DATA420-18S2 (C)

Assignment 2

The Million Song Dataset (MSD)

Student name: Shun Li

Student number: 65322005

Introduction

In this assignment, we will study a collection of datasets referred to as the Million Song Dataset (MSD), a project initiated by The Echo Nest and LabROSA. The Echo Nest was a research spin-off from the MIT Media Lab established with the goal of understanding the audio and textual content of recorded music, and was acquired by Spotify after 10 years for 50 million Euro.

First part, we will do some data processing, which includes removing the mismatched songs, and using audio feature attribute name and type from audio/attributes directory to define schemas for audio features automatically.

Next part, we will focus on the Top MAGD datasets to use numerical representations of song's audio waveform to predict genre, through classification algorithms.

Last part, we will use Taste Profile dataset to development a song recommendation system based on collaborative filtering model ,such as Alternating Least Squares (ALS).

Data processing

Q1

Read through the documentation above and figure out how to read from each of the datasets. Make sure that you only read a subset of the larger datasets, so that you can develop your code efficiently without taking up too much of your time and the cluster resources.

(a) Give an overview of the structure of the datasets, including file formats, data types, and the expected level of parallelism that you can expect to achieve from HDFS.

The data is in four parts:

msd/audio: which contains three directories(attributes, features, statistics)
attributes directory consists of 13 different csv files that containing the feature names and types ; these 13 files correspond to the following 13 different features directory respectively to define schemas of these features;
features directory consists of 13 different directories, each directory has 8 csv files.
Statistics directory has one csv file.

msd/genre: which has three different tsv. files

msd/main: it includes one directory called summary,and this summary directory consist of two csv files.

msd/tasteprofile: it has two different directories(mismatches; triplets)
mismatches directory contains two txt files.
triplets directory contains 8 tsv files.

(b) Look up the repartition method. Do you think this method will be useful?

Usually, one of principles for partition is that make the number of partitions as possible as equal to the number of cluster cores;

In addition, the size of the partition block also influences the Spark performance : the smaller the partition block, the more partitions are needed to process a same file, more computational tasks for handling partitions;the bigger the partition block, the time handling one block are longer.

So ,some times ,we need to adapt Repartition method , which is a merge of shuffle and change of number of partitions , to make the number of partitions and size of each partition are ideal for our job.

But for this dataset, each file are matched with blocks and can make full use of these blocks, which is consistent with the partition principles.

So, we do not need to use this method.

(c) Count the number of rows in each of the datasets. How do the counts compare to the total number of unique songs?

In term of reading file , we adapt the regularization method to read each whole dataset once, replacing that just reading one file . This can improve the efficiency of file reading .

```
audio_attributes = (spark.read.csv("/data/msd/audio/attributes/*"))
```

Then ,using count command to count each dataset.

The following is the result:

Dataset	**Row_count**
audio_attributes	3929
audio_features	12927867
audio_statistics	992866
genre	1103077
main_summary	2000002
tasteprofile_mismatches	20032
tasteprofile_triplets	48373586

Q2

Complete the following data preprocessing.

(a) Filter the Taste Profile dataset to remove the songs which were mismatched.

Firstly , pick small subset to check the form of each dataset:

(1) Triplets:

```
hdfs dfs -text hdfs:///data/msd/tasteprofile/triplets.tsv/part-00000.tsv.gz| head -10
```

```
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOAKIMP12A8C130995     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOAPDEY12A81C210A9     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBBMDR12A8C13253B     2
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBFNSP12AF72A0E22     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBFOVM12A58A7D494     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBNZDC12A6D4FC103     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBSUJE12A6D4F8CF5     2
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBVFZR12A6D4F8AE3     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBXALG12A8C13C108     1
#b80344d063b5ccb3212f76538f3d9e43d87dca9e     SOBXHDL12A81C204C0     1
```

Each triplet file contains three columns: user; song; play_count.

(2) Mismatches:

```
hdfs dfs -text hdfs:///data/msd/tasteprofile/mismatches/sid_mismatches.txt | head -10
```

```
#ERROR: <SOUMNSI12AB0182807 TRMMGKQ128F9325E10> Digital Underground  -  The Way We Swing  !=  Linkwood  -  Whats up with the Underground
#ERROR: <SOCMRBE12AB018C546 TRMMREB12903CEB1B1> Jimmy Reed  -  The Sun Is Shining (Digitally Remastered)  !=  Slim Harpo  -  I Got Love If You Want It
#ERROR: <SOLPHZY12AC468ABA8 TRMMBOC12903CEB46E> Africa HiTech  -  Footstep  !=  Marcus Worgull  -  Drumstern (BONUS TRACK)
#ERROR: <SONGHTM12A8C1374EF TRMMITP128F425D8D0> Death in Vegas  -  Anita Berber  !=  Valen Hsu  -  Shi Yi
#ERROR: <SONGXCA12A8C13E82E TRMMAYZ128F429ECE6> Grupo Exterminador  -  El Triunfador  !=  I Ribelli  -  Lei M
#ERROR: <SOMBCRC12A67ADA435 TRMMNVU128EF343EED> Fading Friend  -  Get us out!  !=  Masterboy  -  Feel The Heat 2000
#ERROR: <SOTDWDK12A8C13617B TRMMNCZ128F426FF0E> Daevid Allen  -  Past Lives  !=  Bhimsen Joshi  -  Raga - Shuddha Sarang_ Aalap
#ERROR: <SOEBURP12AB018C2FB TRMMPBS12903CE90E1> Cristian Paduraru  -  Born Again  !=  Yespiring  -  Journey Stages
#ERROR: <SOSRJHS12A6D4FDAA3 TRMWMEL128F421DA68> Jeff Mills  -  Basic Human Design  !=  M&T  -  Drumsettester
#ERROR: <SOIYAAQ12A6D4F954A TRMWHRI128F147EA8E> Excepter  -  OG  !=  The Fevers  -  Não Tenho Nada (Natchs Scheint Die Sonne)
```

each row is just a single string, and the songid and trackid that we need are included in this part

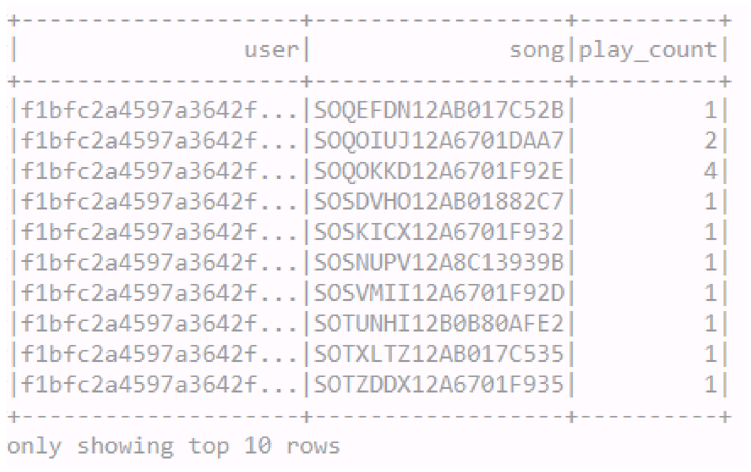
#ERROR: <SOIYAAQ12A6D4F954A   TRMWHRI128F147EA8E>

So, for this job, we separate it into three step:(see code for more details)

Step1: Loading the triplets data (attention: the form of triplets dataset is tsv,we need to use

sep = "\t" to remove “\”)

##   Triplets:

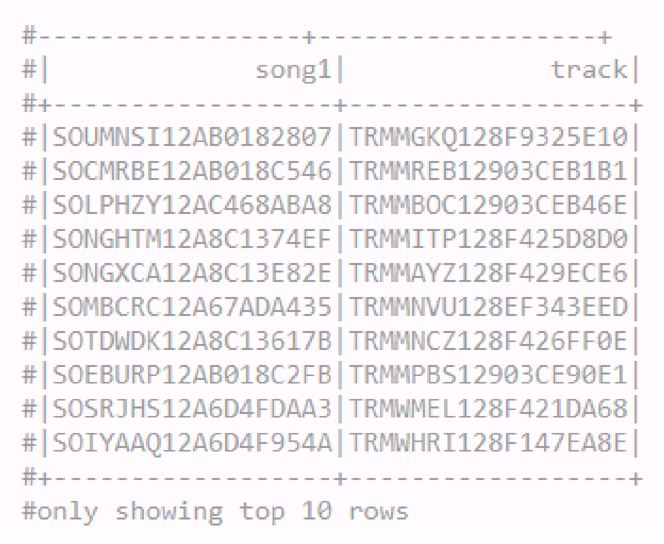


```
+--------------------+------------------+----------+
|                user|              song|play_count|
+--------------------+------------------+----------+
|f1bfc2a4597a3642f...|SOQEFDN12AB017C52B|         1|
|f1bfc2a4597a3642f...|SOQOIUJ12A6701DAA7|         2|
|f1bfc2a4597a3642f...|SOQOKKD12A6701F92E|         4|
|f1bfc2a4597a3642f...|SOSDVHO12AB01882C7|         1|
|f1bfc2a4597a3642f...|SOSKICX12A6701F932|         1|
|f1bfc2a4597a3642f...|SOSNUPV12A8C13939B|         1|
|f1bfc2a4597a3642f...|SOSVMII12A6701F92D|         1|
|f1bfc2a4597a3642f...|SOTUNHI12B0B80AFE2|         1|
|f1bfc2a4597a3642f...|SOTXLTZ12AB017C535|         1|
|f1bfc2a4597a3642f...|SOTZDDX12A6701F935|         1|
+--------------------+------------------+----------+
only showing top 10 rows
```

Step2: load mismatch dataset and exact the songid and trackid (attention: the   form of

Mismatches dataset is txt)

## Mismatches:

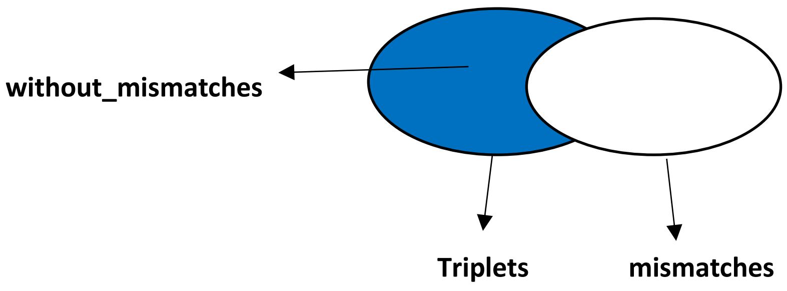


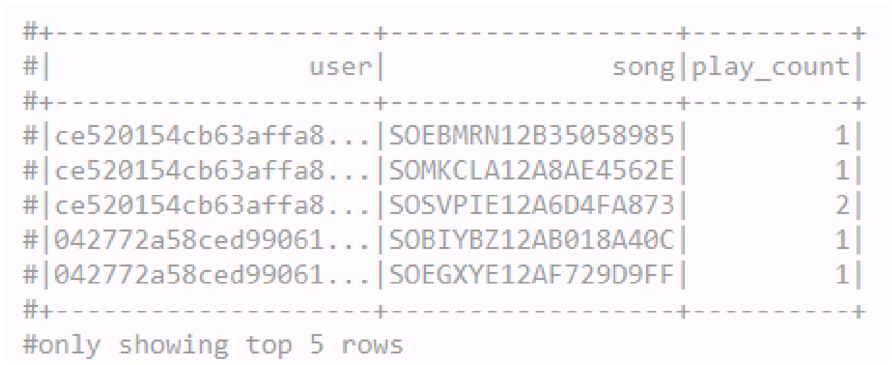
```
#------------------+------------------+
#|             song1|             track|
#+------------------+------------------+
#|SOUMNSI12AB0182807|TRMMGKQ128F9325E10|
#|SOCMRBE12AB018C546|TRMMREB12903CEB1B1|
#|SOLPHZY12AC468ABA8|TRMMBOC12903CEB46E|
#|SONGHTM12A8C1374EF|TRMMITP128F425D8D0|
#|SONGXCA12A8C13E82E|TRMMAYZ128F429ECE6|
#|SOMBCRC12A67ADA435|TRMMNVU128EF343EED|
#|SOTDWDK12A8C13617B|TRMMNCZ128F426FF0E|
#|SOEBURP12AB018C2FB|TRMMPBS12903CE90E1|
#|SOSRJHS12A6D4FDAA3|TRMWMEL128F421DA68|
#|SOIYAAQ12A6D4F954A|TRMWHRI128F147EA8E|
#+------------------+------------------+
#only showing top 10 rows
```

Step3:   remove the songs which were mismatched.

Here, using **“left_anti” join** method to complete the job:

tasteprofile_triplets.join(mismatches,tasteprofile_triplets.song == mismatches.song1,"left_anti")



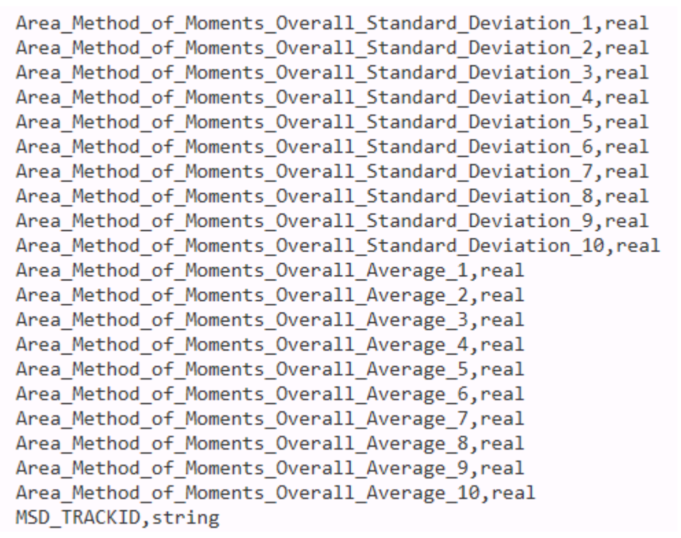


```
#+--------------------+------------------+----------+
#|                user|              song|play_count|
#+--------------------+------------------+----------+
#|ce520154cb63affa8...|SOEBMRN12B35058985|         1|
#|ce520154cb63affa8...|SOMKCLA12A8AE4562E|         1|
#|ce520154cb63affa8...|SOSVPIE12A6D4FA873|         2|
#|042772a58ced99061...|SOBIYBZ12AB018A40C|         1|
#|042772a58ced99061...|SOEGXYE12AF729D9FF|         1|
#+--------------------+------------------+----------+
#only showing top 5 rows
```

In conclusion, totally there are 48,373,586 user - song - play count triplets and overall 20,032 different songs are mismatched, after removing these mismatched songs, there remains 45,795,100 user - song - play count triplets.

(b) Load the audio feature attribute names and types from the audio/attributes directory and use them to define schemas for the audio features themselves. Note that the attribute files and feature datasets share the same prefix and that the attribute types are named consistently. Think about how you can automate the creation of StructType by mapping attribute types to pyspark.sql.types objects.

using the names and types in audio/attributes to define the schemas of audio/features



```
Area_Method_of_Moments_Overall_Standard_Deviation_1,real
Area_Method_of_Moments_Overall_Standard_Deviation_2,real
Area_Method_of_Moments_Overall_Standard_Deviation_3,real
Area_Method_of_Moments_Overall_Standard_Deviation_4,real
Area_Method_of_Moments_Overall_Standard_Deviation_5,real
Area_Method_of_Moments_Overall_Standard_Deviation_6,real
Area_Method_of_Moments_Overall_Standard_Deviation_7,real
Area_Method_of_Moments_Overall_Standard_Deviation_8,real
Area_Method_of_Moments_Overall_Standard_Deviation_9,real
Area_Method_of_Moments_Overall_Standard_Deviation_10,real
Area_Method_of_Moments_Overall_Average_1,real
Area_Method_of_Moments_Overall_Average_2,real
Area_Method_of_Moments_Overall_Average_3,real
Area_Method_of_Moments_Overall_Average_4,real
Area_Method_of_Moments_Overall_Average_5,real
Area_Method_of_Moments_Overall_Average_6,real
Area_Method_of_Moments_Overall_Average_7,real
Area_Method_of_Moments_Overall_Average_8,real
Area_Method_of_Moments_Overall_Average_9,real
Area_Method_of_Moments_Overall_Average_10,real
MSD_TRACKID,string
```

In the audio/attribute, each file contains 21 rows, each row has two column (name, type).

```
hdfs dfs -text hdfs:///data/msd/audio/features/msd-jmir-area-of-moments-all v1.0.csv/part-00000.csv.gz | head -1
```

from the result(see code), we can know that in audio/features dataset, each row has 21 columns, the name of each column is consistent with the name column in audio/attribute dataset, also the type of them are same as types column in audio/attribute dataset.In another words, the name and type for **ith** column in audio/feature are consistent with the **ith** row's name column and type column in audio/attributes (**i** is positive integer from 1 to 21)

So, basing on this logic, writing a function to make the StructType defining automatical, more details can be seen in the code part.

Audio similarity

Q1

There are multiple audio feature datasets, with different levels of detail. Pick one of the small datasets to get started.

(a) The audio features are continuous values, obtained using methods such as digital signal processing and psycho-acoustic modeling.

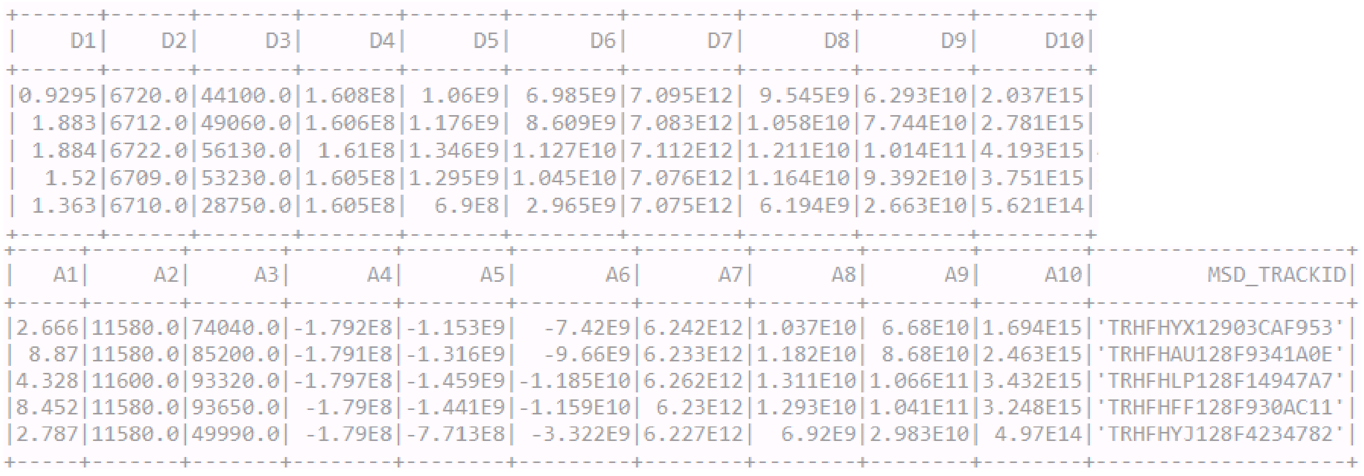
Produce descriptive statistics for each feature column in the dataset you picked. Are any features strongly correlated?

Here, I change the name of column to make them more concise, for example: change

" **Area_Method_of_Moments_Overall_Standard_Deviation_1**" to "**D1**";

"**Area_Method_of_Moments_Overall_Average_10**" to "**A1**"

And so on.

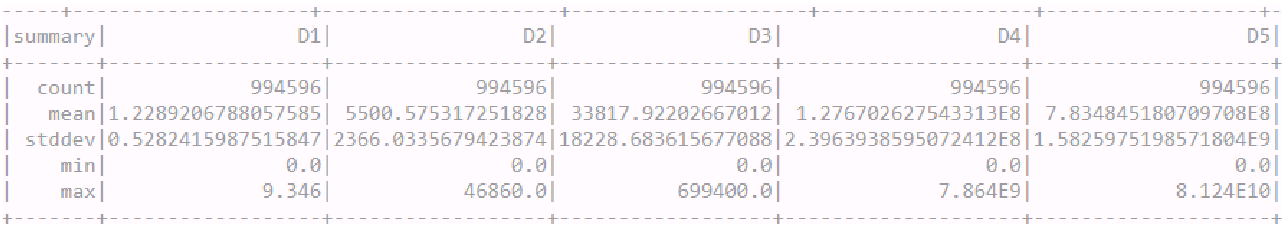


```
+------+------+-------+-------+-------+--------+--------+--------+--------+--------+
|    D1|    D2|     D3|     D4|     D5|      D6|      D7|      D8|      D9|     D10|
+------+------+-------+-------+-------+--------+--------+--------+--------+--------+
|0.9295|6720.0|44100.0|1.608E8| 1.06E9| 6.985E9|7.095E12| 9.545E9|6.293E10|2.037E15|
| 1.883|6712.0|49060.0|1.606E8|1.176E9| 8.609E9|7.083E12|1.058E10|7.744E10|2.781E15|
| 1.884|6722.0|56130.0| 1.61E8|1.346E9|1.127E10|7.112E12|1.211E10|1.014E11|4.193E15|
|  1.52|6709.0|53230.0|1.605E8|1.295E9|1.045E10|7.076E12|1.164E10|9.392E10|3.751E15|
| 1.363|6710.0|28750.0|1.605E8|  6.9E8| 2.965E9|7.075E12| 6.194E9|2.663E10|5.621E14|
+------+------+-------+-------+-------+--------+--------+--------+--------+--------+
+-----+-------+-------+--------+--------+---------+--------+--------+--------+--------+-------------------+
|   A1|     A2|     A3|      A4|      A5|       A6|      A7|      A8|      A9|     A10|        MSD_TRACKID|
+-----+-------+-------+--------+--------+---------+--------+--------+--------+--------+-------------------+
|2.666|11580.0|74040.0|-1.792E8|-1.153E9|  -7.42E9|6.242E12|1.037E10| 6.68E10|1.694E15|'TRHFHYX12903CAF953'|
| 8.87|11580.0|85200.0|-1.791E8|-1.316E9|  -9.66E9|6.233E12|1.182E10| 8.68E10|2.463E15|'TRHFHAU128F9341A0E'|
|4.328|11600.0|93320.0|-1.797E8|-1.459E9|-1.185E10|6.262E12|1.311E10|1.066E11|3.432E15|'TRHFHLP128F14947A7'|
|8.452|11580.0|93650.0| -1.79E8|-1.441E9|-1.159E10| 6.23E12|1.293E10|1.041E11|3.248E15|'TRHFHFF128F930AC11'|
|2.787|11580.0|49990.0| -1.79E8|-7.713E8| -3.322E9|6.227E12| 6.92E9|2.983E10| 4.97E14|'TRHFHYJ128F4234782'|
+-----+-------+-------+--------+--------+---------+--------+--------+--------+--------+-------------------+
```

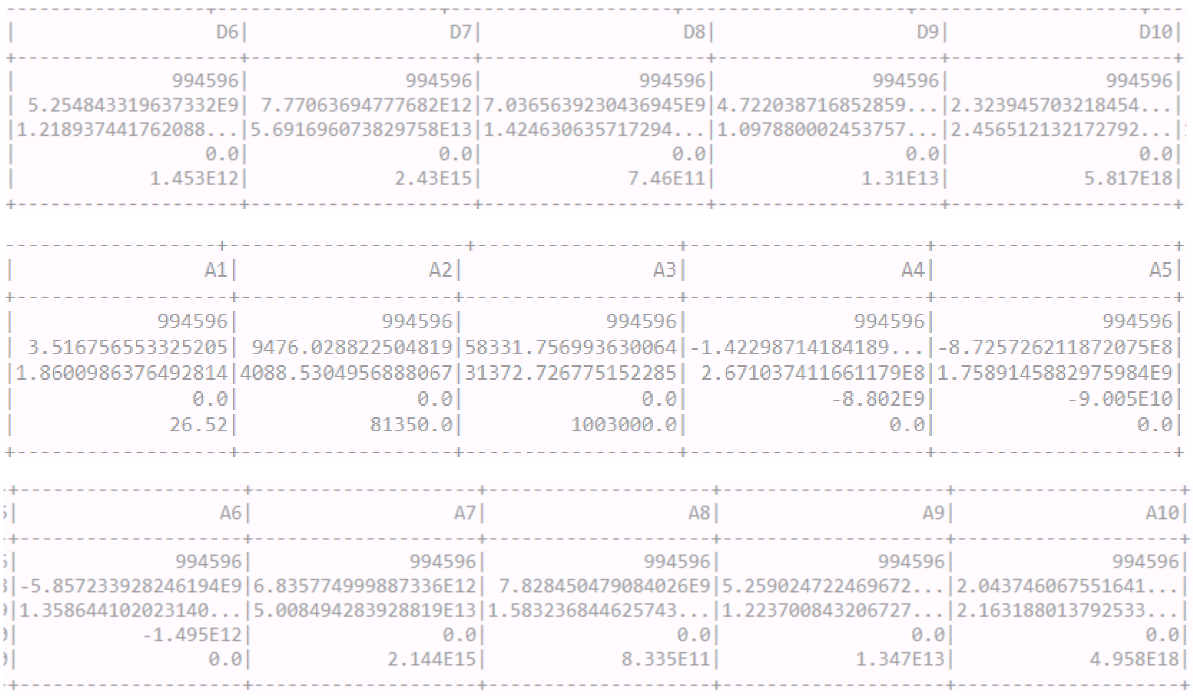
Next , I found that in the column MSD_TRACKID, even the type of it is string, it covers single quotes , like **'TRHFHYX12903CAF953'** , so we need to remove them.Here, I used the same way that I exacted the songid and trackid form mismatch directory.

Then,I tried to drop the column MSD_TRACKID to make other columns suitable for next descriptive statistics. Just using the **describe()** command to produce descripitive statistics for each feature column.

The result is here:



```
-----+-------------------+------------------+-------------------+--------------------+-------------------+-
|summary|                 D1|                D2|                 D3|                  D4|                 D5|
+-------+-------------------+------------------+-------------------+--------------------+-------------------+
|  count|             994596|            994596|             994596|              994596|             994596|
|   mean|1.2289206788057585| 5500.575317251828| 33817.92202667012| 1.276702627543313E8| 7.834845180709708E8|
| stddev|0.5282415987515847|2366.0335679423874|18228.683615677088|2.3963938595072412E8|1.5825975198571804E9|
|    min|                0.0|               0.0|                0.0|                 0.0|                0.0|
|    max|              9.346|           46860.0|           699400.0|             7.864E9|           8.124E10|
+-------+-------------------+------------------+-------------------+--------------------+-------------------+
```



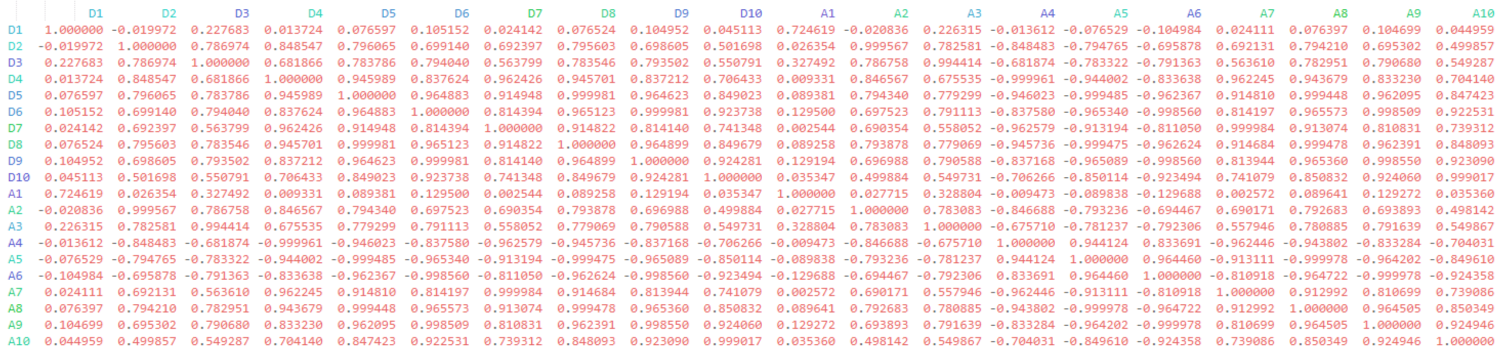
```
+------------------+------------------+------------------+------------------+------------------+
|                D6|                D7|                D8|                D9|               D10|
+------------------+------------------+------------------+------------------+------------------+
|            994596|            994596|            994596|            994596|            994596|
| 5.254843319637332E9| 7.77063694777682E12|7.0365639230436945E9|4.722038716852859...|2.323945703218454...|
|1.218937441762088...|5.691696073829758E13|1.424630635717294...|1.097880002453757...|2.456512132172792...|
|               0.0|               0.0|               0.0|               0.0|               0.0|
|          1.453E12|           2.43E15|           7.46E11|           1.31E13|          5.817E18|
+------------------+------------------+------------------+------------------+------------------+

+------------------+------------------+------------------+------------------+------------------+
|                A1|                A2|                A3|                A4|                A5|
+------------------+------------------+------------------+------------------+------------------+
|            994596|            994596|            994596|            994596|            994596|
| 3.516756553325205| 9476.028822504819|58331.756993630064|-1.42298714184189...|-8.725726211872075E8|
|1.8600986376492814|4088.5304956888067|31372.726775152285| 2.671037411661179E8|1.7589145882975984E9|
|               0.0|               0.0|               0.0|           -8.802E9|          -9.005E10|
|             26.52|           81350.0|         1003000.0|               0.0|               0.0|
+------------------+------------------+------------------+------------------+------------------+

-+------------------+------------------+------------------+------------------+------------------+
|                A6|                A7|                A8|                A9|               A10|
-+------------------+------------------+------------------+------------------+------------------+
|            994596|            994596|            994596|            994596|            994596|
|-5.857233928246194E9|6.835774999887336E12| 7.828450479084026E9|5.259024722469672...|2.043746067551641...|
|1.358644102023140...|5.008494283928819E13|1.583236844625743...|1.223700843206727...|2.163188013792533...|
|         -1.495E12|               0.0|               0.0|               0.0|               0.0|
|               0.0|          2.144E15|          8.335E11|          1.347E13|          4.958E18|
-+------------------+------------------+------------------+------------------+------------------+
```

The summary table gives the count , mean , standard deviation, minimum and maximum for each feature columns.we can find that the mean and standard deviation vary too much between different feature columns.

Are any features strongly correlated?

Here, first thing that I need to do is remove some rows tha contain missing value.Then, change the original dataframe into a RDD. Next,calculate a correlation matrix using Pearson's method.

Here is the result:



```
           D1        D2        D3        D4        D5        D6        D7        D8        D9       D10        A1        A2        A3        A4        A5        A6        A7        A8        A9       A10
D1   1.000000 -0.019972  0.227683  0.013724  0.076597  0.105152  0.024142  0.076524  0.104952  0.045113  0.724619 -0.020836  0.226315 -0.013612 -0.076529 -0.104984  0.024111  0.076397  0.104699  0.044959
D2  -0.019972  1.000000  0.786974  0.848547  0.796065  0.699140  0.692397  0.795603  0.698605  0.501698  0.026354  0.999567  0.782581 -0.848483 -0.794765 -0.695878  0.692131  0.794210  0.695302  0.499857
D3   0.227683  0.786974  1.000000  0.681866  0.783786  0.794040  0.563799  0.783546  0.793502  0.550791  0.327492  0.786758  0.994414 -0.681874 -0.783322 -0.791363  0.563610  0.782951  0.790680  0.549287
D4   0.013724  0.848547  0.681866  1.000000  0.945989  0.837624  0.962426  0.945701  0.837212  0.706433  0.009331  0.846567  0.675535 -0.999961 -0.944002 -0.833638  0.962245  0.943679  0.833230  0.704140
D5   0.076597  0.796065  0.783786  0.945989  1.000000  0.964883  0.914948  0.999981  0.964623  0.849023  0.089381  0.794340  0.779299 -0.946023 -0.999485 -0.962367  0.914810  0.999448  0.962095  0.847423
D6   0.105152  0.699140  0.794040  0.837624  0.964883  1.000000  0.814394  0.965123  0.999981  0.923738  0.129500  0.697523  0.791113 -0.837580 -0.965340 -0.998560  0.814197  0.965573  0.998509  0.922531
D7   0.024142  0.692397  0.563799  0.962426  0.914948  0.814394  1.000000  0.914822  0.814140  0.741348  0.002544  0.690354  0.558052 -0.962579 -0.913194 -0.811050  0.999984  0.913074  0.810831  0.739312
D8   0.076524  0.795603  0.783546  0.945701  0.999981  0.965123  0.914822  1.000000  0.964899  0.849679  0.089258  0.793878  0.779069 -0.945736 -0.999475 -0.962624  0.914684  0.999478  0.962391  0.848093
D9   0.104952  0.698605  0.793502  0.837212  0.964623  0.999981  0.814140  0.964899  1.000000  0.924281  0.129194  0.696988  0.790588 -0.837168 -0.965089 -0.998560  0.813944  0.965360  0.998550  0.923090
D10  0.045113  0.501698  0.550791  0.706433  0.849023  0.923738  0.741348  0.849679  0.924281  1.000000  0.035347  0.499884  0.549731 -0.706266 -0.850114 -0.923494  0.741079  0.850832  0.924060  0.999017
A1   0.724619  0.026354  0.327492  0.009331  0.089381  0.129500  0.002544  0.089258  0.129194  0.035347  1.000000  0.027715  0.328804 -0.009473 -0.089838 -0.129688  0.002572  0.089641  0.129272  0.035360
A2  -0.020836  0.999567  0.786758  0.846567  0.794340  0.697523  0.690354  0.793878  0.696988  0.499884  0.027715  1.000000  0.783083 -0.846688 -0.793236 -0.694467  0.690171  0.792683  0.693893  0.498142
A3   0.226315  0.782581  0.994414  0.675535  0.779299  0.791113  0.558052  0.779069  0.790588  0.549731  0.328804  0.783083  1.000000 -0.675710 -0.781237 -0.792306  0.557946  0.780885  0.791639  0.549867
A4  -0.013612 -0.848483 -0.681874 -0.999961 -0.946023 -0.837580 -0.962579 -0.945736 -0.837168 -0.706266 -0.009473 -0.846688 -0.675710  1.000000  0.944124  0.833691 -0.962446 -0.943802 -0.833284 -0.704031
A5  -0.076529 -0.794765 -0.783322 -0.944002 -0.999485 -0.965340 -0.913194 -0.999475 -0.965089 -0.850114 -0.089838 -0.793236 -0.781237  0.944124  1.000000  0.964460 -0.913111 -0.999978 -0.964202 -0.849610
A6  -0.104984 -0.695878 -0.791363 -0.833638 -0.962367 -0.998560 -0.811050 -0.962624 -0.998560 -0.923494 -0.129688 -0.694467 -0.792306  0.833691  0.964460  1.000000 -0.810918 -0.964722 -0.999978 -0.924358
A7   0.024111  0.692131  0.563610  0.962245  0.914810  0.814197  0.999984  0.914684  0.813944  0.741079  0.002572  0.690171  0.557946 -0.962446 -0.913111 -0.810918  1.000000  0.912992  0.810699  0.739086
A8   0.076397  0.794210  0.782951  0.943679  0.999448  0.965573  0.913074  0.999478  0.965360  0.850832  0.089641  0.792683  0.780885 -0.943802 -0.999978 -0.964722  0.912992  1.000000  0.964505  0.850349
A9   0.104699  0.695302  0.790680  0.833230  0.962095  0.998509  0.810831  0.962391  0.998550  0.924060  0.129272  0.693893  0.791639 -0.833284 -0.964202 -0.999978  0.810699  0.964505  1.000000  0.924946
A10  0.044959  0.499857  0.549287  0.704140  0.847423  0.922531  0.739312  0.848093  0.923090  0.999017  0.035360  0.498142  0.549867 -0.704031 -0.849610 -0.924358  0.739086  0.850349  0.924946  1.000000
```

From this correlation matrix , we can see that the absolute value of correlations between some feature column are very close to 1, it means they are very strongly correlated. So, I have summary the result here:

D3 and A3 ; D4 and A4 ; D5 and D8,A5,A8 ; D6 and D9,A6,A9

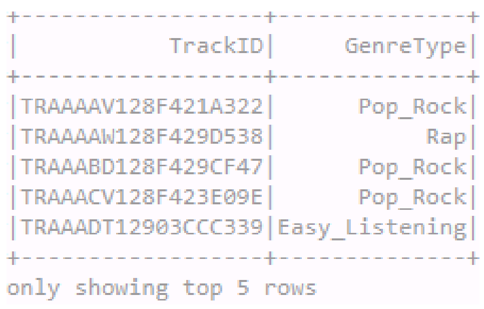
D7 and A7 ; D8 and D5, A5,A8 ; D9 and D6,A6,A9 ; D10 and A10

A5 and A8 ; A6 and A9.

Both of them are very strongly correlated.

(b) Load the MSD All Music Genre Dataset (MAGD).

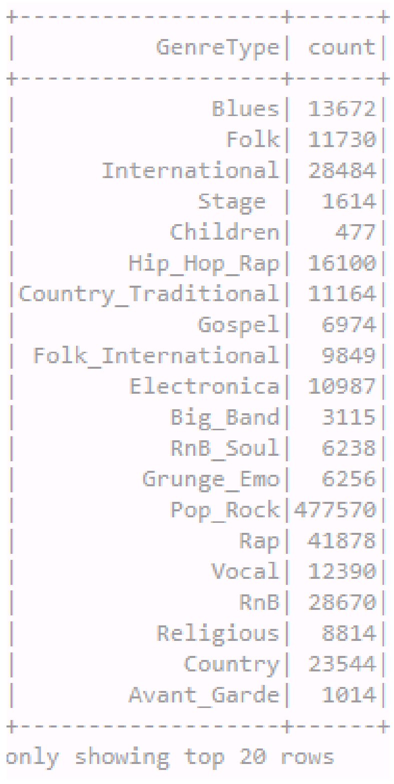
As we have loaded the other dataset, using regularization method to load all MAGD datasets, and we need to pay attention that they are tsv files. The MAGD has two columns: TrackID and GenreTYpe.



```
+------------------+-------------+
|           TrackID|    GenreType|
+------------------+-------------+
|TRAAAAV128F421A322|     Pop_Rock|
|TRAAAAW128F429D538|          Rap|
|TRAAABD128F429CF47|     Pop_Rock|
|TRAAACV128F423E09E|     Pop_Rock|
|TRAAADT12903CCC339|Easy_Listening|
+------------------+-------------+
only showing top 5 rows
```

Visualize the distribution of genres for the songs that were matched.

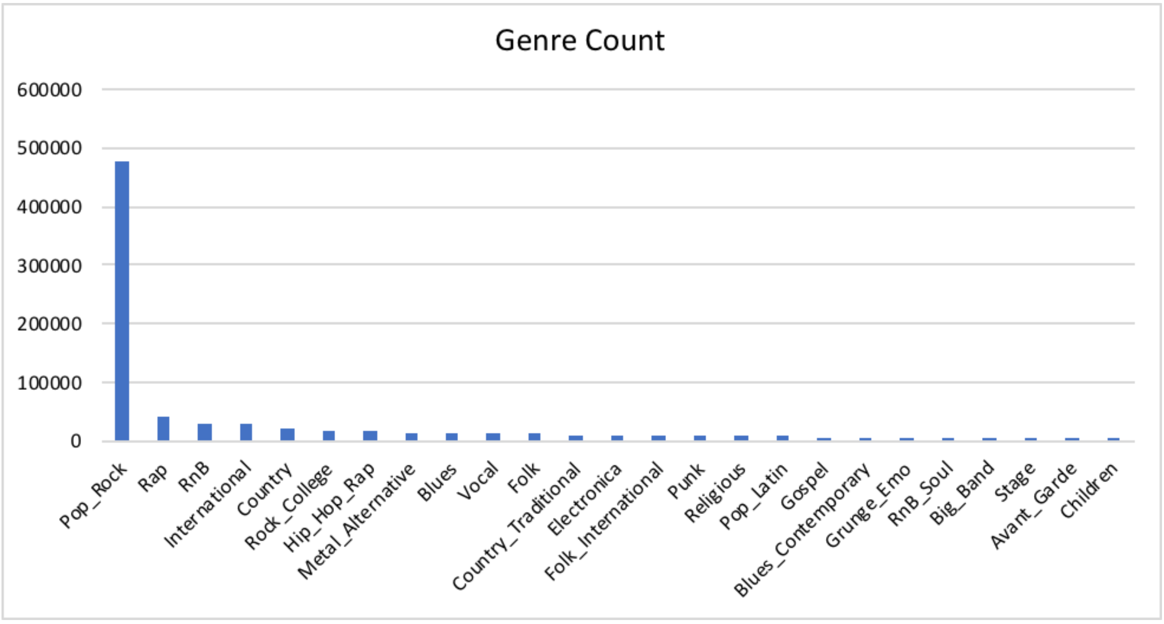
Here, firstly I need to group the rows by the genretype.Then ,because the song id has been matched with trackid, so I just need to count :how mang different trackid are covered in each genretype?



```
+-------------------+------+
|          GenreType| count|
+-------------------+------+
|              Blues| 13672|
|               Folk| 11730|
|      International| 28484|
|             Stage |  1614|
|           Children|   477|
|        Hip_Hop_Rap| 16100|
|Country_Traditional| 11164|
|             Gospel|  6974|
| Folk_International|  9849|
|        Electronica| 10987|
|           Big_Band|  3115|
|           RnB_Soul|  6238|
|         Grunge_Emo|  6256|
|           Pop_Rock|477570|
|                Rap| 41878|
|              Vocal| 12390|
|                RnB| 28670|
|          Religious|  8814|
|            Country| 23544|
|        Avant_Garde|  1014|
+-------------------+------+
only showing top 20 rows
```

Next, I tried to save the result into my outputs directory in HDFS, and use **copyToLocal** command to download to my local home directory.

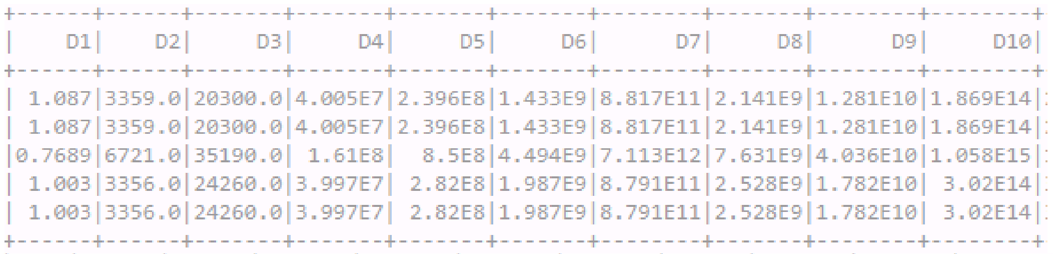
After finishing this , I used R language to produce the bar chart:



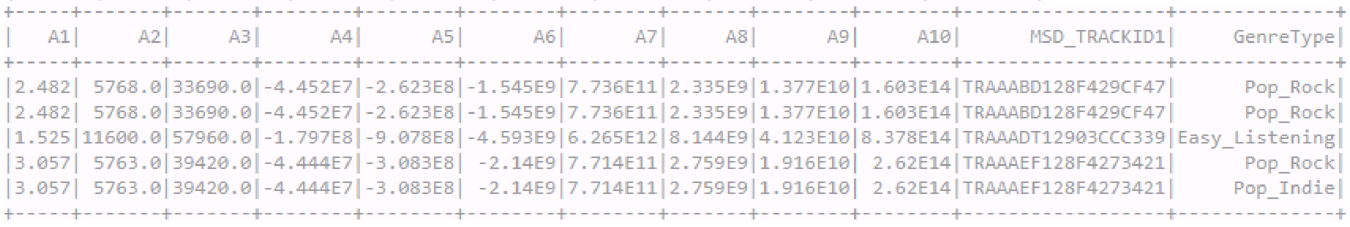
It is very obvious that the **pop-Rock songs** takes up a relative large proportion. Next, the second one is Rap and RNB.

(c) Merge the genres dataset and the audio features dataset so that every song has a label.

For features dataset, it has a MSD_TrackID column ; also the genres dataset has the trackID,and the GenreType lable are included in the genres . So just use **join** command by trackid that they shared with each other to merge them together .



```
+------+------+-------+-------+-------+-------+--------+-------+--------+--------+
|    D1|    D2|     D3|     D4|     D5|     D6|      D7|     D8|      D9|     D10|
+------+------+-------+-------+-------+-------+--------+-------+--------+--------+
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|2.141E9|1.281E10|1.869E14|
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|2.141E9|1.281E10|1.869E14|
|0.7689|6721.0|35190.0| 1.61E8|  8.5E8|4.494E9|7.113E12|7.631E9|4.036E10|1.058E15|
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11|2.528E9|1.782E10| 3.02E14|
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11|2.528E9|1.782E10| 3.02E14|
+------+------+-------+-------+-------+-------+--------+-------+--------+--------+
```



```
+-----+-------+-------+--------+--------+--------+--------+-------+--------+--------+------------------+--------------+
|   A1|     A2|     A3|      A4|      A5|      A6|      A7|     A8|      A9|     A10|      MSD_TRACKID1|     GenreType|
+-----+-------+-------+--------+--------+--------+--------+-------+--------+--------+------------------+--------------+
|2.482| 5768.0|33690.0|-4.452E7|-2.623E8|-1.545E9|7.736E11|2.335E9|1.377E10|1.603E14|TRAAABD128F429CF47|      Pop_Rock|
|2.482| 5768.0|33690.0|-4.452E7|-2.623E8|-1.545E9|7.736E11|2.335E9|1.377E10|1.603E14|TRAAABD128F429CF47|      Pop_Rock|
|1.525|11600.0|57960.0|-1.797E8|-9.078E8|-4.593E9|6.265E12|8.144E9|4.123E10|8.378E14|TRAAADT12903CCC339|Easy_Listening|
|3.057| 5763.0|39420.0|-4.444E7|-3.083E8| -2.14E9|7.714E11|2.759E9|1.916E10| 2.62E14|TRAAAEF128F4273421|      Pop_Rock|
|3.057| 5763.0|39420.0|-4.444E7|-3.083E8| -2.14E9|7.714E11|2.759E9|1.916E10| 2.62E14|TRAAAEF128F4273421|     Pop_Indie|
+-----+-------+-------+--------+--------+--------+--------+-------+--------+--------+------------------+--------------+
```

only showing top 5 rows

Q2

First you will consider binary classification only.

(a) Research and choose three classification algorithms from the spark.ml library. Justify your choice of algorithms, taking into account considerations such as explainability, interpretability, predictive accuracy, training speed, hyperparameter tuning, dimensionality, and issues with scaling.

Here, I want to choose the three classification algorithms: Logistics Regression; Navie Bay; Random Forest.

(1) For logistics regression:

it assumes that the data are subject to Bernoulli distribution. By using the method of maximization of likelihood function, and applying to predict the parameters.

Advantages:

The form of it is simple, the model is very interpretable;
The training speed is fast ;
Low memory footprint,it just needs to store the eigenvalues for each dimension;

Disadvantages:

The predictive accuracy is not good , because it is very simple, very similar to linear model
Very hard to deal with unbalanced data.

(2) For Navie Bay

Navie bay is a series of simple probability classifiers , which is based on bayes theorem .

But it has a important hypothesis : each variable is need to be independent.

Advantages:

It originated from the classical mathematics theory, so the classification efficiency is stable;
Good performance for small-scale data, and can handle multiple classification tasks;
Less sensitive to missing value

Disadvantages:

The algorithm is relatively simple, so the predict accuracy is not good.

(3) For Random Forest:

The theory of it is that the selection of columns and selection of rows are randomized to generate mang classification trees, and then results of these classification trees are summarized together.

Advantages:

Due to randomness, random forest is not easy to fall into overfitting;

Able to process data with high dimensions(many features)

The training speed is fast;

Parallelization processing can be implemented

Based on the descriptive statistics from Q1 part (a), decide what processing you should apply to the audio features before using them to train each model.

Because some feature column are strongly correlated , it means the features exist **multicollinearity**,so before using them to train each model, we need to solve this probelm.The methods that we can choose are :

# 1. keep important explanatory variables ,and remove secondary or alternative explanatory variables
# 2. Principle component analysis
# 3. Ridge regression
# 4. Stepwise regression
# and so on .

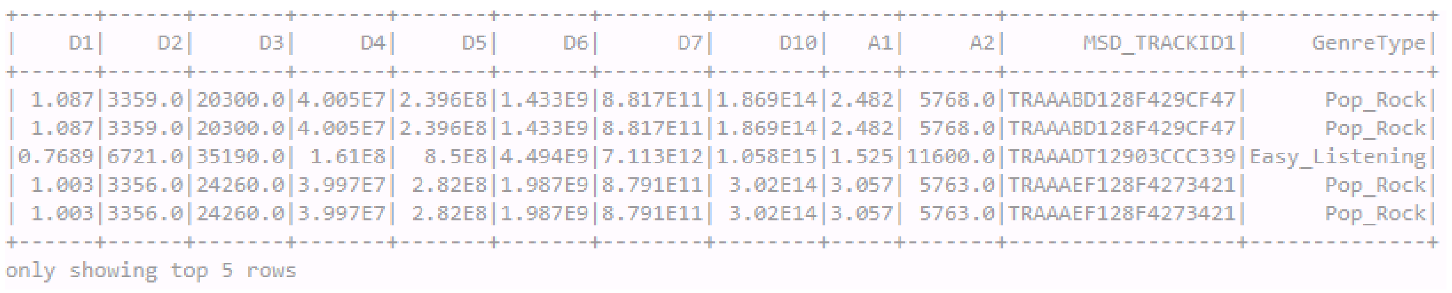
we can choose one method to avoid the multicollinearity.Here, I just choosed the most simple one : remove all the unimportant variables:

["D8","D9","A3","A4","A5","A6","A7","A8","A9","A10"]

Here , I wrote a function to make the removing more efficient:

```
correlation_columns = ["D8","D9","A3","A4","A5","A6","A7","A8","A9","A10"]
 for i in correlation_columns:
 genre_song  = genre_song.drop(i)
```

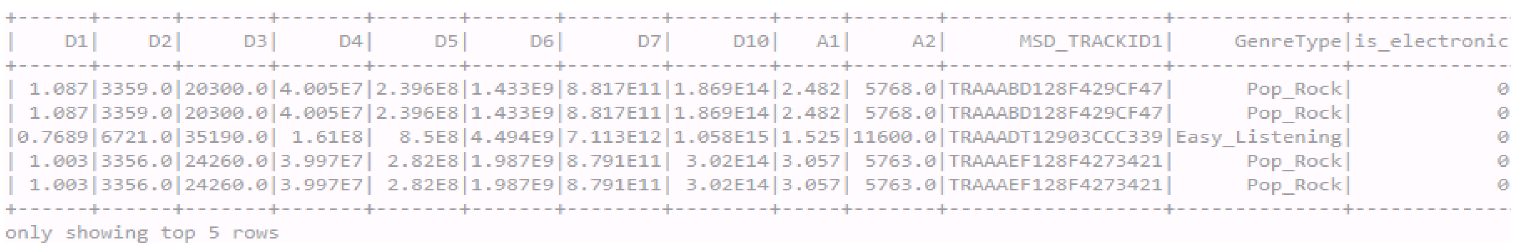
so ,the result :



```
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+
|    D1|    D2|     D3|     D4|     D5|     D6|      D7|     D10|   A1|     A2|       MSD_TRACKID1|     GenreType|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|1.869E14|2.482| 5768.0|TRAAABD128F429CF47|      Pop_Rock|
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|1.869E14|2.482| 5768.0|TRAAABD128F429CF47|      Pop_Rock|
|0.7689|6721.0|35190.0| 1.61E8|  8.5E8|4.494E9|7.113E12|1.058E15|1.525|11600.0|TRAAADT12903CCC339|Easy_Listening|
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11| 3.02E14|3.057| 5763.0|TRAAAEF128F4273421|      Pop_Rock|
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11| 3.02E14|3.057| 5763.0|TRAAAEF128F4273421|      Pop_Rock|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+
only showing top 5 rows
```

(b) Convert the genre column into a column representing if the song is "Electronic" or some other genre as a binary label.

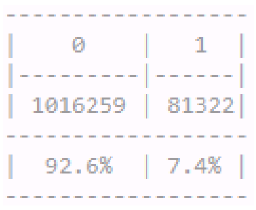
Using **withcolumn** command to create a new column called "**is _electronic**",and if the **GenreType** is electronic, assign the new column with **1**; otherwise ,assign the new column with **0**. Then ,the new table:



```
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+-------------
|    D1|    D2|     D3|     D4|     D5|     D6|      D7|     D10|   A1|     A2|       MSD_TRACKID1|     GenreType|is_electronic
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+-------------
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|1.869E14|2.482| 5768.0|TRAAABD128F429CF47|      Pop_Rock|            0
| 1.087|3359.0|20300.0|4.005E7|2.396E8|1.433E9|8.817E11|1.869E14|2.482| 5768.0|TRAAABD128F429CF47|      Pop_Rock|            0
|0.7689|6721.0|35190.0| 1.61E8|  8.5E8|4.494E9|7.113E12|1.058E15|1.525|11600.0|TRAAADT12903CCC339|Easy_Listening|            0
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11| 3.02E14|3.057| 5763.0|TRAAAEF128F4273421|      Pop_Rock|            0
| 1.003|3356.0|24260.0|3.997E7| 2.82E8|1.987E9|8.791E11| 3.02E14|3.057| 5763.0|TRAAAEF128F4273421|      Pop_Rock|            0
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+--------------+-------------
only showing top 5 rows
```

What is the class balance of the binary label?

for this question, let's check how many percentage of observations are "Electronic" .And how many percentages of observations are others.if the percentage of "Electronic" is much less than others, it means the data are is unbalance.



```
------------------
|    0     |   1  |
|---------|------|
| 1016259 | 81322|
------------------
|  92.6%   | 7.4% |
------------------
```

For this dataset, the percentage of electronic songs is 7.4%, and the percentage of other- type songs is 92.6%.so, now , it is a severely unbalanced dataset.

The result is consistent with the bar chart that we have produced before(majority of songs are pop-rock songs, not electronic.

(c) Split the dataset into training and test sets. Note that you may need to take class balance into account using a sampling method such as stratification, subsampling, or oversampling. Justify your choice of sampling method.

Since the dataset is unbalanced, we need to balance it by the following methods:

# up sample/oversampling :

produce more instances of the under represented class.

# down sample/ undersampling :

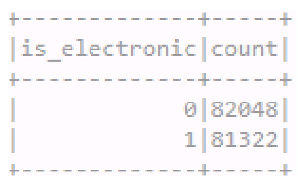
discard instances of over represented class

# both/ stratification :

do a metric of up and down samplimng

Here, I use the **subsampling nethod** to solve the problem, that is picking all of rows that the GenerType of song is electronic, and pick the similar number of rows from other –types songs:

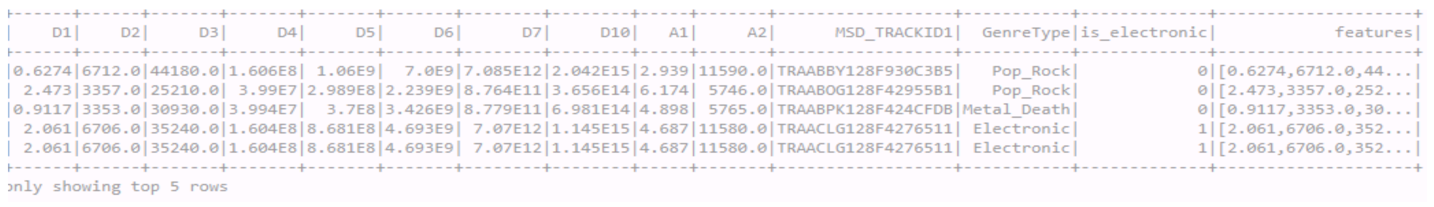
genre_sample = genre_song.sampleBy("is_electronic", fractions={0: 0.08, 1:1.0}, seed=0)



```
+-------------+-----+
|is_electronic|count|
+-------------+-----+
|            0|82048|
|            1|81322|
+-------------+-----+
```

Now, the number of rows that the genertype is electronic is 82048; the other kinds of songs is 81322. The dataset currently has been balanced.

Then, I need to create a new column called “features” to make it suitable for the algorithms that I choosed , the value of “features” is a vector which depends on all the features values.



```
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+-----------+------------+--------------------+
|    D1|    D2|     D3|     D4|     D5|     D6|      D7|     D10|   A1|     A2|       MSD_TRACKID1|  GenreType|is_electronic|            features|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+-----------+------------+--------------------+
|0.6274|6712.0|44180.0|1.606E8| 1.06E9|  7.0E9|7.085E12|2.042E15|2.939|11590.0|TRAABBY128F930C3B5|   Pop_Rock|           0|[0.6274,6712.0,44...|
| 2.473|3357.0|25210.0| 3.99E7|2.989E8|2.239E9|8.764E11|3.656E14|6.174| 5746.0|TRAABOG128F42955B1|   Pop_Rock|           0|[2.473,3357.0,252...|
|0.9117|3353.0|30930.0|3.994E7|  3.7E8|3.426E9|8.779E11|6.981E14|4.898| 5765.0|TRAABPK128F424CFDB|Metal_Death|           0|[0.9117,3353.0,30...|
| 2.061|6706.0|35240.0|1.604E8|8.681E8|4.693E9| 7.07E12|1.145E15|4.687|11580.0|TRAACLG128F4276511| Electronic|           1|[2.061,6706.0,352...|
| 2.061|6706.0|35240.0|1.604E8|8.681E8|4.693E9| 7.07E12|1.145E15|4.687|11580.0|TRAACLG128F4276511| Electronic|           1|[2.061,6706.0,352...|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+-----------+------------+--------------------+
only showing top 5 rows
```

Now, we can split the data , here, I randomsplit the dataset into 60% for traing data, and 40% for test data.

When I tried to print the split result, it is like this:

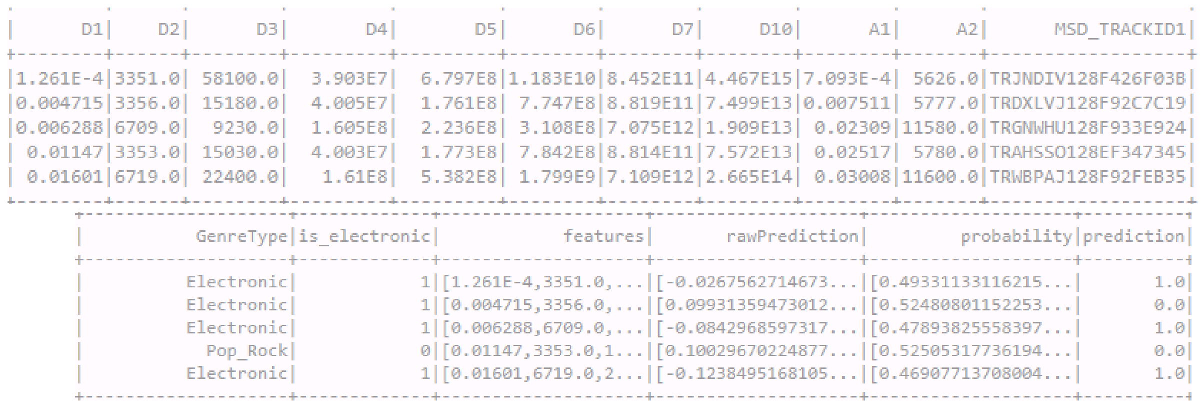
```
Training Dataset Count: 98581

Test Dataset Count: 64611
```

So , 98581 rows are for training ,and 64611 rows are for testing.

(d) Train each of the three classification algorithms that you chose in part (a).

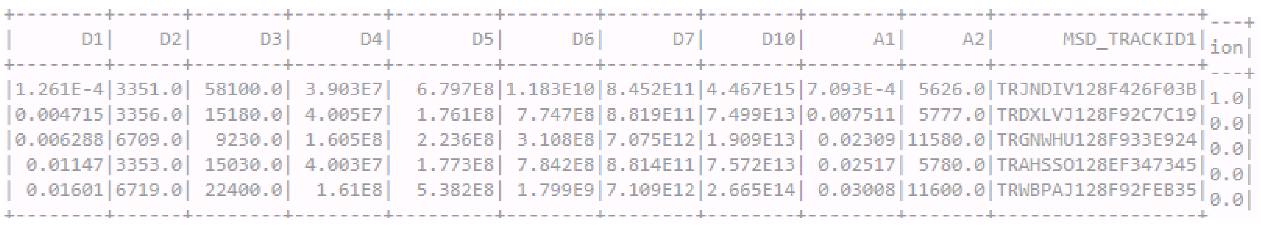
Here, I have trained the three classification algorithms ,and use the each model to predict the test data.

Logistics Regression:



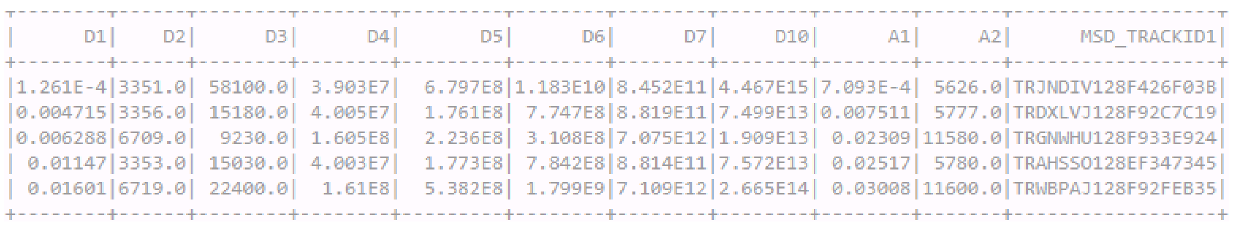
```
|      D1|    D2|       D3|       D4|       D5|       D6|       D7|      D10|       A1|      A2|      MSD_TRACKID1|
+--------+------+---------+---------+---------+---------+---------+---------+---------+-------+------------------+
|1.261E-4|3351.0|  58100.0|  3.903E7|  6.797E8|1.183E10|8.452E11|4.467E15|7.093E-4|  5626.0|TRJNDIV128F426F03B|
|0.004715|3356.0|  15180.0|  4.005E7|  1.761E8|  7.747E8|8.819E11|7.499E13|0.007511|  5777.0|TRDXLVJ128F92C7C19|
|0.006288|6709.0|   9230.0|  1.605E8|  2.236E8|  3.108E8|7.075E12|1.909E13| 0.02309|11580.0|TRGNWHU128F933E924|
| 0.01147|3353.0|  15030.0|  4.003E7|  1.773E8|  7.842E8|8.814E11|7.572E13| 0.02517|  5780.0|TRAHSSO128EF347345|
| 0.01601|6719.0|  22400.0|   1.61E8|  5.382E8|  1.799E9|7.109E12|2.665E14| 0.03008|11600.0|TRWBPAJ128F92FEB35|
+--------+------+---------+---------+---------+---------+---------+---------+---------+-------+------------------+
      +--------------------+------------+--------------------+--------------------+--------------------+----------+
      |           GenreType|is_electronic|            features|        rawPrediction|          probability|prediction|
      +--------------------+------------+--------------------+--------------------+--------------------+----------+
      |          Electronic|           1|[1.261E-4,3351.0,...|[-0.0267562714673...|[0.49331133116215...|       1.0|
      |          Electronic|           1|[0.004715,3356.0,...|[0.09931359473012...|[0.52480801152253...|       0.0|
      |          Electronic|           1|[0.006288,6709.0,...|[-0.0842968597317...|[0.47893825558397...|       1.0|
      |            Pop_Rock|           0|[0.01147,3353.0,1...|[0.10029670224877...|[0.52505317736194...|       0.0|
      |          Electronic|           1|[0.01601,6719.0,2...|[-0.1238495168105...|[0.46907713708004...|       1.0|
      +--------------------+------------+--------------------+--------------------+--------------------+----------+
```

naïve Bayes:

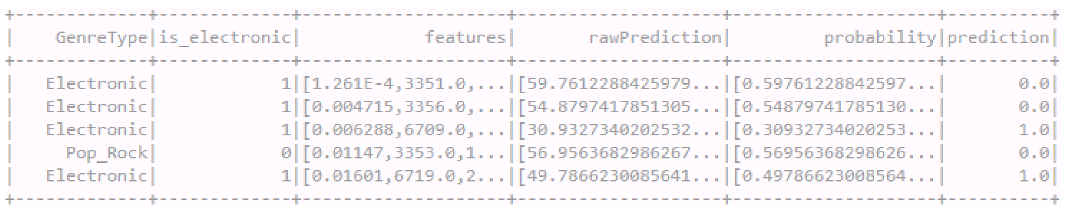


```
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+---+
|      D1|    D2|      D3|      D4|       D5|      D6|      D7|     D10|      A1|     A2|       MSD_TRACKID1|ion|
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+---+
|1.261E-4|3351.0| 58100.0| 3.903E7|  6.797E8|1.183E10|8.452E11|4.467E15|7.093E-4| 5626.0|TRJNDIV128F426F03B|1.0|
|0.004715|3356.0| 15180.0| 4.005E7|  1.761E8| 7.747E8|8.819E11|7.499E13|0.007511| 5777.0|TRDXLVJ128F92C7C19|0.0|
|0.006288|6709.0|  9230.0| 1.605E8|  2.236E8| 3.108E8|7.075E12|1.909E13| 0.02309|11580.0|TRGNWHU128F933E924|0.0|
| 0.01147|3353.0| 15030.0| 4.003E7|  1.773E8| 7.842E8|8.814E11|7.572E13| 0.02517| 5780.0|TRAHSSO128EF347345|0.0|
| 0.01601|6719.0| 22400.0|  1.61E8|  5.382E8| 1.799E9|7.109E12|2.665E14| 0.03008|11600.0|TRWBPAJ128F92FEB35|0.0|
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+
```

Random Forest:



```
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+
|      D1|    D2|      D3|      D4|       D5|      D6|      D7|     D10|      A1|     A2|       MSD_TRACKID1|
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+
|1.261E-4|3351.0| 58100.0| 3.903E7|  6.797E8|1.183E10|8.452E11|4.467E15|7.093E-4| 5626.0|TRJNDIV128F426F03B|
|0.004715|3356.0| 15180.0| 4.005E7|  1.761E8| 7.747E8|8.819E11|7.499E13|0.007511| 5777.0|TRDXLVJ128F92C7C19|
|0.006288|6709.0|  9230.0| 1.605E8|  2.236E8| 3.108E8|7.075E12|1.909E13| 0.02309|11580.0|TRGNWHU128F933E924|
| 0.01147|3353.0| 15030.0| 4.003E7|  1.773E8| 7.842E8|8.814E11|7.572E13| 0.02517| 5780.0|TRAHSSO128EF347345|
| 0.01601|6719.0| 22400.0|  1.61E8|  5.382E8| 1.799E9|7.109E12|2.665E14| 0.03008|11600.0|TRWBPAJ128F92FEB35|
+--------+------+--------+--------+---------+--------+--------+--------+--------+-------+------------------+
```



```
+------------+------------+--------------------+--------------------+--------------------+----------+
|   GenreType|is_electronic|            features|       rawPrediction|         probability|prediction|
+------------+------------+--------------------+--------------------+--------------------+----------+
|  Electronic|           1|[1.261E-4,3351.0,...|[59.7612288425979...|[0.59761228842597...|       0.0|
|  Electronic|           1|[0.004715,3356.0,...|[54.8797417851305...|[0.54879741785130...|       0.0|
|  Electronic|           1|[0.006288,6709.0,...|[30.9327340202532...|[0.30932734020253...|       1.0|
|    Pop_Rock|           0|[0.01147,3353.0,1...|[56.9563682986267...|[0.56956368298626...|       0.0|
|  Electronic|           1|[0.01601,6719.0,2...|[49.7866230085641...|[0.49786623008564...|       1.0|
+------------+------------+--------------------+--------------------+--------------------+----------+
```

(e) Use the test set to compute the compute a range of performance metrics for each model, such as precision, accuracy, and recall.

Now, I has applied each algorithms to the test dataset and get the result about performance metrics : (more details can be get from the codes)

<u>Logistics Regression:</u>

```
um positive: 36254

um negative: 28357

precision: 0.5920174325591658

recall: 0.6623156205640931

accuracy: 0.6017086873752148
```

Naïve Bayes:

```
um positive: 27297

um negative: 37314

precision: 0.4982965161006704

recall: 0.4197370857248658

accuracy: 0.49700515392115896
```

Random Forest:

```
um positive: 29158

um negative: 35453

precision: 0.6427738528019754

recall: 0.5783496883293218

accuracy: 0.6273080435220009
```

(f) Use cross-validation to tune the hyperparameters for each model.

I choose 5-fold cross-validation to complete the job,

Logistics Regression:

```
num positive: 64433

num negative: 178

precision: 0.5014045597752704

recall: 0.9969450101832994

accuracy: 0.5012459178777607
```

Naïve Bayes:

```
num positive: 27297

 num negative: 37314

precision: 0.4982965161006704

recall: 0.4197370857248658

accuracy: 0.49700515392115896
```

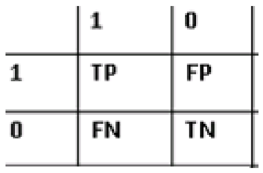
Random Forest:

```
num positive: 29158

 num negative: 35453

 precision: 0.6427738528019754

recall: 0.5783496883293218

 accuracy: 0.6273080435220009
```

How has this changed your performance metrics?

Here, I just analysis the Logistics Regression to give an example about how to use confusion matrix to evaluate the performance about the algorithm. The analysis for other two algorithms is similar.

First, we need to know the component of confusion matrix. Like the table shows, the left 1 and 0 are **predict value**, the top 1 and 0 are **actual value**; so , we can get **TP**(actual:1,predict:1) **FP**(actual:0,predict :1) **FN**(actual:1,predict:0) ,**TN**(actual:0,predict:0)



	1	0
1	TP	FP
0	FN	TN

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

Accuracy = (TP + TN) / total = (TP + TN) / (TP+FP+FN+TN)

According to the result, we can find that the precision has decreased from 0.59 to 0.50, it means among all of the songs that we predicted as “Electronic”, the ratio of the correct result has been decreased; the recall has increase dramatically from 0.66 to 0.99, it suggests that among the all correct predicting results, the percentage about “Electronic” songs is very high; also the accuracy has increased as well ,it shows that we can predict more correct results.

So, generally speaking, after applying cross-validation, the model can performance better than before.

(g) Comment on the relative performance of each model and of the classification algorithms overall, taking into account the performance metrics that you computed in parts (e) and (f).

When comparing these three different algorithms,

precision : Random Forest(0.64), Logistics Regression(0.50),Naïve Bayes(0.49)

Recall : Logistics Regression (0.66), Random Forest (0.58), Naïve Bayes (0.42)

Accuracy: Random Forest (0.62), Logistics Regression (0.60), Naïve Bayes (0.49)

So, in general, Navie Bayes is the worst algorithms for this case, the precision ,recall and accuracy about it were not good; compared with logistics , Random Forest has the better precision and accuracy.

Q3

Next you will extend your work above to predict across all genres .

(a)Look up and explain the difference between the one vs. one and the one vs. all multiclass classification strategies, and describe how you would implement each of these strategies in Spark. Do you expect either one of these strategies to perform better than the other?

The difference between these two strategies is about how to split the data for training. Suppose you have N data-samples with C classes.

One-vs-One: Here, you pick 2 classes at a time, and train a two-class-classifier using samples from the selected two classes only (other samples are ignored in this step). You repeat this for all the two class combinations. So you end up with N(N-1)/2 classifiers. And while testing, you do voting among these classifiers.
One-vs-All : Here, you pick one class and train a two-class-classifier with the samples of the selected class on one side and all the other samples on the other side. Thus, you end up with N classifiers. While testing, you simply classify the sample as belonging to the class with maximum score among the N classifiers.

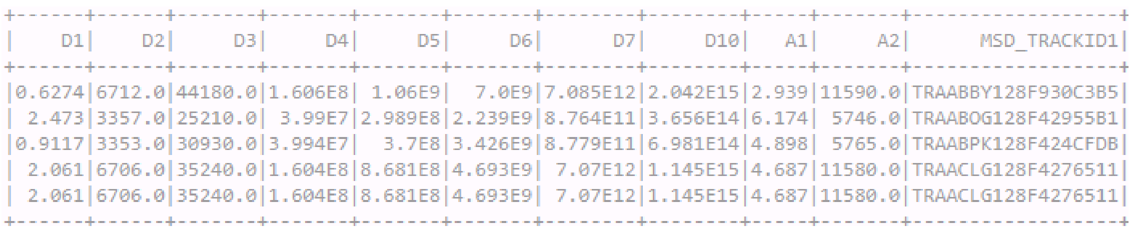
Tips:
It is a bit difficult to understand, so I will give an example here.
suppose we have a 3 class problem, with class labels c1, c2, and c3. And let samples be x1, x2, ... Let the classifiers be f1, f2, ... So, suppose your training data is { {x1, c1}, {x2, c1}, {x3, c2}, {x4,c1}, {x5, c2}, {x6, c3}, {x7, c3} }.
Then:
One-vs-One:
f1: trained with the subset { {x1, c1}, {x2, c1}, {x3, c2}, {x4,c1}, {x5, c2} }, for classes c1 and c2
f2: trained with the subset { {x3, c2}, {x5, c2}, {x6, c3}, {x7, c3} }, for classes c2 and c3.
f3: trained with the subset { {x1, c1}, {x2, c1}, {x4,c1}, {x6, c3}, {x7, c3} }, for classes c2 and c3.
One-vs-All:
f1: trained with { {x1, c1}, {x2, c1}, {x3, ~c1}, {x4,c1}, {x5, ~c1}, {x6, ~c1}, {x7, ~c1} }, for class c1 and the rest (~c1, i.e, NOT c1).
f2: trained with { {x1, ~c2}, {x2, ~c2}, {x3, c2}, {x4, ~c2}, {x5, c2}, {x6, ~c2}, {x7, ~c2} }, for class c2 and the rest (~c2, i.e, NOT c2).
f3: trained with { {x1, ~c3}, {x2, ~c3}, {x3, ~c3}, {x4, ~c3}, {x5, ~c3}, {x6, c3}, {x7, c3} }, for class c3 and the rest (~c3, i.e, NOT c3)

So, in Sark, One VS all (also called “OneVsRest “) is implemented as an `Estimator`. For the base classifier, it takes instances of `Classifier` and creates a binary classification problem for each of the k classes. The classifier for class i is trained to predict whether the label is i or not, distinguishing class i from all other classes. Predictions are done by evaluating each binary classifier and the index of the most confident classifier is output as label.

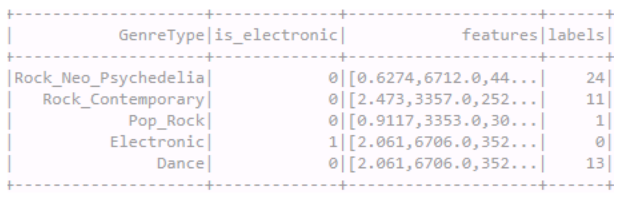
(b) Convert the genre column into an integer index that represents the genre consistently. Find a way that requires the least amount of work by hand.

Here ,I applied the **StringIndexer** to convert the genre column into an integer index ,which is shown in the label column .It means that if one GenreType happens more times , it will be indexed a better/smaller index number. For example, the Pop_Rock takes up the maximum proportion, so it will be assigned with number 1. But ,now the value of label column is double ,not integer, we need to change the type of it.Here, I use **cast** common to change it.

genre_sample.withColumn("labels", F.round(genre_sample.label,2).cast("integer")).drop("label")



```
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+
|    D1|    D2|     D3|     D4|     D5|     D6|      D7|     D10|   A1|     A2|       MSD_TRACKID1|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+
|0.6274|6712.0|44180.0|1.606E8| 1.06E9|  7.0E9|7.085E12|2.042E15|2.939|11590.0|TRAABBY128F930C3B5|
| 2.473|3357.0|25210.0| 3.99E7|2.989E8|2.239E9|8.764E11|3.656E14|6.174| 5746.0|TRAABOG128F42955B1|
|0.9117|3353.0|30930.0|3.994E7|  3.7E8|3.426E9|8.779E11|6.981E14|4.898| 5765.0|TRAABPK128F424CFDB|
| 2.061|6706.0|35240.0|1.604E8|8.681E8|4.693E9| 7.07E12|1.145E15|4.687|11580.0|TRAACLG128F4276511|
| 2.061|6706.0|35240.0|1.604E8|8.681E8|4.693E9| 7.07E12|1.145E15|4.687|11580.0|TRAACLG128F4276511|
+------+------+-------+-------+-------+-------+--------+--------+-----+-------+------------------+
```



```
+-------------------+------------+--------------------+------+
|          GenreType|is_electronic|            features|labels|
+-------------------+------------+--------------------+------+
|Rock_Neo_Psychedelia|           0|[0.6274,6712.0,44...|    24|
|   Rock_Contemporary|           0|[2.473,3357.0,252...|    11|
|            Pop_Rock|           0|[0.9117,3353.0,30...|     1|
|          Electronic|           1|[2.061,6706.0,352...|     0|
|               Dance|           0|[2.061,6706.0,352...|    13|
+-------------------+------------+--------------------+------+
```

only showing top 5 rows

(c)Repeat Q2 parts (c) - (f) using one of the classification algorithms that you used for binary classification, choosing performance metrics that are relevant for multiclass classification instead. Make sure you take into account the class balance in your comments.

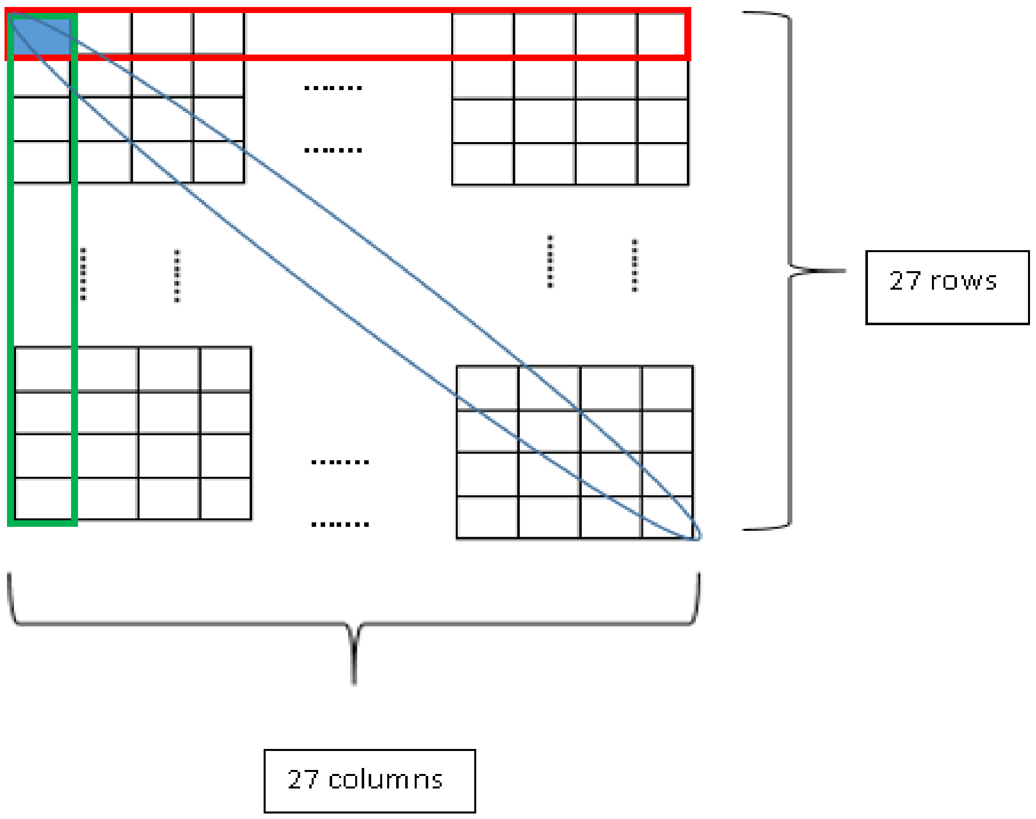
Here, I choose the logistics regression , and using the similar way to calculate the precision , recall and accuracy. First , I tried to print out the whole confusion matrix, but since there are 27 different column ,so if I print it out , it looks so big. So, supposing that we have get this matrix, the structure is like the following picture shows.(you can check the code with the picture together)

the value in blue block is that **prediction is "Electronic"** and **actual is also "Electronic"----TP**

the sum of value in red block is that **prediction is "Electronic"----nP**

the sum of value in green block is that **actual is "Electronic"-------NN**

the blue oval is the sum of value in the clinodiagonal, this sum is that **all correct prediction for each type-------MM**



Then I can use the same logic to calculate each one of them.The result:

```
precision for "Electronic: 0.5234785321800628

recall for "Electronic : 0.9279744346116028

accuracy for "Electronic: 0.45565478010887517
```

How has the performance of your model been affected by the inclusion of multiple genres?

Compared with the binary classification with logistics regression, the precision has decresed from 0.59 to 0.53; recall has increased from 0.66 to 0.92; accuracy has decreased from 0.6 to 0.46.

It means that the logistics regression model has been affected a lot by the inclusion of other genres , causing the result s change a lot.

Song recommendations

Q1

First it will be helpful to know more about the properties of the dataset before you being training the collaborative filtering model.

(a)

How many unique songs are there in the dataset? How many unique users?

For triplets , grouping the user , and then count all of the users, then we can get the number of unique users:

tasteprofile_triplets.groupBy("user").agg(F.count("user").alias("count"))

1,019,318 unique users

Still group the triplets by the song column, then we can get the number of unique songs:

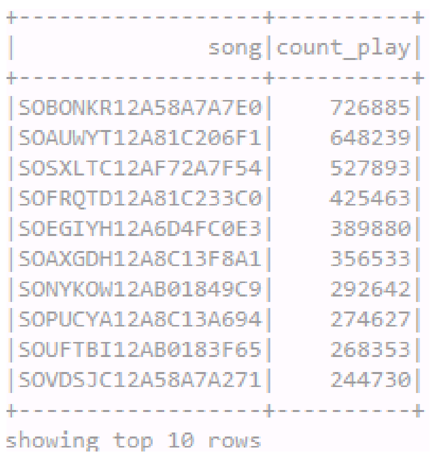
tasteprofile_triplets.groupBy("song").agg(F.count("song").alias("count"))

384,546 unique MSD songs

(b)

How many different songs has the most active user played? What is this as a percentage of the total number of unique songs in the dataset?

Using the result that we grouped by the songs, then sum up the play_count for each song, sort the result by descending order.



```
+------------------+----------+
|              song|count_play|
+------------------+----------+
|SOBONKR12A58A7A7E0|    726885|
|SOAUWYT12A81C206F1|    648239|
|SOSXLTC12AF72A7F54|    527893|
|SOFRQTD12A81C233C0|    425463|
|SOEGIYH12A6D4FC0E3|    389880|
|SOAXGDH12A8C13F8A1|    356533|
|SONYKOW12AB01849C9|    292642|
|SOPUCYA12A8C13A694|    274627|
|SOUFTBI12AB0183F65|    268353|
|SOVDSJC12A58A7A271|    244730|
+------------------+----------+
showing top 10 rows
```

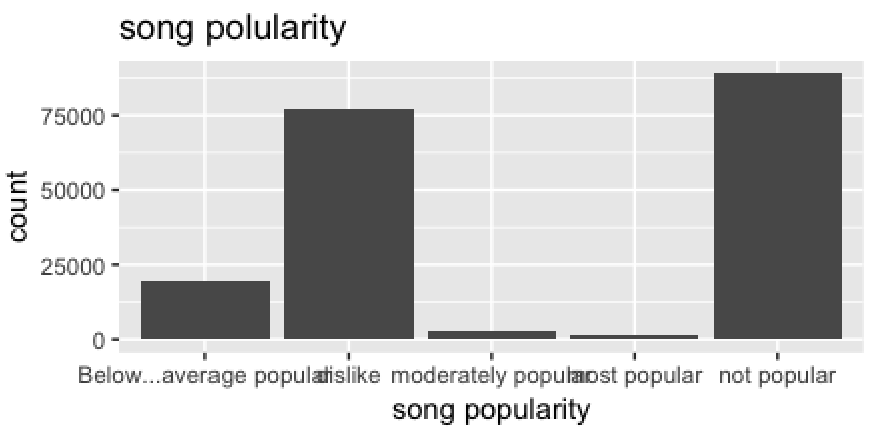
Here, I definited that if the number of play_count of a song **is over 10,000**, the song is a most active. So ,then calculate the sum of the rows which play_count is over 10,000 , we get the result:

1693 songs are most active. The percentage of total number of unique songs Is **4.392%(**1693/38546 = 0.04392154827997717)

(c)

Visualize the distribution of song popularity and the distribution of user activity. What is the shape of these distributions?

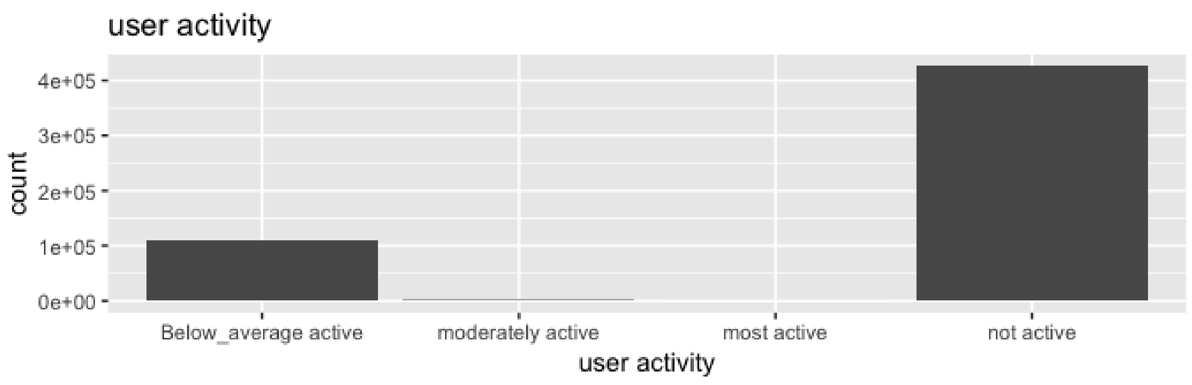
Exact the result that we group the triplets by songs and users ,then save them into HDFS ,and copy the result into my local home directory, using the R language to create the related bar chart to do the visualization:



Here, when I used the R language, I used the **case_when** command to create a ordinal column called popularity, the ordinal has five different levels :

```
data$popularity <- case_when(
            data$count >= 10000 ~ "most popular",
            data$count < 10000 & data$count >= 5000 ~ "moderately popular",
            data$count < 5000 & data$count >= 1000 ~ "Below average popular",
            data$count < 1000 & data$count >= 100~ "not popular",
            data$count < 100 ~ "dislike")
```

From the bar chart ,we can find that the **majority of songs are not popular** (100<= play_count <=1000) **and dislike**(play_count <= 100) , just a very few of songs are most and moderately popular.



Using the same way to create a ordinal column called active , the ordinal has four different levels:

```
data$active <- case_when(
                         data$count >= 1000 ~ "most  active",
                         data$count <1000 & data$count >= 500 ~ "moderately  active",
                        data$count < 500 & data$count >= 100 ~ "Below_average active",
                        data$count < 100 & data$count >= 0~ "not active")
```

From the bar chart ,we can find that the **majority of users are not active** (play_count <100) and **below_average active** (100<=play_count <500) , just a very few of songs are moderately active and most active.

(d)

Collaborative filtering determines similar users and songs based on their combined play history. Songs which have been played only a few times and users who have only listened to a few songs will not contribute much information to the overall dataset and are unlikely to be recommended.

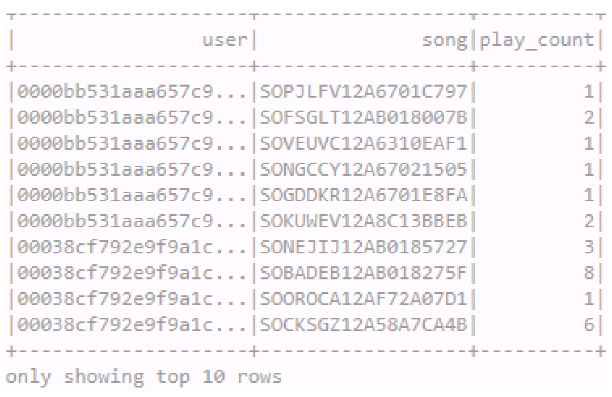
Create a clean dataset of user-song plays by removing songs which have been played less than N times and users who have listened to fewer than M songs in total. Choose sensible values for N and M and justify your choices, taking into account (a) and (b).

First , I have choose the N:10,000 ,which means if a song has been played more than 10,000, it can be viewed most popular, like we have done in question (b), I just want to get the most popular songs; choose M : 100, which means if a user has played the songs more than 100 times , he will be choosen, like we have done in question (c), I want to remove the users who are not acitive.

Here, we have get the result: **1693 songs are most popular ; 112105 users is active.**

So, next , my job is to recommend the most popular songs to the active users.

Here, I remove the songs which are not popular to get the sub-dataset, then remove the users who are not active from the sub-dataset to get the final clean dataset:



```
+--------------------+------------------+----------+
|                user|              song|play_count|
+--------------------+------------------+----------+
|0000bb531aaa657c9...|SOPJLFV12A6701C797|         1|
|0000bb531aaa657c9...|SOFSGLT12AB018007B|         2|
|0000bb531aaa657c9...|SOVEUVC12A6310EAF1|         1|
|0000bb531aaa657c9...|SONGCCY12A67021505|         1|
|0000bb531aaa657c9...|SOGDDKR12A6701E8FA|         1|
|0000bb531aaa657c9...|SOKUWEV12A8C13BBEB|         2|
|00038cf792e9f9a1c...|SONEJIJ12AB0185727|         3|
|00038cf792e9f9a1c...|SOBADEB12AB018275F|         8|
|00038cf792e9f9a1c...|SOOROCA12AF72A07D1|         1|
|00038cf792e9f9a1c...|SOCKSGZ12A58A7CA4B|         6|
+--------------------+------------------+----------+
only showing top 10 rows
```

And the number of rows in this clean dataset is **3,905,520**. Before cleaning the dataset , it has 48,373,586 rows. So, the percentage of this clean dataset is `8.07%` `(3905520/48373586)`.

(e)

Split the user-song plays into training and test sets. Make sure that the test set contains at least 20% of the plays in total.

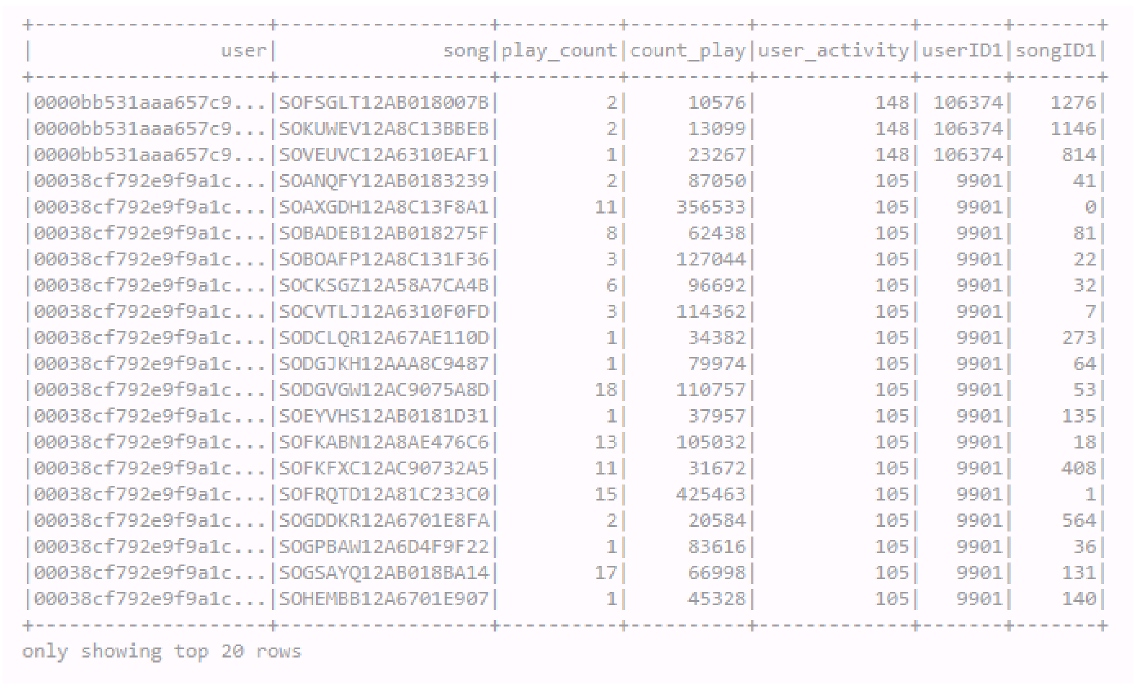
Note that due to the nature of the collaborative filtering model, you must ensure that every user in the test set has some user-song plays in the training set as well. Explain why this is required and how you have done this while keeping the selection as random as possible.

for this question, we must ensure that every user in the test set has some user-song plays in the training set as well ,it means the training and test dataset have the same unique users.

We need to make sure that every user in the test set has some user-song plays in the training set as well, it means that the training data and test data have the same unique users. Because in the next job, when we train the collaborative filtering model, we need to recommend the every user.

Suppose that if the user is just in testing data, not covered in the training data, we can not recommend him according to the training result .

But ,at begining, I use **StringIndexer** and **cast("integer")** command to convert the user and song column into integer index that represents them.(see code to get more details). Now , the clean dataset is like this:



```
+--------------------+------------------+----------+----------+-------------+-------+-------+
|                user|              song|play_count|count_play|user_activity|userID1|songID1|
+--------------------+------------------+----------+----------+-------------+-------+-------+
|0000bb531aaa657c9...|SOFSGLT12AB018007B|         2|     10576|          148| 106374|   1276|
|0000bb531aaa657c9...|SOKUWEV12A8C13BBEB|         2|     13099|          148| 106374|   1146|
|0000bb531aaa657c9...|SOVEUVC12A6310EAF1|         1|     23267|          148| 106374|    814|
|00038cf792e9f9a1c...|SOANQFY12AB0183239|         2|     87050|          105|   9901|     41|
|00038cf792e9f9a1c...|SOAXGDH12A8C13F8A1|        11|    356533|          105|   9901|      0|
|00038cf792e9f9a1c...|SOBADEB12AB018275F|         8|     62438|          105|   9901|     81|
|00038cf792e9f9a1c...|SOBOAFP12A8C131F36|         3|    127044|          105|   9901|     22|
|00038cf792e9f9a1c...|SOCKSGZ12A58A7CA4B|         6|     96692|          105|   9901|     32|
|00038cf792e9f9a1c...|SOCVTLJ12A6310F0FD|         3|    114362|          105|   9901|      7|
|00038cf792e9f9a1c...|SODCLQR12A67AE110D|         1|     34382|          105|   9901|    273|
|00038cf792e9f9a1c...|SODGJKH12AAA8C9487|         1|     79974|          105|   9901|     64|
|00038cf792e9f9a1c...|SODGVGW12AC9075A8D|        18|    110757|          105|   9901|     53|
|00038cf792e9f9a1c...|SOEYVHS12AB0181D31|         1|     37957|          105|   9901|    135|
|00038cf792e9f9a1c...|SOFKABN12A8AE476C6|        13|    105032|          105|   9901|     18|
|00038cf792e9f9a1c...|SOFKFXC12AC90732A5|        11|     31672|          105|   9901|    408|
|00038cf792e9f9a1c...|SOFRQTD12A81C233C0|        15|    425463|          105|   9901|      1|
|00038cf792e9f9a1c...|SOGDDKR12A6701E8FA|         2|     20584|          105|   9901|    564|
|00038cf792e9f9a1c...|SOGPBAW12A6D4F9F22|         1|     83616|          105|   9901|     36|
|00038cf792e9f9a1c...|SOGSAYQ12AB018BA14|        17|     66998|          105|   9901|    131|
|00038cf792e9f9a1c...|SOHEMBB12A6701E907|         1|     45328|          105|   9901|    140|
+--------------------+------------------+----------+----------+-------------+-------+-------+
only showing top 20 rows
```

Then , from my point , I split this clean dataset into 4 steps:

Step1:

Randomsplit data into 70% for training , and 30% for testing;

Then use the **dropDuplicates** command to get the unique user in training and testing data.

n = training_data.dropDuplicates(["user"])

```
m=  testing_data.dropDuplicates(["user"])
```

step2:

get the co-owned users from the two unique user datasets.

```
shared_data = m.join(n,"user","inner")

shared_data = shared_data.select("user")
```

step3:

filter the beginning training and testing data to make them just cover these co-owned users.

```
training_data = training_data.join(shared_data,"user","inner")

testing_data = testing_data.join(shared_data,"user","inner")
```

step4:

check whether the every user that in the test set is also covered in the training set

```
a = training_data.dropDuplicates(["user"])

b =  testing_data.dropDuplicates(["user"])
```

the result has shown that :a = b = shared_data = 109104.

so , we can determine that every user in the test set has user-song-plays in training set as well.

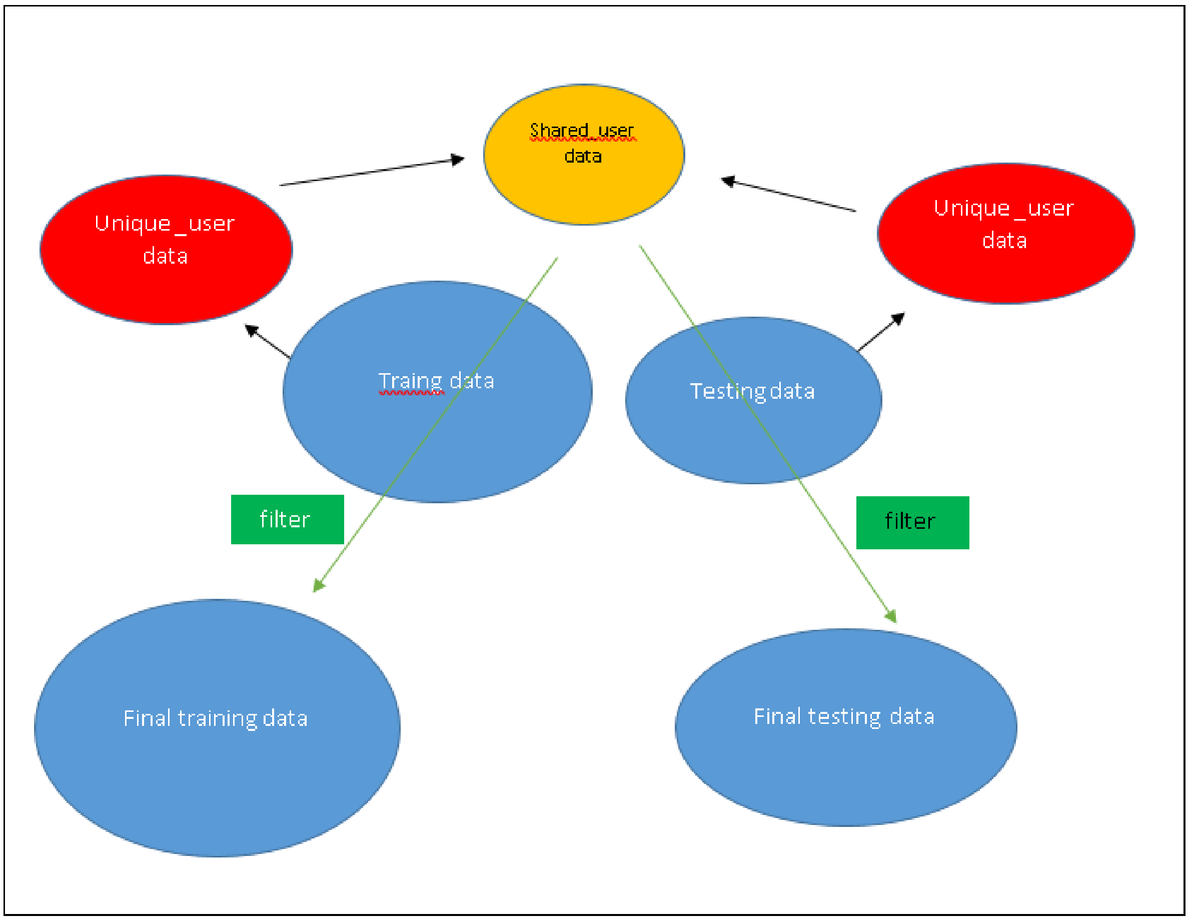
The final split result :

Training Dataset Count: 2721715 (69.90%)

Test Dataset Count: 1172052 (30.10%)

Tips:

I have drawn a bried picture to show the processing:



Q2

Next you will train the collaborative filtering model.

(a) Use the spark.ml library to train an implicit matrix factorization model using Alternating Least Squares (ALS).

Here, use the Alternating Least Squares (ALS) to train the matrix factorization model. here , the parameter in ALS : **usecol** is “userID1”; **itemCol** is “songID”, **ratingCo**l is “play_count”.

from pyspark.ml.recommendation import ALS

als = ALS(userCol="userID1", itemCol="songID1", ratingCol="play_count")

model = als.fit(training_data)

we also can use **dir(model)** to check the model .it has some methods such as

'recommendForAllItems'

'recommendForAllUsers',

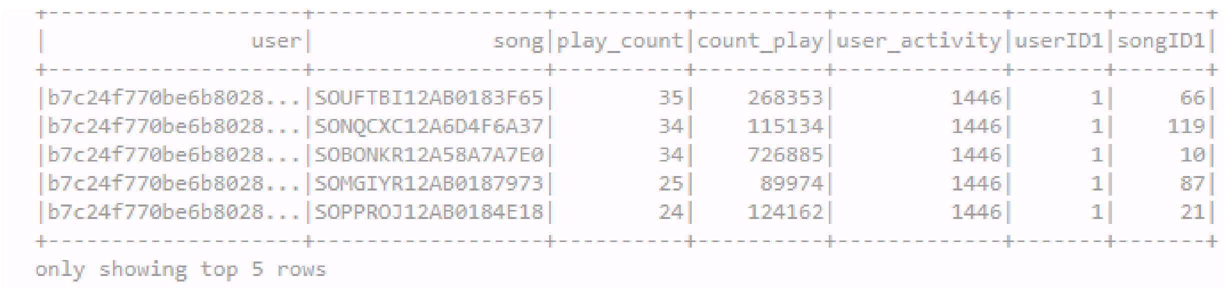
'recommendForItemSubset',

'recommendForUserSubset'

So, in the next part, we can use one of them to give some recommendations.

(b) Select a few of the users from the test set by hand and use the model to generate some recommendations. Compare these recommendations to the songs the user has actually played. Comment on the effectiveness of the collaborative filtering model.

Here, I choose the userID is 1,username is (b7c24f770be6b8028...), then for the test set , I have ordered the songs depending on the play_counts for each song, and get the top 5 songs.



```
+--------------------+------------------+----------+----------+-------------+-------+-------+
|                user|              song|play_count|count_play|user_activity|userID1|songID1|
+--------------------+------------------+----------+----------+-------------+-------+-------+
|b7c24f770be6b8028...|SOUFTBI12AB0183F65|        35|    268353|         1446|      1|     66|
|b7c24f770be6b8028...|SONQCXC12A6D4F6A37|        34|    115134|         1446|      1|    119|
|b7c24f770be6b8028...|SOBONKR12A58A7A7E0|        34|    726885|         1446|      1|     10|
|b7c24f770be6b8028...|SOMGIYR12AB0187973|        25|     89974|         1446|      1|     87|
|b7c24f770be6b8028...|SOPPROJ12AB0184E18|        24|    124162|         1446|      1|     21|
+--------------------+------------------+----------+----------+-------------+-------+-------+
only showing top 5 rows
```

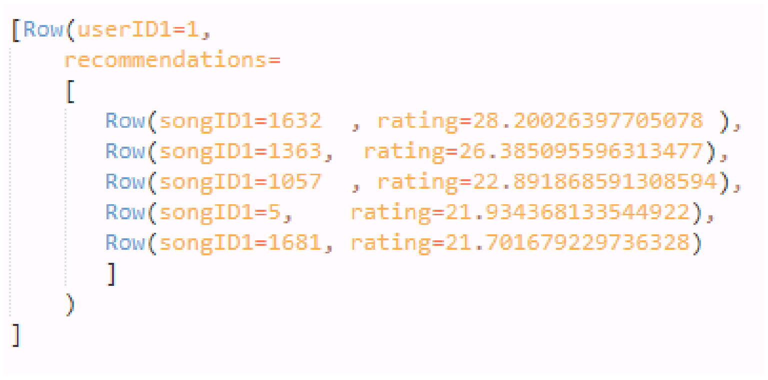
Then , using **recommendForAllUsers** command to recommend every user, select the user whose userID1 is 1) and get the top 5 songs recommendations for this user.

users_recomnend = model.recommendForAllUsers(5)

user1 = users_recomnend.filter(users_recomnend.userID1 ==1)

user1.collect()

The recommendation result:



```
[Row(userID1=1,
    recommendations=
    [
       Row(songID1=1632  , rating=28.20026397705078 ),
       Row(songID1=1363,  rating=26.385095596313477),
       Row(songID1=1057  , rating=22.891868591308594),
       Row(songID1=5,    rating=21.934368133544922),
       Row(songID1=1681, rating=21.701679229736328)
       ]
    )
]
```

when we check the songs'type, majority of the songs that recommended are the same kind of songs that the user has actually played. So, it proves that the collaborative filtering model is very efficient.

(c) Use the test set of user-song plays and recommendations from the collaborative filtering model to compute the following metrics

• Precision @ 5

• NDCG @ 10

• Mean Average Precision (MAP)

Here, I tried to get the **RankingMetrics** for **predictionAnd Labels**, then use this materic to compute these metrics.

metrics = RankingMetrics(predictionAndLabels)

metrics.precisionAt(5)

metrics.ndcgAt(10)

metrics.meanAveragePrecision

But the challenge here is that I failed to get this metrics, because it need to convert the dataframe into RDD, and then the following codes to calculate :

testData = ratings.map(lambda p: (p.user, p.product))

model.recommendForAllUsers()

predictions = model.predictAll(testData).map(lambda r: ((r.user, r.product), r.rating))

ratingsTuple = ratings.map(lambda r: ((r.user, r.product), r.rating))

predictionAndLabels = predictions.join(ratingsTuple).map(lambda tup: tup[1])

So, please give me some suggestions here.

Look up these metrics and explain why they are useful in evaluating the collaborate filtering model.

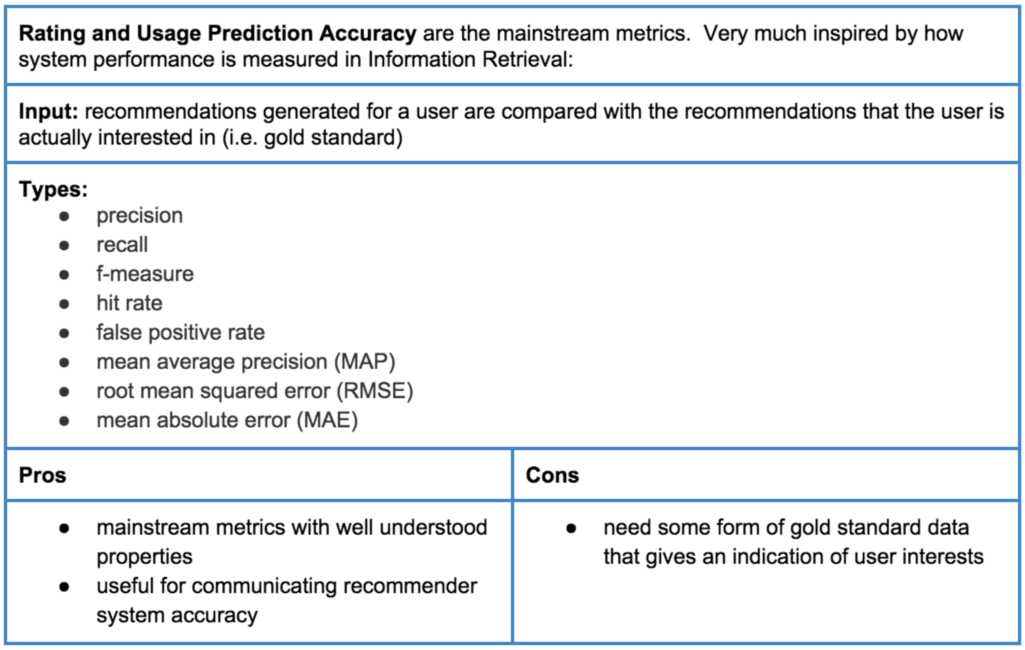
When talking about NDCG @10,we need to be familiar with CG(cumulative gain). For example, if the system recommends five songs for a user, the user gives each rank for each songs(5,3,2,1,2), so the CG is sum of them(13).But, CG has not considered the order of rankings.So, we need to consider it again, and then introduce the DCG(discounted CG). But DCG has not considered the recommendation table and the actual number of valid result in each search, so we use NDCG, NDCG @10 is that NDCG whose ranking level is from 1 to 10. It is between the range(0 to 1). If it is close to 1, it mean the recommendation system is more accurate.
Since it has considered the order of ranking , recommendation table and actual number of valid result, so it is very useful in evaluating the system.

Before learning about MAP, we need to know about AP(Average Precision).first, let's give an example, if we use google to search a keyword, we return 10 results. The best case is that the 10 results are all relevant information that we need. But if only part of it is relevant ,just 5 , and the 5 results are shown in the front, it is also not bad. The worst result is that these 5 five result is returned from the 6th.The AP is compute this . And MAP is the mean of all the users's AP .

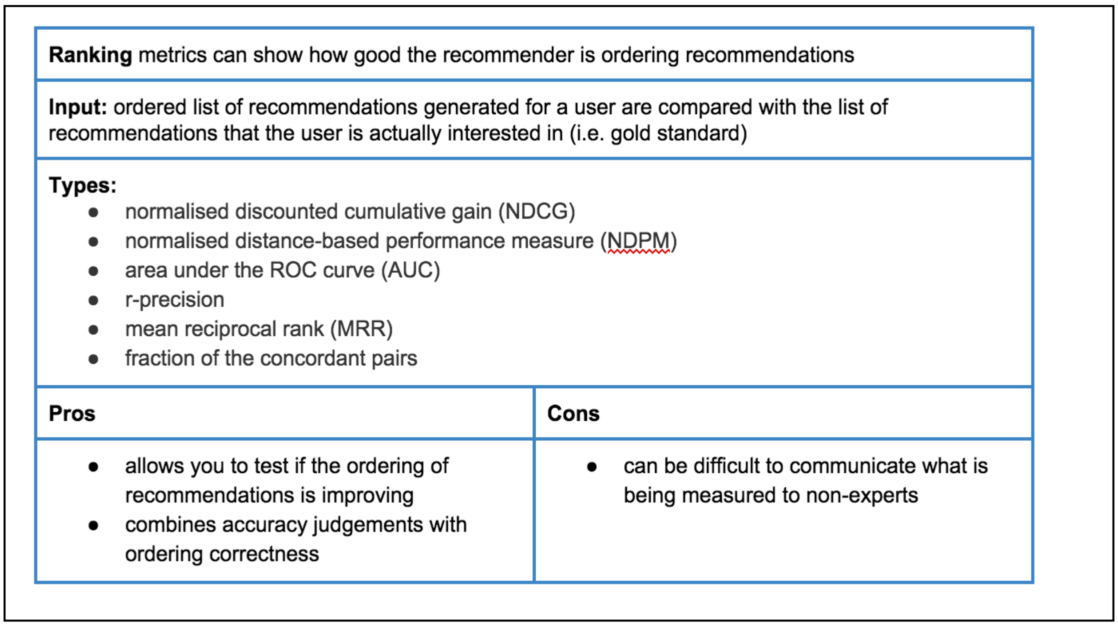
From the definition ,we can find MAP considers the result's sequence.

Tips:

Here is a summary for evaluation metrics from different families:



Rating and Usage Prediction Accuracy are the mainstream metrics. Very much inspired by how system performance is measured in Information Retrieval:	
Input: recommendations generated for a user are compared with the recommendations that the user is actually interested in (i.e. gold standard)	
Types: ● precision ● recall ● f-measure ● hit rate ● false positive rate ● mean average precision (MAP) ● root mean squared error (RMSE) ● mean absolute error (MAE)	
Pros	**Cons**
● mainstream metrics with well understood properties ● useful for communicating recommender system accuracy	● need some form of gold standard data that gives an indication of user interests



Ranking metrics can show how good the recommender is ordering recommendations	
Input: ordered list of recommendations generated for a user are compared with the list of recommendations that the user is actually interested in (i.e. gold standard)	
Types: ● normalised discounted cumulative gain (NDCG) ● normalised distance-based performance measure (NDPM) ● area under the ROC curve (AUC) ● r-precision ● mean reciprocal rank (MRR) ● fraction of the concordant pairs	
Pros	**Cons**
● allows you to test if the ordering of recommendations is improving ● combines accuracy judgements with ordering correctness	● can be difficult to communicate what is being measured to non-experts

Assuming that you could measure future user-song plays based on your recommendations, what other metrics could be useful?

From my point, I think the root mean squared error(RMSE) and mean absolute error(MAE) can be useful.

Q3

The method used to train the collaborative filtering model above is one of many.

(a)Explain the limitations of using this model to generate recommendations for a streaming music service such as Spotify.

The collaborative filtering systems has two different ways:

User-based collaborative filtering algorithm;

item-based collaborative filtering algorithm

Here, the model that we used is user-based collaborative filtering algorithm , it has a limitation about **time-effectiveness**. For example, if a user does some new behaviours, the model might not change the recommendations immediately.It needs time to generate some changes; also, for a new user, if he just does some behaviours for very few items, the recommendation system can not create a personalized recommendations for him, because the user similarity table are calcuted at regular intervals , not immediately.

In addition, this systemis to find users sets that are similar to the target users's interests, and then Fundthe items that the other users in the users sets like, but the target user has not heard before , and recommend these items to the target users. So,users may not be trusting the recommendations that the system made, its **recommendation's convicdon** is not good.

In what situations would this model be unable to produce recommendations for a specific user and why?

Like what I have commented before, if a user does some new behaviours,for example, he listens to some new songs what is the most annoying for him, the model might not change the recommendations immediately,because the user similarity table are calcuted at regular intervals , not immediately.

In what situations would this model produce low quality recommendations and why?

For these situations, this model will produce low quality recommendations:

(1) **New Users Problems**:
for a new user, if he just does some behaviours for very few items, the recommendation system can not create a good recommendation for him, it needs some time and more records;

(2) **Different type of items problems**:
Here, the type is not like the different genretypes in the song-recommendation that we have experienced before. It means that the different product types.the most famous example is Beer & Baby paper pants in wal-mart .This kind of recommendations can not be generated by using the collaborative filtering model.

(b)Suggest two other recommendation systems that could be used in each of these scenarios respectively.

Here, for New Users Problems, I suggest the **knowledge-based** recommendation system.

The collaborative filtering mode usually depends on the vast previous user-items records , but new user have not done enough records, maybe they are just single-pass buyers. For example, a new user bought a luxury goods durning a long time, he can be viewed as a single-pass buyer, and it is very difficult to recommend , because of deficiency of historical information .

But if we applied the knowledge- based recommendation, it can be solved.it relies on detailed knowledge of the itmes'characteristic, and use some personalized approaches to guide the users to find the interesting or useful items among a number of potential candidates.

For Different type of items problems:, I recommendate **the Association Rule-based Recommendation**. It reflects the interdependence and correlation between two things.Then ,making association-rules mining from the customers's purchase record database.

So, this recommendation system is very ideal for the implementation of different types-goods reconmmendations.

(c) Based on the song metadata provided in the million song dataset summary, is there any other business logic that could be applied to the recommendations to improve the real world performance of the system?

From the summary, actually the song metadata has plenty of variables , not only song ID, the track ID.It also has the artist ID, and 51 other fields, such as the year, title, artist tags, and various audio properties such as loudness, beat, tempo, and time signature.

So, we can choose use more variables to build more different recommendation systems. Like recommending the users the songs depend on artists ,or build recommendation systems to recommend the artists which kind of songs they need to write depend on condition about the user – song plays. Applying the different recommendations for different targets, and it can make the whole music industry more and more flourishing.

At last, the users get the songs that they want to appreciate, the artists can write the songs in which they want to express their own emotions and get the economic benefits that they deserve. Also, music companies can provide better services.