IBM Coursera Advanced Data Science Capstone Project

Montassar Mhamdi

OUTLINES

- DATASET - USE CASE
- DATA EXPLORATION
- DATA CLEANSING - DATA AGGREGATION
- MODEL DEFINITION
- MODEL TRAINING
- MODEL EVALUATION

DATASET used for Music recommendation :

- Published by the audio recommendation system **Audiscrobbler**.
- It includes **148,111 unique users** and **1,631,028 unique artists**.

DATA EXPLORATION

misspelledArtistID

standardArtistID

```
MisspelledIDs StandardIDs
{1092764: 1000311,
 1095122: 1000557,
 6708070: 1007267,
 10088054: 1042317,
 1195917: 1042317,
 1112006: 1000557,
 1187350: 1294511,
 1116694: 1327092,
 6793225: 1042317,
 1079959: 1000557,
 6789612: 1000591,
 1262241: 1000591,
 6791455: 1000591,
 6694867: 1000591,
 10141141: 1113738,
 1295140: 1000591,
 1027859: 1252408,
 2127019: 1000591,
```

...

artistID

name

```
ArtistDF.show()

+--------+--------------------+
|artistID|                name|
+--------+--------------------+
| 1134999|        06Crazy Life|
| 6821360|        Pang Nakarin|
|10113088|Terfel, Bartoli- ...|
|10151459| The Flaming Sidebur|
| 6826647|   Bodenstandig 3000|
|10186265|Jota Quest e Ivet...|
| 6828986|       Toto_XX (1977|
|10236364|         U.S Bombs -|
| 1135000|artist formaly kn...|
|10299728|Kassierer - Musik...|
|10299744|         Rahzel, RZA|
| 6864258|      Jon Richardson|
| 6878791|Young Fresh Fello...|
|10299751|          Ki-ya-Kiss|
| 6909716|Underminded - The...|
|10435121|             Kox-Box|
| 6918061|  alexisonfire [wo!]|
| 1135001|         dj salinger|
| 6940391|The B52's - Chann...|
|10475396|            44 Hoes|
+--------+--------------------+
only showing top 20 rows
```

userID

artistID

playcount

```
userArtistDF.show()

+-------+--------+---------+
| userID|artistID|playcount|
+-------+--------+---------+
|1000002|       1|       55|
|1000002| 1000006|       33|
|1000002| 1000007|        8|
|1000002| 1000009|      144|
|1000002| 1000010|      314|
|1000002| 1000013|        8|
|1000002| 1000014|       42|
|1000002| 1000017|       69|
|1000002| 1000024|      329|
|1000002| 1000025|        1|
|1000002| 1000028|       17|
|1000002| 1000031|       47|
|1000002| 1000033|       15|
|1000002| 1000042|        1|
|1000002| 1000045|        1|
|1000002| 1000054|        2|
|1000002| 1000055|       25|
|1000002| 1000056|        4|
|1000002| 1000059|        2|
|1000002| 1000062|       71|
+-------+--------+---------+
only showing top 20 rows
```

DATA EXPLORATION

```
totalUsers = userArtistDF.count()
print(" total number of users : {}".format(totalUsers))
distinctUsers = userArtistDF.select('userID').distinct().count()
print(" total number of distinct users : {}".format(distinctUsers))
```

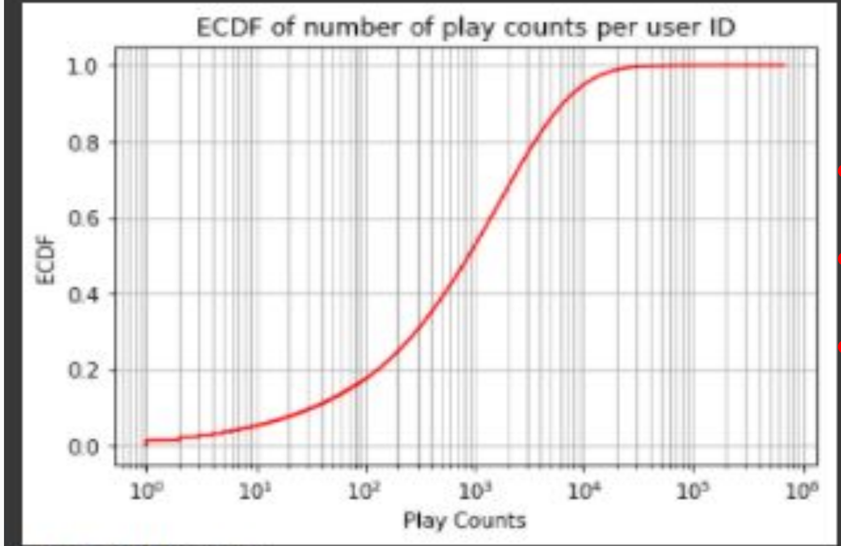
```
total number of users : 24296858
total number of distinct users : 148111
```

```
[13] distinctArtists = userArtistDF.select('artistID').distinct().count()
     print(" total number of distinct artists : {}".format(distinctArtists))
```

```
total number of distinct artists : 1631028
```

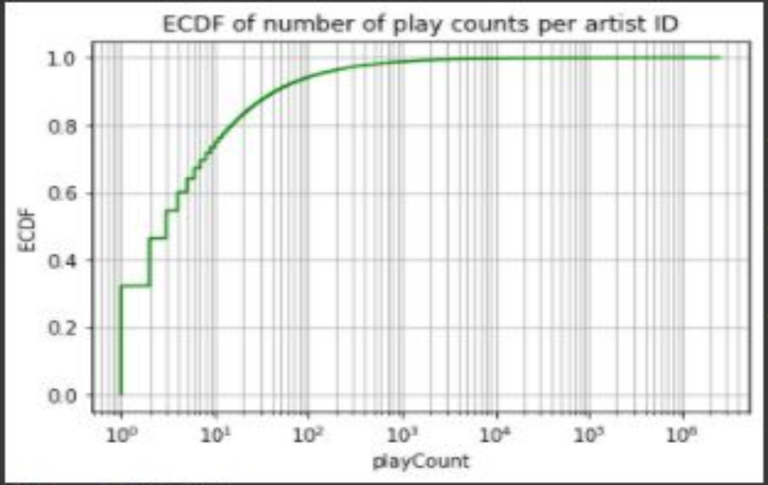
```
maxUserID = userArtistDF.agg({'userID' : 'max'}).show()
maxUserID = userArtistDF.agg({'userID' : 'min'}).show()
maxUserID = userArtistDF.agg({'artistID' : 'max'}).show()
maxUserID = userArtistDF.agg({'artistID' : 'min'}).show()
```

```
+-----------+
|max(userID)|
+-----------+
|    2443548|
+-----------+

+-----------+
|min(userID)|
+-----------+
|         90|
+-----------+

+-------------+
|max(artistID)|
+-------------+
|     10794401|
+-------------+

+-------------+
|min(artistID)|
+-------------+
|            1|
+-------------+
```




```
Sum = 371638969
Mean = 2509.1922206993404
Max = 674412
Min = 1
Percentile 25% = 204.0
Percentile 50% = 892.0
Percentile 75% = 2800.0
Percentile 90% = 6484.0
Percentile 95% = 10120.0
Percentile 97% = 13297.399999999965
The percentage of users who listen less than 10 times P(YUsr<=10) = 5.229%
```

- The average play count of users is equal to 2509.
- 97% of users have play counts are less than average.
- 5.3% of users have play counts less than 10. So it's quite*- difficult to recommend artists for them.



- The average of play count per artist is 228.
- %95 of artists are listened less the average.
- artists who are listened once are about the third of population.
- Top 5 artists (~2.6%) can be recommended for all users.

```
Sum = 371638969
Mean = 227.85566464830768
Max = 2502130
Min = 1
Percentile 25% = 1.0
Percentile 50% = 3.0
Percenrile 75% = 11.0
Percentile 90% = 45.0
Percentile 95% = 126.0
The percentage of artists how are listened once P(YArt=1) = 32.185%
The percentage of artists how are listened less than 1000 P(YArt<=1000) = 98.744%
Top listened artits :
 [1425942 1542806 1930592 2259185 2502130]
Top listened artists and their percentage = 2.599%
```

DATA CLEANSING & AGGREGATION

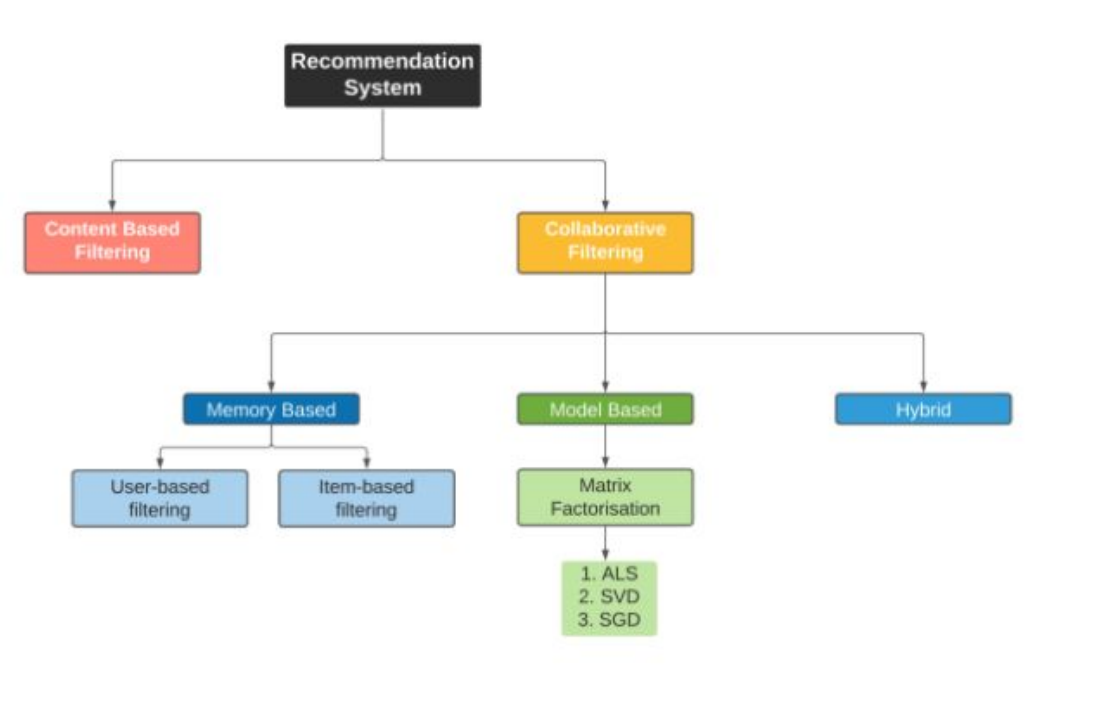
- Replacing the misspelled artist IDs (misspelled artists names) with one unique standard ID.

```
[ ] updatedUserArtistDF.show()

+-------+--------+---------+
| userID|artistID|playcount|
+-------+--------+---------+
|1000002|       1|       55|
|1000002| 1000006|       33|
|1000002| 1000007|        8|
|1000002| 1000009|      144|
|1000002| 1000010|      314|
|1000002| 1000013|        8|
|1000002| 1000014|       42|
|1000002| 1000017|       69|
|1000002| 1000024|      329|
|1000002| 1000025|        1|
|1000002| 1000028|       17|
|1000002| 1000031|       47|
|1000002| 1000033|       15|
|1000002| 1000042|        1|
|1000002| 1000045|        1|
|1000002| 1000054|        2|
|1000002| 1000055|       25|
|1000002| 1000056|        4|
|1000002| 1000059|        2|
|1000002| 1000062|       71|
+-------+--------+---------+
only showing top 20 rows
```

MODEL DEFINITION : **Block ALS**

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
```

Alternating Least Squares (ALS) matrix factorisation attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. X * Yt = R. Typically these approximations are called ‘factor’ matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.



MODEL TRAINING :

80% of data for training, 20% for test

```
[14] dataSet = userArtist.map(lambda r: Rating(float(r[0]), float(r[1]), int(r[2]))).repartition(30).cache()
```

```
trainData, testData = dataSet.randomSplit([.8, .2], 1)
trainData.cache()
testData.cache()
```

```
PythonRDD[11] at RDD at PythonRDD.scala:53
```

```
[17] t0 = time()
model = ALS.trainImplicit(trainData, rank, iterations, lambda_, alpha)
t1 = time()

print('training finished after {}s', format(t1-t0))
```

```
training finished after {}s 402.24497270584106
```

```
model.save(sc, 'capstoneProjectArtistsRecommendation.spark')
print('Model is saved.')
```

```
Model is saved.
```

```
rank, iterations, lambda_, alpha = 10, 5, 0.01, 40
```

: Trained Model Parameters

Parameters:

ratings - RDD of Rating objects with userID, productID, and rating

rank - number of features to use (also referred to as the number of latent factors)

iterations - number of iterations of ALS

lambda - regularization parameter

blocks - level of parallelism to split computation into

seed - random seed for initial matrix factorization model

MODEL EVALUATION :

AUC = The area under a ROC curve plot which plots recall (true positive rate) against fallout (false positive rate) for increasing recommendation set size.
TP = most recommended items.
FP = unrecommended items.

```
t0 = time()
auc = computeAUC( testData, allItemsIDs2, model.predictAll)
t1 = time()
print("AUC=",auc)
print("Predictions finished after {}" .format(t1-t0))
```

```
AUC= 0.9603278483259308
Predictions finished after 293.67347383499146
```