CS181 Practical 3 : Preferences in Streaming Music

Team [J.Q.G]
Jeewon (Grace) Hwang, Qing Zhao, and Jing Wen

April 2016

1 Technical Approach

In this practical,we are tasked with predicting people's tastes in music. Specifically, we are predicting the playtimes that different users would have listened to tracks of different artists over a span of time. The data sets we are given in the first place are:

- Basic self-reported demographic information about many, but not all, of the users, such as sex, age, and location
- Name of artists and their MusicBrainz1 ID. There are about 233,000 users and 2,000 artists
- The training set with over 4.1M (user, artist) pairs
- The test set of a similar size

Our objective is to predict how many times a user will listen to a new artist. Unlike the previous two practicals, the problem this time does not fall into typical regression or classification categories, thus in this report we'll describe that we have tried a set of naive algorithms, unsupervised learning algorithms like clustering (K-Means), as well as supervised learning algorithms such as regression models (Generalised Linear Regression), Random Forest and so on.

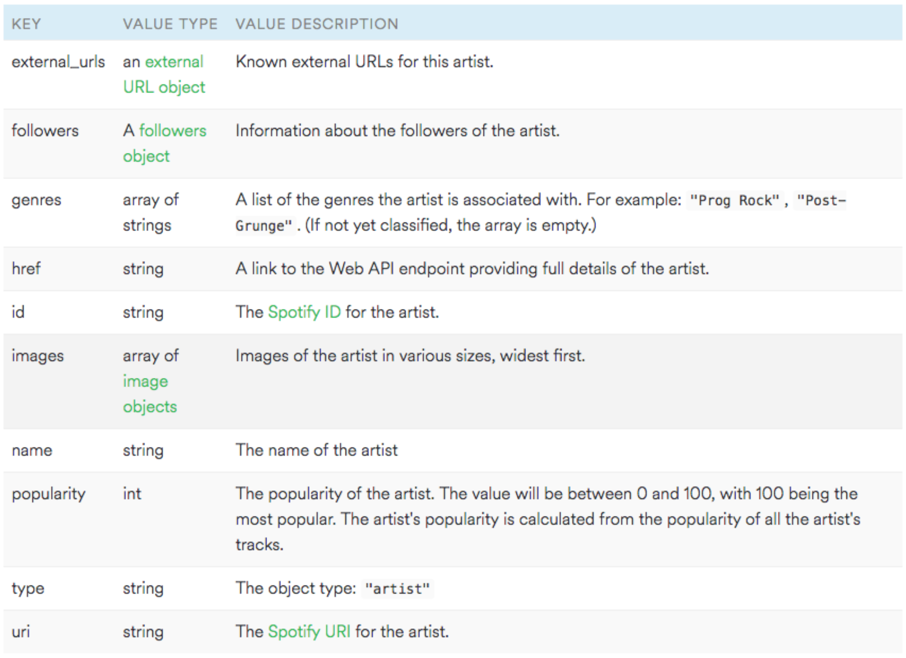
1) Features Engineering

Impute Missing values : Age, Gender
When we looked up the user-profile data, not every data for each user is filled up. Quite a lot of user did not have sex and age data (19535 users had missing sex-value, 44842 for age value). Therefore, to utilize these sex and age features, we imputed those missing values by looking up the rest user's data. For instance, when we compare the ratio of Male and Female, it was around 3:1, so according to this probability, we filled the missing sex value. We applied the same logic for computing missing Age value.

Scrap more data from external database to Add more Features
So far, all the information we have for artists consists only of artist names and their MusicBrainz1 ID, and for users we only have their sex, age, and country. Apparently, to make predictions about how many times a user will listen to a new artist, these features are not sufficient, especially on the artist side. For users, it was almost impossible to get any additional information, whereas it is much easier to have more insights about artists by exploring some public music API, such as MusicBrainz1, Spotify and Wikipedia.

Therefore, to begin with, we decided to implement our own scrap-function using BeautifulSoup to fetch the genre info for each artist from Wikipedia.

In addition, we used 'Spotify Web API', which lets us fetch data from the Spotify music catalog. Figure 1 is the list of information we could get about a given artist. Among them we extract "followers", "genre" and "popularity" as our new features.



KEY	VALUE TYPE	VALUE DESCRIPTION
external_urls	an external URL object	Known external URLs for this artist.
followers	A followers object	Information about the followers of the artist.
genres	array of strings	A list of the genres the artist is associated with. For example: `"Prog Rock"`, `"Post-Grunge"`. (If not yet classified, the array is empty.)
href	string	A link to the Web API endpoint providing full details of the artist.
id	string	The Spotify ID for the artist.
images	array of image objects	Images of the artist in various sizes, widest first.
name	string	The name of the artist
popularity	int	The popularity of the artist. The value will be between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of all the artist's tracks.
type	string	The object type: `"artist"`
uri	string	The Spotify URI for the artist.

Figure 1: Spotify Artist Object (Full)

Transform Categorical data
We noticed that there are many categorical data in our training set, such as country, gender, userID, artistID. Simply mapping them into a range of index would not work well since the index values would be computed by our model, which is dubious since they are just categorical data that does not reflect any numerical significance. To address this problem, we converted them into one-hot-encoded vector. By doing this, the distance between either two vectors will always be one.

2) Feature Reduction

Country Re-mapping
By transforming *country* into one-hot-encoded form, we got around 250 more new features, which gives rise to the curse of dimensionality. These 250 features actually contain only 1 piece of information - country. To reduce dimensionality and introduce correlations among countries, we mapped these countries to 5 different continents which is still in the form of one-hot-encoded vector.

PCA
Let's set up as follow. Assuming there are D total number of artists and N total users, and let the number of play of user i for all the artists to be $\vec{u_i} = (u_{1i}, \cdots, u_{di})$. U is a $N \times D$ matrix defined as below:

$$U_{N,D} = \begin{pmatrix} \cdots & \vec{u_1} \cdots \\ & \vdots \\ \cdots & \vec{u_N} \cdots \end{pmatrix}$$

Thus, we can transform our training data to this $N \times D$ matrix. Then, we apply PCA on this matrix to reduce its dimension and include them as additional features for users. This would allow us to include some information regarding artists played by users.

Data Transformation using Sparse Matrix
Since our data set was huge compared to the previous Practicals, it was challenging for us every-time to deal with the large volume of data. Therefore, we looked for efficient way that can compress the data representation.

Then we found out the Sparse Matrix from Scipy package. We wanted to construct a matrix U that has all distinct pair of ($user_i$, $artist_j$) to analyze the similarity. We have $233,296$ users and $2,000$ artists, and hence U has $233,296 \times 2,000 = 466,572,000$ elements. On the other hand, we only have $4,154,804$ pairs of (user, artist) in the training data, which accounts for less than 0.9% of the elements. This implies the matrix U is very sparse. As our data is sparse, we can take advantage of storing them as sparse matrices so that we can compute more efficiently. This transformation allowed us speed up our computation and analysis significantly.

3) Models

K-means

Since we now have features, such as age, country, continent, and music played, we can group users into clusters $\{C_1, \cdots, C_k\}$ based on these information using 'K-Means' algorithm.

After we group users into clusters, we compute the mean u_k for $i \in C_k$, and transform it to a proportion matrix P. Then p_{ij} indicates that for a user i, the proportional of time he or she would listen to artist j. Then we make prediction based on this matrix and clusters. First, we find the cluster that user i belongs to, and then we can find corresponding p_{ij}. Our prediction of user playtimes will be the product of p_{ij} and the number of total plays of the user j.

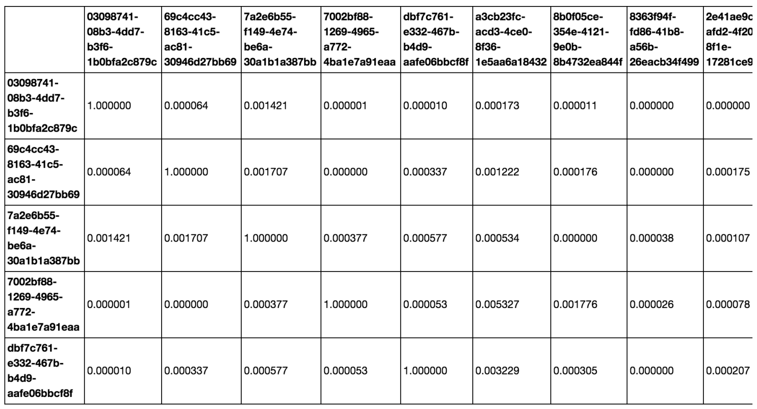
Similarity Matrix

Another approach we took is to solve the problem from a different angle. Instead of grouping users and studying their behaviors, we look into the similarity of the artists. Given the user-artist matrix U, we can compute the cosine between vectors of each pair of artists, and hence construct a similarity matrix for artists.

With the similarity matrix, we can find the nearest neighbors for each artist. When we need to predict the number of times that a user will play the music of an artist, we can find the list of nearest of neighbors of the artist. Then, we go over the list and see if the user has listened to any of the neighbor artists before. If so, we would use the number of play of the nearest neighbor artists as our prediction.

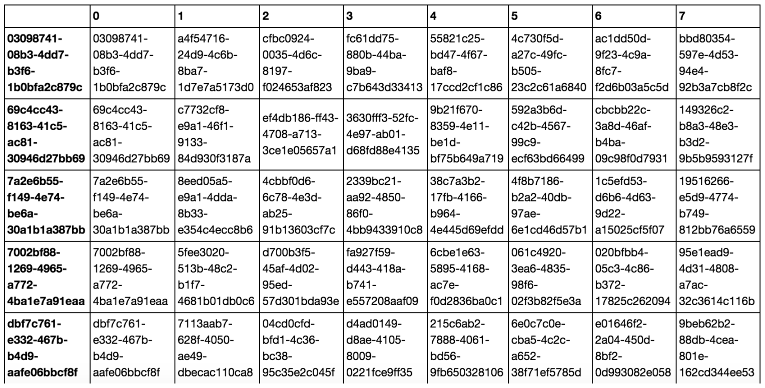
The **cosine similarity** between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them. Here we are using this as a measure of the similarity between artists. Detailed steps are as below:

1. Let $\vec{a_i}$ be the ith column of the user-artist matrix U, i.e. a vector for artist i. Compute the similarity matrix S (see Figure 2)
$$S_{ij} = \frac{\vec{a_i}\vec{a_j}}{||\vec{a_i}||||\vec{a_j}||}$$
2. Then apply 'NearestNeighbors' algorithm to find the K nearest neighbors for each artist. (see Figure 3)
3. To make prediction for user i listening to artist j, we find artists j's nearest neighbor whom user i has listened to and use the number of play as our prediction. If the user does not listen to any of the nearest neighbors of the artist, we will use the user median number of play as our prediction.



	03098741-08b3-4dd7-b3f6-1b0bfa2c879c	69c4cc43-8163-41c5-ac81-30946d27bb69	7a2e6b55-f149-4e74-be6a-30a1b1a387bb	7002bf88-1269-4965-a772-4ba1e7a91eaa	dbf7c761-e332-467b-b4d9-aafe06bbcf8f	a3cb23fc-acd3-4ce0-8f36-1e5aa6a18432	8b0f05ce-354e-4121-9e0b-8b4732ea844f	8363f94f-fd86-41b8-a56b-26eacb34f499	2e41ae9c-afd2-4f20-8f1e-17281ce9
03098741-08b3-4dd7-b3f6-1b0bfa2c879c	1.000000	0.000064	0.001421	0.000001	0.000010	0.000173	0.000011	0.000000	0.000000
69c4cc43-8163-41c5-ac81-30946d27bb69	0.000064	1.000000	0.001707	0.000000	0.000337	0.001222	0.000176	0.000000	0.000175
7a2e6b55-f149-4e74-be6a-30a1b1a387bb	0.001421	0.001707	1.000000	0.000377	0.000577	0.000534	0.000000	0.000038	0.000107
7002bf88-1269-4965-a772-4ba1e7a91eaa	0.000001	0.000000	0.000377	1.000000	0.000053	0.005327	0.001776	0.000026	0.000078
dbf7c761-e332-467b-b4d9-aafe06bbcf8f	0.000010	0.000337	0.000577	0.000053	1.000000	0.003229	0.000305	0.000000	0.000207

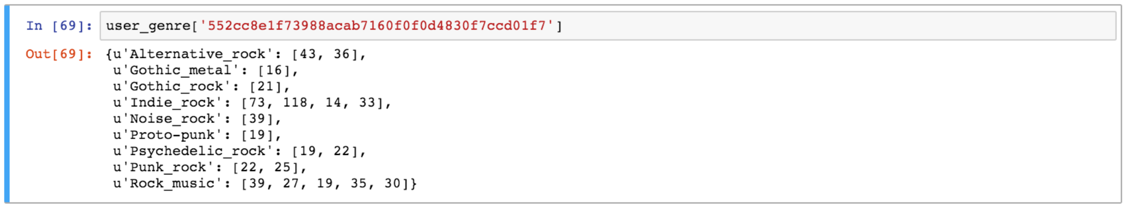
Figure 2: Artist Similarity Matrix Sample



	0	1	2	3	4	5	6	7
03098741-08b3-4dd7-b3f6-1b0bfa2c879c	03098741-08b3-4dd7-b3f6-1b0bfa2c879c	a4f54716-24d9-4c6b-8ba7-1d7e7a5173d0	cfbc0924-0035-4d6c-8197-f024653af823	fc61dd75-880b-44ba-9ba9-c7b643d33413	55821c25-bd47-4f67-baf8-17ccd2cf1c86	4c730f5d-a27c-49fc-b505-23c2c61a6840	ac1dd50d-9f23-4c9a-8fc7-f2d6b03a5c5d	bbd80354-597e-4d53-94e4-92b3a7cb8f2c
69c4cc43-8163-41c5-ac81-30946d27bb69	69c4cc43-8163-41c5-ac81-30946d27bb69	c7732cf8-e9a1-46f1-9133-84d930f3187a	ef4db186-ff43-4708-a713-3ce1e05657a1	3630fff3-52fc-4e97-ab01-d68fd88e4135	9b21f670-8359-4e11-be1d-bf75b649a719	592a3b6d-c42b-4567-99c9-ecf63bd66499	cbcbb22c-3a8d-46af-b4ba-09c98f0d7931	149326c2-b8a3-48e3-b3d2-9b5b9593127f
7a2e6b55-f149-4e74-be6a-30a1b1a387bb	7a2e6b55-f149-4e74-be6a-30a1b1a387bb	8eed05a5-e9a1-4dda-8b33-e354c4ecc8b6	4cbbf0d6-6c78-4e3d-ab25-91b13603cf7c	2339bc21-aa92-4850-86f0-4bb9433910c8	38c7a3b2-17fb-4166-b964-4e445d69efdd	4f8b7186-b2a2-40db-97ae-6e1cd46d57b1	1c5efd53-d6b6-4d63-9d22-a15025cf5f07	19516266-e5d9-4774-b749-812bb76a6559
7002bf88-1269-4965-a772-4ba1e7a91eaa	7002bf88-1269-4965-a772-4ba1e7a91eaa	5fee3020-513b-48c2-b1f7-4681b01db0c6	d700b3f5-45af-4d02-95ed-57d301bda93e	fa927f59-d443-418a-b741-e557208aaf09	6cbe1e63-5895-4168-ac7e-f0d2836ba0c1	061c4920-3ea6-4835-98f6-02f3b82f5e3a	020bfbb4-05c3-4c86-b372-17825c262094	95e1ead9-4d31-4808-a7ac-32c3614c116b
dbf7c761-e332-467b-b4d9-aafe06bbcf8f	dbf7c761-e332-467b-b4d9-aafe06bbcf8f	7113aab7-628f-4050-ae49-dbecac110ca8	04cd0cfd-bfd1-4c36-bc38-95c35e2c045f	d4ad0149-d8ae-4105-8009-0221fce9ff35	215c6ab2-7888-4061-bd56-9fb650328106	6e0c7c0e-cba5-4c2c-a652-38f71ef5785d	e01646f2-2a04-450d-8bf2-0d993082e058	9beb62b2-88db-4cea-801e-162cd344ee53

Figure 3: Nearest Neighbor for Each Artist

Mixture Gaussian Model : each Genre form Clusters in each User
We believed that the users' playtimes of an artist is associated with the genre that the artist belongs to. If two artists, A and B, belong to the same genre, the user who plays the music of A a lot would also tend to play the music of B a lot. Therefore, if we can cluster artists into different genres, then we will be able to estimate more accurately the playtimes of a certain music by either taking mean or median of playtimes in the same genre that this user had tried before. So we collected 'Genre' data from Wikipedia for each artist, and Figure 4 represents the following idea.



```
In [69]: user_genre['552cc8e1f73988acab7160f0f0d4830f7ccd01f7']

Out[69]: {u'Alternative_rock': [43, 36],
          u'Gothic_metal': [16],
          u'Gothic_rock': [21],
          u'Indie_rock': [73, 118, 14, 33],
          u'Noise_rock': [39],
          u'Proto-punk': [19],
          u'Psychedelic_rock': [19, 22],
          u'Punk_rock': [22, 25],
          u'Rock_music': [39, 27, 19, 35, 30]}
```

Figure 4: Genre Clusters per User

So we transformed this problem into a **Mixture Gaussian Model** problem, since we can model each playtimes as a variable that we want to infer, and they are coming from the combination of the probability of each different genres that a user already have tried. So likelihood comes from Normal distribution (we assumed that natural data comes from normal distribution at this moment) for a specific genre that this artist belongs to and prior comes from probability of membership among different genres that a certain user tried before. So the problem can be described as below,

$$p(x_{ij}) = \sum_{k=1}^{K} \pi_k N(x_{ij}|\mu_k, \Sigma_k)$$

where x_{ij} is playtimes for (user i, artist j), and K is the number of different Genres that the (user i) have tried before in train data. If a user haven't tried yet a genre that new artist in test data set belongs to, then we plugged user-median instead.

Random Forest
Since in our case there are many features that are categorical, and the output of our predictive model is playtimes, it is very natural for us to relate the problem to a set of decision trees where the paths from root to leaf represents classification rules based on country, age, gender, etc. Based on this, our intuition is that by fitting a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and averaging over all decision trees, we could improve the predictive accuracy and control over-fitting.

Features we use to feed into our model are:
age, gender, total_plays, continent, user, artist, genre, followers and popularity.

Since Scikit Learn only accepts numerical data, as previously discussed in the "feature engineering" and "feature reduction", we convert user, artist, continent, genre and gender into numerical representation and used the same codes as previous practicals to implement random forest regressor. The results will be shown and discussed later in our report.

Bayesian Ridge Regression
Besides Random Forest with all the nice properties of decision trees that aligns with our application scenarios, we were also interested in trying Bayesian Ridge Regression based on the following three reasons:

- it adapts to the data at hand
- it can be used to include regularization parameters in the estimation procedure.
- more importantly, we are interested in learning whether the number of plays has a linear relationship with our predictors.

For specific implementation, we used GridSearchCV and BayesianRidge provided by scikit-learn to come up with best set of parameters and make predictions. The result are shown in the next section.

Extremely Naive Approach : $Constant \times User\_Median$
If you stop here and jump directly to the result part, you'll find out that no matter how hard we try in feature engineering and modelling, we could beat the user median baseline! So while keep modifying our existing models, we start to think that maybe the naive way of simply computing user median makes more sense, and maybe it's a good idea to just follow this simple yet "expressive" trajectory! After some exploratory data analysis (see "Discussion" part), we found most user's play number follows slightly skewed distribution. Based on this finding, we speculated that rather than predicting play number according to user median, we could add an weight on the median to fit the skewed distribution. Using the given MAE function, we could test different weight w and select the w which gives us lower MAE value than $w = 1$.

2 Results

MAE Test Score from Kaggle

As we see in the Table 1 below, we realized that other models that do not utilize per-user-median did not work out well to beat the benchmarks. From this, we decided to come up with totally different approach that can make a subtle improvement from the per-user-median. Therefore, we come up with Extremely Naive Approach, and Table 2 is the result for our experiments. The reasoning behind this approach is dealt above in the 'Extremely Naive Approach' part and in the Discussion.

Table 1: Test Results Summary

Approach	Version	Test Score
K-Means	K=10	252.79
	K=100	243.07
Similarity (KNN)	K=100	176.67
	K=285	197.49
Bayesian Ridge Regression		386.20
Random Forest		387.91
Genre Median		157.47
User Median	Median	137.79
	$0.95 \times Median$	137.68

Validation Score for Extremely Naive Approach

In order to choose appropriate constant in the Extremely Naive Approach, we splitted our train set into 2 chunk - one for train data and the other for validation purpose. Table 2 describes the validation score (MAE) among different constant values.

Table 2: Validation Score for Extremely Naive Approach

Constant value	Validation Score	Kaggle Score
c = 1.05	139.098	
c = 1.03	138.697	
c = 1.01	138.380	
c = 1.00	138.253	137.790
c = 0.99	138.153	
c = 0.97	138.018	
c = 0.96	137.987	
c = 0.95	**137.979**	**137.680**
c = 0.94	137.998	
c = 0.93	138.041	
c = 0.80	141.142	

We found out that c = 0.95 case gave us the least MAE Validation score, so we submitted it to Kaggle as our final model that beats the per-user-median benchmark.

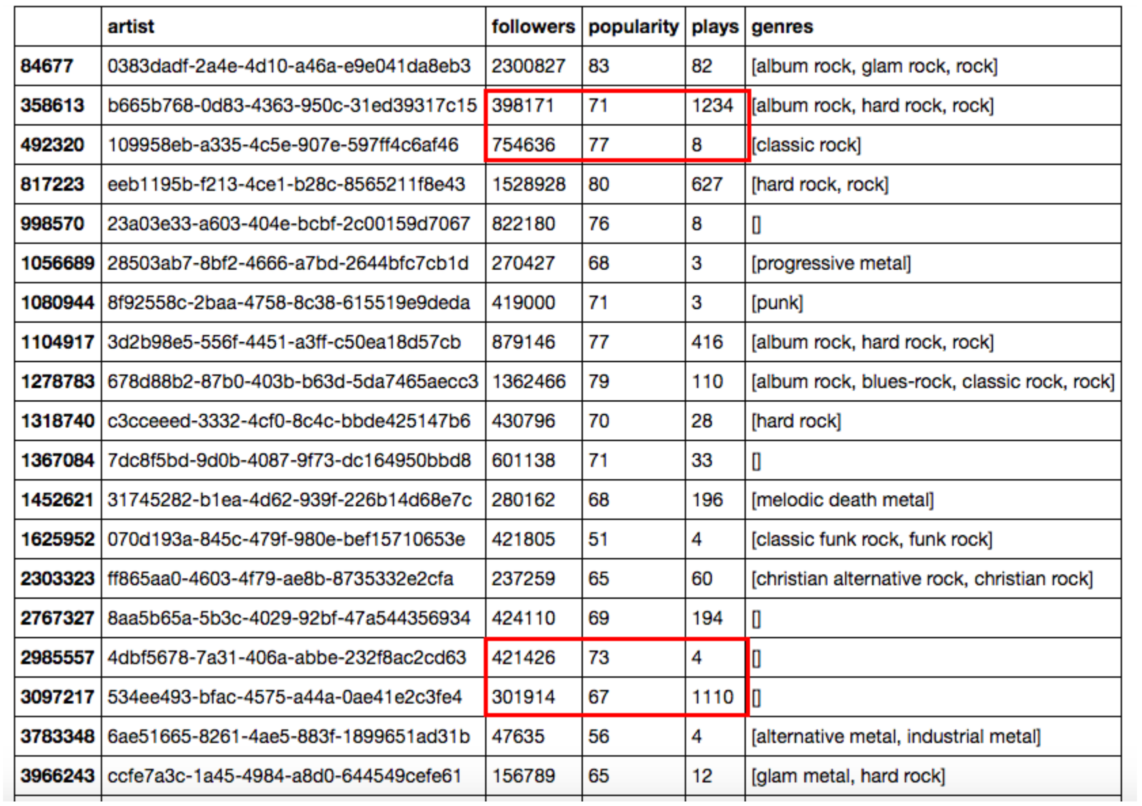
3 Discussion

Feature Engineering

Aside from the existing features, we scrapped more information about artists from Spotify API, including genre, artist popularity, and the number of followers each artist has, hoping to get more insights to the artist aspect as well as users' taste. However, as suggested by the results in last section, they did not work out well. To figure out the reason behind this, we randomly picked out some users and looked into the artists information he or she follows. Figure 5 shows one of the samples we draw with user id '092aa297bc75015ec397cf095a3781cec0d19c00'.

Consider the rows labeled with red square, two artists with roughly the same degree of popularity, number of followers and genres have tremendously different playtimes. And as we go through multiples samples, this happens quite frequently, which lead us to believe that the features we have so far are not informative enough to help us make any solid predictions or clustering, therefore no matter how we change our models, we won't be able to establish any concrete results with insufficient bricks.

Another notable finding is that the genres are too "fine-grained". By simply scanning through the genres of all the artists, it is obvious that the user likes rock music. However, under this category, we have "album rock", "glam rock", "hard rock", "classic rock", "blues-rock"... When we use 0-1 vector representation for genre, the vector gets unnecessarily big, leading to the sparsity of data and even inaccurate results.



	artist	followers	popularity	plays	genres
84677	0383dadf-2a4e-4d10-a46a-e9e041da8eb3	2300827	83	82	[album rock, glam rock, rock]
358613	b665b768-0d83-4363-950c-31ed39317c15	398171	71	1234	[album rock, hard rock, rock]
492320	109958eb-a335-4c5e-907e-597ff4c6af46	754636	77	8	[classic rock]
817223	eeb1195b-f213-4ce1-b28c-8565211f8e43	1528928	80	627	[hard rock, rock]
998570	23a03e33-a603-404e-bcbf-2c00159d7067	822180	76	8	[]
1056689	28503ab7-8bf2-4666-a7bd-2644bfc7cb1d	270427	68	3	[progressive metal]
1080944	8f92558c-2baa-4758-8c38-615519e9deda	419000	71	3	[punk]
1104917	3d2b98e5-556f-4451-a3ff-c50ea18d57cb	879146	77	416	[album rock, hard rock, rock]
1278783	678d88b2-87b0-403b-b63d-5da7465aecc3	1362466	79	110	[album rock, blues-rock, classic rock, rock]
1318740	c3cceeed-3332-4cf0-8c4c-bbde425147b6	430796	70	28	[hard rock]
1367084	7dc8f5bd-9d0b-4087-9f73-dc164950bbd8	601138	71	33	[]
1452621	31745282-b1ea-4d62-939f-226b14d68e7c	280162	68	196	[melodic death metal]
1625952	070d193a-845c-479f-980e-bef15710653e	421805	51	4	[classic funk rock, funk rock]
2303323	ff865aa0-4603-4f79-ae8b-8735332e2cfa	237259	65	60	[christian alternative rock, christian rock]
2767327	8aa5b65a-5b3c-4029-92bf-47a544356934	424110	69	194	[]
2985557	4dbf5678-7a31-406a-abbe-232f8ac2cd63	421426	73	4	[]
3097217	534ee493-bfac-4575-a44a-0ae41e2c3fe4	301914	67	1110	[]
3783348	6ae51665-8261-4ae5-883f-1899651ad31b	47635	56	4	[alternative metal, industrial metal]
3966243	ccfe7a3c-1a45-4984-a8d0-644549cefe61	156789	65	12	[glam metal, hard rock]

Figure 5: Sample Artist Information. Here we focus on the correlation between (*followers*, *popularity*, *genres*) and *plays*

Models

Median vs. Mean per User

Since the per-user-median benchmark is quite competent, we concluded that we have to utilize per-user-median, and make subtle changes on it to propose improved model. Therefore, we tried an experiment by modifying the sample code in the part of taking median per user, into taking mean value instead.

Table 3: Median vs Mean

	Per-User-Median	Per-User-Mean
Kaggle score	137.79146	177.68353

From the Table 3, we realize that the Per-User-Mean approach gave us even worse result. Combined with Figure 5 (User in Figure 5 had range of from 4 to 1200), we realized that for most users, the number of plays has long tails (meaning the variance is large) and the bumps is skewed (or there are multiple bumps), leading to the huge disparity between Median and Mean values. This finding shed light on the fact that clustering models have better results than those which have prior assumption that *plays* follow normal distribution.

K-means and Similarity Matrix

We start with k-means algorithm, hoping to identify artist preference of each group. However, the result is not promising. Although we observe an improvement in accuracy score when we cluster users into 100 groups, the improvement is small. It does not justify that we should try models with more clusters. Two things may go wrong with this approach:

- First, the features we include are not good enough to allow us form good clusters. Users in the same cluster may not have similar preferences on artists.

- Another thing may go wrong is our method to make prediction of number of plays. As we have discussed in the Technical Approach, matrix U is very sparse. This implies, we may end up with Proportion Matrix with a lot of zeros or very small values. Thus, when we make predictions, we are more prone to underestimate the number of plays and not able to beat the User Median Benchmark.

Then, we move on to look at the similarities among artists, hoping to make predictions based on the similarities of artists. Searching through $K = 100$ nearest neighbors gives us a better result than searching through $K = 285$. Since we use user median when we are not able find the K nearest neighbor of the artist in interest, it means that it is not a good idea to use the number of plays of the > 100th nearest neighbors as our prediction. It also explains why user median has been perform well.

Random Forest and Bayesian Ridge Regression
As we can tell from Table 1, Random Forest and Bayesian Ridge Regression fail to give us satisfying result. The reasons are multi-fold:

- As discussed in the *feature engineering* part, our features are not informative enough to generate concrete training results;
- Since Random Forest and Bayesian Regression in scikit-learn only accept numerical format, we have to convert the string ID to index (i.e. 0,1,2... One-hot encoding doesn't work with 233286 users and 2000 artist, and PCA makes no sense); the handling of user ID and artist ID is problematic, but so far we haven't come up with any better solutions;
- For Bayesian Ridge Regression, with so many categorical features, the relationship between number of plays and all the features is very likely to be non-linear, meanwhile, we have already proved that the *plays* distribution is skewed and does not follow normal distribution, which renders the model to be full of flaws.

4 Future Improvement

Here is a list of things we wish to try if we are given more time:

- We read the recommended paper "MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS" and the approach seems very promising. Matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation in the paper, which is easy to be generalized to our case of predicting number of plays for music.
- Since the model that ultimately helps us to beat the user median benchmark is the one that simply multiplying the median with a factor that handles the skewed distribution, we are thinking that instead of applying one global factor to all users, we could come up with a model to characterise the distribution of each user's playtimes, based on which we could generate tailored factors for each user, hence further decrease MAE score.
- Judging from results, the similarity matrix for artist actually works pretty well. Since the similarities characterize artists' correlations, whereas user median focus on user's behavior, our intuition is that by trying out different number of nearest neighbors K (the smaller K is, the less artist feature weighs and the more user median contributes to the prediction), we could find the "sweet spot" between user and artist features, therefore produce more reliable results.