STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY

# EE627 Final Report: Yahoo Music Recommendation

Source Code: [https://github.com/JMavorah/EE627-Final-Project](https://github.com/JMavorah/EE627-Final-Project)

Jack Mavorah, Dylan Zenner

December 16, 2021

## 1. Introduction

Recommendation engines have become a hot topic, especially in the past couple of years due to the pandemic having a huge effect on society. With an increasing number of individuals shopping online it has become apparent that businesses with some sort of recommendation system in place generate the most revenue. The Yahoo Music Recommendation project aims to build out a recommendation engine that is able to predict three tracks that a user will like based on a wide range of features. While this project does not build out a recommendation system for an online business, the algorithms utilized are ones which can be used for any type of recommendation engine. So, the overall process can be replicated to suit the needs of the individual business.

The project was accomplished using the Kaggle data science platform. The predictions produced by each method were organized into a common file format and submitted to Kaggle. Kaggle then compared the predictions to the ground truth data and provided an accuracy score, rating all of the teams participating in the project.

## 2. Methods

### 2.1 Algorithm Implementations

Testing data included a list of users, each paired with six different tracks. The goal was to predict three tracks that the user will enjoy. Seven different algorithms were used in the development of the recommendation engine. The programming was done in Python, mostly using Python notebooks in Google Colab. PySpark (python library to make use of Apache Spark clustering capabilities) was used for clusterization. A brief introduction to the algorithms and their implementations are detailed below:

#### 2.1.1 Hierarchy Search:

This was the simplest algorithm used. It is based on training data which includes data from users' previous ratings. The hierarchy includes track, album, artist, and finally genre. For each user and track in the test data, the training data is searched to determine if that user has rated that track previously. Then a search is conducted to determine if that user has rated the album to which that song belongs, then for a rating of the artist to which the song belongs, and then its genre. These ratings are then combined to give a score to each of every user's songs. The tracks are then sorted and thresholded, and the top three songs are recommended. This was accomplished using basic Python programming.

As mentioned, this algorithm is simple to implement but relies on previously existing user data (user-driven prediction). This means that if there is a lack of information for a user (or some inaccurate information), this method will not result in good predictions.

**2.1.2 Alternating Least Squares Matrix Factorization (ALS):**

This algorithm is one of the simpler and most well known when it comes to recommendation engines. Matrix factorization algorithms are some of the most successful examples of latent factor models, which work by attempting to explain and predict ratings by characterizing both users and the items they are rating using numerous factors. If an item and user share a highly correlated factor, a recommendation may be made. Matrix factorization attempts to estimate the ratings matrix as the product of two lower-rank matrices: $\mathbf{q_i}$ for the items, and $\mathbf{p_U}$ for the users. The approximate rating of any item by any user is then given as $\hat{\mathbf{r}}_{\mathbf{ui}} = \mathbf{q_i}^{\mathbf{T}}\mathbf{p_U}$. Both the **p** and **q** matrices are unknown. ALS approaches the problem by alternating fixing one of the matrices, once each matrix is completely fixed, the other matrix is recomputed by solving a least-squares problem. The benefit is ALS is the potential for high parallelization. The elements of the matrices can be calculated independently of each other, and thus these calculations can be spread across large amounts of compute power. In this implementation, PySpark was used for parallelization, and the ALS model built into PySpark was used. The ALS input parameters (max iterations, rank, regularization parameter) were varied to optimize the accuracy of the prediction.

**2.1.3 Logistic Regression:**

Another well known algorithm which was implemented using the built in LogisticRegression model in PySpark. Logistic regression is best for when the target is categorical. In this case the goal was to predict if a user likes a given track or not. If the user likes the track then they are assigned a ‘1’ and ‘0’ otherwise. This is an example of a categorical target.

**2.1.4 Decision Tree Classification:**

A supervised machine learning algorithm which splits the data continuously according to a certain parameter (again implemented using the model built into PySpark). There are two types of decision tree classifiers, classification trees and regression trees. In a classification tree the outcome is categorical in nature. For this project if a user liked a track they were assigned a ‘1’ and ‘0’ otherwise. This is an example of a categorical output.

**2.1.5 Gradient Boosted Tree Classification:**

Another supervised machine learning algorithm (built into PySpark) and arguably one of the best. Gradient boosted trees combine various other weak predictors in order to create an additive predictive model, similar to forming an ensemble of numerous algorithms together.

**2.1.6 Random Forest Classification:**

This algorithm is created by combining many different individual decision trees in an ensemble. Each decision tree in a random forest will return a prediction and the prediction with the most votes will be the prediction. For example, here the goal is to predict if a user will like a track (1) or not (0). Say there are 10 decision trees in a given random forest. If seven of those trees predict that the user will like the track and three of those trees predict that the user will not like the track then the prediction on whether the user likes the track or not will be a 1 (likes the track) since a majority of the trees predicted that the user will like the track.

### 2.1.7 Ensemble Modeling:

Ensemble modeling is used to exploit multiple different models for the same task. After obtaining the predictions from all of the aforementioned models, an ensemble was formed to generate a prediction based on all of them together. Each prediction set can be formatted as a vector, which can then be stacked to form a matrix. Now, a vector of weights for each solution vector must be formed in order to determine how heavily each model should be considered in the final ensemble. The weights can be determined through a least-squares problem. Before solving for the weights, the values of the prediction vectors were changed from 1 (for a recommendation) and 0 (for no recommendation) to 1 and -1. This allows a simple multiplication with the ground truth to determine if the prediction was correct (if the prediction and truth match, the result will be 1, and if they do not match, the result will be -1). Here, there is no access to the ground truth itself, only to the scores assigned to each model by Kaggle. As mentioned, the weights can be found with a least-squares problem, $a_{LS} = (S^TS)^{-1}S^Tx$, where S is the matrix of solution vectors, and $S^Tx$ is the accuracy of each model needed to calculate the weights. $S^Tx$ can be calculated to be the vector of $[N(2P_i-1)]$, where $P_i$ is the correct rate score provided by Kaggle. Finally, the solution vector obtained from the ensemble is $S_{ensemble} = \sum a_i s_i = S{*}a_{LS} = S(S^TS)^{-1}S^Tx$. The values in this vector are then sorted and thresholded to ones and zeros (for recommending or not recommending tracks) to form the final prediction.

## 3. Results (Best scores)

Below are the scores provided by Kaggle for each model used. An interesting point to note is that multiple versions of the hierarchy search were done. The first only took tracks, albums, and artists into account, and the second attempt used genres as well. The model that included the genres actually performed worse, and this is probably due to the fact that the ratings of each category were rated equally (a rating for a track, album, or artist would be a better predictor than a rating for a genre). Varying the weights of each category would likely generate a much better predictor. Also note that the ensemble predictor had by far the highest rated score. It was very interesting to see models that did not necessarily perform well on their own increase the accuracy of the final predictors when used in an ensemble.

| Algorithm | Score |
|---|---|
| Alternating Least Squares Regression (ALS) | 0.73128 |
| Logistic Regression | 0.84465 |
| Decision Tree Classification | 0.82301 |
| Gradient Boosted Tree Classification | 0.84420 |
| Random Forest Classification | 0.84498 |
| Hierarchy Search | 0.84150 |
| Ensemble | 0.87483 |

## 4. Conclusions

This project explored the realm of recommendation engines by utilizing many different algorithms to predict if a user would like a track or not with the Yahoo music dataset. While a respectable final prediction score (and position on the team leaderboard) was achieved, there are definitely areas for improvement. All of the code used can be found in the GitHub repository linked on the cover page. Overall, the project was an excellent way to obtain practical experience with recommendation engine algorithms as a whole.