Mechanical & Industrial Engineering
UNIVERSITY OF TORONTO

Amazon Music Rating Course Project Report

MIE 1624 Introduction to Data Science and Analytics

Prepared for: Prof. Oleksandr Romanko

Group 4: Kangni Wang, Wenmo Qiu, Jing Ma, Zhanhao Ye,

Ning Du, Xinyuan Zhu, Yonghao Li

1. Introduction

There are three main objectives that we want to achieve when completing this course project. As a start, we want to create a model (machine learning or neural networks) to accurately predict Amazon music ratings based on user reviews. To choose the best model, we use the mean squared error (MSE) as the performance metric. After modeling, we will implement three types of recommender systems (popularity model, item-based collaborative filtering, and matrix factorization) using our data. Additionally, we will introduce "Spotifinder", an app that can help people connect with other people who have similar tastes in music. Utilizing NLP analysis to process users' reviews regarding musicians, melodies, or lyrics, "Spotifinder" will be able to combine social networking with music entertainment.

2. Data Cleaning & Preprocessing

To predict ratings related to music reviews on Amazon, we first need to remove unnecessary content and highlight keywords in the "reviewText" and "summary" columns through several steps. To prepare the data for NLP modeling, we discard meaningless content such as punctuations, URLs, HTML tags, and line breaks from the text features using regular expressions. After tokenization, we use the NLTK library to remove stop words that do not add value to the reviews. Additionally, we use lemmatization to show words in their root forms (based on the text) to avoid repetitions. We then put tokenized words into one string for vectorization. In order to avoid data exposure in the validation set, we split "train.csv" into train set and validation set before vectorizing the text features.

3. Exploratory Data Analysis

We can see from both *Figure 2* and *Figure 3* that most people tend to give review scores that are close to the full mark for music ratings. The percentages of ratings at 1 and 2 were similarly low. When comparing across categories, "Jazz" and "Classical" receives more 5-points than the other genres, while "Alternative rock" received more 1-points than the others. Although "Pop" and "Alternative rock" received the greatest number of reviews, their average feedback was not as good as the reviews for "Jazz" and "Classical". For "Jazz" and "Classical", users tend to leave good ratings but they rarely write reviews. *Figure 4* shows the "reviewText" word cloud, which displays the popular words from our text data. We can see that the words used in music reviews are mostly positive, such as wonderful, great, and love.

4. Model Implementation

A comparison of model performances is shown in the table in Appendix.

4.1 Linear Regression

The Ridge regression model achieved a better validation MSE (0.654) compared to the Lasso regression. MSEs for the training set and the validation set indicate that there is no significant overfitting. Based on *Figure 5*, the model makes a good prediction on the rating 3 group, but for the other rating classes, it either overestimates or underestimates the true values. Overall, linear regression has low computational costs and is relatively easy to implement, while overfitting can be simply avoided by regularizations.

4.2 Ordinal Logistic Regression

For this model, we tuned penalty (Ridge and Lasso) and C (representing the inverse of regularization strength; a smaller C specifies stronger regularization, making it harder for the model to overfit). The ordinal logistic regression model was a straightforward approach, but the model performed worse than linear regression and it came with a longer training time.

4.3 Naïve Bayes

The multinomial naïve Bayes model is a simple algorithm that is good at text classification. It has been broadly used in text mining. This model assumes that the features are independent. However, there might be a positive relationship between “summary” and “reviewText” such that the basic assumption of the multinomial naïve Bayes model became unsatisfied, leading to bad model performance.

4.4 Support Vector Machine

The goal of SVR is to locate an optimal decision boundary at a distance from the original hyperplane such that the data points closest to the hyperplane or the support vectors are within the boundary line. The SVM model is effective in high dimensional spaces, robust to outliers, and less likely to overfit. However, it did not perform well in our case since it was not suitable for overlapping target classes in a large dataset.

4.5 Extra-tree Regressor

The extra-tree regressor is similar to random forest in that it fits several randomized decision trees on various sub-samples of the data, then uses averaging to improve the predictive accuracy while controlling overfitting. However, the model did not perform ideally and took too long to tune. This could be due to the complexity of the model and the large size of input features.

4.6 Random Forest

The random forest regressor consists of an ensemble of uncorrelated individual decision trees. Each individual tree only uses a subset of the entire dataset for training, and the majority vote becomes the model prediction. One of the advantages for RF is the use of bootstrap aggregation which allows each individual tree to randomly sample from the dataset with replacement, such that small changes in the data will not impact overall performance significantly. However, the performance was not ideal given its long tuning time and high computational costs.

4.7 XGBoost Regressor

XGBoost is the most advanced version of boosting ensemble tree models, which has an option to penalize complex models using both L1 and L2 regularizations. The best model exhibited some degrees of overfitting which could be due to the model complexity and the large size of input features. To account for this, multiple parameters were tuned (max depth, gamma, lambda, alpha, and early stopping rounds). A detailed list of hyperparameters is displayed in Appendix.

4.8 Light Gradient Boost Machine

In the light gradient boost machine model, we used the “number of boosts around” to control the training iterations and to prevent underfitting. We also employed “window of early stopping” to avoid overfitting. In this model, since we used many strategies in the hyperparameter tuning process to prevent overfitting, there was no overfitting issue. The best model resulted in a good accuracy with the given features, so LightGBM worked well for this case. Predictions are visualized in *Figure 6*.

4.9 Multilayer Perceptron (best performance so far)
Multilayer perceptron model consists of layers of nodes, each node is a neuron that uses a nonlinear activation function. With a backpropagation approach for model training, it can identify a dataset that is not linearly separable. In *Figure 7*, we can see that the model overestimates the rating 5 class but does a relatively great job predicting the other classes. It also exhibits some degrees of overfitting. The MLPRegressor performed better than XGB because multilayer perceptron networks could better fit nonlinear data with higher accuracy than tree-based models. A detailed list of hyperparameters is displayed in Appendix.

5. Recommender System
Recommender system is a useful information filtering tool that helps users who are searching for information efficiently from massive amounts of data. In this project, we built three different recommending systems using relatively simple algorithms.

5.1 Popularity Model
The popularity model simply recommends items based on the most popular choices. The model results are shown in *Figure 8*. Consequently, the model recommended the same items with the highest overall ratings to all reviewers.

5.2 Collaborative Filtering Model (item-based)
The recommender created by the collaborative filtering model is based on item-item similarities, which means that the users/reviewers will probably be recommended the items that are similar to their already-highly-rated items. *Figure 9* shows the resulting recommendations to the reviewers in the test set. Different recommendations for each reviewer suggest that the model produces a different set of personalized recommendations for different users.

5.3 Matrix Factorization
Using matrix factorization, we can find latent features that determine how a reviewer rates an item by decomposing the R matrix into constituent parts such that the product of these parts generates the original matrix. Our prediction is $R_{pred} = P \times Q^T$.
We applied this method only to a subset of our dataset since most of the ratings in the R matrix were missing, meaning that we had more “real” training data than test data. This resulted in the inaccuracy of our predictions (validation MSE was 1.2). Among the predictions, only a small portion was actually required to be predicted and we had to discard the redundant predicted ratings, which was a waste of computational resources and led to inefficiency.

6. Application: Spotifinder
Our application “Spotifinder” combines the functionalities of music recommendation and social networking. When users leave comments in this app, the texts will be used as features to fit into our machine learning models to predict the ratings. According to the ratings, our model will then recommend friends with similar music preferences. With the popularity model, we can push music to our users based on its popularity. With the collaborative filtering model, we recommend music based on their similarities with already-highly-rated items for each user. Conceptual designs for the Spotifinder app are shown in the appendix.

7. Results and Conclusions

To visualize the dataset, *Figure 1* shows that the most popular music category is "Pop" with 50% of the total reviews. "Alternative Rock" is the second popular with 28.5%, while "Dance & Electronic" has only 6.3% which is the least popular among all. We conclude that people prefer more catchy and rhythmic songs.

For feature selection, we initially tried to include all features ("category", "price", "summary", and "reviewText") into the model, but the resulting accuracies were low. We then changed our strategy to fit models with just the text columns (i.e., "summary" and "reviewText"). However, summary and review text could be correlated since the same sentiments or keywords written in the review are likely to show up in the summary. This might be problematic for some models where feature co-independencies are assumed. Therefore, in some of the models, we only used "reviewText" as the input feature or we joined the texts in "summary" and "reviewText".

We implemented word frequency (WF) and TF-IDF separately for all models. For the Multilayer perceptron model, we also tried using the word2vec approach, but the vectorizer's output was not compatible with the model. As a result, all models performed better with the TF-IDF approach. This is due to the fact that WF only generates vectors containing the count of occurrences for each word, while TFIDF takes into account the potential importance or the lack of it for each word when vectorizing the texts. The multilayer perceptron model achieved the lowest MSE.

After comparing the MSE results of the models mentioned above in section 4, the models that performed best include the Light Gradient Boost machine model (0.5831), the XGBoost model (0.6187), and the MLP model (0.4951). These three models gave good results and accuracies because they were suitable for our data, which was sparse with an imbalanced class distribution. For example, class 5 had more than 100,000 samples while class 1 only had less than 20,000 samples. In the Light GBM model, instead of using level-wise splitting methods, leaf-wise splitting was used to grow trees so that the loss could be minimized more quickly with the same number of leaves. MLPRegressor performed well because multilayer perceptron networks could better fit nonlinear data with higher accuracy than tree-based models. As a feedforward artificial neural network, MLP generalized better as it could outperform other models in extracting patterns and detecting trends from complex input data. With the help of adaptive learning rate and activation functions, MLP converged to the global minimum more quickly and consistently. The multilayer architecture also helped the model process large datasets.

The other models failed to predict accurately mainly because they are not suitable for our dataset. The SVM model failed because it did work well with large datasets with overlapping target classes. For random-tree, it is plausible that due to the sparsity and not-axis-aligned properties of our dataset, for some nodes, the bootstrapped sample and the random subset of features would collaborate to produce an invariant feature space. There was no productive split to be had, so it was unlikely that the children of this node would be at all helpful. Since random-tree gave poor results and extra-tree took a step further by using a random threshold at each split due to the random nature of the threshold at each split, it would produce an even poorer result than the random-tree model. For naïve Bayes, all features were required to be independent, but "reviewText" and "summary" were correlated in both content and sentiment wise. Hence, the basic requirement of implementing Naive Bayes was not met and the prediction failed.

Appendix

Table of Model Performances and Hyperparameters

Models	Features Used	Best Hyperparameters	Max_Feature from TF-IDF	Validation MSE
Linear Regression	joined “reviewText” and “summary”	Lasso Alpha = 0.001, Ridge Alpha = 5	2000	0.654
Ordinal Logistic Regression	joined “reviewText” and “summary”	C = 5, regularizer = ridge	2000	0.701
Naive Bayes	“reviewText” and “summary”	Laplace smoothing factor = 0.01	1600	0.960
SVR	“reviewText”	C = 1	20000	0.656
Extra-tree Regressor	“reviewText” and “summary”	criterion = gini, max_depth = 17, max_feature = “sqrt”, n_estimator = 30	1200	1.316
Random Forest	“reviewText”	max_depth = 50, n_estimator = 150, min_samples_split = 20	15000	0.732
XGBoost Regressor	“reviewText”	(See below)	15000	0.618
Light GBM	“reviewText”	num_boost_round = 500, early_stopping_rounds = 5	8000	0.583
Multilayer Perceptron (MLP)	joined “reviewText” and “summary”	(See below)	15000	0.495

XGBRegressor hyperparameters tuned:

- Learning rate: the step size shrinkage used to prevent overfitting is set to 0.3 because this leads to the more optimal performance of the model.
- N estimator: the number of trees to build is set to 900 due to the complexity of the data.
- Max depth: this is a positive number representing the maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. Since the training feature is complex and high-dimensional, max depth is set to 50. Although this leads to some overfitting, a smaller number reduces the model’s ability to learn properly from the training data, while a larger number leads to more overfitting. Both would generate worse MSE performance.
- Min child weight: this is a positive number representing the minimum sum of instance weight (hessian) needed in a child node. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. After trying a range of numbers, this is set to 100 for optimal model performance.
- Gamma: this is a positive number representing the minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be. This is set to 0.5 because a smaller number easily leads to overfitting while a large number makes the model performance worse.
- Reg lambda: L2 regularization term on weights. Increasing this value leads to a more conservative model. This is set to 50 so that the model focuses on the most relevant aspects of the training data and does not overfit too much.

- Objective: this is set to “reg: squarederror” to make sure that the model is computing regression with squared loss.
- Colsample_bytree: the subsample ratio of columns for each tree is set to default value 1 for best validation performance.
- Colsample_bylevel: the subsample ratio of columns for each level is set to default value 1 for best validation performance.
- Subsample: the ratio of the training instances randomly sampled by the model before growing trees. This is set to the default value 1 because we are trying to minimize validation error, and although a smaller ratio can prevent overfitting, the validation performance would be worse.
- Seed: this random number seed is set to make sure that the model has a relatively consistent output.
- N_jobs: this is set to -1 so that the model can run on all processors for a faster speed.
- Early stopping rounds: this is set to 5 so that if the validation error has not improved in 5 boosting rounds, the model will stop training.

MLPRegressor hyperparameters tuned:

- Activation: this is set to “relu” (the rectified linear unit function) for optimal performance. Using “tanh” or “logistic” (sigmoid) function leads to worse performance.
- Alpha: the L2 penalty (regularization term) parameter is set to 0.5 because the default 0.0001 value does not provide a strong enough regularization for the model overfitting.
- Learning rate init: this is set to the default value of 0.001 for optimal performance.
- Learning rate: this is set to “adaptive” so that the learning rate is constant at 0.001 as long as training loss keeps decreasing. If two consecutive epochs fail to decrease training loss by at least 1e-4 then the current learning rate is divided by 5.
- Max iter: determines the number of epochs and is set to 5000 for optimal performance. Using other numbers leads to worse performance.
- Batch size: the size of mini-batches for stochastic “adam” optimizer is set to 1024 for optimal performance. Using other powers of 2 batch sizes does not generate better results.
- Random state: determines random number generation for weights and bias initialization, train-test split if early stopping is used, and batch sampling when the solver is “sgd” or “adam”. This random number seed is set to 1 to make sure that the model has a relatively reproducible output.
- Early stopping: this is set to True so that the model will terminate training when the validation score is not improving.
- Validation fraction: this determines the percentage of validation data used in model training. This is set to 0.1 so that 10% of data will be set aside for validation. Setting it to more or less would lead to worse performance.

Figure 1: Category Distribution

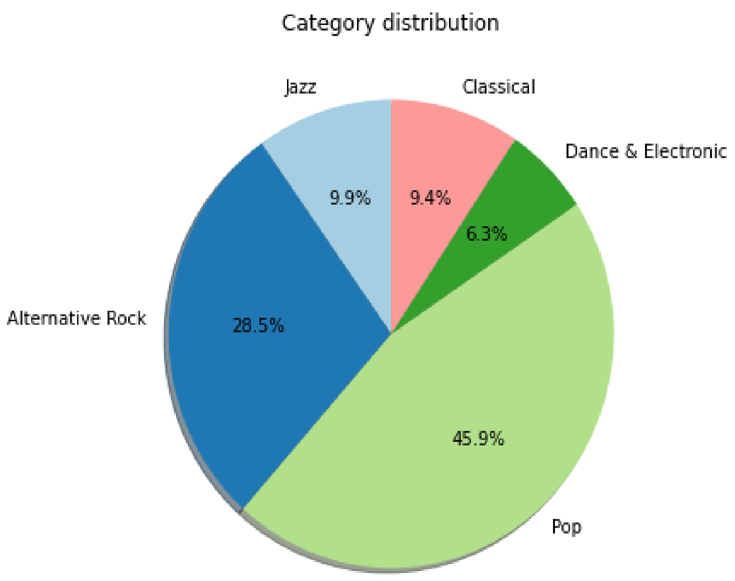


Figure 2: Overall Rating Distribution

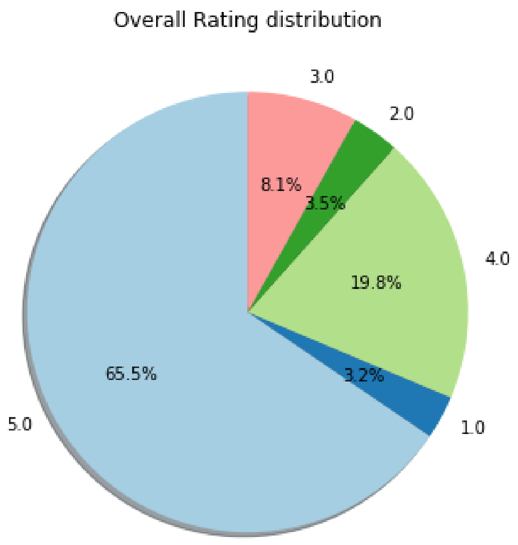


Figure 3: Music Category vs. Overall Rating

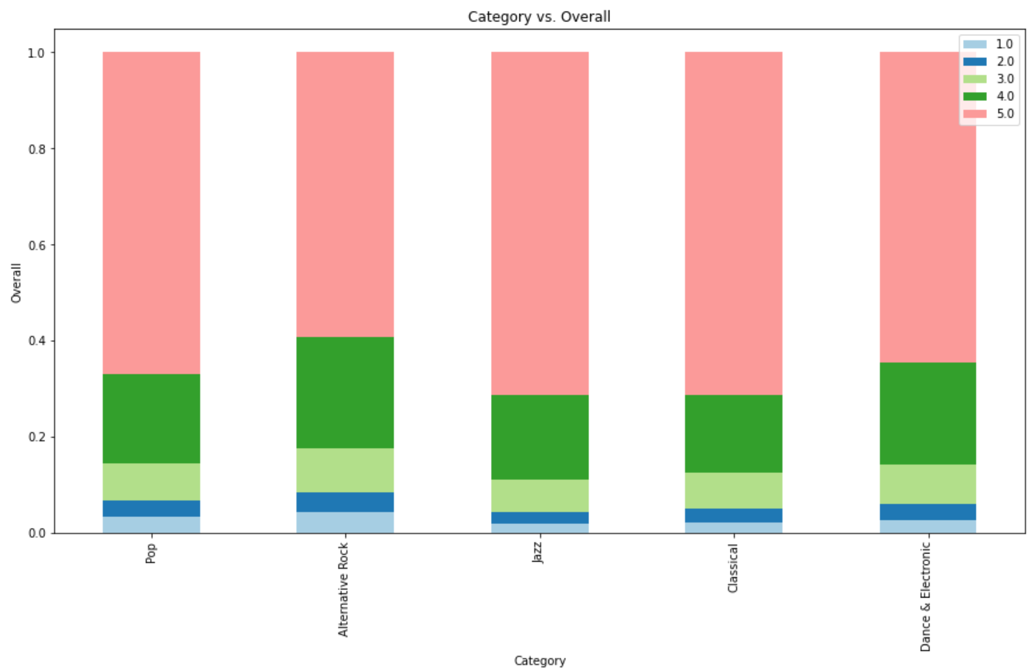


Figure 4: "ReviewText" Word Cloud

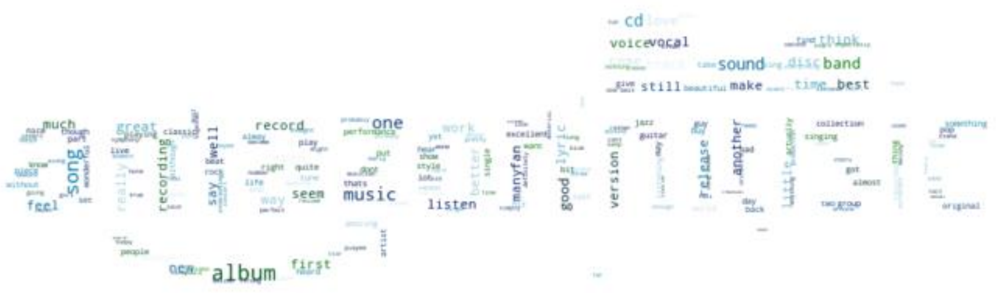


Figure 5: Linear Regression Model Prediction

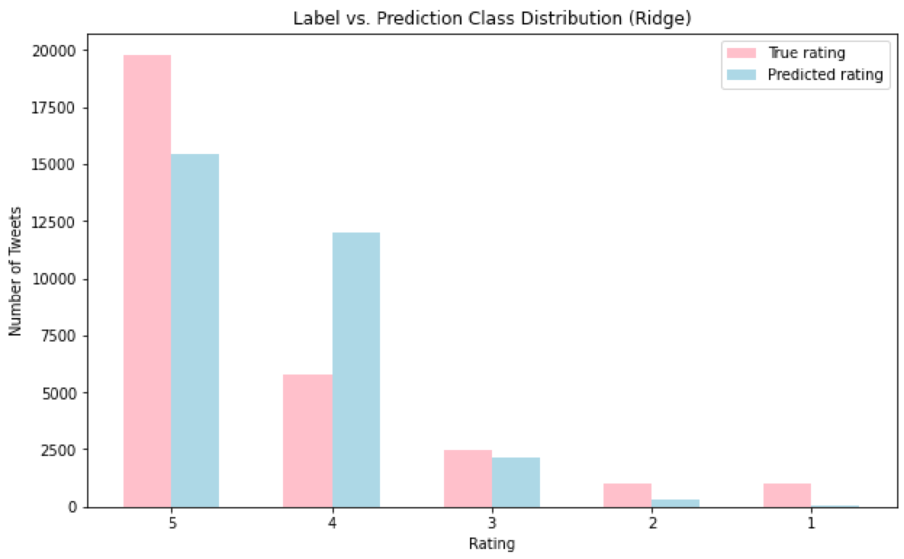


Figure 6: Light Gradient Boost Machine Model Prediction

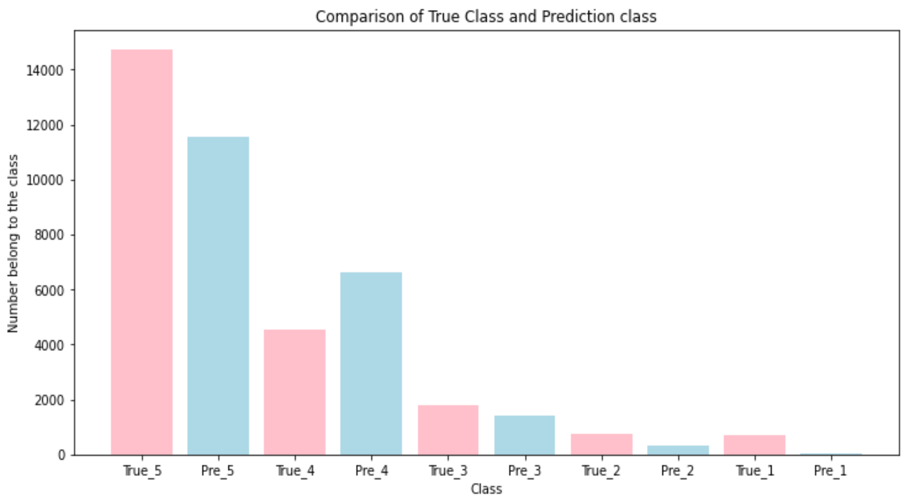


Figure 7: Multilayer Perceptron Model Prediction

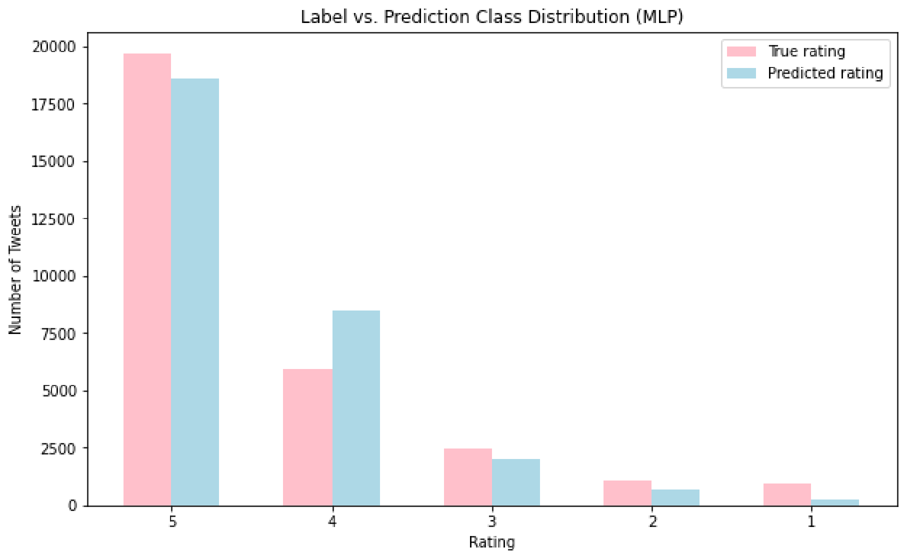
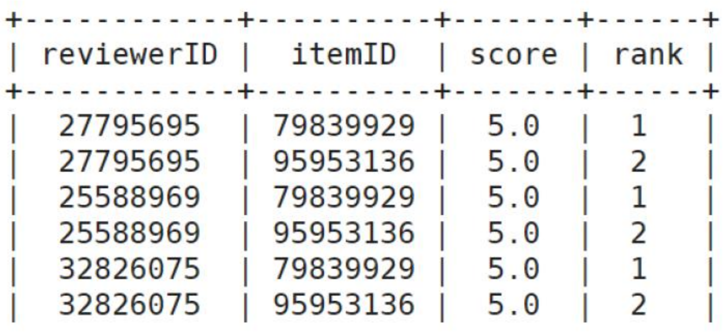
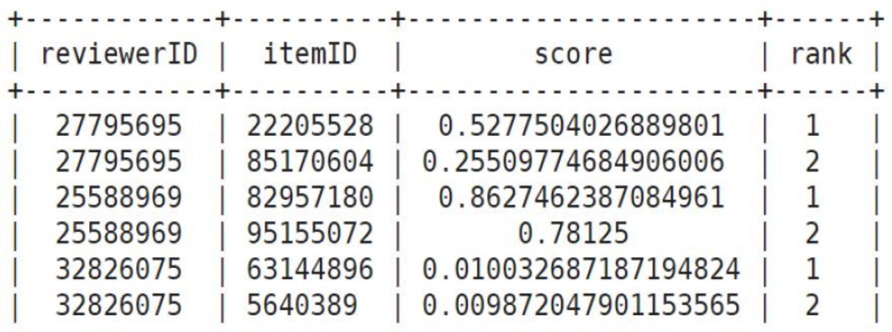


Figure 8: Recommendation by Popularity Model



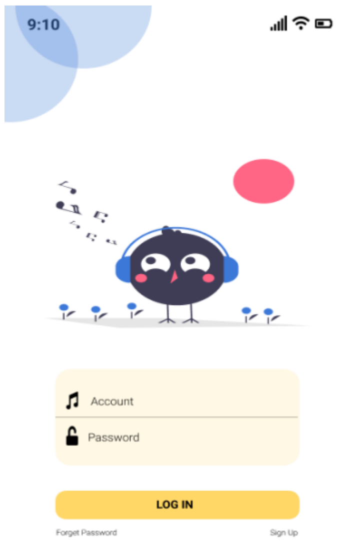
```
+------------+----------+-------+------+
| reviewerID |  itemID  | score | rank |
+------------+----------+-------+------+
|  27795695  | 79839929 |  5.0  |  1   |
|  27795695  | 95953136 |  5.0  |  2   |
|  25588969  | 79839929 |  5.0  |  1   |
|  25588969  | 95953136 |  5.0  |  2   |
|  32826075  | 79839929 |  5.0  |  1   |
|  32826075  | 95953136 |  5.0  |  2   |
```

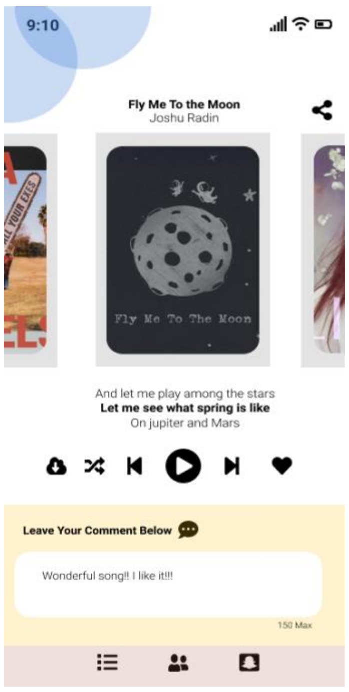
Figure 9: Recommendation by Item-based Collaborative Filtering Model



```
+------------+----------+----------------------+------+
| reviewerID |  itemID  |        score         | rank |
+------------+----------+----------------------+------+
|  27795695  | 22205528 |  0.5277504026889801  |  1   |
|  27795695  | 85170604 | 0.25509774684906006  |  2   |
|  25588969  | 82957180 |  0.8627462387084961  |  1   |
|  25588969  | 95155072 |       0.78125        |  2   |
|  32826075  | 63144896 | 0.010032687187194824 |  1   |
|  32826075  | 5640389  | 0.009872047901153565 |  2   |
```

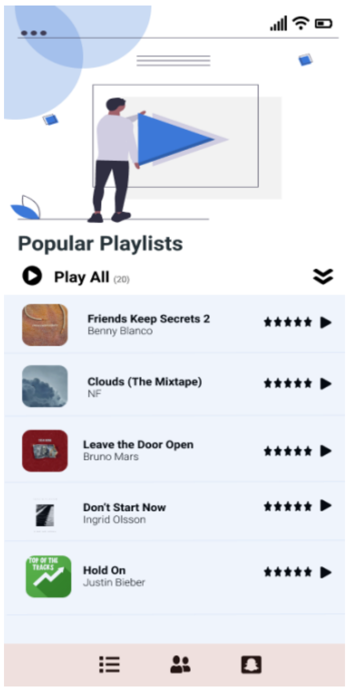
Figure 10: Spotifinder App



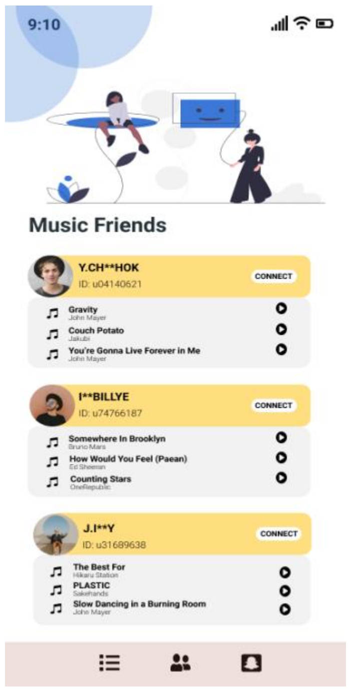


And let me play among the stars
Let me see what spring is like
On jupiter and Mars

Wonderful song!! I like it!!!



Friends Keep Secrets 2	Benny Blanco	★★★★★
Clouds (The Mixtape)	NF	★★★★★
Leave the Door Open	Bruno Mars	★★★★★
Don't Start Now	Ingrid Olsson	★★★★★
Hold On	Justin Bieber	★★★★★



Y.CHHOK** ID: u04140621

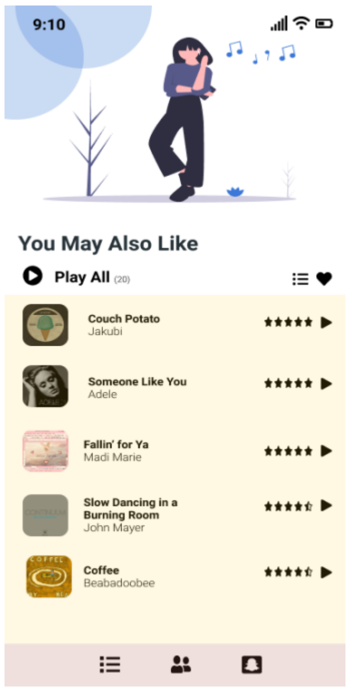
Gravity	John Mayer
Couch Potato	Jakubi
You're Gonna Live Forever in Me	John Mayer

IBILLYE** ID: u74766187

Somewhere In Brooklyn	Bruno Mars
How Would You Feel (Paean)	Ed Sheeran
Counting Stars	OneRepublic

J.IY** ID: u31689638

The Best For	Hikaru Station
PLASTIC	Sakehands
Slow Dancing in a Burning Room	John Mayer



Couch Potato	Jakubi	★★★★★
Someone Like You	Adele	★★★★★
Fallin' for Ya	Madi Marie	★★★★★
Slow Dancing in a Burning Room	John Mayer	★★★★☆
Coffee	Beabadoobee	★★★★☆