The Million Song Dataset (MSD)

Data Analysis using Spark

Name: Shi Chen
E-mail: yuejianyingmm@icloud.com

Contents

Background Information

Million Song Dataset (MSD) is an integrated database of Million Songs. This project initiated by The Echo Nest and LabROSA. There are other 7 datasets are involved in the this study. However, in this report, it will focus on the MSD, Taste Profile and Top MAGD datasets. It mainly discuss the audio similarity by using machine learning model to classify the genre of songs and try to do the song recommendations.

Processing Part

Q1

Q1 (a) Directory tree

```
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -ls -R hdfs:///data/msd | awk '{print $8}'| sed -e 's/[^-][^\/]*\//--/g' -e
's/^/ /' -e 's/-/ | /'
  | -------audio
  | ---------attributes
  | -----------msd-jmir-area-of-moments-all-v1.0.attributes.csv
  | -----------msd-jmir-lpc-all-v1.0.attributes.csv
  | -----------msd-jmir-methods-of-moments-all-v1.0.attributes.csv
  | -----------msd-jmir-mfcc-all-v1.0.attributes.csv
  | -----------msd-jmir-spectral-all-all-v1.0.attributes.csv
  | -----------msd-jmir-spectral-derivatives-all-all-v1.0.attributes.csv
  | -----------msd-marsyas-timbral-v1.0.attributes.csv
  | -----------msd-mvd-v1.0.attributes.csv
  | -----------msd-rh-v1.0.attributes.csv
  | -----------msd-rp-v1.0.attributes.csv
  | -----------msd-ssd-v1.0.attributes.csv
  | -----------msd-trh-v1.0.attributes.csv
  | -----------msd-tssd-v1.0.attributes.csv
  | ---------features
  | -----------msd-jmir-area-of-moments-all-v1.0.csv
  | -------------part-00000.csv.gz
  | -------------part-00001.csv.gz
```

......

```
 | ---------statistics
 | -----------sample_properties.csv.gz
 | -------genre
 | ---------msd-MAGD-genreAssignment.tsv
 | ---------msd-MASD-styleAssignment.tsv
 | ---------msd-topMAGD-genreAssignment.tsv
 | -------main
 | ---------summary
 | -----------analysis.csv.gz
 | -----------metadata.csv.gz
 | -------tasteprofile
 | ---------mismatches
 | -----------sid_matches_manually_accepted.txt
 | -----------sid_mismatches.txt
 | ---------triplets.tsv
 | -----------part-00000.tsv.gz
 | -----------part-00001.tsv.gz
 | -----------part-00002.tsv.gz
 | -----------part-00003.tsv.gz
 | -----------part-00004.tsv.gz
 | -----------part-00005.tsv.gz
 | -----------part-00006.tsv.gz
 | -----------part-00007.tsv.gz
```

The datasets include four directories: audio, genre, main and tasteprof, in each directory, there are several subdirectory. In each sub directory, there are several sub-sub directory, some of them using *".cv"* format or *".tsv "* format, beneath that the auto file format for large size is *".csv.gz"* or *".tsv.gz"*. Some small size files are using *".txt"* format.

The datasets contain numeric data, and string.

The 24 directories are paralleled distributed in 204 blocks in the different work nodes in HDFS. When the size of the file is less than 128MB, it will be store in one block, if the size of the file is larger than 128 MB, it will be distributed in more than one block.

```
Status: HEALTHY
 Number of data-nodes:  32
 Number of racks:               1
 Total dirs:                    24
 Total symlinks:                0

Replicated Blocks:
 Total size:    13896584474 B
 Total files:   133
 Total blocks (validated):      204 (avg. block size 68120512 B)
 Minimally replicated blocks:   204 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    4
 Average block replication:     8.0
 Missing blocks:                0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)

Erasure Coded Block Groups:
 Total size:    0 B
 Total files:   0
 Total block groups (validated):        0
 Minimally erasure-coded block groups:  0
 Over-erasure-coded block groups:       0
 Under-erasure-coded block groups:      0
 Unsatisfactory placement block groups: 0
 Average block group size:      0.0
 Missing block groups:          0
 Corrupt block groups:          0
 Missing internal blocks:       0
FSCK ended at Tue Sep 29 20:24:45 NZDT 2020 in 6 milliseconds


The filesystem under path '/data/msd' is HEALTHY
```

Q1 (b)

"rdd = sc.parallelize()…rdd.repartiton(n)" it can help to define the number of RDDs. It is good for unbalanced size and cannot parallel loaded files. In this case, most of the single files (especially gzip files) are less 640 MB, it seems unnecessary to repartition, it will add extra shuffle cost.

Q1 (C)

The large files are in "/data/msd/audio/features/" directory. So we count the number of rows in each sub directory of features, we got the number of lines (songs) of each datasets. They all less than 1000,000. It means there may be some missing records.

Output

```
# Q1 (C) How many lines in each datasets
hdfs dfs -cat /data/msd/audio/attributes/* | wc -l                                                        # 3929

hdfs dfs -cat /data/msd/audio/features/msd-jmir-area-of-moments-all-v1.0.csv/* | gunzip | wc -l             # 994623
hdfs dfs -cat /data/msd/audio/features/msd-jmir-lpc-all-v1.0.csv/* | gunzip | wc -l                          # 994623
hdfs dfs -cat /data/msd/audio/features/msd-jmir-methods-of-moments-all-v1.0.csv/* | gunzip | wc -l           # 994623
hdfs dfs -cat /data/msd/audio/features/msd-jmir-mfcc-all-v1.0.csv/* | gunzip | wc -l                         # 994623
hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-all-all-v1.0.csv/* | gunzip | wc -l                 # 994623
hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-derivatives-all-all-v1.0.csv/* | gunzip | wc -l     # 994623
hdfs dfs -cat /data/msd/audio/features/msd-marsyas-timbral-v1.0.csv/* | gunzip | wc -l                       # 995001
hdfs dfs -cat /data/msd/audio/features/msd-mvd-v1.0.csv/* | gunzip | wc -l                                   # 994188
hdfs dfs -cat /data/msd/audio/features/msd-rh-v1.0.csv/* | gunzip | wc -l                                    # 994188
hdfs dfs -cat /data/msd/audio/features/msd-rp-v1.0.csv/* | gunzip | wc -l                                    # 994188
hdfs dfs -cat /data/msd/audio/features/msd-ssd-v1.0.csv/* | gunzip | wc -l                                   # 994188
hdfs dfs -cat /data/msd/audio/features/msd-trh-v1.0.csv/* | gunzip | wc -l                                   # 994188
hdfs dfs -cat /data/msd/audio/features/msd-tssd-v1.0.csv/* | gunzip | wc -l                                  # 994188

hdfs dfs -cat /data/msd/audio/statistics/* | gunzip | wc -l                                                  # 992866

hdfs dfs -cat /data/msd/genre/msd-MAGD-genreAssignment.tsv | wc -l                                           # 422714
hdfs dfs -cat /data/msd/genre/msd-MASD-styleAssignment.tsv | wc -l                                           # 273936
hdfs dfs -cat /data/msd/genre/msd-topMAGD-genreAssignment.tsv | wc -l                                        # 406427

hdfs dfs -cat /data/msd/main/summary/analysis.csv.gz |gunzip | wc -l                                         # 1000001
hdfs dfs -cat /data/msd/main/summary/metadata.csv.gz |gunzip | wc -l                                         # 1000001

hdfs dfs -cat /data/msd/tasteprofile/mismatches/sid_matches_manually_accepted.txt | wc -l                    # 938
hdfs dfs -cat /data/msd/tasteprofile/mismatches/sid_mismatches.txt | wc -l                                   # 19094
hdfs dfs -cat /data/msd/tasteprofile/triplets.tsv/* | gunzip | wc -l                                         # 48373586
```

```
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-area-of-moments-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-lpc-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-methods-of-moments-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-mfcc-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-all-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-jmir-spectral-derivatives-all-all-v1.0.csv/* | gunzip | wc -l
994623
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-marsyas-timbral-v1.0.csv/* | gunzip | wc -l
995001
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-mvd-v1.0.csv/* | gunzip | wc -l
994188
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-rh-v1.0.csv/* | gunzip | wc -l
994188
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-rp-v1.0.csv/* | gunzip | wc -l
994188
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-ssd-v1.0.csv/* | gunzip | wc -l
994188
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-trh-v1.0.csv/* | gunzip | wc -l
994188
[sch405@canterbury.ac.nz@mathmadslinux1p ~]$ hdfs dfs -cat /data/msd/audio/features/msd-tssd-v1.0.csv/* | gunzip | wc -l
994188
```

Q2

Q2 (a)

Strategy

(1) Remove the records which should not be put into mismatched by using left_anti join.
(2) Remove the mismatched records from triplets by using left_anti joi

Step

(1) Preparing configuration: import libraries, define SparkSession, SparkContext, and compute suitable number of partitions. As shown below, there are 32 default partitions.

```
# -----------------------------------------------------------------------------
# Imports
# -----------------------------------------------------------------------------


# Python and pyspark modules required

import sys

from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.sql import functions as F


# Required to allow the file to be submitted and run using spark-submit instead
# of using pyspark interactively

spark = SparkSession.builder.getOrCreate()
sc = SparkContext.getOrCreate()

# Compute suitable number of partitions

conf = sc.getConf()

N = int(conf.get("spark.executor.instances"))
M = int(conf.get("spark.executor.cores"))
partitions = 4 * N * M      # 32
```

(2) Define the schema and load data to python (master node memory), read the file into lines and split one line into separate feature strings and store them in a tuple structure. Then repeat last step with “for loop” to each line in the lines. Then combine all the line tuple into a new list. We can do that because the size of the file is not big (938 lines).

```
# -----------------------------------------------------------------------------
# Load
# -----------------------------------------------------------------------------

# Define the data frame schema
mismatches_schema = StructType([
    StructField("song_id", StringType(), True),
    StructField("song_artist", StringType(), True),
    StructField("song_title", StringType(), True),
    StructField("track_id", StringType(), True),
    StructField("track_artist", StringType(), True),
    StructField("track_title", StringType(), True)
])

# Load the data from local file to master node python (not spark), and doing the data processing directly.
# The dataset uploading should be small, because this process is happen in master node memory.
with open("/scratch-network/courses/2020/DATA420-20S2/data/msd/tasteprofile/mismatches/sid_matches_manually_accepted.txt", "r") as f:   #open the local file read as f
    lines = f.readlines()                           # assign object "lines" as a list which contains each line in the
    sid_matches_manually_accepted = []              # A new list which used to store the information of each line
    for line in lines:                              # Iterate the line string process (split the feature string and comebine them with tuple structure) to each line
        if line.startswith("< ERROR: "):            # Find the target information pattern and skip the useless information
            a = line[10:28]                         # Using index to stract the "song_id" information as a string.
            b = line[29:47]                         # Using index to stract the "track_id" information as a string.
            c, d = line[49:-1].split("  !=  " )     # Using '!=' to split the residual string into two parts.
            e, f = c.split("  -  ")                 # Using '-' to split c into two parts "song_artist" and "song_title".
            g, h = d.split("  -  ")                 # Using '-' to split d into two parts "track_artist" and "track_title".
            sid_matches_manually_accepted.append((a, e, f, b, g, h))   # append the tuple of each line into the new list during the for loop.

matches_manually_accepted = spark.createDataFrame(sc.parallelize(sid_matches_manually_accepted, 8), schema=mismatches_schema)
                                                    # upload and parallelize the result list into spark as dataframe with the defined schema
matches_manually_accepted.cache()                   # cache the dataframe
matches_manually_accepted.show(10, 40)

print(matches_manually_accepted.count())            # 488   Why it is not 938?
```

```python
with open("/scratch-network/courses/2020/DATA420-20S2/data/msd/tasteprofile/mismatches/sid_mismatches.txt", "r") as f:
    lines = f.readlines()
    sid_mismatches = []
    for line in lines:
        if line.startswith("ERROR: "):
            a = line[8:26]
            b = line[27:45]
            c, d = line[47:-1].split("  !=  ")
            e, f = c.split("  -  ")
            g, h = d.split("  -  ")
            sid_mismatches.append((a, e, f, b, g, h))

mismatches = spark.createDataFrame(sc.parallelize(sid_mismatches, 64), schema=mismatches_schema)
mismatches.cache()
mismatches.show(10, 40)

print(mismatches.count())  # 19094
```

```
In [14]: matches_manually_accepted.show(10, 40)
    ...:
+------------------+----------------+----------------------------------------+------------------+----------------------------------------+----------------------------------------+
|           song_id|     song_artist|                              song_title|          track_id|                            track_artist|                             track_title|
+------------------+----------------+----------------------------------------+------------------+----------------------------------------+----------------------------------------+
|SOFQHZM12A8C142342|    Josipa Lisac|                                 razloga|TRMWMFG128F92FFEF2|                            Lisac Josipa|                            1000 razloga|
|SODXUTF12AB018A3DA|      Lutan Fyah|     Nuh Matter the Crisis Feat. Midnite|TRMWPCD12903CCE5ED|                                 Midnite|                   Nah Matter the Crisis|
|SOASCRF12A8C1372E6|Gaetano Donizetti|L'Elisir d'Amore: Act Two: Come sen v...|TRMHIPJ128F426A2E2|Gianandrea Gavazzeni_ Orchestra E Cor...|L'Elisir D'Amore_ Act 2: Come Sen Va ...|
|SOITDUN12A58A7AACA|     C.J. Chenier|                              Ay, Ai Ai|TRMHXGK128F42446AB|                          Clifton Chenier|                               Ay_ Ai Ai|
|SOLZXUM12AB018BE39|           許志安|                                男人最痛|TRMRSOF12903CCF516|                                Andy Hui|                        Nan Ren Zui Tong|
|SOTJTDT12A8C13A8A6|               S|                                       h|TRMNKQE128F427C4D8|                             Sammy Hagar|                 20th Century Man (Live)|
|SOGCVWB12AB0184CE2|               H|                                       Y|TRMUNCZ128F932A95D|                                Hawkwind|                25 Years (Alternate Mix)|
|SOKDKGD12AB0185E9C|     影山ヒロノブ|Cha-La Head-Cha-La (2005 ver./DRAGON ...|TRMOOAH12903CB4B29|                         Takahashi Hiroki|Maka fushigi adventure! (2005 Version...|
|SOPPBXP12A8C141194|    Αντώνης Ρέμος|                        O Trellos - Live|TRMXJDS128F42AE7CF|                            Antonis Remos|                               O Trellos|
|SODQSLR12A8C133A01|   John Williams|Concerto No. 1 for Guitar and String ...|TRWHMXN128F426E03C|                 English Chamber Orchestra|II. Andantino siciliano from Concerto...|
+------------------+----------------+----------------------------------------+------------------+----------------------------------------+----------------------------------------+
```

```
In [12]: mismatches = spark.createDataFrame(sc.parallelize(sid_mismatches, 64), schema=mismatches_schema)
    ...: mismatches.cache()
    ...: mismatches.show(10, 40)
+------------------+------------------+----------------------------------------+------------------+----------------+----------------------------------------+
|           song_id|       song_artist|                              song_title|          track_id|    track_artist|                             track_title|
+------------------+------------------+----------------------------------------+------------------+----------------+----------------------------------------+
|SOUMNSI12AB0182807|Digital Underground|                        The Way We Swing|TRMMGKQ128F9325E10|        Linkwood|         Whats up with the Underground|
|SOCMRBE12AB018C546|        Jimmy Reed|The Sun Is Shining (Digitally Remaste...|TRMMREB12903CEB1B1|      Slim Harpo|              I Got Love If You Want It|
|SOLPHZY12AC468ABA8|     Africa HiTech|                                Footstep|TRMMBOC12903CEB46E|  Marcus Worgull|                Drumstern (BONUS TRACK)|
|SONGHTM12A8C1374EF|    Death in Vegas|                            Anita Berber|TRMMITP128F425D8D0|       Valen Hsu|                                  Shi Yi|
|SONGXCA12A8C13E82E| Grupo Exterminador|                            El Triunfador|TRMMAYZ128F429ECE6|       I Ribelli|                               Lei M'Ama|
|SOMBCRC12A67ADA435|     Fading Friend|                             Get us out!|TRMMNVU128EF343EED|      Masterboy|                    Feel The Heat 2000|
|SOTDWDK12A8C13617B|      Daevid Allen|                              Past Lives|TRMMNCZ128F426FF0E|  Bhimsen Joshi|            Raga - Shuddha Sarang_ Aalap|
|SOEBURP12AB018C2FB|  Cristian Paduraru|                              Born Again|TRMMPBS12903CE90E1|      Yespiring|                         Journey Stages|
|SOSRJHS12A6D4FDAA3|        Jeff Mills|                       Basic Human Design|TRMWMEL128F421DA68|            M&T|                          Drumsettester|
|SOIYAAQ12A6D4F954A|          Excepter|                                      OG|TRMWHRI128F147EA8E|     The Fevers|Não Tenho Nada (Natchs Scheint Die So...|
+------------------+------------------+----------------------------------------+------------------+----------------+----------------------------------------+
only showing top 10 rows
```

```python
triplets_schema = StructType([
    StructField("user_id", StringType(), True),
    StructField("song_id", StringType(), True),
    StructField("plays", IntegerType(), True)
])

# The triplets file is large, so load to spark from hdfs directly.
triplets = (
    spark.read.format("csv")
    .option("header", "false")
    .option("delimiter", "\t")
    .option("codec", "gzip")
    .schema(triplets_schema)
    .load("hdfs:///data/msd/tasteprofile/triplets.tsv/")
    .cache()
)
triplets.cache()
triplets.show(10, 50)
triplets.count()     # 48373586

# Remove the records which should not be put into mismatched by using left_anti join  # Drop only one record...why
mismatches_not_accepted = mismatches.join(matches_manually_accepted, on="song_id", how="left_anti")

# Remove the mismatched records from triplets by using left_anti join  # Drop 2,578,475 mismatched records
triplets_not_mismatched = triplets.join(mismatches_not_accepted, on="song_id", how="left_anti")

# Repartition the result to make it more balance to computate
triplets_not_mismatched = triplets_not_mismatched.repartition(partitions).cache()

print(mismatches_not_accepted.count())  # 19093
print(triplets.count())                 # 48373586
print(triplets_not_mismatched.count())  # 45795111
```

Q2 (b) Schema

This will organise our dataset in formal way, which will benefit the model pre-processing. It is because some model can only dealing with numeric values.

Process

(1) Create a list store the names of all the “features” datasets.
(2) Create a schema dictionary by using the information in attributes datasets to store the feature variable name and correlated schema type. This can be used as schema for the features dataset.

(3) Schema can only be the StructType or string.
(4) In the schema, the column name can be redefined in more simple way, it will save memory.

Code

```
# Processing Q2 (b)

# The command below is checking the unique attributes schema in the attributes file in hdfs
# hdfs dfs -cat "/data/msd/audio/attributes/*" | awk -F',' '{print $2}' | sort | uniq

# NUMERIC
# real
# real
# string
# string
# STRING

# Map the attribute type into formal schema data type
audio_attribute_type_mapping = {
  "NUMERIC": DoubleType(),
  "real": DoubleType(),
  "string": StringType(),
  "STRING": StringType()
}
```

```
# Map the attribute type into formal schema data type
audio_attribute_type_mapping = {
  "NUMERIC": DoubleType(),
  "real": DoubleType(),
  "string": StringType(),
  "STRING": StringType()
}

# Create a list store all the audio dataset names in the audio/attributes
audio_dataset_names = [
  "msd-jmir-area-of-moments-all-v1.0",
  "msd-jmir-lpc-all-v1.0",
  "msd-jmir-methods-of-moments-all-v1.0",
  "msd-jmir-mfcc-all-v1.0",
  "msd-jmir-spectral-all-all-v1.0",
  "msd-jmir-spectral-derivatives-all-all-v1.0",
  "msd-marsyas-timbral-v1.0",
  "msd-mvd-v1.0",
  "msd-rh-v1.0",
  "msd-rp-v1.0",
  "msd-ssd-v1.0",
  "msd-trh-v1.0",
  "msd-tssd-v1.0"
]
audio_dataset_schemas = {}    # Creat a new dictionary
for audio_dataset_name in audio_dataset_names:
  print(audio_dataset_name)

  audio_dataset_path = f"/scratch-network/courses/2020/DATA420-20S2/data/msd/audio/attributes/{audio_dataset_name}.attributes.csv"
  with open(audio_dataset_path, "r") as f:
    rows = [line.strip().split(",") for line in f.readlines()]
    # string strip() function will remove spaces at the beginning and at the end of the string:

  # you could rename feature columns with a short generic name

    rows[-1][0] = "track_id"
    for i, row in enumerate(rows[0:-1]):
        row[0] = f"feature_{i:04d}"

# Define the schema of audio_dataset
  audio_dataset_schemas[audio_dataset_name] = StructType([
    StructField(row[0], audio_attribute_type_mapping[row[1]], True) for row in rows
  ])

  s = str(audio_dataset_schemas[audio_dataset_name])
  print(s[0:50] + " ... " + s[-50:])
```

```
# msd-jmir-area-of-moments-all-v1.0
# StructType(List(StructField(Area_Method_of_Moments ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-jmir-lpc-all-v1.0
# StructType(List(StructField(LPC_Overall_Standard_D ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-jmir-methods-of-moments-all-v1.0
# StructType(List(StructField(Method_of_Moments_Over ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-jmir-mfcc-all-v1.0
# StructType(List(StructField(MFCC_Overall_Standard_ ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-jmir-spectral-all-all-v1.0
# StructType(List(StructField(Spectral_Centroid_Over ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-jmir-spectral-derivatives-all-all-v1.0
# StructType(List(StructField(Spectral_Centroid_Over ... e,true),StructField(MSD_TRACKID,StringType,true)))
# msd-marsyas-timbral-v1.0
# StructType(List(StructField(Mean_Acc5_Mean_Mem20_Z ... Type,true),StructField(track_id,StringType,true)))
# msd-mvd-v1.0
# StructType(List(StructField("component_0",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
# msd-rh-v1.0
# StructType(List(StructField("component_0",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
# msd-rp-v1.0
# StructType(List(StructField("component_1",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
# msd-ssd-v1.0
# StructType(List(StructField("component_0",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
# msd-trh-v1.0
# StructType(List(StructField("component_0",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
# msd-tssd-v1.0
# StructType(List(StructField("component_1",DoubleTy ... ,true),StructField(instanceName,StringType,true)))
```

Audio similarity

Q1

Q1 (a) Audio features correlation

Process

(1) Clean data, remove the mismatches observations.
(2) Schema
(3) Correlation logic, detect all the correlations value larger than threshold then minimize the number of features (diagonal correlation value = "1")
(4) Remove the high correlated features and check the new correlation again

Output

Features dataset: "/data/msd/audio/features/msd-jmir-spectral-all-all-v1.0.csv/"
Number of features: 16
Threshold = 0.7
num_correlated_columns_not_the_same: 40

There are strong correlated feature pairs (correlation > 0.7):
[(0, 1), (0, 7), (1, 7), (2, 4), (2, 5), (2, 10), (2, 12), (2, 13), (4, 5), (4, 10), (4, 12), (4, 13), (5, 10), (5, 12), (8, 9), (8, 15), (9, 15), (10, 12), (10, 13), (12, 13)]

Number of features after removed the high correlated features: 8

IPython: home/sch405

```
In [23]: print((correlations > threshold).astype(int))
[[1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0]
 [0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1]
 [0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0]
 [0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1]]
```

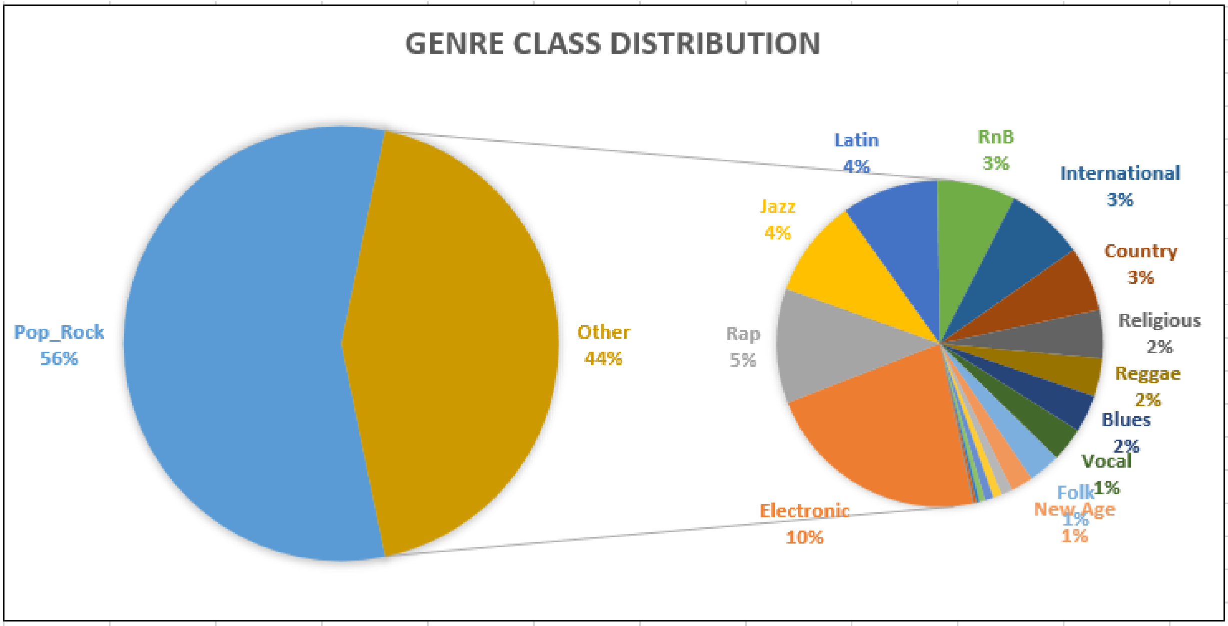
After removing the strong correlated feature pairs. Check the correlation matrix is like below.

IPython: home/sch405

```
In [39]: print((correlations_new > threshold).astype(int))
[[1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 1]]

In [40]:
    ...: print(correlations_new.shape) # (8, 8)
    ...:
    ...: print(num_correlated_columns_not_the_same) # [0 0]
    ...:
(8, 8)
[0 0]
```

Q1 (b) Genre class distribution visualization



Q1 (c) Merging genres and audio features and data cleaning

Process

Join on “track_id”, if the “track_id” are in different string format (such as with quotation marks). Using *df.withColumn("track_id", F.regexp_replace("track_id", "\"",""))* to uniform.

Output

```
In [83]: features_merge_genres.show(10,40)
+------------------+--------+--------+-------+-------+--------+--------+-------+------+--------+
|          track_id|  f_0002|  f_0004| f_0005| f_0006|  f_0010|  f_0012| f_0013|f_0014|   genre|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+--------+
|TRAAABD128F429CF47|0.001519|0.001557|0.05665| 0.0558| 0.00109|0.002902| 0.1126|0.5304|Pop_Rock|
|TRAAADZ128F9348C2E|0.001638|0.002156|0.07978|0.07485|0.001605|0.005229| 0.2006|0.4524|    null|
|TRAAAND12903CD1F1B|0.003171| 0.00271|0.08154|0.07511|0.002439|0.004914| 0.1759|0.5884|    null|
|TRAAAVL128F93028BC|9.198E-4|0.001212|0.04155|0.04491|5.823E-4| 0.00197|0.07356|0.5935|    null|
|TRAABPK128F424CFDB|0.004769|0.002363|0.07297| 0.0443|0.003815|0.005105| 0.1974|0.6286|Pop_Rock|
|TRAABYN12903CFD305|4.123E-4|0.001247|0.04712|0.08117|2.826E-4|0.002039|0.07816|0.5358|    null|
|TRAACER128F4290F96|0.004536|0.002502|0.09102|0.05125|0.004631|0.006002| 0.2269|0.5374|Pop_Rock|
|TRAACWF12903CA0AD7|0.003022|0.002332|0.07864|0.05194|0.001936|0.004375| 0.1636|0.5263|    null|
|TRAADAD128F9336553|0.001876|0.001833|0.06366|0.04211|0.001212|0.003198| 0.1184|0.5068|    null|
|TRAADRX12903D0EFE8|0.003009|0.001986| 0.0709|0.05733|0.003535|0.006095| 0.2382|0.5164|    null|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+--------+
only showing top 10 rows
```

 # Attention: there are a large amount of “null” value of "genre" (# 573999 / 994615 = 57.71%)

Conclusion:

1. To benefit model training, here we have to iteratively remove the strong correlated features;

2. Dealing with missing value, there are a lot of things need to considered, “not available”, “not applicable”, “missing completely at random” “missing at random”, “missing not at random”. After all these are clear, we can decide the appropriate method to deal with the missing value. In this case, we just assume that the missing value are “not available” and “missing completely at random” , so we can use partly delete to remove all the observations contain “null”, after delete there are still have a sufficient dataset which has 420,616 observations.

IPython: home/sch405

```
In [93]:
   ...: features_merge_genres = (
   ...:     features_merge_genres. filter(~F.col("genre").isNull()))

In [94]: features_merge_genres.show(10,100)
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+
|          track_id|  f_0002|  f_0004| f_0005| f_0006|  f_0010|  f_0012| f_0013|f_0014|     genre|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+
|TRAAABD128F429CF47|0.001519|0.001557|0.05665| 0.0558| 0.00109|0.002902| 0.1126|0.5304|  Pop_Rock|
|TRAABPK128F424CFDB|0.004769|0.002363|0.07297| 0.0443|0.003815|0.005105| 0.1974|0.6286|  Pop_Rock|
|TRAACER128F4290F96|0.004536|0.002502|0.09102|0.05125|0.004631|0.006002| 0.2269|0.5374|  Pop_Rock|
|TRAADYB128F92D7E73|0.002983|0.002261|0.07965|0.06292|0.001982|0.004733| 0.1773|0.5126|      Jazz|
|TRAAGHM128EF35CF8E|0.005852|0.002468|0.07639|0.06356|0.003312|0.005462| 0.1978| 0.609|Electronic|
|TRAAGRV128F93526C0|0.002131|0.001519|0.04777|0.04523|0.002354|0.004584| 0.1764|0.5403|  Pop_Rock|
|TRAAGTO128F1497E3C|0.001236|0.001789|0.06716|0.08579|3.271E-4| 0.00162| 0.0619|0.6142|  Pop_Rock|
|TRAAHAU128F9313A3D|8.682E-4|9.592E-4| 0.0332|0.06616|8.643E-4|0.002507|0.09907|0.5874|  Pop_Rock|
|TRAAHEG128E07861C3|  0.0088|0.003949|  0.131|0.05231|0.003686|0.004698| 0.1726|0.6321|       Rap|
|TRAAHZP12903CA25F4|0.002841|0.002088|0.07393|0.06671|9.622E-4|0.002193|0.08333|0.6052|       Rap|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+
only showing top 10 rows


In [95]: features_merge_genres.count()
Out[95]: 420616
```

Q2

Q2 (a) Three classification algorithms

They are Logistic Regression, Decision tree, and Random forest

Logistic Regression: It can dealing with both binary classification and multiclass classification. Compared to kernel methods and network methods, because it is low dimensional method, it is more transparent and interpretable. It is relatively fast to train, only involves two hyperparameters (α, λ). But Logistic Regression has relatively high level assumption (linear relationship, residual are normal and have zero mean, no auto correlation, no perfect multicolinerity, homoscedasticity). So before training, removing variables that have large absolute correlations will be helpful.

Decision tree: Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions (Spark.apache, n.d).

Random forest: It is robust method, do not have too strict assumption (unless the levels of nominal variable should less than 15). Easy to train and fast though may not perform best in accuracy. Because it is made of a lot of sub tree models, it is not such so transparent. It is not as good as the last two in explainability and interpretability .

Conclusion:

The processing should unless apply: missing value imputation, remove strong correlated features.

Q2 (b) Convert “genre” column into binary variable

IPython: home/sch405

```
In [100]: features_merge_genres_label.show(10, 100)
     ...: (
     ...:     features_merge_genres_label
     ...:     .groupBy("is_rap")
     ...:     .count()
     ...:     .show(2)
     ...: )
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+------+
|          track_id|  f_0002|  f_0004| f_0005| f_0006|  f_0010|  f_0012| f_0013|f_0014|     genre|is_rap|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+------+
|TRAAABD128F429CF47|0.001519|0.001557|0.05665| 0.0558| 0.00109|0.002902| 0.1126|0.5304|  Pop_Rock|     0|
|TRAABPK128F424CFDB|0.004769|0.002363|0.07297| 0.0443|0.003815|0.005105| 0.1974|0.6286|  Pop_Rock|     0|
|TRAACER128F4290F96|0.004536|0.002502|0.09102|0.05125|0.004631|0.006002| 0.2269|0.5374|  Pop_Rock|     0|
|TRAADYB128F92D7E73|0.002983|0.002261|0.07965|0.06292|0.001982|0.004733| 0.1773|0.5126|      Jazz|     0|
|TRAAGHM128EF35CF8E|0.005852|0.002468|0.07639|0.06356|0.003312|0.005462| 0.1978| 0.609|Electronic|     0|
|TRAAGRV128F93526C0|0.002131|0.001519|0.04777|0.04523|0.002354|0.004584| 0.1764|0.5403|  Pop_Rock|     0|
|TRAAGTO128F1497E3C|0.001236|0.001789|0.06716|0.08579|3.271E-4| 0.00162| 0.0619|0.6142|  Pop_Rock|     0|
|TRAAHAU128F9313A3D|8.682E-4|9.592E-4| 0.0332|0.06616|8.643E-4|0.002507|0.09907|0.5874|  Pop_Rock|     0|
|TRAAHEG128E07861C3|  0.0088|0.003949|  0.131|0.05231|0.003686|0.004698| 0.1726|0.6321|       Rap|     1|
|TRAAHZP12903CA25F4|0.002841|0.002088|0.07393|0.06671|9.622E-4|0.002193|0.08333|0.6052|       Rap|     1|
+------------------+--------+--------+-------+-------+--------+--------+-------+------+----------+------+
only showing top 10 rows

+------+------+
|is_rap| count|
+------+------+
|     0|399717|
|     1| 20899|
+------+------+


In [101]: print(20899 / 399717)
0.05228449127757886
```

Conclusion

Using the cleaning data (after removing strong correlated variables and missing observations), the class of this dataset is very unbalanced, the number of positive class ("is_rap") observations is only 5.23% of the negative class ("other"). We have to using down sampling, up sampling, weight or hybird methods to solve class imbalance problem. And it is suggested to do stratified train/test split.

Q2 (c) Train/ Test split

Strategy

- Using stratified train/test split
- Down Sampling training data

```
print_class_balance(features_merge_genres_label, "is_rap")

# is_rap
# 420616
  # is_rap   count     ratio
# 0      0  399717  0.950313
# 1      1   20899  0.049687

print_class_balance(training, "training")

# training
# 336492
  # is_rap   count     ratio
# 0      0  319773  0.950314
# 1      1   16719  0.049686

print_class_balance(test, "test")

# test
# 84124
  # is_rap  count     ratio
# 0      0  79944  0.950311
# 1      1   4180  0.049689
```

Q2 (d) Train models

Strategy

- Using down sampling training data to train the model (comparing the result from table 1, down sampling achieve higher auroc, and benefit calculation)
- Then using test data to test model performance

Table 1: Model performance (under sampling training data)

Performance	Logistic Regression (No sampling)	Logistic Regression (Down sampling)
Precision	0.2958	0.2392
recall	0.9470	0.5986
accuracy	0.8855	0.8855
auroc	0.8519	0.8592

Q2 (e) Model performance metrics

Output

Table 2: Model hyperparameters (under sampling training data)

Methods	Hyperparameter	Hyperparameter Value
Logistic Regression	α, λ	default (elasticNetParam=0.0, regParam=0.0)
Random Forest	numTrees, maxDepth, maxBins	numTrees = 20, maxDepth = 4, maxBins = 2
Decision Tree	maxDepth, maxBins	maxDepth=5, maxBins=32

Table 3: Model performance (under sampling)

Performance	Logistic Regression	Random Forest	Decision Tree
Precision	0.2392	0.1240	0.1240
Recall	0.5986	0.7200	0.7200
Accuracy	0.8855	0.7334	0.7334
AUROC	0.8592	0.7771	0.7607

Q2 (f) Discuss the model performance

According to Table3, compared to the other two, Logistic Regression (LR) achieves more balanced precision and recall, and relatively higher accuracy and AUROC. In this case, overall, Logistic Regression performs best. The performances of Random Forest (RF) and Decision Tree (DT) are quite similar. Both of them have higher recall and lower precision than LR, they are more radical in predict as more positive class as possible. It can be implied that, in imbalance binary classification problem, if the negative class observations are much more than the positive class. The more serious the imbalance, the higher threshold is suggested to use to improve the RF and DT performance.

Q3

Q3 (a) Hyper parameters tuning

Table 4: Model performance (under sampling training data)

Algorithms	Logistic Regression	Random Forest	Decision Tree
Hyper parameters	α(elasticNetParam)	numTrees	
		maxDepth	maxDepth
	λ(regParam)	maxBins	maxBins

From table 4, in logistic regression, there are two hyperparameters (λ, α) using as the parameter to control the penalty from ridge regression part and the LASSO regression part. The hyperparameter λ is the penalty to control overfitting. As λ increasing, the effect from less important predictors will be minimize. hyperparameter λ is the penalty to control the model complexity. When there are a large number of variables that are not strong correlate to the target variable, it is necessary to increase α. In this case, there are only eight predictors, λ, α should reasonablely close to 0 rather than 1.

In Random Forest and Decision Tree, the hyperparameters all about the number of depth, bins, trees, which is used to control the model complexity. In this case, it is a long size data set, with 420,616 observations, but 8 predictors. The simple tree model, with less tree involved, low level of depth and a small amount of bins will be suitable.

Compare to Table 2, the hyperparameter I choose in Q2 RandomForest and Decision Tree are litter larger, it can be improved. As well as the Logistic Regressing, the (λ, α) can be slightly increase.

Q3 (b) Tuning hyperparameters (Cross-Validation)

Here I only use Tuning hyperparameters to train Logistic Regression.

Table 5: Tuning hyperparameters (under sampling training data)

Methods	Hyperparameter	Hyperparameter Value
Logistic Regression	α, λ	elasticNetParam=[0.1,0.5], regParam=[0.0,0.2]
Cross-Validation	Folds	numFolds=5

Table 6: Model performance Comparision

Performance	Logistic Regression (before tuning)	Logistic Regression (after tuning)
Precision	0.2392	0.255
Recall	0.5986	0.4849
Accuracy	0.8855	0.904
AUROC	0.8592	0.8218

According to Table 6, after tuning, the overall performance of model seems slightly worse. But the precision and accuracy slightly improved. Why? What kind of change?

Table 6: Confusion Matrix Comparation

Logistic Regression (before tuning)		Logistic Regression (after tuning)	
# actual total:	84124	actual total:	84124
# actual positive:	4180	# actual positive:	4180
# actual negative:	79944	# actual negative:	79944
# nP:	10460	# nP:	7946
# nN:	73664	# nN:	76178
# TP:	2502	# TP:	2027
# FP:	7958	# FP:	5919
# FN:	1678	# FN:	2153
# TN:	71986	# TN:	74025
# precision:	0.2392	# precision:	0.25510
# recall:	0.5986	# recall:	0.4849
# accuracy:	0.8855	# accuracy:	0.9040
# auroc:	0.8592	# auroc:	0.82180

According to Table 7, after tuning, the number of positive class predictions decrease, the number of negative class predictions increase. This tuned model seems more affected by imbalance classes.

Q4

Q4 (a) Multiclass classification

Process

(1) Add label column “Class” to convert the nominal variable “genre” into numeric variable “Class” (value range from 1 to 21).
(2) Parameter preparation: features.
(3) UDF used to output class balance result.
(4) Train/test split → down sampling training (did not do – coding problem) → using Logistic Regression Model to train the data (has no time to do the hyperparameter tuning) → using test data to check the performance → using multiclass Classification Evaluator to evaluate the performance (write the code, have not achieve the result, because has not solve the down sampling part).

Q4 (b) Add label column “Class”

```
# Add a label column "Class"
features_merge_genres_label = (
    features_merge_genres.withColumn("Class",
                                     when(F.col("genre").contains("Rap"), 1)
                                     .when(F.col("genre").contains("Jazz"),2)
                                     .when(F.col("genre").contains("Blues"),3)
                                     .when(F.col("genre").contains("Pop_Rock"),4)
                                     .when(F.col("genre").contains("Classical"),5)
                                     .when(F.col("genre").contains("Reggae"),6)
                                     .when(F.col("genre").contains("Religious"),7)
                                     .when(F.col("genre").contains("Vocal"),8)
                                     .when(F.col("genre").contains("Easy_Listening"),9)
                                     .when(F.col("genre").contains("RnB"),10)
                                     .when(F.col("genre").contains("Latin"),11)
                                     .when(F.col("genre").contains("Folk"),12)
                                     .when(F.col("genre").contains("Country"),13)
                                     .when(F.col("genre").contains("Stage"),14)
                                     .when(F.col("genre").contains("Electronic"),15)
                                     .when(F.col("genre").contains("International"),16)
                                     .when(F.col("genre").contains("Children"),17)
                                     .when(F.col("genre").contains("Avant_Garde"),18)
                                     .when(F.col("genre").contains("New Age"),19)
                                     .when(F.col("genre").contains("Comedy_Spoken"),20)
                                     .when(F.col("genre").contains("Holiday"),21)
                                     )
)
```

```
features_merge_genres_label.show(10, 100)

# +------------------+--------+--------+-------+-------+--------+--------+------+------+----------+-----+
# |          track_id|  f_0002|  f_0004| f_0005| f_0006|  f_0010|  f_0012|f_0013|f_0014|     genre|Class|
# +------------------+--------+--------+-------+-------+--------+--------+------+------+----------+-----+
# |TRAAABD128F429CF47|0.001519|0.001557|0.05665| 0.0558| 0.00109|0.002902|0.1126|0.5304|  Pop_Rock|    4|
# |TRAAAEF128F4273421|9.944E-4|0.001172|0.04344|0.05534|8.953E-4|0.002842|0.1133|0.5422|  Pop_Rock|    4|
# |TRAAAIR128F1480971|0.009601|0.003531| 0.1192|0.05424|0.006124|0.007223|0.2708|0.5909|       RnB|   10|
# |TRAAAMO128F1481E7F|0.001631|0.001468|0.04735|0.05611|0.001698|0.004041|0.1478|0.5223| Religious|    7|
# |TRAACLG128F4276511| 0.01205|0.005275| 0.1502|0.02528|0.008857|0.008184|0.2824| 0.607|Electronic|   15|
# |TRAADMZ128F422F2F8|0.004146|0.002341|0.07626|0.03364|0.003002|0.004696|0.1716|0.5663|  Pop_Rock|    4|
# |TRAADYB128F92D7E73|0.002983|0.002261|0.07965|0.06292|0.001982|0.004733|0.1773|0.5126|      Jazz|    2|
# |TRAAFTE128F429545F|0.003126|0.002083|0.07685|0.06178|0.001778|  0.0037|0.1433|0.5693|  Pop_Rock|    4|
# |TRAAGAV128F4241242|0.004237|0.001988|0.06203|0.04199|0.004097|0.005174|0.2041|0.5961|  Pop_Rock|    4|
# |TRAAGCG128F421CC9F|0.002427|0.002146|0.07174|0.06791|0.001557|0.003849|0.1413|0.4322|     Latin|   11|
# +------------------+--------+--------+-------+-------+--------+--------+------+------+----------+-----+
# only showing top 10 rows
```

Q4 (c) Train model and evaluate.

(1) Train/ test split

```
training = temp
for c in classes:
    training = training.where((col("Class") != i) | (col("Row") < class_counts[i] * 0.8))

training.cache()
training.show(10, 100)

# +------------------+--------------------------------------------------------------------+-----+-----------+---------------------+---+
# |          track_id|                                                            Features|Class|         id|               Random|Row|
# +------------------+--------------------------------------------------------------------+-----+-----------+---------------------+---+
# |TRBFGKU128F14ACF03| [4.109E-4,8.021E-4,0.02586,0.05477,1.907E-4,0.001016,0.03645,0.578]|   14|60129543768| 0.002019489582244516|  1|
# |TRUCHSB128F93526B5|  [0.008982,0.003162,0.1039,0.07135,0.003995,0.004255,0.1568,0.6671]|   14|94489307683| 0.002756944229875047|  2|
# |TRLNGGN128F42756BF|  [0.002674,0.002171,0.07238,0.07497,0.00118,0.003457,0.1265,0.5848]|   14|51539623341| 0.003189916694055883|  3|
# |TRYTIAY128F42926F3|[5.138E-4,0.001175,0.04247,0.05181,4.623E-4,0.002354,0.08987,0.5608]|   14|60129575604|0.0035103452633515886|  4|
# |TRIETGR128F932E959| [0.001149,0.001712,0.06316,0.06828,8.954E-4,0.003018,0.1152,0.5554]|   14|68719487623|0.0035803650018962907|  5|
# |TRTKKGY12903CD57BF|      [0.0012,0.00178,0.06759,0.0757,0.001097,0.003681,0.1429,0.5314]|   14|51539633895| 0.003844871418016149|  6|
# |TRUMZMV128F426F39C|     [4.627E-5,3.597E-4,0.01288,0.07497,3.0E-5,5.668E-4,0.0211,0.561]|   14|25769831502| 0.003881911731826171|  7|
# |TRZQMGV128F4221006| [7.967E-4,0.001429,0.05161,0.05449,9.169E-4,0.003832,0.1457,0.5059]|   14|25769838400| 0.004387503289432382|  8|
# |TRKBEHN128F933C43A|  [0.00133,0.001613,0.05757,0.06013,9.459E-4,0.002897,0.1107,0.5273]|   14|42949686592| 0.005227422061659048|  9|
# |TRXGSHC128F4272A41|  [0.001983,0.001931,0.06453,0.04932,0.001145,0.002837,0.1053,0.544]|   14|17179900421| 0.005832145102972586| 10|
# +------------------+--------------------------------------------------------------------+-----+-----------+---------------------+---+
# only showing top 10 rows
```

Table 6: Multiclass Class Balance (whole dataset)

Class	count	ratio
4	237649	56.50%
15	40665	9.67%
1	20899	4.97%
2	17774	4.23%
11	17504	4.16%
10	14314	3.40%
16	14194	3.37%
13	11689	2.78%
7	8780	2.09%
6	6931	1.65%
3	6801	1.62%
8	6182	1.47%
12	5789	1.38%
19	4000	0.95%
20	2067	0.49%
14	1613	0.38%
9	1535	0.36%
18	1012	0.24%
5	555	0.13%
17	463	0.11%
21	200	0.05%

(2) Down sampling training data

This multiclass is imbalanced should find a way (down sampling, up sampling, weight …) To solve this problem before to do the model training.

Code is ready to have a test, no time to run and modify.

```
# -------------------------------------
# Muticlass classification Downsampling
# -------------------------------------


train_class_count = (training.groupBy(F.col("Class")).agg(F.countDistinct(F.col("Features"))))


for c in classes:
    training_downsampled = (
    training
    .withColumn("Random", rand())
    .where((col("Class") != i) | ((col("Class") == i) & (col("Random") < 21 * ( int(train_class_count.filter(F.col("Class") == i))/ 336593))))
    )
training_downsampled.cache()

print_class_balance(training_downsampled, "training_downsampled")


class_counts = (
    features
    .groupBy("Class")
    .count()
    .toPandas()
    .set_index("Class")["count"]
    .to_dict()
)
classes = sorted(class_counts.keys())
```

(3) Training and Evaluate

Code is ready, no time to run and modify.

```
# -----------------------------------------------------------------
# Muticlass classification Downsampling & LogisticRegression
# -----------------------------------------------------------------
lr = LogisticRegression(featuresCol='Features', labelCol='Class')
lr_model = lr.fit(training)
predictions = lr_model.transform(test)

predictions.cache()
```

```
# ------------------------------------------------
# Muticlass classification & performance metrics
# ------------------------------------------------

"""
from pycm import *

y_actu =[]
y_pred = []
cm = ConfusionMatrix(actual_vector=y_actu, predict_vector=y_pred) # Create CM From Data
cm.classes
cm.table
print(cm)
cm.matrix()
cm.normalized_matrix()

"""

evaluator_metrics = ["f1", "WeightedPrecision", "weightedRecall", "accuracy"]

def print_multiClass_metrics(predictions, labelCol="Class", predictionCol="prediction", rawPredictionCol="rawPrediction"):
    total = predictions.count()
    for i in evaluatro_metrics:
        metric_value = (MCE.evaluate(predictions,{MCE.metricName: i})
    )
    print(f"{metri:25s}{metric_value}")


print_multiClass_metrics(predictions)
```

Song recommendations

Q1

Q1 (a) Count unique songs an unique users

Result

There are 378310 unique songs and 1019318 unique users.

Q1 (b) How many different songs has the most active user played?

Process

1. Find the max play_count user;

2. Count unique songs listened by the target user.

Result

There are 195 unique songs has the most active user played. The percentage is 0.019%.

Q1 (c) Visualize the distribution of song popularity and the distribution of user activity

Process

1. Distribution of song popularity means treat song as object, to do the play count. Group the play_count to see how many songs in different popular groups.

2. Distribution of user activity means treat user as object, to do the paly count. Group the play_count to see how many users in different activity groups.

Visualisation

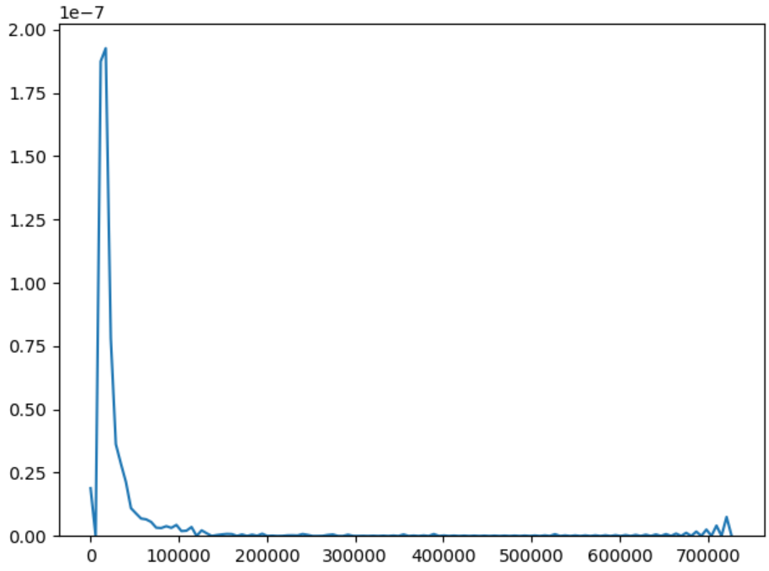


Figure 1 Distribution of song popularity

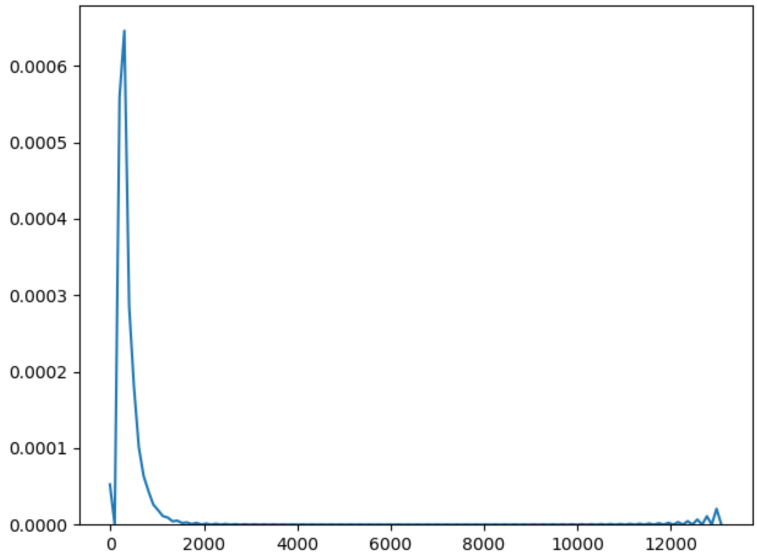


Figure 2 Distribution of user activity

Result

The shape of the these two distribution are all “Right skewed”.

Q1 (d) Create a clean dataset

Process

1. N: user_song_count_threshold (removing users who have listened to fewer than N songs);

   M: song_user_count_threshold (removing songs played less than M users);

2. Combine song popularity and user activity distribution, and the output of song_counts.approxQuantile() and user_counts.approxQuantile(), and the business logic to determine the suitable threshold N, M.

Logic

(1) Overall

Table 7: Song popularity_song/play count per user

```
# +----------------------------------------+----------+----------+
# |user_id                                 |song_count|play_count|
# +----------------------------------------+----------+----------+
# |093cb74eb3c517c5179ae24caf0ebec51b24d2a2|195       |13074     |
# |119b7c88d58d0c6eb051365c103da5caf817bea6|1362      |9104      |
# |3fa44653315697f42410a30cb766a4eb102080bb|146       |8025      |
# |a2679496cd0af9779a92a13ff7c6af5c81ea8c7b|518       |6506      |
# |d7d2d888ae04d16e994d6964214a1de81392ee04|1257      |6190      |
# |4ae01afa8f2430ea0704d502bc7b57fb52164882|453       |6153      |
# |b7c24f770be6b802805ac0e2106624a517643c17|1364      |5827      |
# |113255a012b2affeab62607563d03fbdf31b08e7|1096      |5471      |
# |99ac3d883681e21ea68071019dba828ce76fe94d|939       |5385      |
# |6d625c6557df84b60d90426c0116138b617b9449|1307      |5362      |
# +----------------------------------------+----------+----------+
```

There are some bad pattern: the first user only listen 195 songs, but played 13074 times, It's almost 67 times per song. This is not a normal actural user (it is not good for training), This kind of outliers are suggested to remove.

```
#               count                mean              stddev min   max
# song_count  1019318   44.92720721109605   54.91113199747355   3  4316
# play_count  1019318  128.82423149596102  175.43956510304616   3 13074

user_counts.approxQuantile("song_count", [0.0, 0.25, 0.5, 0.75, 1.0], 0.05)
user_counts.approxQuantile("play_count", [0.0, 0.25, 0.5, 0.75, 1.0], 0.05)

# [3.0, 20.0, 32.0,  58.0,  4316.0]
# [3.0, 35.0, 71.0, 173.0, 13074.0]
```

Figure 3 User_counts statistics output

Figure 3suggest the distribution of the number of played songs and the distribution of the times of play.

We can focus on min value, the very less, one user only listened 3 songs; one user only played 3 times.

Average: each user listen 32 songs and plays 71 times. Average, 2-3 times per song.

Table 8: Song popularity_song/play count per user

```
# +------------------+----------+----------+
# |song_id           |user_count|play_count|
# +------------------+----------+----------+
# |SOBONKR12A58A7A7E0|84000     |726885    |
# |SOSXLTC12AF72A7F54|80656     |527893    |
# |SOEGIYH12A6D4FC0E3|69487     |389880    |
# |SOAXGDH12A8C13F8A1|90444     |356533    |
# |SONYKOW12AB01849C9|78353     |292642    |
# |SOPUCYA12A8C13A694|46078     |274627    |
# |SOUFTBI12AB0183F65|37642     |268353    |
# |SOVDSJC12A58A7A271|36976     |244730    |
# |SOOFYTN12A6D4F9B35|40403     |241669    |
# |SOHTKMO12AB01843B0|46077     |236494    |
# +------------------+----------+----------+
# only showing top 10 rows
```

```
#              count              mean             stddev  min     max
# user_count  378310  121.05181200602681  748.6489783736941   1   90444
# play_count  378310   347.1038513388491  2978.605348838212   1  726885

song_counts.approxQuantile("user_count", [0.0, 0.25, 0.5, 0.75, 1.0], 0.05)
song_counts.approxQuantile("play_count", [0.0, 0.25, 0.5, 0.75, 1.0], 0.05)

# [1.0, 4.0, 15.0, 44.0, 90444.0]    # output will change everytime, why?
# [1.0, 7.0, 30.0, 111.0, 726885.0]  # output will change everytime, why?
```

Figure 4 song_counts statistics output

From Figure 4, the min, there are song only played by one user, only played once. So this song definitely not popular and obviousely it should not be put in the recommend list.

We are doing recommend songs to user, not find users for paticular song. So we want to focus on the most efficient way to recommen the most possible song that the user will interested in. Under the assumption that one user listen to one song, he must at least listen to two other songs So we donnot want invovle these low frequency count. We want to make our ideal set smaller, make model faster to train. The low frequency users may need other particular business strategy.

If we want to make it smaller, through out users and songs below the some kind of threshold, based on the value they been evaluate with. We can use these quantail output to do that. For example, keep 50% users and 50% songs. It will reduce our overall size of the dataset.

Eventually, I choose N = 45, M = 5.

(2) After Limiting

```
print(triplets_not_mismatched.count() - triplets_limited.count()) # 18,140,867

print(triplets_limited.count() / triplets_not_mismatched.count()) # 0.6038689151774301
```

Figure 5

```
#              limiting     total      percentage
# user_count  298387 / 1019318 =     0.2927
# song_count  81391 /   378310   =    0.2151
```

Figure 6

Collaborative filter algorithm will be more effective when having more information per user. Maximaze the information per user withought throng out too many users. It will achieve good results with less noise but better set of inputs. And it will take less time to calculate. Consider the effect, even we through away some users, not including them in the algorithm, therefore we cannot make recommendation for these users.

These 40% users don't have enough reactivation anyway, so these less information cannot ensure the model will work for them, maybe other easy approaches may perform batter on these users rather than the Collaborative filter, such as business logic alone, popularity based recommondation, chatagory recommendation.

Above all, after we removed the users who listened to fewer than 45 songs, removing songs played less by 5 users. Our algorithm still involved more than 29% users and more than 21% songs.

Q1 (e) Train/Test splitting

Every user in the test set has to be in the training set as well. It is because collaborative filtering algorithms are based on using the interaction between users and items to infer user preference. If there is no records in the training data, there is no learning about this user's history.

How to do it?

Using window function Window.partitionBy("user_id"). It will random distribute every user_id in each partition, when generate train/test from each partition.

Q2 (a) ALS

```
# +------------------+----------------------------------------+-----+---------------+---------------+------------+
# |song_id           |user_id                                 |plays|user_id_encoded|song_id_encoded|prediction  |
# +------------------+----------------------------------------+-----+---------------+---------------+------------+
# |SOAYETG12A67ADA751|00007ed2509128dcdd74ea3aac2363e24e9dc06b|2    |230291.0       |668.0          |0.0028589617|
# |SOBWGGV12A6D4FD72E|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |21967.0        |9.727796E-5 |
# |SODESWY12AB0182F2E|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |50140.0        |3.8436243E-5|
# |SOELPFP12A58A7DA4F|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |47791.0        |4.8647635E-5|
# |SOGEWRX12AB0189432|00007ed2509128dcdd74ea3aac2363e24e9dc06b|2    |230291.0       |43771.0        |5.0902414E-5|
# |SOHEWBM12A58A7922A|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |65419.0        |1.1359467E-5|
# |SOHYKCX12A6D4F636F|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |56345.0        |3.0861494E-5|
# |SOICJAD12A8C13B2F4|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |33517.0        |8.227438E-5 |
# |SOINVHR12AB0189418|00007ed2509128dcdd74ea3aac2363e24e9dc06b|2    |230291.0       |53030.0        |4.825835E-5 |
# |SOOKJWB12A6D4FD4F8|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |25135.0        |1.6038015E-4|
# |SOOTFWU12A6D4FB8FB|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |39749.0        |3.0221636E-5|
# |SOPFUBI12A58A79E33|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |70396.0        |1.8131863E-5|
# |SORFZWW12A6D4F742C|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |5890.0         |6.564613E-4 |
# |SOVGNWE12A6D4FB90A|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |50432.0        |2.6998241E-5|
# |SOVMWUC12A8C13750B|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |803.0          |0.005757871 |
# |SOWXPFM12A8C13B2EC|00007ed2509128dcdd74ea3aac2363e24e9dc06b|1    |230291.0       |44347.0        |3.326081E-5 |
# |SOBKTVL12A8C13C031|00009d93dc719d1dbaf13507725a03b9fdeebebb|1    |295314.0       |11850.0        |7.210085E-4 |
# |SOCDAMA12A6D4FB5B6|00009d93dc719d1dbaf13507725a03b9fdeebebb|1    |295314.0       |14476.0        |4.3145713E-4|
# |SOCQVSB12A58A80F8B|00009d93dc719d1dbaf13507725a03b9fdeebebb|1    |295314.0       |60283.0        |2.313962E-5 |
```

Figure 7

Q2 (b) Test by hand

```
# +---------------+---------------------+
# |user_id_encoded|recommended_songs    |
# +---------------+---------------------+
# |14             |[2, 12, 0, 9, 8]     |
# |18             |[0, 30, 7, 11, 42]   |
# |25             |[9, 12, 8, 2, 27]    |
# |38             |[12, 61, 193, 19, 25]|
# |46             |[43, 7, 30, 11, 124] |
# |50             |[30, 32, 0, 35, 48]  |
# |73             |[7, 11, 2, 0, 39]    |
# |97             |[30, 0, 7, 42, 88]   |
# |161            |[11, 6, 88, 94, 90]  |
# |172            |[186, 70, 2, 224, 230]|
# +---------------+---------------------+
# only showing top 10 rows
```

Figure 8

```
# +---------------+------------------------------------------------------------------------
# |user_id_encoded|relevant_songs
# +---------------+------------------------------------------------------------------------
# |14             |[8958, 1253, 41647, 51149, 48312, 39234, 88, 11003, 11321, 3701, 11464, 36159, 73110, 62906, 14207, 4471, 15096, 13685, 22512, 35070, 9259, 49657, 25586, 4329, 7, 13533, 2443, 64, 8619, 11268, 27709, 4877, 23827, 17900, 61325, 10362, 12092,
# |18             |[57748, 8272, 75246, 70502, 15817, 1414, 43842, 62020, 48978, 8019, 7129, 41392, 22807, 10116, 78850, 40107, 1327, 13190, 367, 2418, 20684, 21996, 43028, 23306, 65416, 26816, 30769, 47057, 79207, 33795, 12959, 1567, 16592, 80337, 11734, 375,
# |25             |[6096, 5125, 19659, 5985, 7682, 1520, 8357, 2369, 191, 3061, 329, 7163, 18001, 170, 8811, 2312, 34811, 207, 24, 1593, 642, 572, 471, 1981, 211, 1052, 1864, 50, 1326, 1356, 44, 2559, 611, 25313, 2681, 481, 42185, 52071, 1477, 3841, 229, 14461
# |38             |[236, 8320, 4270, 20340, 32416, 50142, 30716, 15550, 273, 62705, 30737, 55730, 3377, 416, 4155, 13130, 8699, 325, 71654, 20951, 16923, 5986, 14393, 17821, 4986, 18580, 11949, 14694, 1521, 8521, 6881, 2732, 36357, 17384, 4524, 5824, 11189, 15
# |46             |[41486, 16109, 8205, 5119, 955, 15442, 16889, 3216, 8291, 6153, 27001, 15890, 241, 1224, 1828, 22429, 179, 44082, 21213, 3230, 50866, 44066, 24262, 53404, 4041, 48423, 16334, 7874, 34105, 31572, 35007, 14767, 9774, 50652, 65353, 53161, 52983
# |50             |[1026, 1265, 4891, 5035, 1872, 9355, 25224, 8115, 9509, 15214, 21091, 19515, 22427, 18621, 4906, 5966, 17010, 3290, 7601, 279, 23741, 15359, 358, 248, 15142, 3017, 12405, 455, 15409, 5951, 9655, 20396, 27122, 15791, 29569, 3953, 30676, 7592,
# |73             |[7737, 21868, 24198, 8538, 13286, 8135, 544, 43178, 9099, 8079, 44350, 26061, 453, 4942, 37111, 3970, 146, 627, 3034, 17837, 2981, 1267, 821, 262, 17590, 1713, 9910, 32600, 8823, 48417, 11575, 7310, 629, 3189, 11888, 8783, 20943, 52789, 1521
# |97             |[10970, 27174, 18223, 13417, 17120, 13327, 52123, 38963, 32585, 37319, 7104, 22175, 12517, 19652, 16850, 38703, 29807, 26506, 10610, 65221, 36950, 63492, 8449, 10598, 3100, 52243, 11058, 54877, 33119, 70486, 9880, 25555, 40630, 574, 8841, 33
# |161            |[8077, 5221, 5499, 4230, 684, 15326, 385, 4981, 7807, 14976, 18409, 9248, 4709, 11228, 5269, 48058, 49319, 79909, 3754, 392, 3741, 14595, 5043, 9855, 251, 3970, 10082, 6818, 18894, 7452, 191, 2965, 10725, 17939, 2397, 11874, 2499, 16056, 599
# |172            |[9517, 5158, 11096, 9899, 13472, 14750, 17000, 20977, 21335, 24721, 49892, 35672, 11462, 20082, 3739, 14419, 26315, 10448, 19712, 7105, 14561, 18953, 33567, 45890, 11415, 4594, 1463, 9986, 15340, 44544, 13372, 14368, 19228, 23008, 3436, 1806
# +---------------+------------------------------------------------------------------------
```

Figure 9

The figure8 is the user recommend songs generated by the ALS algorithm, the figure 9 is the song that user actually played.

By comparing the songs in these two tables, In the user example I chose, I didn't find the same song.

If there is does not perform good in this case.

Q2 (c) Evaluation Metrics

Table 9: Model performance Comparison

Performance	Value
Precision@5	0.0490
NDCG @10	0.0333
Mean Average Precision (MAP)	0.0064

Why these evaluation metrics are useful?
1. They can be generate on time.
2. Some of the evaluation metrics not only consider the possible preference but also the recommend order. It is quite useful when there are a few of limiting positions can be used to recommend the most possible user favour items.

Limitations
1. Cannot use for new user, who has not have enough data can be trained.
2. From business logic, when we want to recommend some new items that does not have user experience data, these metrics' evaluation result may mislead.

Alternative method
1. Checking for test items in recommendations.

2. Real time, user reaction evaluation.
3. Revenue

What other metrics

1. If the future user-song plays can be measured, it can be transform as binary problem. If user plays the recommend song, can be labelled as “1”, otherwise “0”. Then “confusion metrics”, AUROC can be used here.

Reference

Spark.apache.org. 2020. *Classification And Regression - Spark 2.2.0 Documentation*. Retrieved on Oct 22th, 2020 from https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#decision-tree-classifier