AAI 627 — Big Data Analytics

Final Project

Music Recommendation System

Spring 2025

Syed Ahmad Shah

Project Overview

The project focused on creating a music recommendation system given sparse training data to predict user ratings for unseen data. For each test user, we were to rank 6 candidate tracks, predicting 3 tracks as liked and 3 as disliked.

The main challenge was that the data did not only include tracks, but also exposed:

- Tracks
- Albums
- Artists
- Genres

As such it is not a simple track recommender, but rather a music hierarchy which should further power our recommendations.

Over the course of the project, I had implemented heuristic methods, fallback strategies, hybrid approaches, autoencoder based learning, ALS experiments, and "exposed-score" based ensemble implementations.

Main Project Highlights

The project started with a rule based implementation centered around the signals provided from album, artist, and genres.

Through experimentation it was noted that the strongest insight came from the album-level preference

Genre based methods (weighted implementations) performed weakly, as genre is a very broad and noisy category when compared to the strong relationships found in album or artist relationships

Implementations

Weighted Hierarchical Average

As my first main strategy, I used a weighted score approach based on:

- Album Score
- Artist Score
- Genre mean

I wanted to give more importance to album level preference as it was the most specific signal, then provide supporting evidence with artist and genre signals.

Result:

Score: 0.759

I learned that the hierarchy direction was appropriate, and that the album level information was indeed far more useful than the broad genre preference.

Maximum Genre Strategy

For this strategy, I emphasized the user's strongest genre by using the maximum genre score rather than the mean. I wanted to test if the weaker genre signals were in fact hurting the final score and hiding the strong genre persistence within the tracks.

Result:

Score: 0.704

It seemed genre was too broad to be a main decision signal and seemed to hurt the final score when the weighting was increased. A track sharing one highly rated genre did not mean the user would like that track. This method showed definitely that I overestimated user preference by attempting to rely on genre history.

Bayesian Global Fallback

In my earlier implementations, I had noted a large percentage of the tracks that were recommended were the result of a fall back I implemented for a default score. So to better utilize this noticeable set of information that was relying on a simple mean fallback, I replaced it with a Bayesian-smoothed global item scores.

This would offer the following benefits:

- It reduced the effect of items with very few ratings
- It gave a more informed score than simply relying on the same constant score everywhere

This implementation and work showed the importance and thought I need to implement for the fallback logic, and not solely focusing on the main implementation as not every case will be covered by my main model. A global prior is far better than a determined default.

"Dig Deeper" Album Sibling Logic

I had added an additional rule to check if a user had rated other tracks from the same album even if the hadn't rated that album specifically. This helped in creating a more personalized album score before falling back to the global popularity.

Result:

- Weighted Average + Bayesian + Dig Deeper: `0.774`
- Max Genre + Bayesian + Dig Deeper: `0.708`

Sibling-track logic was a great personalization improvement, it did help, however the larger issue was still the overall sparsity of the available data.

Adaptive Weight Normalization

Instead of just combining the album, artist, and genre score together, I only used signals for the items that the user had directly rated, then normalized the weights. So rather than having a fallback value for each category if the user is unable to provide one, we just remove it from the calculation.

- This prevented strong real signals from being diluted by the weak fallback values
- It gave preference to real data
- Scoring was more deterministic and personalized

So say we have album information, but are missing genre and artist information

$$\begin{aligned} \text{Score} &= 0.5 * \text{album\_score} \\ \text{Total weight} &= 0.5 \\ \text{Normalized final score} &= (0.5 * \text{album\_score})/0.5 = \text{album\_score} \end{aligned}$$

Score: 0.792

Autoencoder Fallback

I believed utilizing a Neural Network as cheating in a way, as we were initially implementing heuristic models and techniques. However, I had learned about Autoencoders in another class "Deep Learning" and wanted to further better my understanding of the model, and how well it can fit the data, so instead I used it as the fallback instead of the main implementation.

I wanted to test just how much of the data that was being left to the fallback made a difference in the results. Instead of replacing the whole recommender with deep learning, I used an autoencoder only for the weakest part, the final fallback stage.

Result:

Score: 0.849

The autoencoder improved personalization in sparse cases quite significantly, and provided a very reasonable score, given I had done little feature engineering and model parameter optimization.

ALS and Multi-ALS

I had also explored collaboraiteve filtering with ALS, with separate models for:

- Track
- Album
- Artist
- Genre

This experiment helped me understand the latent factor recommendation, however it did not perform as well as the hybrid hierarchy based models

- raw multi-ALS blend: `0.653`
- rank blend: `0.646`
- normalized blend: `0.688`

Spark ML

In addition to the previous approaches, I had also explored supervised learning methods utilizing Spark ML. The goal was to determine if classification models could be trained efficiently to predict user preference.

The Spark ML experiment involved several different models:

- Decision Tree
- Logistic Regression
- Gradient Boosted Trees
- Random Forest

Scores:

- Decision tree = 0.892
- Logistic Regression = 0.910
- Gradient Boosted Trees: 0.915
- Random Forest = 0.918

9. Ensembling Logic

Given I have made 45 entries into Kaggle and have received scores for each submission. I was able to combine the multiple submissions into a weighted ensemble. By creating a weighted ensemble, we would be able to deterministically get an idea of combining model results to achieve a better prediction.

The ensemble followed a linear-combination approach:

- Gather 20 records of submissions
- Normalize predictions to $\{-1, 1\}$
- Treat each submission as a solution vector
- Estimate weights from known scores
- Combine into a final ranking score

Score: 0.921

This implementation performed the best as it was an informed decision from the previous 20 different implementations and variations that were tested. The Ensemble method was useful when several models were capturing different strengths, reducing the overreliance on any one model and improving stability. This allows for an informed combination of results to provide a stronger output.

Results Table

Item Tested	Result
Ensembling Logic	0.921
Random Forest	0.918
Gradient Boosted Trees	0.915
Logistic Regression	0.910
Decision tree	0.892
Autoencoder Fallback	0.849
Adaptive Weight Normalization	0.792
Weighted Average + Bayesian + Dig Deeper	0.774
Weighted Hierarchical Average	0.759
Max Genre + Bayesian + Dig Deeper	0.708
Maximum Genre Strategy	0.704
normalized blend	0.688
raw multi-ALS blend	0.653
rank blend	0.646

Closing Reflection

The most important lesson I learned throughout this project is that a good recommender system does not come from using an overly complex model, instead its about understanding the structure and relations of the data, identifying bottlenecks, and combining the right methods for different parts of the problem.

Thank You for your guidance and support throughout this class and project. This class gave me a better understanding of recommendation systems, and has genuinely aided me in better understanding how to approach real world data problems through experimentation and analysis. One of the notable things I have learned in this class is the concept of Matrix Factorization, which I have already implemented in another class and am excited to utilize it for a real-world problem. Hope you have a great summer break!