AAI 627 Final Project Report
Yahoo Music Recommender

Mourad Deihim & Daniel Gural

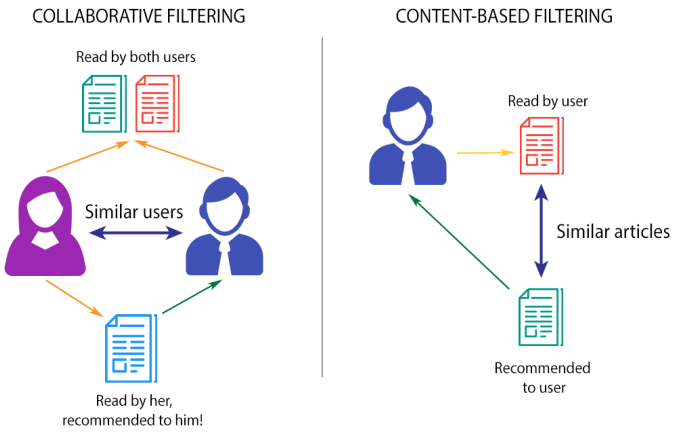
Outline

1. Project Overview
2. Data Preprocessing
3. Initial Approach
4. Matrix Factorization
5. Machine Learning Algorithms
6. Ensemble Method
7. Final Results & Conclusion

Project Overview

The objective of this project was to build a music recommender system based on the Yahoo Music Dataset. Recommendation systems are frequently found in today's world on music apps, social media, as well as video platforms. Many of the core business deliverables of companies like Spotify, Apple, Netflix, Facebook, and Youtube rely on robust recommender systems. Without it, they wouldn't be able to show their customers new and interesting songs, posts, or videos they otherwise would've missed. This helps create a more personalized experience for the user on these platforms.

There are many different types of recommender systems. These systems use different methods of determining what a user may like based on different factors. I will briefly highlight several popular recommender systems. The first and most popular method is the Collaborative Recommender System. This system takes in aggregate ratings, and generates new recommendations based on the inter-user comparison. This works well to capture complex tastes as well as unique groups of users. It runs on the assumption that people who have enjoyed the same content in the past will continue to do so in the future. The next is Content Based Recommender Systems. Recommendations are made here based on the features that may be associated with a given object. Whether it's some key words, video/song length, beats per minute or something else, these characteristics are studied for users on existing content and used to find similar content. Below is a picture that shows the differences between these first two systems.



Another common approach is Demographic Based Recommender Systems. It is a relatively low complexity method of recommending by classifying users into a specific group. The user then receives content based on the demographic or demographics that they find themselves in. Knowledge Based Recommender Systems work with a fundamental knowledge about the items in its collections. If it is able to detect a user's needs based on features or past searches, it may be able to recommend an item to fulfill that user's need. The final type of recommender system is a Hybrid System which is simply a combination of two or more of the previously stated systems. Some common hybrids are content and collaborative as well as switching hybrid recommender that switches the recommender it uses based on certain features.

Our goal of the project was to assign to a user relevant songs based on their taste. This took on the form in the test set as a group of 6 songs. We were to create different algorithms to try our best to have the highest accuracy of recommendations for these 6 songs. The data we have to prepare from is a hierarchy of track, album, artist, and genres. It is possible for a track to have many different genres as well. These tracks are paired with a user and given a rating by the user. Below are some summary statistics of the data set.

Number of Unique Users	249012
Number of Items	296111
Number of Ratings (Train)	61944406
Number of Ratings (Test)	607032

Data Preprocessing

Finding the right preprocessing for a dataset is essential for good performance. It can easily make or break a model. Data preprocessing is something that our group struggled with greatly. It can be seen quickly enough that training on just track ratings is not good enough for a model to be proficient. Doing so results in the model over recommending or not. We needed more from the data. Our group set out to get some additional ratings. Our first mistake happened early and turned out to be a fundamental one that set us back greatly. We tried to take the entire train data set of over 12 million tracks and create a new set that included average artist rating and artist and maybe more. However, we tried too much as this became incredibly complex to compute, eating up an enormous amount of RAM on Google Colab and never finishing. Running the preprocessing for days would only end in memory errors as ram ran out. The major breakthrough we had was that we did not need all the data given us to make an accurate model. We only needed data from the users we were looking at and the specific tracks we were rating. This cut down computations by an enormous factor and made preprocessing much faster. After this we created a file to be used to train all of our future models off of and performed great for us. This was a great lesson on shooting too far trying to overachieve and backfiring. Luckily, we were able to fix this mistake and still create great models in the end.

The two files that were given to the group for building and testing the models described in this paper are trainIdx2_matrix.txt and testTrack_hierarchy.txt. The trainIdx2_matrix.txt provides item ids that correspond to a trackID, albumID, artistID, or 21 possible genreIDs. These IDs come with a score that a specific user has given them from 0-100, however each user has not given a score to every single item id, so the data is rather sparse. The testTrack_hierarchy.txt

provides a hierarchy of these items on a user-item basis, which were then used to populate a new output file with all available scores for a user-track pair.

Initial Approach

Our initial approach to recommending songs to each user based on the training data given was quite simplistic, however proved to give impressive results. Given the data from "'trainIdx2_matrix.txt', we were able to determine the rating each user assigned to various item ids, that correspond to albums, artists, and genre. The format of our data can be seen in the image below:

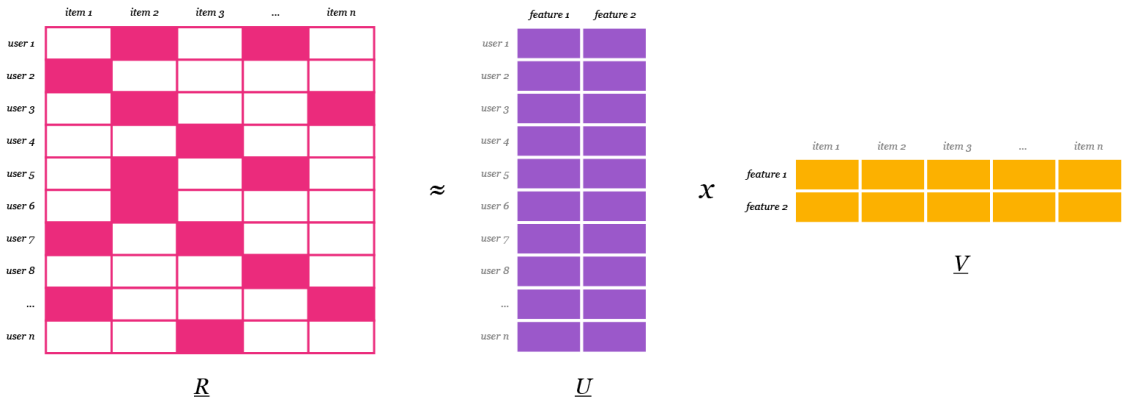
UserID	TrackID	Album Rating	Artist Rating
199810	208019	0.0	0.0
199810	74139	0.0	0.0
199810	9903	0.0	0.0
199810	242681	0.0	0.0
199810	18515	0.0	70.0

Since the genre columns were so sparse, the scores weren't included in our initial attempts. From here, we loaded the test track hierarchy data, which includes the corresponding artist, album, and genre id (though genre wasn't included) for a user-track pair. Next, the artist and album id's for the corresponding user-item pair were used to search the score given by the training data and populate a dataframe that had a column for user id, track id, artist score, and album score. If the item id for the user-track pair did not exist in the training data, no score was applied and no prediction was applied to the missing cells. After getting these scores, our first approach was to find the mean of the scores we determined that were given to each user-track pair. The data frame was sorted by ascending value, first by user, and then by the mean that we calculated. The

highest three mean scores were the tracks that were recommended for each user and the lowest three means were not recommended. This method resulted in an unexpectedly high prediction accuracy of 84.797%. The resulting predictions file was submitted to the Kaggle competition page as “prediction_mean.csv”.

Matrix Factorization

The next step that we took in the process of improving our music recommendations was to account for the user rating sparsity. Since a user does not have a rating for every single item id, it leaves significant room for error in making recommendations. In order to mitigate this, we utilized matrix factorization methods. The benefit of matrix factorization is that if explicit scores are not given, they can be inferred using implicit or explicit feedback, which can reflect their preference based on their other behavior. Matrix factorization works by decomposing the user-item matrix as a product of two other lower dimensionality matrices, which allows us to use latent factors to represent the user preferences in a lower dimensional space. One matrix is representative of a user matrix in which rows are users and columns are the latent factors. In the second matrix or item matrix, the rows can represent latent factors while the columns represent items. A visual of the matrix decomposition can be seen in the figure below.



In the sparse user-item matrix we can calculate the predicted rating (r_{ui}) the user (u) will give an item (i) as shown in the formula below, where H is the User Matrix and W is the Item Matrix.

$$\tilde{r}_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

The purpose of matrix factorization is to minimize the error between true rating and this predicted rating, in order to learn the latent factor vectors $q_i^T p_u$. This is demonstrated by the formula below, where k is the set of user-item pairs for which r is known.

$$\arg\min_{\mathbf{q}^*,\mathbf{p}^*} \sum_{u,i,\in\kappa} \left(\hat{r}_{ui} - \mathbf{q}_i^T \mathbf{p}_u\right)^2 + \lambda\left(\|\mathbf{q}_i\|^2 + \|\mathbf{p}_u\|^2\right)$$

Our approach utilizes Alternating Least Squares or ALS to minimize the equation above. ALS ensures that each iteration decreases the error until convergence by using parallelization, to alternate between minimizing the loss of the user and the item loss. To determine the best configuration of our ALS model we implemented a grid search to fine tune our hyperparameters. After training the model on our training data, we determined the RMSE for the predictions was 0.29. The best parameters discovered are listed below:

Now that we have a tuned ALS model, we are able to apply it to the test data to make predictions for the missing ratings of each user-item id pair to create a new dataset that is no longer sparse.

```
**Best Model from Training**
  Rank: 5
  MaxIter: 5
  RegParam: 0.01
  RMSE: 28.413087645002843
```

These missing values were the empty track, album, artist, and genre ratings, and the newly generated rating. The first 5 rows of the generated matrix with the newly populated scores included can be seen below:

UserID	TrackID	track_score	album_score	artist_score
199810	74139	65	61	46
199810	242681	56	63	63
199810	9903	78	0	0
199810	208019	64	53	0
199810	105760	70	65	67

Machine Learning Algorithms

The next step for our project was to implement Machine Learning Algorithms on the data. For these algorithms, the training data was received from a new file, test2_new.txt, which came with labeled predictions. The machine learning classifiers used are forms of supervised learning, where to train the algorithms, they must have a target label to find. It is much easier to train the models on a binary recommendation scale compared to guessing a rating. Before training, a pipeline was used to turn the features of the data set into a single vector, the accepted form of input for the ML models. In total, we tried six different machine learning models: Factorization Machine, Random Forest Classifier, Decision Tree, SVM Classifier, Logistic Regression, and Gradient Boosted Tree Classifier. We used cross validation to get the best parameters for each of the models.

The first model is the Factorization Machine. Factorization machines are able to estimate parameters accurately even with very sparse data. Coupled with the fact that they train with linear complexity, this allows them to be fit with very large data sets, making them a great fit for recommendation systems. The formula can be seen below. The first two terms are the intercept

$$f(x) = w_0 + \sum_{p=1}^{P} w_p x_p + \sum_{p=1}^{P-1} \sum_{q=p+1}^{P} \langle v_p, v_q \rangle x_p x_q$$

and linear terms of the formula. The last term is the pairwise interactions. Using matrix factorization, our group was able to achieve an accuracy of 85.78%

The next model is Decision Tree Classifier. It is a classifier that is structured with nodes. The first node is the root node and all the following are leaf nodes. Decision trees split the data

based on the largest information gain. This reduction of uncertainty over multiple nodes ultimately allows the tree at a set node to make a classification on the object it is observing. For our group, Decision Tree Classifier received a accuracy of 85.57%

The following model is the Random Forest Classifier. It is a collection of decision tree classifiers trained on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and better address overfitting. For our group, we received an accuracy of 85.58% for this classifier.

In addition, we used a SVM classifier, a popular ML algorithm for classification and regression. It is able to deliver good accuracy results while not requiring significant computations. SVMs aim to find a hyperplane in the N-dimensional data set that best separates the classes of data. SVMs try to find the hyperplane that has the maximum distance between points of both classes in order to classify future points with more confidence. The ROC was .873 on our best tuned model. For SVM, our group received an accuracy of 84.66%

Another ML algorithm used in this project was Logistic Regression. It is very popular due to its ability to predict categorical datasets. Logistic regression can be used to predict a binary outcome by using binomial logistic regression. The formula for classification for logistic regression as we learned in class is seen below. The ROC for this algorithm is 0.875. Our group

$$\text{Prob.}(\mathbf{y}=1)=\frac{1}{1+\exp(-[b_0+b_1\mathbf{x}_1+b_2\mathbf{x}_2+\cdots+b_n\mathbf{x}_n])}$$

for this classifier received an accuracy of 85.53%

The final classifier used by our group was Gradient-Boosted Decision Tree. Similar to random forest, it is a collection of decision trees that are trained. The differences are that gradient boosting machines combine the trees at the beginning and calculate the residuals and train from there while random forests combine at the end of the process. Our group received an accuracy of

85.56% using the Gradient-Boosted Decision Tree.

Ensemble

Ensemble learning tries to combine multiple models to create a better performing model. Different models may have captured different patterns or properties of the dataset and by combining the models could detect with better accuracy. We used a least squares solution as seen below. **S** is the score matrix of predicted results. We reform **S^Tx** as **N(2P-1)** with N being the

$$\arg\min_{\mathbf{a}} \|\mathbf{x} - \mathbf{S}\mathbf{a}\|^2 \qquad \Rightarrow \qquad \mathbf{a}_{\mathsf{LS}} = \left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{x}$$

number of predictions and P being the correct rate of the model matrix. This gives us the coefficient needed to make a prediction on the data set. Using the final equation seen below,

$$\mathbf{s}_{\mathsf{ensemble}} = a_1\mathbf{s}_1 + a_2\mathbf{s}_2 + \cdots + a_K\mathbf{s}_K = \mathbf{S}\cdot\mathbf{a}_{\mathsf{LS}} = \mathbf{S}\left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{x}$$

we are able to obtain our final ensemble prediction. For our group, we tried many different forms of ensemble learning combinations. With 36 different combinations of ensemble models tried, we were unable to find one that surpassed our best model. This may be due to our models picking up the same patterns or being heavily correlated. In the future with more features and different models, ensembles could still be used to enhance our results still.

Results

Model	Accuracy
Mean	84.797%
Factorization Machine	85.78%
Decision Tree Classifier	85.57%
Random Forest Classifier	85.58%
SVM	84.66%
Logistic Regression	85.53%
Gradient Boosted D-Tree	85.56%
Ensemble	84.8%

Conclusion

In the process of completing this project, our group was able to obtain a much better grasp of the topics discussed in class by applying it to a real world situation. We certainly struggled at first, especially with processing our data. This occurred for two reasons: 1) there was a significant learning curve before we became familiar with PySpark and how to effectively manipulate the data frames that were obtained from the given txt files. 2) The sparsity of the data proved to be a problem of its own that needed to be overcome. Despite our initial struggles, we managed to process the data, and went even further to make predictions for the missing scores

using matrix factorization. Though this wasn't perfect with a training RMSE of 28, it helped broaden our understanding of matrix decomposition and the fundamental mathematical concepts of the existing loss minimization algorithms such as the alternating least squares method. From the sparse Yahoo data that was given, the group was then able to make accurate predictions for songs that a user would like or dislike. We made use of many different models to make our predictions, from our most simple, which is a mean of all given scores, to the multiple fine tuned machine learning algorithms that were described. Lastly was our ensemble model, in which we attempted countless combinations of our other models in order to achieve a higher accuracy. Out of all of the models that were utilized, we observed that the most accurate was the factorization machine, which produced an accuracy of 85.78% on the test data. In the future, to further improve the results that were achieved, other machine learning algorithms could certainly be implemented. Furthermore, different ensemble configurations with these models or completely different ensemble methods such as a bagging decision tree could be implemented to improve our accuracy. The greatest change our group would like to execute in the future is more features to train on for these models. Taking into account things like mean, variance, or count of the ratings on artists, albums and genres could greatly affect our models performance and lead to better overall accuracy. We would be very interested to see the cost to benefit ratio of training and predicting with these added features as well.