Music Recommender System

Team Name:
Flerken
Team members:
Xinghan Qin
Haoran Peng
Yueran Liu

Abstract

In this project, our team built a prediction model for the music recommender system. By giving more than 1 million user data including the score rating on artist, album, and genre of different tracks by Yahoo music users, we were asked to do data profiling first and then train our data to derive a model that's able to predict whether a specific like it or not.

To solve the task, we used Python in the entire process to load the data, make the dictionary to match all the features mentioned above to their track ID. After we got our sort data in a list in matrix form, we are finally able to apply four different classifiers by using the PySpark package. The results showed that we were able to make predictions about users' preferences with over 87% accuracy.

I.Data process

By giving more than 1 million user data including the score rating on artist, album, and genre of different tracks by Yahoo music users, we need to change the format which is convenient for us to handle.

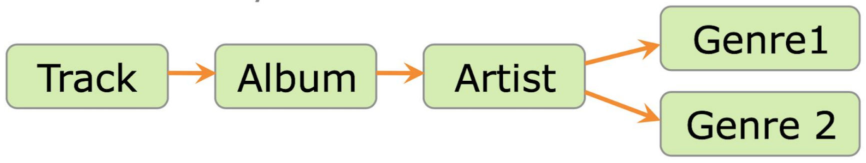
Data set description

The data we used as following:

- albumData2.txt
- artistData2.txt
- genreData2.txt
- trackData2.txt
- trainItem2.txt
- testItem2.txt
- test2.txt

Feature addition

First of all, because our data is stored in a hierarchy structure, we decided to get some basic features based on this structure:



Then, to make a more accurate score and apply more data in the training data, we added more features:

1. Relative tracks: for each object track, we have one album. Hence, for the tracks under this album, we called them relative tracks. Then, we decided to use the maximum score, minimum score and mean score to represent all the relative tracks.

2. Relative albums: for each object track, we have one artist. Hence, for the album under this artist, we called them the relative album. Then, we decided to use the maximum score, minimum score and mean score to represent all the relative albums.
   However, for the tracks under all the relative albums, we decided not to count them because they were not very related to our object track.

To achieve this, we created the “albumData.txt” and “artistData.txt” by using “trackData2.txt”:

```python
directory_name = "../../ee627a-2019fall"
testing_file_name = directory_name + "/testItem2.txt"
track_relationship_file_name = directory_name + "/trackData2.txt"

testing_file = open(testing_file_name, "r")
track_relationship_file = open(track_relationship_file_name, "r")
```

```python
print("Reading the track file ...")
dictrack = {}
albumTrack = {}
artistTrack = {}

for line in track_relationship_file:
    arr_track = line.strip().split('|')
    trackId = arr_track[0]
    albumId = arr_track[1]
    artistId = arr_track[2]
    if albumId != "None":
        tList = albumTrack.get(albumId)
        if tList == None:
            tList = []
        tList.append(trackId)
        albumTrack[albumId] = tList
    if artistId != "None" and albumId != "None":
        aList = artistTrack.get(artistId)
        if aList == None:
            aList = []
        aList.append(albumId)
        artistTrack[artistId] = aList
    otherID = arr_track[1:]
    dictrack[trackId] = otherID
print("Track file reading completed!")
```

```python
import csv
output = open("albumData_final.txt", "w")
output.write("albumId, all trackId in this album\n")
for albumId in albumTrack.keys():
    trackList = albumTrack.get(albumId)
    output.write(albumId)
    for trackId in trackList:
        output.write("," + trackId)
    output.write("\n")
```

```
albumId, all trackId in this album
106710,1,27943,95638,121434,126349,129878,129986,146801
280977,2,32098,37511,38304,72429,86624,128064,128978,142182,148087,164901,255111
38422,3,278462
119529,4,737,1057,22984,29439,48042,64537,99402,122002,148910,173813,193001,220350,222426
```

```python
import csv
output = open("artistData_fianl.txt", "w")
output.write("artistId, all albumId in this album\n")
for artistId in artistTrack.keys():
    albumList = set(artistTrack.get(artistId))
    output.write(artistId)
    for albumId in albumList:
        output.write("," + albumId)
    output.write("\n")
```

```
artistId, all albumId in this album
281667,237691,231576,106710,16320
233685,213241,180698,135761,152425,692,286126,106833,165758,130306,51795,66949,280977
219136,208450,38422,231573,142513
166863,92336,162148,22396,175518,153283,60110,150323,119529,121688,76204
294690,218479,167580,128171,147681,22766,200857,3460,112274,97383,180993,202045,16742,253490,835
44649,101746,68167,214870,120378,274309,59150
142773,181766,246442,141522,103130,83551,58498,184519
```

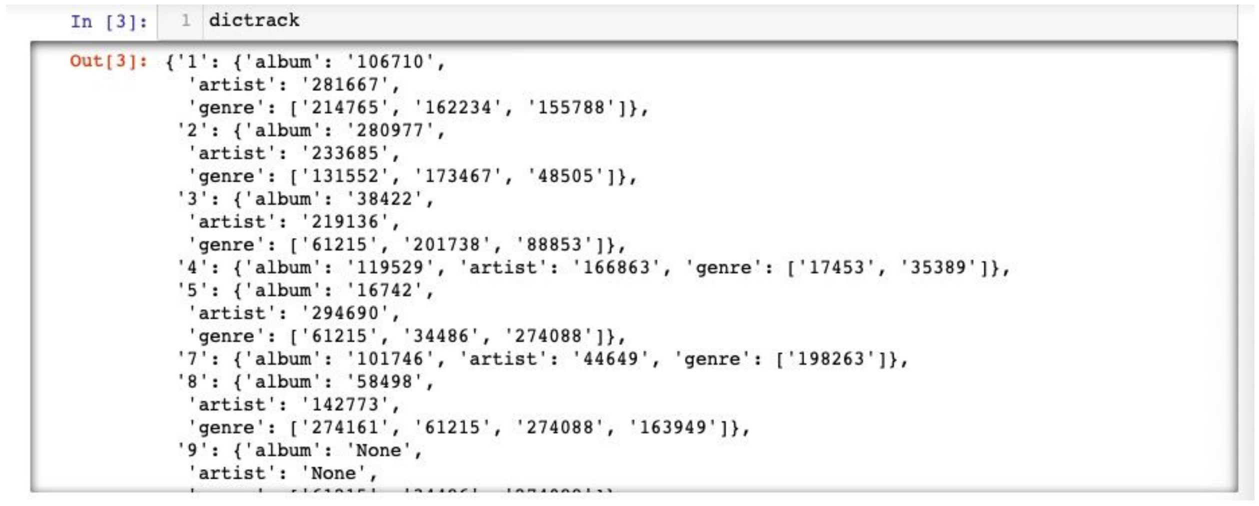
II.Track relation and score

To get our sorted data in a list in matrix form and fetch the related rating scores in the training data for the same users, we first read track files to make a dictionary. The key of the track dictionary is track id and the value of it is album, artist and genre id.

```python
In [1]:
directory_name = "../ee627a-2019fall"
testing_file_name = directory_name + "/test2.txt"
track_file_name = directory_name + "/trackData2.txt"
testing_file = open(testing_file_name, "r")
tracking_file = open(track_file_name, "r")
```

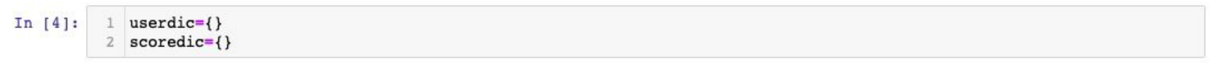
```python
In [2]:
print("Reading the track file ...")
dictrack = {}

for line in tracking_file:
    arr_track = line.strip().split('|')
    trackID = arr_track[0]
    otherID = arr_track[1:]
    trackdic={}
    for others in otherID:
        trackdic['album']=otherID[0]
        trackdic['artist']=otherID[1]
        trackdic['genre']=otherID[2:]
    dictrack[trackID] = trackdic
print("Track file reading completed!")
```

```
Reading the track file ...
Track file reading completed!
```

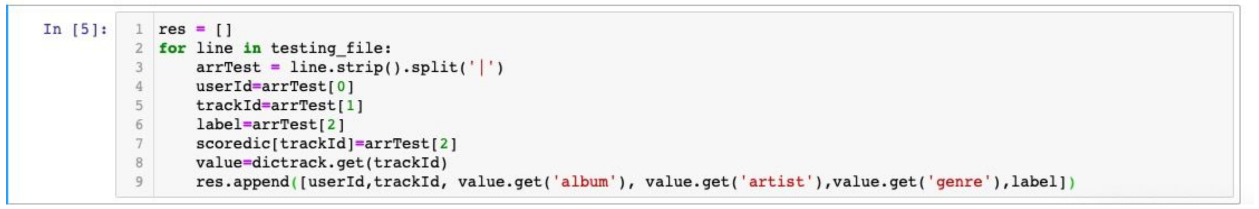


```
In [3]: dictrack

Out[3]: {'1': {'album': '106710',
          'artist': '281667',
          'genre': ['214765', '162234', '155788']},
         '2': {'album': '280977',
          'artist': '233685',
          'genre': ['131552', '173467', '48505']},
         '3': {'album': '38422',
          'artist': '219136',
          'genre': ['61215', '201738', '88853']},
         '4': {'album': '119529', 'artist': '166863', 'genre': ['17453', '35389']},
         '5': {'album': '16742',
          'artist': '294690',
          'genre': ['61215', '34486', '274088']},
         '7': {'album': '101746', 'artist': '44649', 'genre': ['198263']},
         '8': {'album': '58498',
          'artist': '142773',
          'genre': ['274161', '61215', '274088', '163949']},
         '9': {'album': 'None',
          'artist': 'None',
```

With the dictionary above we wrote a list of track relation with features, each element of the list came up with: Userid, label, trackid, albumid, artistid, and all genreid

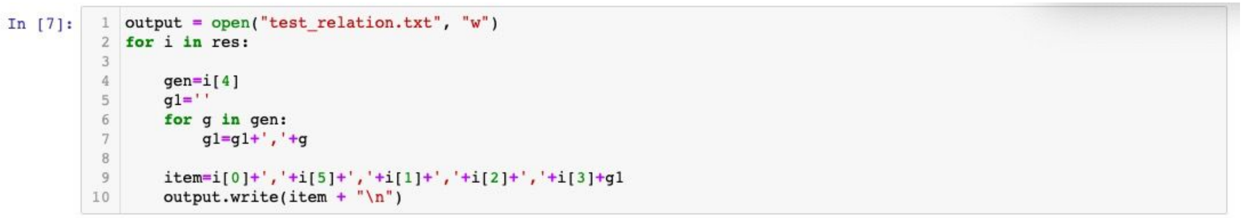


```
In [4]: userdic={}
        scoredic={}
```

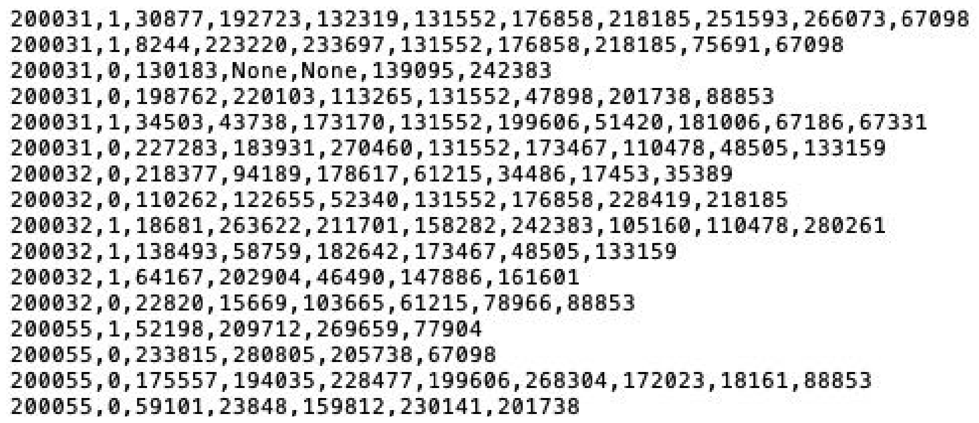


```
In [5]: res = []
        for line in testing_file:
            arrTest = line.strip().split('|')
            userId=arrTest[0]
            trackId=arrTest[1]
            label=arrTest[2]
            scoredic[trackId]=arrTest[2]
            value=dictrack.get(trackId)
            res.append([userId,trackId, value.get('album'), value.get('artist'),value.get('genre'),label])
```

With this list, we wrote a new file of test relation. Each line of the file is the element in track relation list

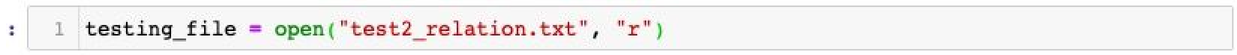


```
In [7]: output = open("test_relation.txt", "w")
        for i in res:

            gen=i[4]
            g1=''
            for g in gen:
                g1=g1+','+g

            item=i[0]+','+i[5]+','+i[1]+','+i[2]+','+i[3]+g1
            output.write(item + "\n")
```

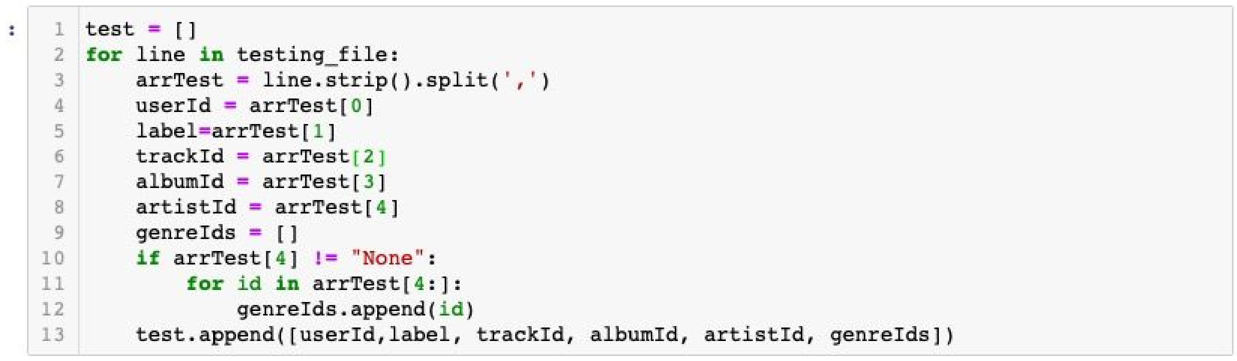


```
200031,1,30877,192723,132319,131552,176858,218185,251593,266073,67098
200031,1,8244,223220,233697,131552,176858,218185,75691,67098
200031,0,130183,None,None,139095,242383
200031,0,198762,220103,113265,131552,47898,201738,88853
200031,1,34503,43738,173170,131552,199606,51420,181006,67186,67331
200031,0,227283,183931,270460,131552,173467,110478,48505,133159
200032,0,218377,94189,178617,61215,34486,17453,35389
200032,0,110262,122655,52340,131552,176858,228419,218185
200032,1,18681,263622,211701,158282,242383,105160,110478,280261
200032,1,138493,58759,182642,173467,48505,133159
200032,1,64167,202904,46490,147886,161601
200032,0,22820,15669,103665,61215,78966,88853
200055,1,52198,209712,269659,77904
200055,0,233815,280805,205738,67098
200055,0,175557,194035,228477,199606,268304,172023,18161,88853
200055,0,59101,23848,159812,230141,201738
```

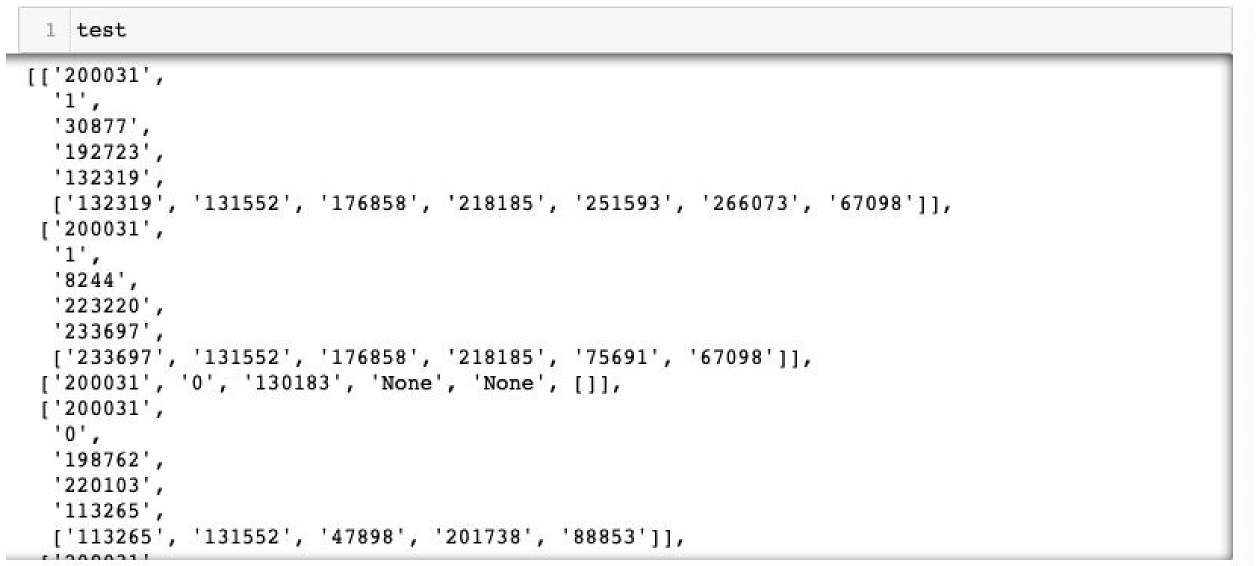
When we had all the features of these tracks, we used a training item to make the dictionary match all the features mentioned above to their track ID.
First of all, we read the test relation file and listed it.



```python
testing_file = open("test2_relation.txt", "r")
```



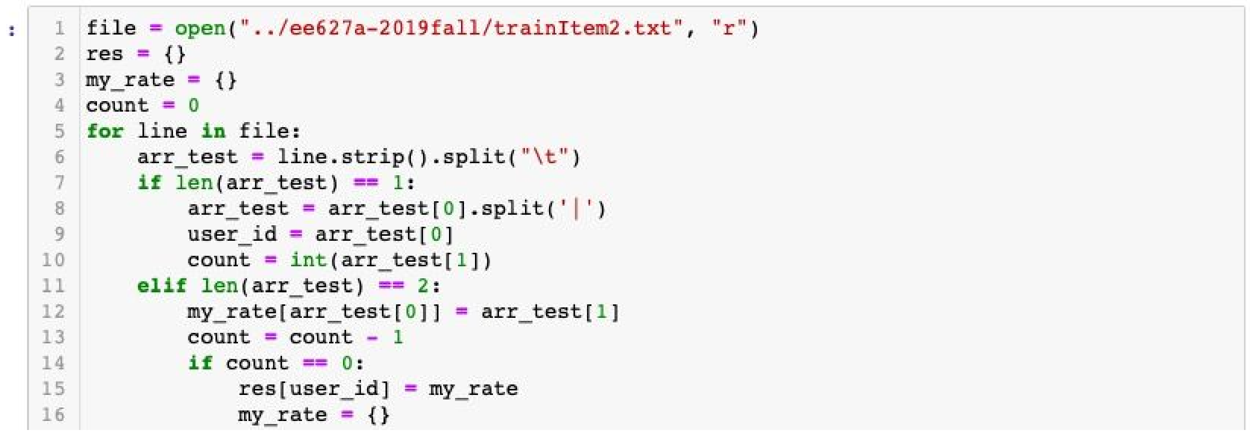
```python
test = []
for line in testing_file:
    arrTest = line.strip().split(',')
    userId = arrTest[0]
    label=arrTest[1]
    trackId = arrTest[2]
    albumId = arrTest[3]
    artistId = arrTest[4]
    genreIds = []
    if arrTest[4] != "None":
        for id in arrTest[4:]:
            genreIds.append(id)
    test.append([userId,label, trackId, albumId, artistId, genreIds])
```



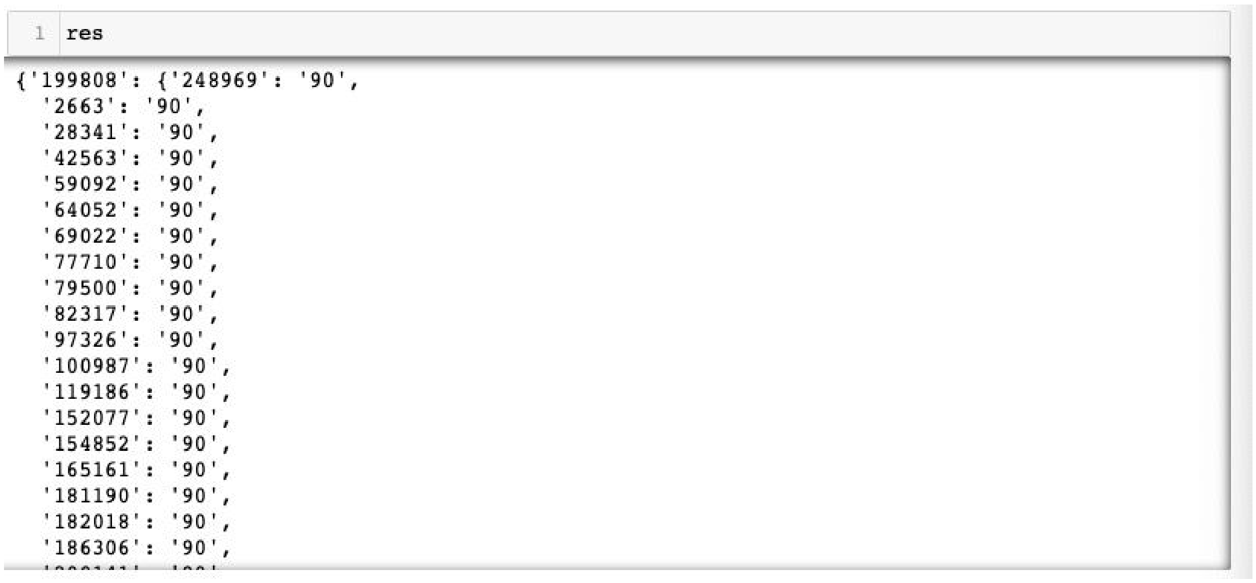
```python
test
```

```
[['200031',
  '1',
  '30877',
  '192723',
  '132319',
  ['132319', '131552', '176858', '218185', '251593', '266073', '67098']],
 ['200031',
  '1',
  '8244',
  '223220',
  '233697',
  ['233697', '131552', '176858', '218185', '75691', '67098']],
 ['200031', '0', '130183', 'None', 'None', []],
 ['200031',
  '0',
  '198762',
  '220103',
  '113265',
  ['113265', '131552', '47898', '201738', '88853']],
```

Second, we read the ‘trainItem2.txt’ file to make it a dictionary, so we can match all scores from this dictionary.



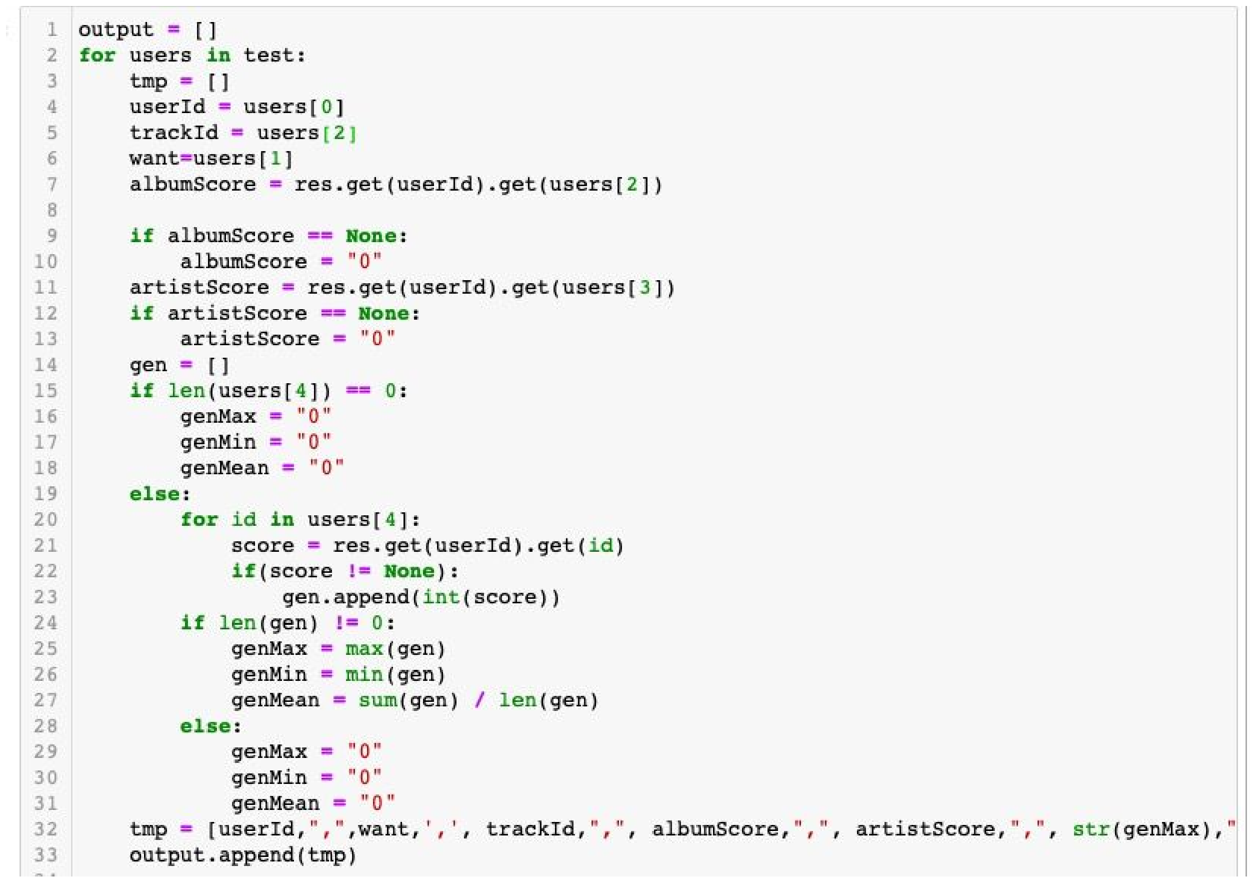
```python
file = open("../ee627a-2019fall/trainItem2.txt", "r")
res = {}
my_rate = {}
count = 0
for line in file:
    arr_test = line.strip().split("\t")
    if len(arr_test) == 1:
        arr_test = arr_test[0].split('|')
        user_id = arr_test[0]
        count = int(arr_test[1])
    elif len(arr_test) == 2:
        my_rate[arr_test[0]] = arr_test[1]
        count = count - 1
        if count == 0:
            res[user_id] = my_rate
            my_rate = {}
```



```
res
```

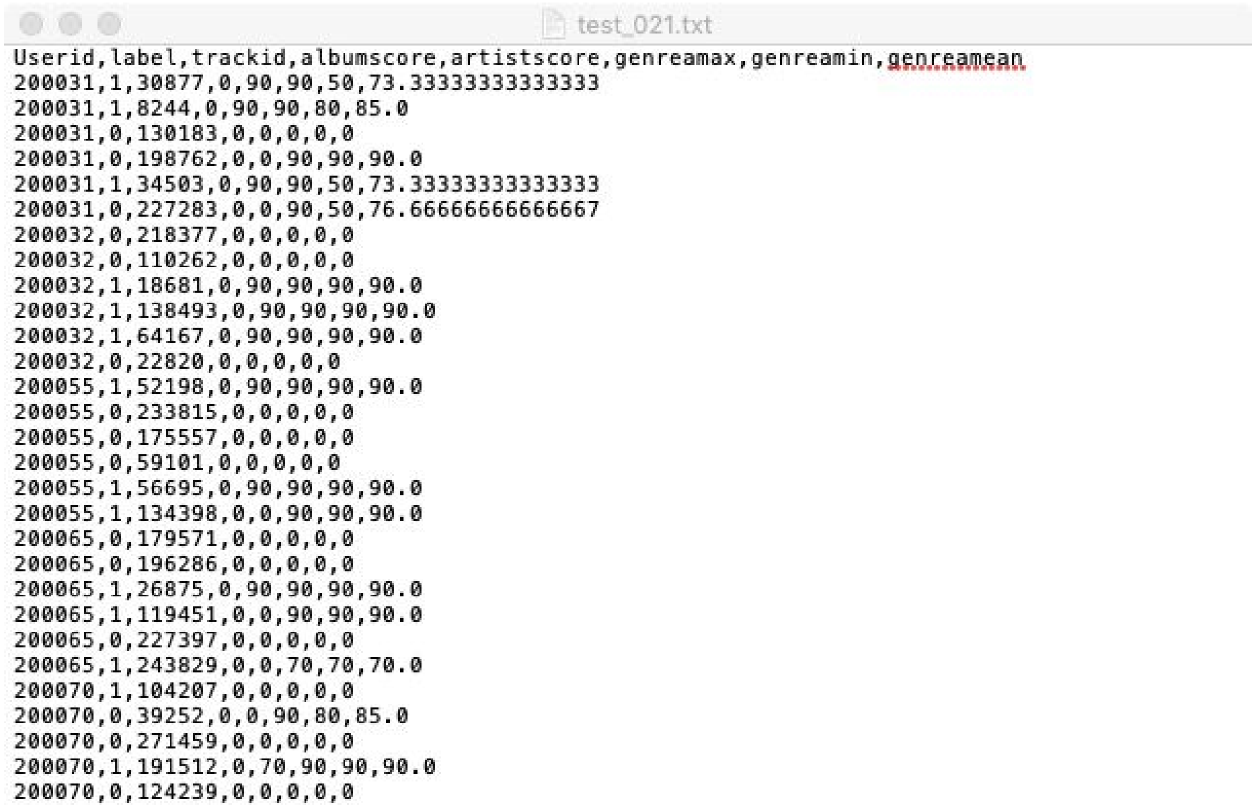
```
{'199808': {'248969': '90',
  '2663': '90',
  '28341': '90',
  '42563': '90',
  '59092': '90',
  '64052': '90',
  '69022': '90',
  '77710': '90',
  '79500': '90',
  '82317': '90',
  '97326': '90',
  '100987': '90',
  '119186': '90',
  '152077': '90',
  '154852': '90',
  '165161': '90',
  '181190': '90',
  '182018': '90',
  '186306': '90',
```

Cause some track features did not have a score, we used 0 instead of None value. What’s more, different tracks may have different kinds of genres. To fit all track, we choose three scores: genre max, genre min, and genre mean.



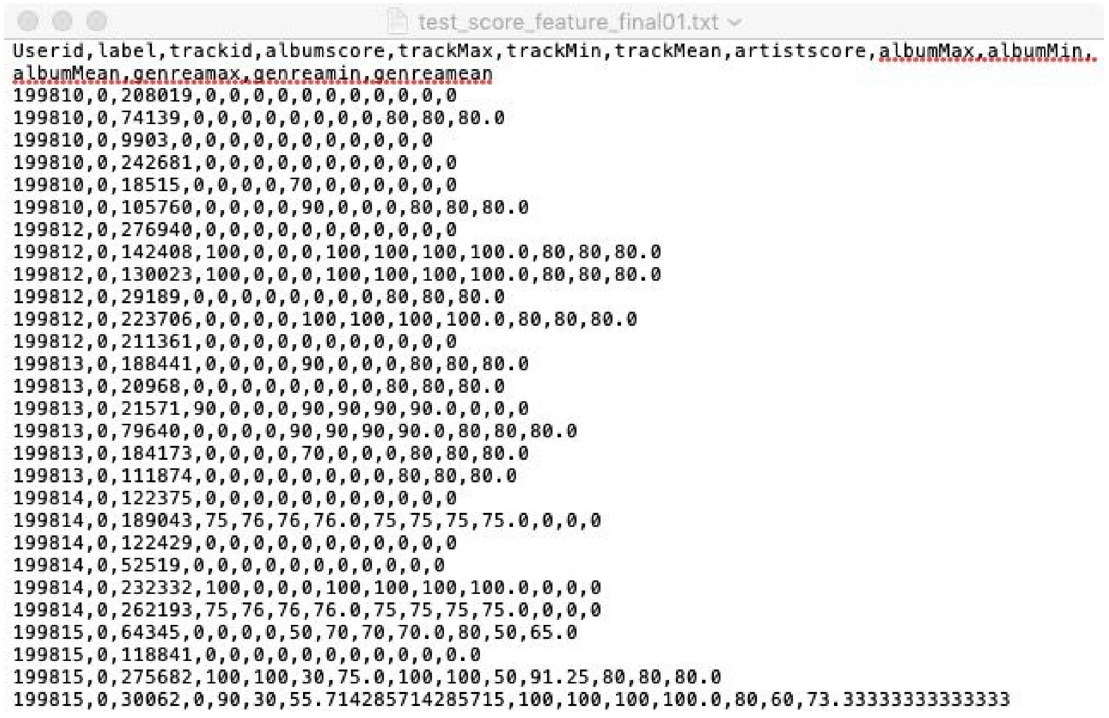
```
output = []
for users in test:
    tmp = []
    userId = users[0]
    trackId = users[2]
    want=users[1]
    albumScore = res.get(userId).get(users[2])

    if albumScore == None:
        albumScore = "0"
    artistScore = res.get(userId).get(users[3])
    if artistScore == None:
        artistScore = "0"
    gen = []
    if len(users[4]) == 0:
        genMax = "0"
        genMin = "0"
        genMean = "0"
    else:
        for id in users[4]:
            score = res.get(userId).get(id)
            if(score != None):
                gen.append(int(score))
        if len(gen) != 0:
            genMax = max(gen)
            genMin = min(gen)
            genMean = sum(gen) / len(gen)
        else:
            genMax = "0"
            genMin = "0"
            genMean = "0"
    tmp = [userId,",",want,',', trackId,",", albumScore,",", artistScore,",", str(genMax),"
    output.append(tmp)
```

The track id with all features(Userid, label, trackid, albumscore, artist score, genreamax, genreamin, genre mean)show as below, we used it in classifier:



```
Userid,label,trackid,albumscore,artistscore,genreamax,genreamin,genreamean
200031,1,30877,0,90,90,50,73.33333333333333
200031,1,8244,0,90,90,80,85.0
200031,0,130183,0,0,0,0,0
200031,0,198762,0,0,90,90,90.0
200031,1,34503,0,90,90,50,73.33333333333333
200031,0,227283,0,0,90,50,76.66666666666667
200032,0,218377,0,0,0,0,0
200032,0,110262,0,0,0,0,0
200032,1,18681,0,90,90,90,90.0
200032,1,138493,0,90,90,90,90.0
200032,1,64167,0,90,90,90,90.0
200032,0,22820,0,0,0,0,0
200055,1,52198,0,90,90,90,90.0
200055,0,233815,0,0,0,0,0
200055,0,175557,0,0,0,0,0
200055,0,59101,0,0,0,0,0
200055,1,56695,0,90,90,90,90.0
200055,1,134398,0,0,90,90,90.0
200065,0,179571,0,0,0,0,0
200065,0,196286,0,0,0,0,0
200065,1,26875,0,90,90,90,90.0
200065,1,119451,0,0,90,90,90.0
200065,0,227397,0,0,0,0,0
200065,1,243829,0,0,70,70,70.0
200070,1,104207,0,0,0,0,0
200070,0,39252,0,0,90,80,85.0
200070,0,271459,0,0,0,0,0
200070,1,191512,0,70,90,90,90.0
200070,0,124239,0,0,0,0,0
```

We also used the same method to add more features, like the max score, min score and mean score of related tracks and albums.



```
Userid,label,trackid,albumscore,trackMax,trackMin,trackMean,artistscore,albumMax,albumMin,
albumMean,genreamax,genreamin,genreamean
199810,0,208019,0,0,0,0,0,0,0,0,0,0,0
199810,0,74139,0,0,0,0,0,0,0,0,80,80,80.0
199810,0,9903,0,0,0,0,0,0,0,0,0,0
199810,0,242681,0,0,0,0,0,0,0,0,0,0,0
199810,0,18515,0,0,0,0,70,0,0,0,0,0,0
199810,0,105760,0,0,0,0,90,0,0,0,80,80,80.0
199812,0,276940,0,0,0,0,0,0,0,0,0,0,0
199812,0,142408,100,0,0,0,100,100,100,100.0,80,80,80.0
199812,0,130023,100,0,0,0,100,100,100,100.0,80,80,80.0
199812,0,29189,0,0,0,0,0,0,0,80,80,80.0
199812,0,223706,0,0,0,0,100,100,100,100.0,80,80,80.0
199812,0,211361,0,0,0,0,0,0,0,0,0,0,0
199813,0,188441,0,0,0,0,90,0,0,0,80,80,80.0
199813,0,20968,0,0,0,0,0,0,0,80,80,80.0
199813,0,21571,90,0,0,0,90,90,90,90.0,0,0,0
199813,0,79640,0,0,0,0,90,90,90,90.0,80,80,80.0
199813,0,184173,0,0,0,0,70,0,0,0,80,80,80.0
199813,0,111874,0,0,0,0,0,0,0,0,80,80,80.0
199814,0,122375,0,0,0,0,0,0,0,0,0,0,0
199814,0,189043,75,76,76,76.0,75,75,75,75.0,0,0,0
199814,0,122429,0,0,0,0,0,0,0,0,0,0,0
199814,0,52519,0,0,0,0,0,0,0,0,0,0
199814,0,232332,100,0,0,0,100,100,100,100.0,0,0,0
199814,0,262193,75,76,76,76.0,75,75,75,75.0,0,0,0
199815,0,64345,0,0,0,0,50,70,70,70.0,80,50,65.0
199815,0,118841,0,0,0,0,0,0,0,0,0,0,0.0
199815,0,275682,100,100,30,75.0,100,100,50,91.25,80,80,80.0
199815,0,30062,0,90,30,55.714285714285715,100,100,100,100.0,80,60,73.33333333333333
```

III.Classifier

At the beginning, we didn’t apply any model. We used our own experience to add weight to the features.

The first time, we ignored all the ‘None’ scores and set the weight of all features like 1. And then, according to the rank of the final score, we recommended the top 3 tracks.

```python
my_track = {}
result = [["TrackID", "Predictor"]]
count = 0

training_dict = read_training_file(training_file_name)
track_dict = read_track_file(track_relationship_file_name)

print("Start to predict rating ...")
for line in testing_file:
    arr_test = line.strip().split('|')
    if len(arr_test) == 2:
        user_id = arr_test[0]
        count = int(arr_test[1])
    else:
        track_id = arr_test[0]
        rate = predict_rate(user_id, track_id, training_dict, track_dict)
        my_track[track_id] = rate
        count = count - 1
        if count == 0:
            my_track_recommendation = predict_recommendation(my_track)
            for i in range(6):
                if i < 3:
                    result.append([user_id + "_" + my_track_recommendation[i], 1])
                else:
                    result.append([user_id + "_" + my_track_recommendation[i], 0])
            my_track = {}
print("Predict rating completed!")

print("Start to write down the prediction ...")
with open(output_file_name, "w") as output_file:
    writer = csv.writer(output_file)
    for item in result:
        writer.writerow(item)
print("Processing completed!")


def predict_rate(user_id, track_id, training_dict, track_dict):
    check_list = track_dict.get(track_id)
    score_dict = training_dict.get(user_id)
    count = 0
    res = 0
    for id in check_list:
        score = score_dict.get(id)
        if score != None:
            count = count + 1
            res = res + int(score)
    if count == 0:
        return 0
    res = res / count
    return res


def predict_recommendation(my_track):
    rates = list(my_track.values())
    tracks = list(my_track.keys())
    rates.sort(reverse=True)
    for item in tracks:
        r = my_track.get(item)
        i = rates.index(r)
        rates[i] = item
    return rates
```

The score was 0.79683.

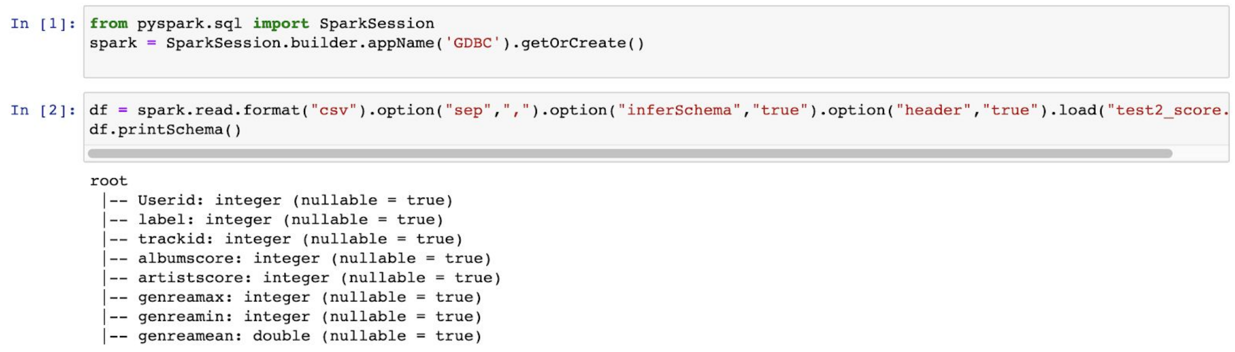
The second time, we set the weight as 0.4 for the album, 0.3 for artist and 0.1 for each general:

```
def predict_rate(user_id, track_id, training_dict, track_dict):
    check_list = track_dict.get(track_id)
    score_dict = training_dict.get(user_id)
    count = 0
    res = 0
    kind = 0    # 0 for album, 1 for artist, 2 and more than 2 for genre
    just_genre = False
    for id in check_list:
        score = score_dict.get(id)
        if kind < 2:
            if score != None:
                if kind == 0:
                    count = count + 0.4
                    res = res + int(score) * 0.4
                else:
                    count = count + 0.3
                    res = res + int(score) * 0.3
        else:
            if kind == 2 and count == 0:
                just_genre = True
            if score != None:
                count = count + 0.1
                res = res + int(score) * 0.1
        kind = kind + 1
    if count == 0:
        return 0
    res = res / count
    if just_genre:
        return res / 2
    return res
```

This time, the score had improvement and it was 0.85442.

Then, we applied four classifier model(Logistics regression model; DecisionTree; Random forest; GBTC) to the training data which had features below:
Userid,label,trackid,albumscore,artistscore,genreamax,genreamin,genreamean

First, we used the same method(VectorAssembler and PipeLine) to change the format of training data(test2, which has 6000 labeled data) to fit the model.



```
In [1]: from pyspark.sql import SparkSession
        spark = SparkSession.builder.appName('GDBC').getOrCreate()

In [2]: df = spark.read.format("csv").option("sep",",").option("inferSchema","true").option("header","true").load("test2_score.
        df.printSchema()

        root
         |-- Userid: integer (nullable = true)
         |-- label: integer (nullable = true)
         |-- trackid: integer (nullable = true)
         |-- albumscore: integer (nullable = true)
         |-- artistscore: integer (nullable = true)
         |-- genreamax: integer (nullable = true)
         |-- genreamin: integer (nullable = true)
         |-- genreamean: double (nullable = true)
```

In [3]:
```python
import pandas as pd
pd.DataFrame(df.take(5), columns=df.columns).transpose()
numeric_features = ['Userid','trackid','label','albumscore','artistscore','genreamax','genreamin','genreamean']

df.select(numeric_features).describe().toPandas().transpose()
```

Out[3]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
Userid	6000	206286.491	3515.749299951101	200031	212234
trackid	6000	149260.0025	86146.45891885243	65	296098
label	6000	0.5	0.5000416718757232	0	1
albumscore	6000	0.0	0.0	0	0
artistscore	6000	23.488666666666667	38.63689523198974	0	100
genreamax	6000	48.80733333333333	43.19745227289881	0	100
genreamin	6000	42.272333333333336	40.789456944842996	0	100
genreamean	6000	45.58248076923077	41.17219269010034	0.0	100.0

In [4]:
```python
from pyspark.ml.feature import VectorAssembler
import pyspark.sql.types as types
df=df.withColumn('label',df['label'].cast(types.IntegerType()))
df=df.withColumn('Userid',df['Userid'].cast(types.IntegerType()))
df=df.withColumn('trackid',df['trackid'].cast(types.IntegerType()))
df=df.withColumn('albumscore',df['albumscore'].cast(types.IntegerType()))
df=df.withColumn('artistscore',df['artistscore'].cast(types.IntegerType()))
df=df.withColumn('genreamax',df['genreamax'].cast(types.IntegerType()))
df=df.withColumn('genreamin',df['genreamin'].cast(types.IntegerType()))
df=df.withColumn('genreamean',df['genreamean'].cast(types.IntegerType()))

assembler=VectorAssembler(inputCols=['Userid','trackid','label','artistscore','genreamax','genreamin','genreamean'],out
```

In [5]:
```python
from pyspark.ml import Pipeline
pipeline=Pipeline(stages=[assembler])
model=pipeline.fit(df)
train=model.transform(df)
```

In [6]:
```python
df1 = spark.read.format("csv").option("sep",",").option("inferSchema","true").option("header","true").load("test_02.txt
df1.printSchema()
```

```
root
 |-- Userid: integer (nullable = true)
 |-- label: integer (nullable = true)
 |-- trackid: integer (nullable = true)
 |-- albumscore: integer (nullable = true)
 |-- artistscore: integer (nullable = true)
 |-- genreamax: integer (nullable = true)
 |-- genreamin: integer (nullable = true)
 |-- genreamean: double (nullable = true)
```

In [7]:
```python
pd.DataFrame(df1.take(5), columns=df1.columns).transpose()
numeric_features = ['Userid','trackid','albumscore','artistscore','genreamax','genreamin','genreamean']

df1.select(numeric_features).describe().toPandas().transpose()
```

Out[7]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
Userid	120000	224372.7714	14155.612277030812	199810	249010
trackid	120000	147871.92235833334	85421.1993529344	1	296099
albumscore	120000	23.685075	38.767846279190096	0	100
artistscore	120000	35.016175	42.73300263592536	0	100
genreamax	120000	35.471716666666666	40.80419698479321	0	100
genreamin	120000	32.74715833333333	39.44356509895082	0	100
genreamean	120000	34.149823609307354	39.70187105524825	0.0	100.0

```
In [8]: df1=df1.withColumn('Userid',df1['Userid'].cast(types.IntegerType()))
        df1=df1.withColumn('label',df1['label'].cast(types.IntegerType()))
        df1=df1.withColumn('trackid',df1['trackid'].cast(types.IntegerType()))
        df1=df1.withColumn('albumscore',df1['albumscore'].cast(types.IntegerType()))
        df1=df1.withColumn('artistscore',df1['artistscore'].cast(types.IntegerType()))
        df1=df1.withColumn('genreamax',df1['genreamax'].cast(types.IntegerType()))
        df1=df1.withColumn('genreamin',df1['genreamin'].cast(types.IntegerType()))
        df1=df1.withColumn('genreamean',df1['genreamean'].cast(types.IntegerType()))

        assembler1=VectorAssembler(inputCols=['Userid','trackid','label','artistscore','genreamax','genreamin','genreamean'],ou
```

```
In [9]: pipeline=Pipeline(stages=[assembler1])
        model1=pipeline.fit(df1)
        test=model1.transform(df1)
```

```
In [10]: print("Training Dataset Count: " + str(train.count()))
         print("Test Dataset Count: " + str(test.count()))

         Training Dataset Count: 6000
         Test Dataset Count: 120000
```

Second, we tried these four classifiers to generate different models. Then apply the models to test data(120000 unlabeled data).

```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label',maxDepth = 3,)
dtModel = dt.fit(train)
```

```
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(featuresCol = 'features',maxIter=10)
gbtModel = gbt.fit(train)
```

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', maxIter=10,regParam=0.3, elasticNetParam=0.8)
lrModel = lr.fit(test)
```

```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label',maxDepth = 3,)
dtModel = dt.fit(test)
```

Finally, we extracted the prediction from the result. Note that something wrong appeared on the prediction (like more than three ‘0’ or ‘1’ prediction in one user) we transferred ordered probability to the final prediction

```
In [18]: pre=predictions1.rdd.map(lambda x: x.prediction).collect()
         user=predictions1.rdd.map(lambda x: x.Userid).collect()
         track=predictions1.rdd.map(lambda x: x.trackid).collect()
```

```
In [19]: output = []
         for i in range(120000):
             output.append([user[i],track[i],pre[i]])
```

```
In [20]: import csv
         with open("GDBCOutput.csv", "w") as output_file:

             writer = csv.writer(output_file)
             writer.writerow(["TrackID", "Predictor"])
             for line in output:
                 item = [str(line[0]) + "_" + str(line[1]), line[2]]
                 writer.writerow(item)
```

.

```
In [3]: file = open("res1.txt","r")
        res = {}
        preId = "199810"
        trackDic = {}
        count = 0
        userId = None
        for line in file:
            arr_track = line.strip().split(',')
            userId = arr_track[0]
            trackId = arr_track[1]
            rate = arr_track[2]
            if userId != preId:
                res[preId] = trackDic
                trackDic = {}
                preId = userId
            trackDic[trackId] = float(rate)
            count += 1
        res[userId] = trackDic
```

```
In [5]: result = []
        for userId in res:
            my_track_recommendation = predict_recommendation(res.get(userId))
            for i in range(6):
                if i < 3:
                    result.append([userId + "_" + my_track_recommendation[i][0], 0])
                else:
                    result.append([userId + "_" + my_track_recommendation[i][0], 1])
```

```
In [7]: import csv
        with open("res_ordered1.csv", "w") as output_file:
            writer = csv.writer(output_file)
            writer.writerow(["TrackID", "Predictor"])
            for line in result:
                writer.writerow(line)
```

Here are the scores of these Models
1.Logistics regression model(0.83256)
2.Decision tree(0.83859)
3.Random forest(0.74292)
4.GBTC(0.83552)

IV.Ensembling

We used the function below to obtain the weight of each submission.

$$\mathbf{a}_{LS} = \left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{x}$$

First, we changed all ‘0’ in these data to ‘-1’ to indicate the prediction and true are matched or not.

```
import pandas as pd
data = pd.read_csv('Submisson_Ensembling0.81954.csv')
for i,row in data.iterrows():
    if data['Predictor'][i] == 0.0:
        data['Predictor'][i] = -1
data.to_csv('Submisson_Ensembling0.81954_new.csv',index=False)
```

Second, we used six submissions to generate the ensemble matrix and score matrix.

```
# load all scores data from submission.csv
pre1=pd.read_csv('Submisson_Ensembling0.81954.csv')
pre2=pd.read_csv('submission0.87005.csv')
pre3=pd.read_csv('LR_odered_0.83256.csv')
pre4=pd.read_csv('output0.85442.csv')
pre5=pd.read_csv('decisionTreeOutput_0.83859.csv')
pre6=pd.read_csv('GBTC_odered_0.80075.csv')
pre7=pd.read_csv('output_ALS0.66016.csv')
pre8=pd.read_csv('submission0.80925.csv')
pre9=pd.read_csv('submission0.84464.csv')
prenew=pd.read_csv('sample_submission.csv')
```

```
#list


s1=pre1.Predictor.tolist()
s2=pre2.Predictor.tolist()
s3=pre3.Predictor.tolist()
s4=pre4.Predictor.tolist()
s5=pre2.Predictor.tolist()
s6=pre3.Predictor.tolist()
s7=pre4.Predictor.tolist()
s8=pre3.Predictor.tolist()
s9=pre4.Predictor.tolist()
#S
S=np.transpose(mat([s1,s2,s3,s4,s5,s6,s7,s8,s9]))
S
```

```
matrix([[0, 0, 1, ..., 1, 1, 1],
        [1, 1, 1, ..., 1, 1, 1],
        [0, 0, 1, ..., 1, 1, 1],
        ...,
        [1, 1, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [1, 1, 0, ..., 0, 0, 0]])
```

```
#score
#STy=k*(2*score_mat-1)
score1=0.81954
score2=0.87005
score3=0.83256
score4=0.85442
score5=0.83859
score6=0.80075
score7=0.66016
score8=0.80925
score9=0.84464
score_mat=np.transpose(mat([score1,score2,score3,score4,score5,score6,score7,score8,score9]))
k=120000
STy=k*(2*score_mat-1)
STy
```

```
matrix([[76689.6],
        [88812. ],
        [79814.4],
        [85060.8],
        [81261.6],
        [72180. ],
        [38438.4],
        [74220. ],
        [82713.6]])
```

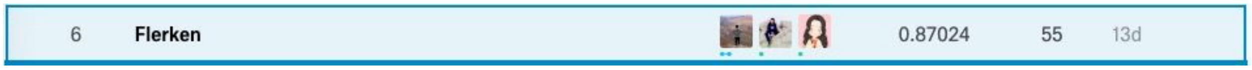
Finally, we generated the function and applied it to the submission matrix then we had the ensembled submission.

```
#a=((STS)^(-1))*STy
STS=np.transpose(S)*S
STS_inverse = np.linalg.pinv(STS)
STS_inverse
a=STS_inverse*STy
#S0=S*a
S0=S*a
S0=S0.T
newS0=np.array(S0).reshape((20000,6)).tolist()
newS0
```

```
output = []
for i in range(120000):
    s = id1[i]
    arr_test = s.strip().split("_")
    string = arr_test[0] + "," + arr_test[1] + "," + str(S0.item(i))
    output.append(string)
```

```
file = open("res1.txt", "w")
for line in output:
    file.write(line +"\n")
```

With the file, we finally got the correct rate up to 0.87024.



6 Flerken 0.87024 55 13d

Conclusion

In this project, we first used the basic method to predict the score in the test data by giving weight to each feature, but it lost some relation between each other. So we reformed the data by generating the relation between different user, track, album, artist and genre to a dictionary form. Then we applied four classifiers to these reformed data and ensemble the above submission together and got the final result(0.87024) and rank(top 8).

But we still have many problems that need to be solved. The classifiers we used are very common, and when we tried to obtain the weight of each submission, the accuracy of our submission did not improve in the beginning. So we have to keep learning in this area to find a better way to construct a more complicated and more accurate model to predict the result.

To get more information, please visit our Github:
https://github.com/OseankingO/627_project.git