Practical 3: Preferences in Streaming Music

Bolei Deng, Matheus C. Fernandes, and Nathan Wolfe
boleideng@g.harvard.edu, fernandes@g.harvard.edu, nwolfe@college.harvard.edu

April 7, 2017

1 Technical Approach

1.1 Feature generation

Initially, we generated features for individual artists and users. For the users, we used their raw age if available as a feature. The users' country and gender data we converted into one-hot vectors for binary features.

For the artists, we gathered a variety of data from MusicBrainz[1]. This data includes: the artist's gender, type (group or individual), age, years since first release, years since last release, years since average release, live percentage of releases (as opposed to studio), album percentage of releases (as opposed to LP/EP), country of origin, and MusicBrainz tags (yielding the genre along with other miscellaneous information). Categorical data such as gender, type, country, and tags were converted into one-hot vectors for binary features.

1.2 Validation and testing

The validation and testing sets were generated by separating a percentage of the input data from the user-artist counts file. For this, we took the dictionary containing all of the data and distributed them into two separate csv files that are called as training and test sets. For the test set, we created two files, one containing no counts to be used for prediction, and one that contains the answers. As a verification, we import the answers and compare them to the predictions using the evaluation:

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N} |\hat{y}_n - y_n|$$

1.3 Algorithm Design

In this practical, we are trying to predict the number of times a user listen to a artist given train data and users', artists' profiles. For each user there is a distribution of listening times to all 2000 artists, so that the whole data set is users number (233,000) by artists number (2000), which is roughly 0.5 billion. While only 4 million user-artist pairs are known, therefore we are missing a lot of informations. To decrease the size of data set, as well as to complete the information, we

[1] `https://musicbrainz.org/`

need to cluster artists to a number of classes. By doing that, we are actually treating every artist in the same class equally. Considering 20 artist classes, we reduce the data set to 233,000 by 20, which contents around 0.5 million elements. Sense we have 4 million train pairs, the information is more complete, i.e., we now have a better understanding of the preference of a user on different classes of artists.

1.3.1 Artist clustering

First, we extract features for individual artists using their MusicBrainz data, as described in section 1.1.

On top of that, we define the popularity of an artist by the total times he or she listened by users. We normalize this value by the maximal play times among all the artists. Popularity closes to 1 means the artist very popular, and closes to 0 means very unpopular. Then the structure of artist features would look like

$$Artist\ features\ =\ Basic\ features\ +\ Popularity$$

With the basic features and popularity, we cluster artist into classes.

1.3.2 User clustering

We have some features for individual users extracted from their basic data, described in section 1.1.

Furthermore with the clustered artists, we now know better about the preference of users. The preference a user to a certain class of artists is defined by the total times he or she listened to all the artists within this class. These numbers, for different artist classes, are normalized by the mean of nonzero values. For zero values, we set them to 1. That means if a user listen to a certain class of artist more times than average times, the preference of this user on this class is > 1; while if less than average, < 1. If the user has 0 listen times to a certain class of artists, we regard that as lost information and add no preference to that class, i.e., set preference to 1. So we add some new features, which is the same number of classes of artists, to user features, that means

$$User\ features\ =\ Basic\ features\ +\ Preference$$

We then cluster users based on these features. For each user class, we report the mean preference of that class, representing the average preference of that class. By doing that we construct the core matrix, which is user class number by artist class number. The schematic diagram of this approach is shown in Fig. 1.

In prediction part, for each incoming user-artist pair, we first find out the user class the user belongs to and the artist class the artist belongs to, then get the preference of the user class to that artist class. Because the user median practically does a relatively good prediction, we use it as a base case and from that determine a perturbation from the mean based on the prediction of our algorithm.

$$Prediction\ =\ User\ median\ +\ Preference$$

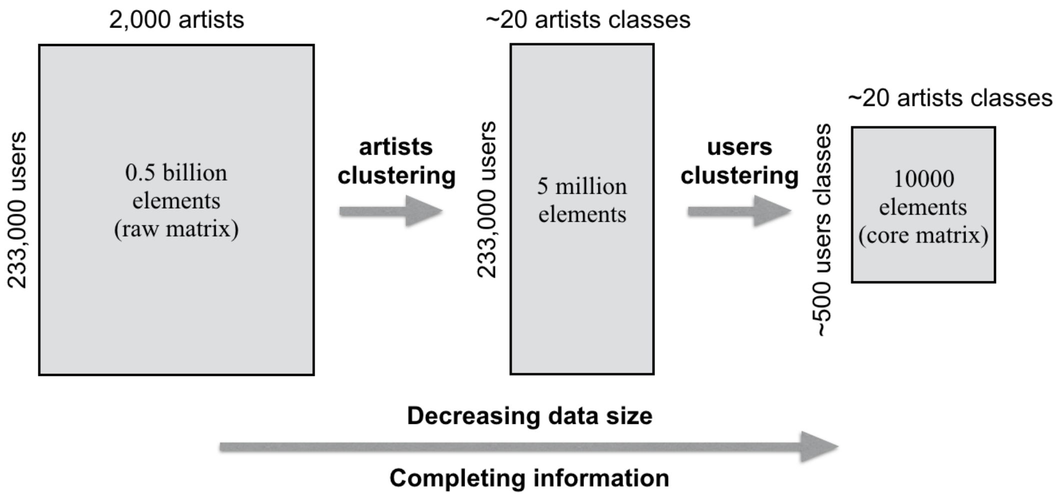

Figure 1: Schematic diagram for the technical approach in this report.

Practically, we do not want to deviate too much from user median. Therefore we compress this perturbation by a certain scale. The scale of this perturbation is empirically determined.

1.4 Clustering Method

For the clustering method, we decided to use K-Means clustering. Because the dataset is very large, K-Means does a better job in terms of efficiency than hierarchical clustering. Additionally, we do not want an individual cluster for each user and each artist as this becomes very computationally expensive given our relatively large dataset. The one difficulty introduced by working with K-Means clustering is that we need to experiment with the number of clusters to see how many are optimal for the prediction. Furthermore, depending on the initialization of the global clusters, we also see different end results for the clusters, as this is a known sensitivity of the K-Means method. To try to mitigate these issues, we attempted different clustering algorithms such as Affinity Propagation, and DBSCAN, none of which seemed to improve our performance.

2 Results

Table 1 shows the results of tests run as described in section 1.2. Except for the test on the baseline User Median method, all tests were run on the same basic model as described in section 1.3. The right-hand “Improvement” column shows the improvement in error from the User Median baseline.

What differed among these tests was the degree of clustering used. The upper half of the table shows the results for an Only Artist Clustering method. For this method no user clustering was used (this is equivalent to treating each user as its own cluster, for $233,000$ user clusters). We varied the number of artist clusters. The lower half of the table shows the results of clustering both users and artists for varying numbers of clusters.

Our results show a sweet spot at 50 artist clusters for a modest improvement on the baseline. However, the introduction of user clusters doesn’t improve the results, but rather hampers them.

Method	User class #	Artist class #	Error	Improvement
USER MEDIAN	N/A	N/A	138.65	- -
ONLY ARTIST CLUSTERING	N/A	10	138.58	0.07
ONLY ARTIST CLUSTERING	N/A	20	138.55	0.10
ONLY ARTIST CLUSTERING	N/A	50	138.47	0.18
ONLY ARTIST CLUSTERING	N/A	100	138.62	0.03
USER & ARTIST CLUSTERING	50	50	138.62	0.03
USER & ARTIST CLUSTERING	100	50	138.59	0.06
USER & ARTIST CLUSTERING	200	50	138.49	0.16
USER & ARTIST CLUSTERING	500	50	138.62	0.03

Table 1: Performance for clustering with different number of user classes and artist classes.

This can be explained by the user data given being insufficient to cluster them effectively, even when data on their number of plays is added as a feature.

Using a model trained on the full train data set, we created a set of predictions and submitted them to Kaggle, seeing a modest 0.035% improvement in error over the user median method. For the Kaggle submission we used the best method determined using the results from Table 1, which was only artist clustering, with 50 artist classes.

3 Discussion

To obtain these results, we began by naively assuming that all of the artists get the average number of plays. This simple approach got us to the baseline of the Kaggle competition. Then, we introduced a perturbation to this mean based on our predictions to improve the score. To obtain the perturbation for each artist, we began by considering only the original features provided to us by the assignment. Using K-Means clustering, as our clustering method did not get us very far from baseline. We then improved our predictions by adding a substantial amount of features on the artist's side. By introducing these features to our model, we were able to improve our predictions to above the baseline. We also tried introducing the genre of each artist, and this made our predictions worse – which tells us that the listeners do not cluster well with the artist genre but instead with the country of origin as well as the release dates.

Github Repository

`https://github.com/matheuscfernandes/cs181-s17-practicals/tree/master/p3`