Music Recommendation

GR5291 Final Project
Instructor: Demissie Alemayehu

Group 22:
Jingyuan Bian (jb4076)
Sizhu Chen (sc4248)
Xiaohan Hu (xh2338)
Yang Chen (yc3335)
Ye Yue (yy2810)
Binhan Wang (bw2544)
Chenghao Yu (cy2475)
Mengjia Huang (mh3781)
Zehan Wang (zw2457)
Lingyi Zhao (lz2570)
Xi Zhang (xz2647)

Introduction & Motivation

Nowadays, the music library is growing rapidly, and so is its variety. Being able to push tracks which match users' tastes is playing an important role in the exploratory feature of music applications. In addition, the ability to successfully predict users' preference can enrich the users' profile database and provide insight on future development. Thus, as a group, our goal is to find efficient and accurate methods to recommend music to enhance user experience and to create business value.

We constructed three algorithms to reach our goal. The first one is a traditional memory-based algorithm which pushes similar tracks according to each user's listening history. The second one is clustering for users, figuring out the optimal size of the groups they have. The third algorithm we used is Alternating Least Squares Method (ALS) for Collaborative Filtering, exploiting other users and items along with their ratings and target user history to recommend an item that target user does not have ratings for.

Dataset

The datasets we will be using in this project is from KKBox, which is Asia's leading music streaming service, holding a library over 30 million tracks. After filtering and cleaning, our data contains 352,681 songs and 30,735 users. The essential variables contain user ID, song ID, genre, language, user age, artist, song length. Based on these fundamental variables, we compiled and extracted several other useful factors, such as the top five popular genre, the top three popular artists, and the top three common languages, which we computed through repeat times of listening.

EDA

First of all, we detected the dataset about users. There are outliers, such as negative values as well as values above 1000, in the age field. Meanwhile, there are 19932 records with 0 as age; this could be either outliers or missing values. After removing these values, we plotted in the age range 1 -100 to show the real distribution. Figure 1 illustrates that the age distribution is skewed right, the mean

age of user is around 35 years old. Though there is a lot of missing gender, Male and female are almost equal.

Secondly, we dig information from the dataset about songs. From figure 2, which is created from Tableau, we can see that the songs in our train dataset mainly distributed in Asia and US. In other words, Chinese songs and English songs account for most of the songs. In addition, the pie chart under figure 3 shows the most popular genres; genre 465 has preponderant influence over the song dataset. Moreover, from the word-cloud in figure 4, we found out Jay Chow and May Day, the mainstays of Chinese Pop music, are the artists whose songs being listened the most repeat times. Thus we can infer this music application is used in China. Meanwhile, in order to make sure whether there is influence of song length on users' preference, we plot the distribution of song length. Figure 5 shows the song length is mainly concentrated around 4 minutes, which is a normal length.

At the same time, in order to explore the relationship between genre and songs, we plot the Top 5 Repeatedly Listened Songs Distribution over Genres (figure 6), and discovered that the genre which includes more songs does not mean it will be repeatedly listened more times. Constructing the bridge between datasets of users and songs, we also pay attention to extract the correlations. From the plot about relationship between users' age and genres they listened (figure 8), we found it is make sense that the age distribution is wider when the genre is more popular. The most popular genre is genre 465, and the age distribution of this genre is the widest. Meanwhile, in order to dig the connection the number of genre and the number of songs he or she listened, we created the scatter plot in terms of different age group (figure 9), and then we found there is a positive correlation between the number of songs and genres listened among every age group.

Algorithms

Baseline

For benchmarking, we write a random algorithm as our baseline. Firstly, we create a song pool that contains all the songs according to their repeatedly listening probabilities as a measure of popularity. If target=0, this song will be included in the song pool once. If target=1, this song will be included in the song pool twice. The more times this song was repeatedly listened, the larger the probabilities this

song would be recommended to the user.

Then we pick up 10 songs from the song pool, which follows a multinomial distribution using relative popularity as parameter. Finally, we calculate the test error to evaluate the baseline algorithm. If the song was not listened or listened just once, we score it 0. If the song was listened and listened more than once, we score it 1. The test accuracy for baseline is 0.0131.

Content-based

The first algorithm we implemented is memory-based. The idea is to create a profile for each user and recommend tracks based on it. In this naive approach, we take genre and language as two features to create a user's profile. The flow of this approach is as follows:

1. Extract all songs a user has listened in the training set.
2. Create a 2-way contingency table of listening history to find the top (language, genre) pair for this user.
3. Filter the music library to find the songs which haven't been listened by this user and are in the top (language, genre) group.
4. Recommend tracks from the filtered pool based on popularity.

Data cleaning:

This algorithm depends the history record of each user, so we want each user to appear in both training and test data sets. Thus, we separate each user's observations into training set and test set with split ratio 7:3. We plan to recommend 10 songs for each user. To be able to evaluate our algorithm's performance, we have to make sure that each user in the test set has at least 10 observations. So we take the top 21000 users according to the number of songs they listened (i.e. the top 21000 users have at least 10 listening records in the test set).

Also, some songs are sorted to multiple genres. To get a more accurate top (language, genre) pair, we expand those observations with multiple-genres songs into multiple observations. For example:

Before expansion:

song_id	target	artist_name	genre_ids
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993|751
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993|751
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993|751

After expansion:

song_id	target	artist_name	genre_ids
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	751
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	751
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	993
kAEMO7m3E9JFVghjtZcj9Zrzb37OJTRj0Ocq8d0hqSg=	0	Enrico Fagone| Walter Zagato| Corrado Giuffredi| Orch...	751

Here are the results with different sized user base:

Number of users	100	1000	5000	21000
Accuracy	0.605	0.470	0.379	0.215

This algorithm's logic is straightforward yet naive. It has its limitation. It highly depends on the user's history information to make a prediction. Hence, it cannot serve the new app users since they have zero listening record. Also, the more songs a user listened, the more accurate our recommendation could be. As we can observe from the table above, the performance is getting worse while we add more users with fewer records to our user base. As we conclude all viable users, the accuracy of this algorithm is about 21.5%.

Clustering

Besides the traditional method stated above we also use another recommendation system for the new music users.

First of all, we want to do the clustering for members and figuring out the optimal size of the groups they have. According to the data we have numerical data (ages, mean of the song lengths, the repeat ratio of the songs ID and the repeat ratio of the Genre ID) and text feature (language, artists information, song information

and etc.). For the numeric data we use K-mean and Birch clustering algorithm to do the clustering (figure 10). For the text feature part we decide to use LDA topic clustering to finish the clustering.

Secondly, the tricky part is how to combine these two algorithms together. Thanks for the essay from the Cornell Library we can use a LDA_Kmeans method to handle with numeric and text feature data based on the previous data.

The object similarity measure is derived from both numeric and categorical attributes. When applied to numeric data the algorithm is identical to k-means. In testing with real data, this algorithm has demonstrated a capability of partitioning data sets in the range of a hundred thousand records, described by more than 20 numeric and categorical attributes, into a 21 clusters in a couple of hours. We use K-prototype to combine two clustering algorithms together. The algorithm is built upon three processes, the initial prototypes selection, initial allocation, and re-allocation.

$$E_l = \sum^{n} y_{il} \sum^{m_r} (x_{ij}^r - q_{lj}^r)^2 + \gamma_l \sum^{n} y_{il} \sum^{m_c} \delta(x_{ij}^c, q_{lj}^c)$$

$$E_l = E_l^r + E_l^c$$

We set clusters and find out the optimal K values for the clusters(figure 11). The optimal cluster number is 21 groups for the users.

After the clustering members and users we finally enter into the recommendation system for both old users based on the previous history and clusters information. For the new users, we might not do the prediction because we did not know their clusters, hence for the new enrolled users, they need to become ‘old’ user first in order to recommend applicable songs for them.

Below figure is the result of our recommendation we have three column inside, the first column is that how many ID in that specific cluster group; the second column ‘All’ means the mean of the accuracy rate of users who did listen to the recommendation songs, ‘Target’ is the mean of the accuracy rate of users who listen the recommendation songs multiple times. We get several information from the data, firstly, NO.9 cluster group have low accuracy because there are only 9 users in the group, but for most groups we have roughly accuracy between 25%-33% which means among four or three recommendation songs they will definitely choose one song as their favorite types. And among recommend songs they choose 25% of the songs they will listen again.

	Number of ID in Group	All	Target
1	189	0.312169312169312	0.232804232804233
2	206	0.280097087378641	0.216019417475728
3	2348	0.289437819420784	0.226192504258944
4	3801	0.287213891081294	0.226098395159169
5	393	0.285241730279898	0.223155216284987
6	3019	0.292050347797284	0.228287512421332
7	253	0.288932806324111	0.21699604743083
8	1369	0.2836376917458	0.222717311906501
9	9	0.133333333333333	0.111111111111111
10	981	0.311213047910296	0.246279306829766
11	1006	0.301888667992048	0.237574552683897
12	434	0.285483870967742	0.23110599078341
13	224	0.288392857142857	0.233928571428571
14	949	0.299894625922023	0.240463645943098
15	574	0.270557491289199	0.205052264808362
16	1834	0.285169029443839	0.221537622682661
17	40	0.2725	0.2075
18	1229	0.295850284784378	0.234906427990236
19	2248	0.294350533807829	0.236076512455516
20	359	0.282729805013928	0.22116991643454
21	219	0.299543378995434	0.236986301369863

That works pretty good, firstly, we use over 30 million data to test the accuracy; secondly, from my personal experience of the music app, every three or four recommended songs there is one song which fit your tastes better than most of the app music recommendation system. However, if we want to furthermore increase our accuracy we need to have more comprehensive data rather than asian users. For instance, NO.9 group might be the music member who only interested in some Indian songs, which might have a bias because of less sample here.

Overall, the results of our recommendation work somewhat efficiently and inexpensive. But if we want to improve more on the recommendation system, more comprehensive data will be required.

Alternating Least Squares (ALS)

The third algorithm we used is Alternating Least Squares Method(ALS) for

Collaborative Filtering.

Collaborative Filtering(CF) is a subset of algorithms that exploit other users and items along with their ratings and target user history to recommend an item that target user does not have ratings for. Fundamental assumption behind this approach is that other users preference over the items could be used recommending an item to the user who did not see the item before. CF differs itself from content-based methods in the sense that user or the item itself does not play a role in recommendation but rather how and which user rated a particular item. (Preference of users is shared through a group of users who selected, purchased or rate items similarly).

- **Alternating Least Square Formulation for Recommender Systems**

  We have users *u* for songs *i* matrix as in the following:
  Qui = 1 *if song i is repeatedly listened by user i*,

  Qui = 0 *if song i is listened but not repeatedly listened by user i*.

Qui is our rating matrix. If we have m users and n songs, then we want to learn a matrix of factors which represent songs. That is, the factor vector for each song and that would be how we represent the song in the feature space. We also want to learn a factor vector for each user in a similar way how we represent the song. Factor matrix for songs $Y \in R^{f \times n}$ and factor matrix for users $X \in R^{m \times f}$. However, we have two unknown variables. Therefore, we will adopt an alternating least squares approach with regularization. By doing so, we first estimate Y using X and estimate X by using Y. After enough number of iterations, we are aiming to reach a convergence point where either the matrices X and Y are no longer changing or the change is quite small. However, there is a small problem in the data. We have neither full user data nor full songs data, this is also why we are trying to build the recommendation engine in the first place. Therefore, we may want to penalize the songs that do not have ratings in the update rule. By doing so, we will depend on only the songs that have ratings from the users and do not make any assumption around the songs that are not rated in the recommendation. Let's call this weight matrix w_{ui} *as such* : *if* $q_{ui} = 0$, $w_{ui} = 0$; *otherwise*, $w_{ui} = 1$. Then, cost functions that we are trying to minimize is in the following:

$$J(x_u) = (q_u - x_u Y) W_u (q_u - x_u Y)^T + \lambda x_u x_u^T$$

$$J(y_i) = (q_i - X y_i) W_i (q_i - X y_i)^T + \lambda y_i y_i^T$$

Note that we need regularization terms in order to avoid the overfitting the data.

Ideally, regularization parameters need to be tuned using cross-validation in the dataset for algorithm to generalize better. Solutions for factor vectors are given as follows:

$$x_u = (Y W_u Y^T + \lambda I)^{-1} Y W_u q_u$$

$$y_i = (X^T W_i X + \lambda I)^{-1} X^T W_i q_i$$

where $W_u \in R^{n \times n}$ and $W_u \in R^{m \times m}$ diagonal matrices. Then, we can get the rating matrix between users and songs.

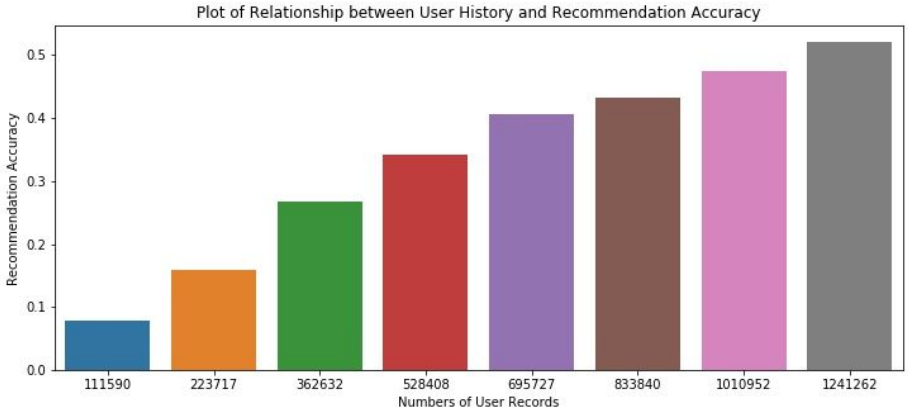
- **Implementation**

We implemented this algorithm using ALS package in MLlib of Pyspark, and used cross validation to select model parameters like the number of latent factors we included, the regularization parameters, etc.

After getting the full rating matrix, we will recommend the top 10 songs which have the highest rating scores to every user. We notice that these songs are included both in training set and testing set, so we did result processing to finally recommended top 10 songs excluding the training set. This part introduced big data volume, so this is why we choose pyspark, which is better at dealing with big data.

Our final result of test error is 0.6775.

From the plot, we can find that there is positive correlation between the numbers of user records and recommendation accuracy.



Summary

Overall, all our algorithms performed better than the baseline random model.

The accuracy drops as we increase the number of users in our data sets.

The content-based algorithm have an accuracy rate of 0.215, which means about 22% of recommended songs suit users' tastes for a user base that is sufficiently large. This algorithm has the lowest accuracy among all three built by us, and it fails to make predictions for new users. Hence, this should not be the final solution. However, it can be considered as an option for the users who listened a lot more songs than others, as the evidence suggests that this algorithm has a fair performance for them.

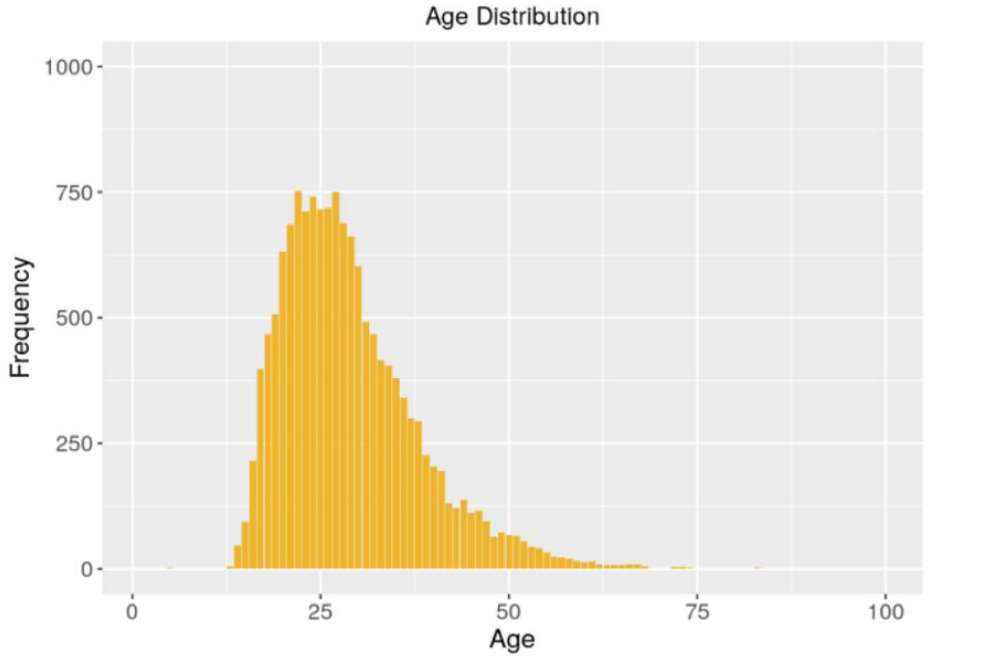
For the clustering method, our accuracy will be roughly around 0.2 to 0.3 for different groups with a weighted mean accuracy rate 0.291, which means among three to four recommended songs there will be one fit users' choices. Looks like this method performance quite well. However if we want to have more higher accuracy we need to consider more comprehensive data and broder cluster information.

For the ALS method, we have better performance, the final accuracy is 0.323. Since ALS is used to deal with the big data, we believe for large datasets it will perform better than most of the methods and algorithms.

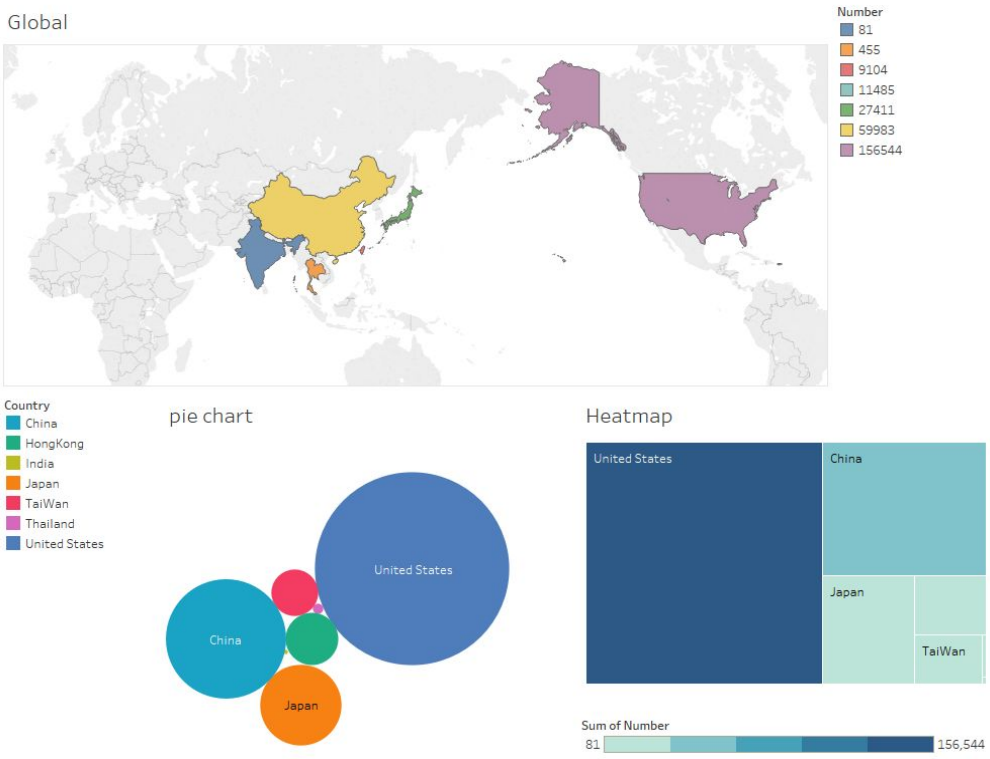
Appendix

Figure

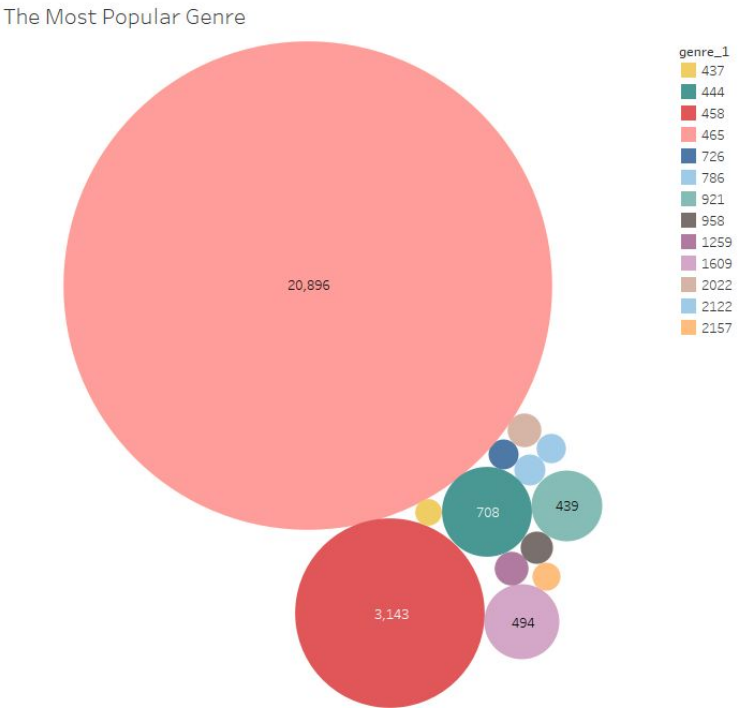
(Figure 1, User Age Distribution)



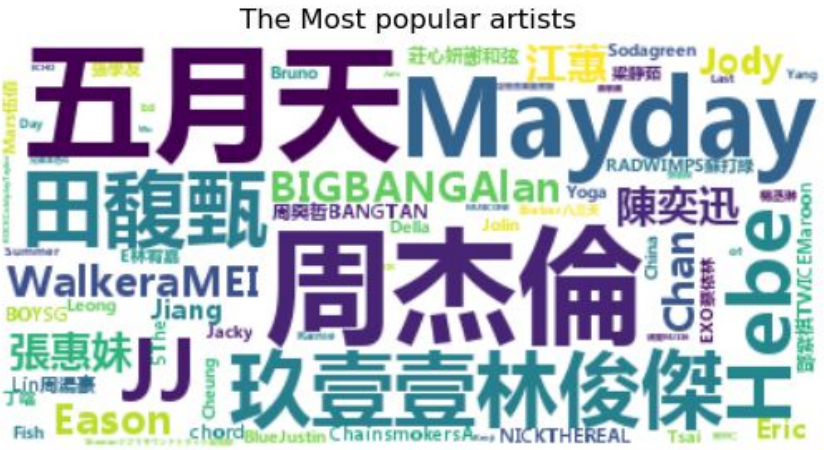
(Figure 2, Song Geometric Distribution Plot)



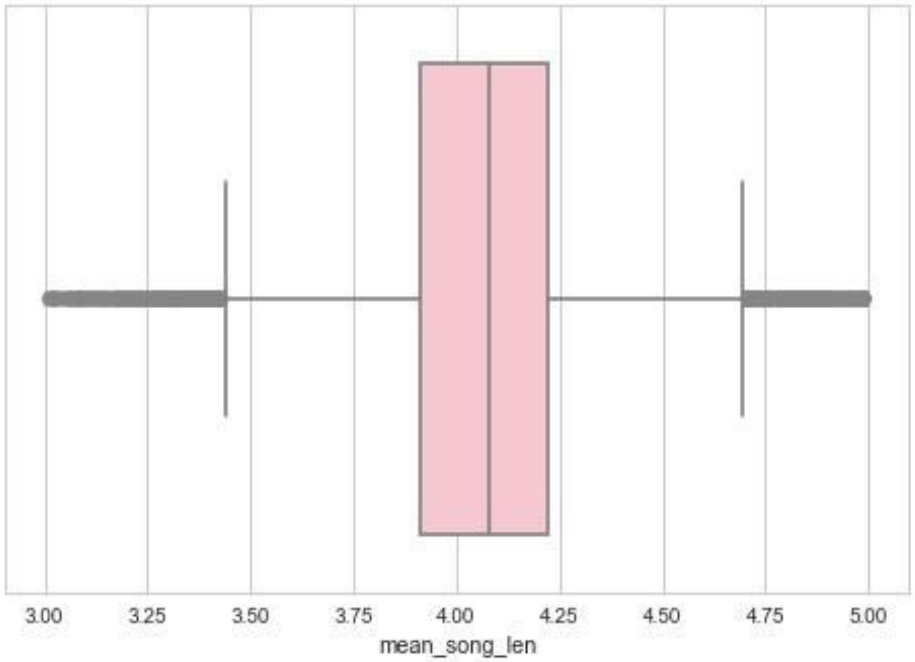
(Figure 3, The Most Popular Genres)



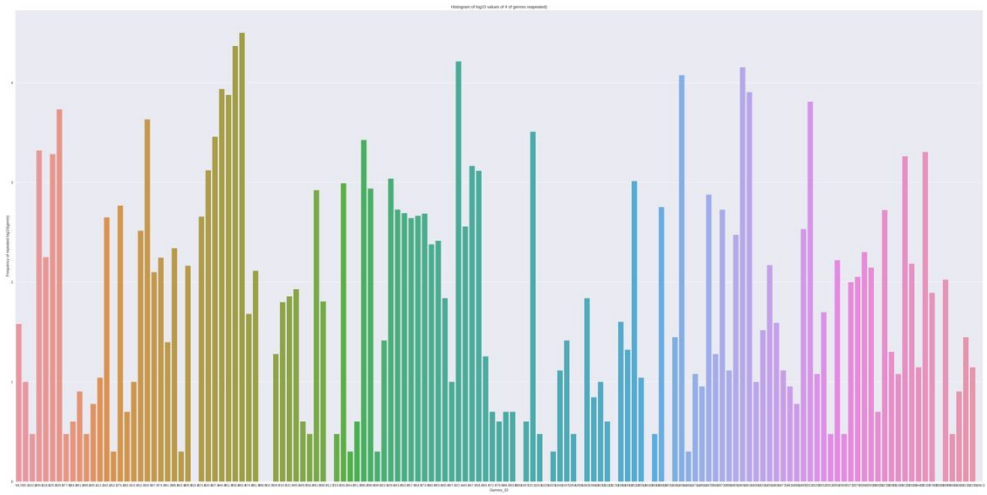
(Figure 4, The Most Popular Artists)



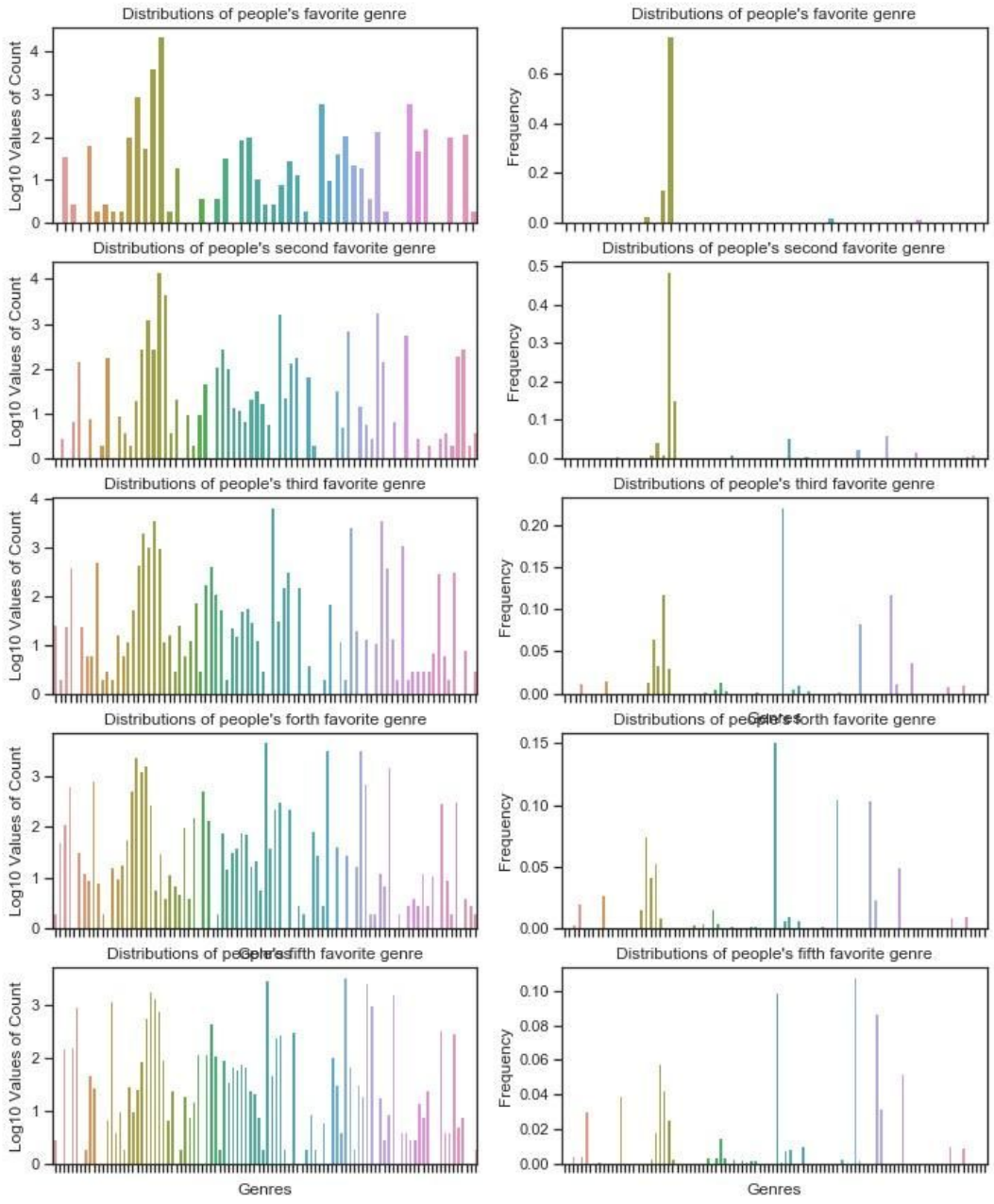
(Figure 5, Boxplot of Song Length)



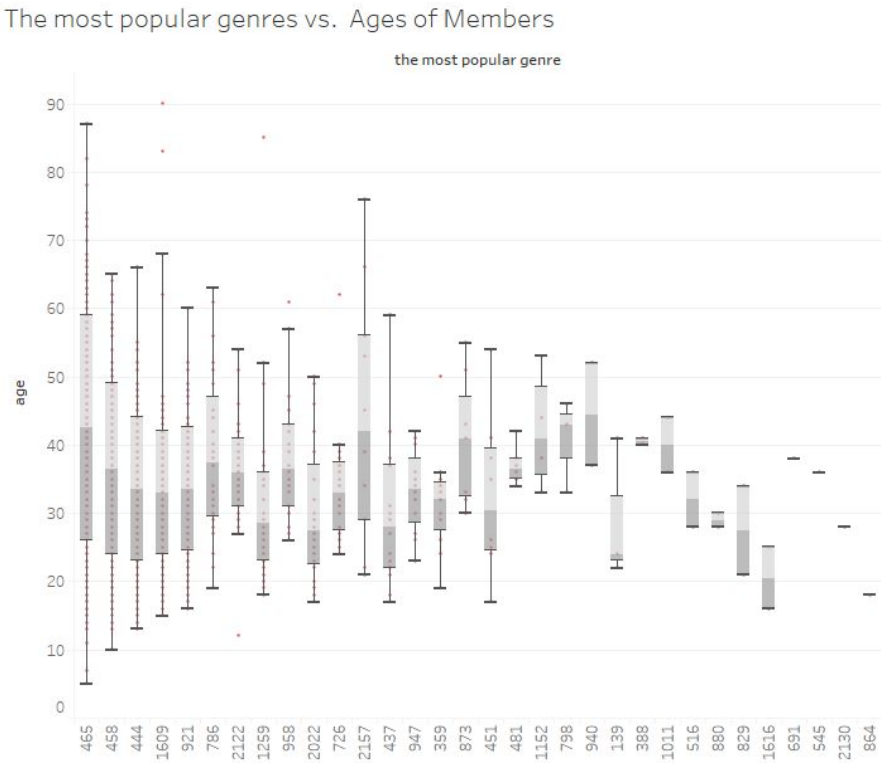
(Figure 6, Top 5 Repeatedly Listened Songs Distribution over Genres)



(Figure 7, Top 5 Popular Songs Distribution)

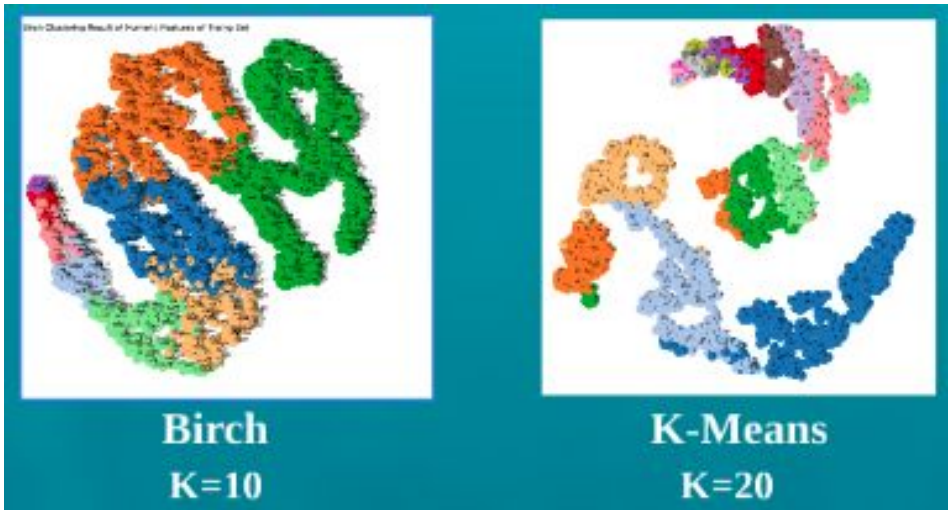


(Figure 8, Genre vs. Age)

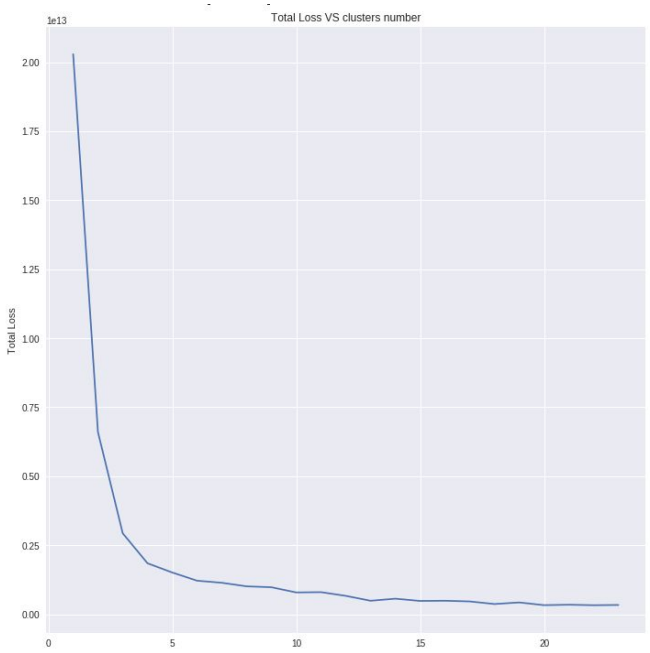


(Figure 9, Number of Songs/Genres listened vs. Age)

(Figure 10, K-mean and Birch clustering)



(Figure 11, LDA_Kmeans K-prototype optimal K clusters)




```
[ ]  np.argmin((ncost))

     21
```

Code:

Content-based:

```
# This file contains the source code of the content-based approach of our music
recommendation
# algorithm.
# Author: Sizhu Chen, Ye Yue, Lingyi Zhao, Binhan Wang

library(dplyr)
library(tidyr)
library(stringr)


##### Load existing train and test data; combine them so that we can split them
later
training<-read.csv("training.csv",stringsAsFactors = F)
testing<-read.csv("testing.csv",stringsAsFactors = F)
total<-rbind(training, testing)

total<-total[-c(3,4,5,10:16)] # drop unwanted variables
# save(total, file="total.RData") # this line is used to store data locally
################################################################################



##### Extract the users with the most listening history
usercount<-count(total, vars=total$msno)

top_10000<-function(df, col){
  temp_df<-df %>%
    arrange(desc(n))%>%
    print
  return(temp_df)
}

# To change userbase size, edit the next line
vvtop<-top_10000(usercount, usercount$n)[1:21000,]
# get the first 21000 users since the other 9000 have fewer songs listened

merge<-subset(total, msno %in% vvtop$vars)
################################################################################

##### Create music library
songpool<-count(total, var=song_id)
names(songpool)<-c("song_id","fre")
```

```r
total_pool<-total[,c(2,5,6)]
total_pool<-total_pool[!duplicated(total_pool),]# nrow=nrow(songpool)
total_pool<-merge(total_pool,songpool,by="song_id")

#expand total_pool by genre

cot1<-rep(NA,nrow(total_pool))
for (i in 1:nrow(total_pool)){
  cot1[i]<-str_count(total_pool$genre_ids[i],"\\|")+1
}
total_pool_expand<- total_pool[rep(row.names(total_pool), cot1),]

genre_split1<-unlist(strsplit(total_pool$genre_ids,"\\|"))
total_pool_expand$genre_ids<-genre_split1
total_pool<-total_pool_expand

total_pool<-total_pool[order(total_pool$fre,decreasing = T),]
##############################################################################

##### Create training and test set
library(parallel)
library(doParallel)
registerDoParallel(detectCores())

vvtop$train_num<-round(vvtop$n*0.7)

train1 <- foreach(i = 1:nrow(vvtop)) %dopar% {
  train1<-data.frame()
  each_user<-subset(merge,msno==vvtop$vars[i])
  set.seed(2018)
  trainset<-each_user[sample(1:nrow(each_user),size = vvtop$train_num[i]),]
}

library(plyr)
train1 <- ldply(train1)

test<-setdiff(merge, train1)

#save(train1, file = "../train21000.RData")
#save(test, file = "../test21000.RData")
##############################################################################

##### Expand multiple genres in training set
cot2<-rep(NA,nrow(train1))
for (i in 1:nrow(train1)){
  cot2[i]<-str_count(train1$genre_ids[i],"\\|")+1
}
```

```r
train_expanded <- train1[rep(row.names(train1), cot2),]
train_expanded$genre_ids<-unlist(strsplit(train1$genre_ids,"\\|"))
trainset<-train_expanded
################################################################################

##### Create user profile
# write a function to get (genre, language) pair for each user
getpair <- function(df) {
  ct <- table(df$language, df$genre_ids)
  lname <- rownames(ct) # language id
  gname <- colnames(ct) # genre id
  m <- max(ct)
  idx <- which(ct == m, arr.ind = T)
  l <- lname[idx[1]]
  g <- gname[idx[2]]
  return(c(l, g))
}

# top pair for each user
feature21 <- foreach(u = vvtop$vars) %dopar% {
  user <- u
  topl <- getpair(trainset[trainset$msno == u, ])[1]
  topg <- getpair(trainset[trainset$msno == u, ])[2]
  print(c(user, topl, topg))
}

feature21 <- ldply(feature21)
colnames(feature21) <- c("user", "language", "genre")
#save(feature21, file = "../feature21000.RData")
################################################################################

##### Make recommendation
recom_table <- foreach (g = feature21$user) %dopar% {
  recom_set<-subset(total_pool,
                    !(song_id %in% train1$song_id[train1$msno==g]))
  recommend<-subset(recom_set,
                    recom_set$genre_ids == feature21$genre[feature21$user==g] &
                      recom_set$language == feature21$language[feature21$user==g])


  if(nrow(recommend)==0){
    recommend<-recom_set$song_id[1:10]
  }
  else{
    recommend<-recommend$song_id[1:10]}
```

```
  print(data.frame(users=rep(g,10),recom_music=recommend))

}

recom_table <- ldply(recom_table)
################################################################################

##### Evaluate performance
# Only count repeating songs as success
test_target1<-test[test$target==1,]

# get performance score for each user
score21 <- foreach (p = unique(recom_table$users)) %dopar% {
  testsong<-test_target1$song_id[test_target1$msno==p]
  score<-sum(testsong %in% recom_table$recom_music[recom_table$users==p])
}
# Transform into overall percentage
sum(unlist(score21))/(10*nrow(vvtop))
```

Clustering:
K-means:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca.fit(num_mat)
X = pca.transform(num_mat)
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Initializing KMeans
kmeans = KMeans(n_clusters=20)
# Fitting with inputs
kmeans = kmeans.fit(X)
# Predicting the clusters
labels = kmeans.predict(X)
# Getting the cluster centers
C = kmeans.cluster_centers_
import pyclustering
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer
from pyclustering.cluster.xmeans import xmeans
```

```
from pyclustering.utils import draw_clusters
initial_centers = kmeans_plusplus_initializer(X, 2).initialize();
xmeans_instance = xmeans(X, initial_centers, ccore = True);
# run cluster analysis
xmeans_instance.process();
# obtain results of clustering
clusters = xmeans_instance.get_clusters();
# display allocated clusters
draw_clusters(X, clusters)
```

Birch_clusters:

LDA:

```
n<- dim(feature_data_text_1)[1]
fd<- array(NA, c(n, 2))
fd[,1]<- feature_data_text_1$msno
fd[,2]<- apply(feature_data_text_1[,c(2,3,6:15)], 1, paste, sep=" ", collapse="
")
library(tm)
library(topicmodels)
fdcorpus<- VCorpus(VectorSource(fd[,2]))
DTM <- DocumentTermMatrix(fdcorpus)
TM <- LDA(DTM, 10, method="Gibbs", control=list(iter = 500, seed = 1234))
terms(TM, 10)
t_f<- as.factor(topics(TM))
which.max(posterior(TM, DTM[2,])$topics)
```

LDA_Kmeans(brief verison, more detialed please see URL):

https://github.com/ElinaBian/Music_Recommendation/blob/master/Algorithm_based_on_clustering/LDA_Kmeans/LDA_Kmeans_k_prototypes.ipynb

```
from collections import defaultdict
import numpy as np
from scipy import sparse
from sklearn.base import BaseEstimator, ClusterMixin
from sklearn.externals.joblib import Parallel, delayed
from sklearn.utils import check_random_state
from sklearn.utils.validation import check_array

def k_prototypes(X, categorical, n_clusters, max_iter, num_dissim, cat_dissim,
                 gamma, init, n_init, verbose, random_state, n_jobs):
    """k-prototypes algorithm"""
    random_state = check_random_state(random_state)
```

```
    if sparse.issparse(X):
        raise TypeError("k-prototypes does not support sparse data.")
    if 'pandas' in str(X.__class__):
        X = X.values

    if categorical is None or not categorical:
        raise NotImplementedError(
            "No categorical data selected, effectively doing k-means. "
            "Present a list of categorical columns, or use scikit-learn's "
            "KMeans instead."
        )
    if isinstance(categorical, int):
        categorical = [categorical]
    assert len(categorical) != X.shape[1], \
        "All columns are categorical, use k-modes instead of k-prototypes."
    assert max(categorical) < X.shape[1], \
        "Categorical index larger than number of columns."

    ncatattrs = len(categorical)
    nnumattrs = X.shape[1] - ncatattrs
    n_points = X.shape[0]
    assert n_clusters <= n_points, "Cannot have more clusters ({}) " \
                                   "than data points ({}).".format(n_clusters,
n_points)

    Xnum, Xcat = _split_num_cat(X, categorical)
    Xnum, Xcat = check_array(Xnum), check_array(Xcat, dtype=None)
    Xcat, enc_map = encode_features(Xcat)
    unique = get_unique_rows(X)
    n_unique = unique.shape[0]
    if n_unique <= n_clusters:
        max_iter = 0
        n_init = 1
        n_clusters = n_unique
        init = list(_split_num_cat(unique, categorical))
        init[1], _ = encode_features(init[1], enc_map)
    if gamma is None:
        gamma = 0.5 * Xnum.std()

    results = []
    seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
    if n_jobs == 1:
        for init_no in range(n_init):
            results.append(k_prototypes_single(Xnum, Xcat, nnumattrs,
```

```
ncatattrs,
                                                n_clusters, n_points, max_iter,
                                                num_dissim, cat_dissim, gamma,
                                                init, init_no, verbose,
seeds[init_no]))
    else:
        results = Parallel(n_jobs=n_jobs, verbose=0)(
            delayed(k_prototypes_single)(Xnum, Xcat, nnumattrs, ncatattrs,
                                         n_clusters, n_points, max_iter,
                                         num_dissim, cat_dissim, gamma,
                                         init, init_no, verbose, seed)
            for init_no, seed in enumerate(seeds))
    all_centroids, all_labels, all_costs, all_n_iters = zip(*results)

    best = np.argmin(all_costs)
    if n_init > 1 and verbose:
        print("Best run was number {}".format(best + 1))

    # Note: return gamma in case it was automatically determined.
    return all_centroids[best], enc_map, all_labels[best], \
        all_costs[best], all_n_iters[best], gamma
class KPrototypes(KModes):

    def __init__(self, n_clusters=8, max_iter=100, num_dissim=euclidean_dissim,
                 cat_dissim=matching_dissim, init='Huang', n_init=10,
gamma=None,
                 verbose=0, random_state=None, n_jobs=1):

        super(KPrototypes, self).__init__(n_clusters, max_iter, cat_dissim,
init,
                                          verbose=verbose,
random_state=random_state,
                                          n_jobs=n_jobs)
        self.num_dissim = num_dissim
        self.gamma = gamma
        self.n_init = n_init
        if isinstance(self.init, list) and self.n_init > 1:
            if self.verbose:
                print("Initialization method is deterministic. "
                      "Setting n_init to 1.")
            self.n_init = 1

    def fit(self, X, y=None, categorical=None):
```

```
        random_state = check_random_state(self.random_state)
             self._enc_cluster_centroids, self._enc_map, self.labels_,
self.cost_,\
            self.n_iter_, self.gamma = k_prototypes(X,
categorical,self.n_clusters,self.max_iter,self.num_dissim,self.cat_dissim,
self.gamma, self.init, self.n_init,self.verbose,random_state,self.n_jobs)
        return self

    def predict(self, X, categorical=None):
          assert hasattr(self, '_enc_cluster_centroids'), "Model not yet
fitted."

        Xnum, Xcat = _split_num_cat(X, categorical)
        Xnum, Xcat = check_array(Xnum), check_array(Xcat, dtype=None)
        Xcat, _ = encode_features(Xcat, enc_map=self._enc_map)
        return _labels_cost(Xnum, Xcat, self._enc_cluster_centroids,
                            self.num_dissim, self.cat_dissim, self.gamma)[0]

    def cluster_centroids_(self):
        if hasattr(self, '_enc_cluster_centroids'):
            return [
                self._enc_cluster_centroids[0],
                decode_centroids(self._enc_cluster_centroids[1], self._enc_map)
            ]
        else:
            raise AttributeError("'{}' object has no attribute
'cluster_centroids_' "
                                 "because the model is not yet fitted.")
import pandas as pd
import numpy as np
from operator import methodcaller
data_num= pd.read_csv('drive/5291/feature_data_numeric_1.csv')
data_text=pd.read_csv('drive/5291/feature_data_text_1.csv')
data_result = pd.merge(data_num, data_text, on='msno', how='outer').dropna()
# feature_data_numeric_1.csv
df=data_result
del df['msno'] ;del df['artist_name1'] ;del df['artist_name2']
syms = np.unique(data_result.msno)
X = df.values
X[:, 0] = X[:, 0].astype(float)
ncost=[]

for i in range(1,31):
```

```
  kproto = KPrototypes(n_clusters=i, init='Cao', verbose=2)
  clusters = kproto.fit_predict(X, categorical=[1, 2])
   ncost.append(kproto.cost_)

for s, c in zip(syms, clusters):
    print("Symbol: {}, cluster:{}".format(s, c))
```

Classification:

```
ldak0<- ldak[ldak$cluster_id==0, ]
ldak0_song<- total[is.element(total$msno, ldak0$msno),]
train0<- data.frame(); test0<- data.frame()
for (i in 1:nrow(ldak0)){
tot<- sum(ldak0_song$msno==ldak0$msno[i])
ldak0$song[i]<- tot
mid_train<- subset(ldak0_song,msno==ldak0$msno[i])
set.seed(2018)
train0<- rbind(train0, mid_train[sample(1:tot, round(tot*0.7)), ])}
test0<- setdiff(ldak0_song, train0)
P_song<- data.frame(matrix(0,length(unique(train0$song_id)),2))
P_song[,1]<- unique(train0$song_id)
for (i in 1:length(unique(train0$song_id))){
 P_song[i,2]<- sum(train0$target[train0$song_id==P_song[i,1]]+1) }
score1<- rep(NA, nrow(ldak0))
score2<- rep(NA, nrow(ldak0))
for (i in 1:nrow(ldak0)){   new_song<- subset(P_song, !(X1 %in%
train0$song_id[train0$msno==ldak0$msno[i]]))
rec<- new_song[order(new_song[,2], decreasing = T),][1:10, ]
test_song1<- test0[test0$msno==ldak0$msno[i], ]
test_song2<- test_song1[test_song1$target==1, ]
score1[i]<- sum(rec[,1] %in% test_song1$song_id)/10
score2[i]<- sum(rec[,1] %in% test_song2$song_id)/10 }
mean(score1) mean(score2)

group_score1<- rep(NA, 21) group_score2<- rep(NA, 21)
 for (j in 0:20){   ldak0<- ldak[ldak$cluster_id==j, ]
ldak0_song<- total[is.element(total$msno, ldak0$msno),]
train0<- data.frame()   test0<- data.frame()
for (i in 1:nrow(ldak0)){     tot<- sum(ldak0_song$msno==ldak0$msno[i])
 ldak0$song[i]<- tot
 mid_train<- subset(ldak0_song,msno==ldak0$msno[i])
set.seed(2018)
train0<- rbind(train0, mid_train[sample(1:tot, round(tot*0.7)), ])}
test0<- setdiff(ldak0_song, train0)
```

```
P_song<- data.frame(matrix(0,length(unique(train0$song_id)),2))
 P_song[,1]<- unique(train0$song_id)
for (i in 1:length(unique(train0$song_id))){
P_song[i,2]<- sum(train0$target[train0$song_id==P_song[i,1]]+1)   }
 score1<- rep(NA, nrow(ldak0))   score2<- rep(NA, nrow(ldak0))
for (i in 1:nrow(ldak0)){
new_song<- subset(P_song, !(X1 %in%
train0$song_id[train0$msno==ldak0$msno[i]]))     rec<-
new_song[order(new_song[,2], decreasing = T),][1:10, ]
test_song1<- test0[test0$msno==ldak0$msno[i], ]
test_song2<- test_song1[test_song1$target==1, ]
 score1[i]<- sum(rec[,1] %in% test_song1$song_id)/10
score2[i]<- sum(rec[,1] %in% test_song2$song_id)/10   }
group_score1[j+1]<-mean(score1)   group_score2[j+1]<-mean(score2)   print(j) }
mean(group_score1) mean(group_score2)
sum(table(ldak$cluster_id)*group_score1)/sum(table(ldak$cluster_id))
sum(table(ldak$cluster_id)*group_score2)/sum(table(ldak$cluster_id))
Error<- matrix(NA, 21,3)
Error[,1]<- table(ldak$cluster_id)
Error[,2]<- group_score1
Error[,3]<- group_score2
colnames(Error)<- c("Number of ID in Group", "All", "Target")
write.csv(Error, file = "Error.csv")
```

ALS:

```
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.regression import LinearRegression
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.classification import RandomForestClassifier
from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark import SQLContext
sc = SparkContext.getOrCreate()
sqlContext = SQLContext(sc)
import pandas as pd
import time
df_train = pd.read_csv("training.csv")
```

```
df_test = pd.read_csv("testing.csv")
df_train['set'] = [1]*df_train.shape[0]
df_test['set'] = [0]*df_test.shape[0]
df = pd.concat([df_train, df_test])
df =
df.drop(['source_system_tab','source_screen_name','source_type','artist_name',

'genre_ids','language','song_length','bd','city','gender','registered_via',
              'registration_init_time','expiration_date'], axis=1)
df.msno = pd.Categorical(df.msno)
df['msno_int'] = df.msno.cat.codes
df.song_id = pd.Categorical(df.song_id)
df['song_id_int'] = df.song_id.cat.codes
df_id_pair = df.drop(['target','set'], axis=1)
df_id_pair.to_csv('strid_intid.csv', index=False)
df_test_dic = df.loc[df['set']==0]
df_test_dic.to_csv("testing_dic.csv", index=False)
df = df.drop(['msno','song_id'], axis=1)
df.to_csv('ALS.csv', header=False, index=False)
df_train = df.loc[df['set']==1]
df_test = df.loc[df['set']==0]
df_train = df_train.drop(['set'],axis=1)
df_test = df_test.drop(['set'],axis=1)
df_train.to_csv('ALStrain.csv', header = False)
df_test.to_csv('ALStest.csv', header = False)

import findspark
findspark.init()

# import ALS and Linear Regression models
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
#from pyspark.ml.recommendation import ALS
from pyspark.mllib.recommendation import ALS
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.regression import LinearRegression
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.classification import RandomForestClassifier
from pyspark.sql import Row
from pyspark.sql import SparkSession
from time import time
from pyspark import SparkContext
```

```
from pyspark import SQLContext
sc = SparkContext.getOrCreate()
sqlContext = SQLContext(sc)
import pandas as pd
import math
# Build a SparkSession; SparkSession provides a single point of entry to
interact with underlying Spark functionality
spark = SparkSession\
       .builder\
       .appName("ALSExample")\
.getOrCreate()
# Load data as RDD, then transform it to DataFrame format
lines = spark.read.text("ALStrain.csv").rdd
parts = lines.map(lambda row: row.value.split(","))
trainingRDD = parts.map(lambda p: (int(p[0]), int(p[1]),float(p[2])))
training_RDD, validation_RDD = trainingRDD.randomSplit([7, 3], seed=520)
validation_for_predict_RDD = validation_RDD.map(lambda x: (x[0], x[1]))
model = ALS.trainImplicit(trainingRDD, rank=26, seed=520, iterations=10,
lambda_=0.001, alpha=0.01)
df_train = pd.read_csv("ALStrain.csv", header=None)
df_test = pd.read_csv("ALStest.csv", header=None)
df_test = df_test.loc[df_test.iloc[:,2]==1]
#df = pd.read_csv("ALSc.csv", header=None)
ID_in_testset = df_test.iloc[:,0].drop_duplicates().tolist()
ID_in_trainset = df_train.iloc[:,0].drop_duplicates().tolist()
ID_in_test_train_set = [x for x in ID_in_testset if x in ID_in_trainset]
len(ID_in_test_train_set)
total_score = 0
for i in range(15000, 18000):
       #print(user_ID)
       #user_ID_idx_needmorerecommend = []
       user_ID = ID_in_test_train_set[i]
       user_train_recorder =
df_train.loc[df_train.iloc[:,0]==user_ID,1].tolist()
       #Recommend_number_max = len(user_train_recorder)+10
       user_recommend = model.recommendProducts(int(user_ID), 60)
       user_test_recommend = pd.DataFrame(user_recommend).iloc[:,1].tolist()
       user_recommend_without_train = [x for x in user_test_recommend if x not
in user_train_recorder]
       len_user_recommend_without_train = len(user_recommend_without_train)
       #print("user index %s" % i)
       if len_user_recommend_without_train < 10:
         Recommend_number_max = len(user_train_recorder)+10
       user_recommend = model.recommendProducts(int(user_ID),
```

```
Recommend_number_max)
      user_test_recommend = pd.DataFrame(user_recommend).iloc[:,1].tolist()
      user_recommend_without_train = [x for x in user_test_recommend if x not
in user_train_recorder]
      len_user_recommend_without_train = len(user_recommend_without_train)
      #print("recommend number %s" % len_user_recommend_without_train)

      user_recommend_without_train_10 = user_recommend_without_train[0:10]
      user_recorder_testset =
df_test.loc[df_test.iloc[:,0]==user_ID].iloc[:,1].tolist()

      num_successful = len([x for x in user_recommend_without_train_10 if x in
user_recorder_testset])
      total_score = total_score + num_successful
#print(num_successful)
print(total_score)
#score = total_score/(len(ID_in_test_train_set)*10)
total_score = 0
for i in range(21000, 24000):
      #print(user_ID)
      #user_ID_idx_needmorerecommend = []
      user_ID = ID_in_test_train_set[i]
      user_train_recorder =
df_train.loc[df_train.iloc[:,0]==user_ID,1].tolist()
      #Recommend_number_max = len(user_train_recorder)+10
      user_recommend = model.recommendProducts(int(user_ID), 60)
      user_test_recommend = pd.DataFrame(user_recommend).iloc[:,1].tolist()
      user_recommend_without_train = [x for x in user_test_recommend if x not
in user_train_recorder]
      len_user_recommend_without_train = len(user_recommend_without_train)
      #print("user index %s" % i)
      if len_user_recommend_without_train < 10:
      Recommend_number_max = len(user_train_recorder)+10
      user_recommend = model.recommendProducts(int(user_ID),
Recommend_number_max)
      user_test_recommend = pd.DataFrame(user_recommend).iloc[:,1].tolist()
      user_recommend_without_train = [x for x in user_test_recommend if x not
in user_train_recorder]
      len_user_recommend_without_train = len(user_recommend_without_train)
      #print("recommend number %s" % len_user_recommend_without_train)

      user_recommend_without_train_10 = user_recommend_without_train[0:10]
      user_recorder_testset =
df_test.loc[df_test.iloc[:,0]==user_ID].iloc[:,1].tolist()
```

```
	num_successful = len([x for x in user_recommend_without_train_10 if x in
user_recorder_testset])
	total_score = total_score + num_successful
#print(num_successful)
print(total_score)
feature_df = pd.read_csv('feature_data_total_1.csv')
feature_df.head()
transfer_df = pd.read_csv('strid_intid.csv')
transfer_df.head()
combine_df = feature_df.merge(transfer_df[['msno','msno_int']], on = 'msno')
combine_df.head()
num_list = []
int_list = []
df_sim = combine_df[['song_listen_num','msno_int']]
for i in range(len(ID_in_test_train_set)):
	user_id = ID_in_test_train_set[i]
	df = df_sim[df_sim['msno_int'] == user_id]
	int_list.append(i)
	if df.shape[0] != 0:
	num = df.iloc[0,0]
	num_list.append(num)
	else:
	num_list.append(0)

	#if i%100 == 0:
	#print(i)
print(int_list[0:10])
sum_list = []
sum_list.append(sum(num_list[0:3000]))
sum_list.append(sum(num_list[3000:6000]))
sum_list.append(sum(num_list[6000:9000]))
sum_list.append(sum(num_list[9000:12000]))
sum_list.append(sum(num_list[12000:15000]))
sum_list.append(sum(num_list[15000:18000]))
sum_list.append(sum(num_list[18000:21000]))
sum_list.append(sum(num_list[21000:24000]))
print(sum_list)
num_list_n = [0,1,2,3,4,5,6,7]
import matplotlib.pyplot as plt
import seaborn as sns

fig = plt.figure(figsize=(12, 5))
fig = plt.subplot(1,1,1)
```

```
sns.barplot(x=sum_list,
y=[0.5204,0.4744,0.4330,0.4065,0.3420,0.2671,0.1597,0.0794])
fig.set_ylabel('Recommendation Accuracy')
fig.set_xlabel('Numbers of User Records')
fig.set_title('Plot of Relationship between User History and Recommendation
Accuracy')
plt.savefig('./relationship.jpg')
```

Link: https://github.com/ElinaBian/Music_Recommendation