q_i, N_i), k) \text{ and: } \text{IDCG}(D,k) = \sum_{j=0}^{\min(|D,k|)-1}$$

Finally, we note that by the construction of the way the metrics are measured, higher metric values correspond to better scores on the training, test, and validation sets. [3]

4 Results

We now present the results based on the setup of the model implementation and evaluation metric sections. We first accumulate the results on the validation set in the below table:

Rank	Reg	MAP	NDCG	Precision
5	0.1	0.02740	0.12085	0.00652
5	1	0.01993	0.10633	0.00629
5	10	0.00003	0.00084	0.00008
10	0.1	0.03297	0.13979	0.00743
10	1	0.02603	0.12677	0.00722
10	10	0.00002	0.00085	0.00007
15	0.1	**0.03648**	0.15117	0.00801
15	1	0.02924	0.13759	0.00777
15	10	0.00040	0.00084	0.00007

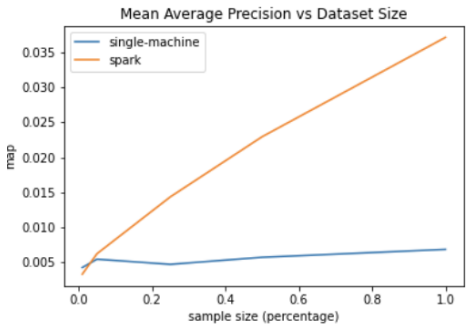
We see that higher ranks tend to have higher metric scores while higher regularization parameters correspond to lower scores. The best configuration occurs at rank 15 and regularization parameter 0.1 We finally run this configuration on the test set and get a MAP of 0.037, precision at 500 of 0.008, and an NDCG at 500 of 0.152. Since these metrics numbers are close to the validation set best numbers, we say this model generalizes fairly well.

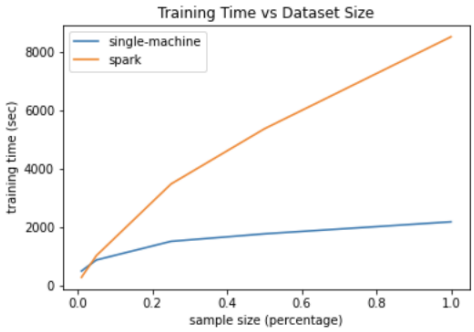
5 Extension 1: Comparison to Single-Machine Implementations

In this section, we introduce LensKit for implementing a recommendation system on a single machine as opposed to Spark's parallel ALS model. LensKit is a Python package for experimenting with and studying recommender systems. It provides support for training, running, and evaluating recommendation systems, as well as functions for both explicit and implicit feedback models.

In this project, we compared Spark's parallel ALS model to LensKit single-machine implementation. We measured both efficiency (model fitting time as a function of dataset size) and the resulting accuracy. A summary of the implementation steps is as follows. We first fit LensKit's ALS model on the training dataset using the hyperparameters that achieved the best mean average precision in Section 4. Next for each distinct user in the test dataset, get 500 recommendations from trained ALS model. We finally compute mean average precision on test dataset.

We ran both Spark's ALS and LensKit ALS on 1%, 5%, 25%, 50%, and 100% of the training dataset with the same hyperparameters. We show two plots comparing the performance of a single-machine implementation and Spark. The first plot shows this comparison in terms of MAP and the second in terms of training time. We see that the MAP is substantially better for the spark model, as compared to the single-machine implementation. In terms of efficiency, it was surprising to see that Spark's ALS takes much longer to train as the dataset size increases. This could be caused by a number of factors, such as high concurrency, inefficient queries, and incorrect configurations.





6 Extension 2: Baseline Models

With recommender systems, a popularity-based model should always be referenced as a baseline model to measure the impact of personalization. To do this, we used the Bias model provided by LensKit to compute track and user bias and subtract these biases out from the play counts.

We first used a training set to create a bias model. Several hyperparameters were tested such as the amount of user and item damping. We iterated through the list $[0.25, 0.5, 1, 2, 5, 10, 15, 30, 50, 100, 150]$ and used every combination of item and user damping to test on the validation set. We used Mean Average Precision (MAP) as the metric that we would use to compare the success of our popularity model as compared to ALS. Finally, using this Bias model we were able to obtain two types of predictions: first to predict the top tracks for each user with user and item bias removed, and second to obtain a sum of counts for each track and rank them in descending order (produces the same ranking for all the users).

We tried both implementations, and ultimately, the best recommendation came from calculating the total number of counts for each song. Since this approach is calculating the sum of utilities, the level of damping was not relevant. In other words, we would still obtain the same set of 500 most popular tracks. The table below shows our final results.

Popularity Baseline Model	MAP
Validation Set	0.05099
Test Set	0.05207

We see the popularity baseline model generalizes well due to the validation and test set scores. This model even outperforms the main model in section 4. However, further hyperparameter tuning in the main model may change this result.

7 Conclusion and Contributions

In this paper, we implement several implicit feedback models for music recommendations using the Million Song Dataset. We find that our model generalizes well on the test set and can be extended to a single-machine implementation as well as being used for a baseline model. Aaron worked on the main model, Max helped with the main model, Ted helped with the main model and did extension 1, and Shaan helped with the main model and extension 2. We all helped writing the paper and accumulating the results.

8 Sources

[1] "Million Song Dataset", The Echo Nest and LabROSA, millionsongdataset.com/.

[2] "Collaborative Filtering." ALS Documentation, PySpark, spark.apache.org/docs/2.4.7/ml-collaborative-filtering.html.

[3] "Evaluation Metrics - RDD-Based API." Evaluation Metrics - RDD-Based API - Spark 2.4.7 Documentation, PySpark, spark.apache.org/docs/2.4.7/mllib-evaluation-metrics.html#ranking-systems.