Artist recommendation based on Yahoo! Music dataset

Jayant Kumar
CS-Graduate
Steven Institute of technology
Home city-Pasighat,India

Malika Thakur
CS-Graduate
Steven Institute of technology
Home city-Chandigarh, India

Team Number : 6

Team Name : A Walk in the Spark

Abstract:

Information retrieval has become an interesting research subject to be explored. In this project, we are planning to present a recommendation of artist names based on the social rating information provided by users. Artist Recommendation system will learn from the history of users rating preferences and suggest the series of artist which the user would love to hear in recent future. The goal of our system would be to generate meaningful recommendations to a collection of users for artist that might interest them. We will present the problem we plan to solve with collaborative filtering algorithms and their corresponding analysis. Then the experimental results are presented and discussed. At last, we are planning to conclude the project and propose some prospective.

1 Introduction:

With rapid growth in musical data, music service providers started feeling a need to manage the inventory of songs and help the users to get best possible artist suggestion. This leads the strong need for good song recommendation system. Artist recommendation system learns from the history of users listening preferences and suggests the series of artist which the user would love to hear in recent future. In this project, we have implemented collaborative filtering algorithms to try to build an effective recommender system. Collaborative filtering algorithms which predict taste of a user by collecting preferences and tastes from many other users is also implemented. Also, we have tried to some different approaches to increase the efficiency of our recommendation system.

2 Data set Analysis:

This dataset represents a snapshot of the Yahoo! Music community's preferences for various songs. The dataset contains over 717 million ratings of 136 thousand songs given by 1.8 million users of Yahoo! Music services. The data was collected between 2002 and 2006.The users and artists are represented by randomly assigned numeric ids so that no identifying information is revealed. The mapping from artist id's to artist, is given. There are 2 sets in this dataset. Part one(ydata-ymusic-artist-names-v1_0.csv) is 2.3MB and part 2(ydata-ymusic-user-artist-ratings-v1_0.csv) is 1.1 Gbytes.

Data set link: https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&guccounter=1

2.1 Artist mapping data:

"ydata-ymusic-artist-names-v1_0.csv" contains the artist_id and name of each musical artist. The snippet of data set is shown below:

```
+--------+-------------+
|artistID|       artist|
+--------+-------------+
|    -100|Not Applicable|
|     -99|Unknown Artist|
| 1000001| "Bobby ""O"""|
| 1000002| "Jimmy ""Z"""|
| 1000003|  '68 Comeback|
| 1000004|  'Til Tuesday|
| 1000005|The (EC) Nudes|
| 1000006|   .38 Special|
| 1000008|         1 + 1|
| 1000009|1 Of The Girls|
+--------+-------------+
only showing top 10 rows
```

2.2 User rating data:

"ydata-ymusic-user-artist-ratings-v1_0.csv" contains user ratings of music artists. It contains 11,557,943 ratings of 98,211 artists by 1,948,882 anonymous users. The format of each line of the file is anonymous_user_id (TAB) artist_id (TAB) rating.

The ratings are integers ranging from 0 to 100, except 255 (a special case that means "never play again"). The snippet of data set is shown below:

```
+-------+--------+------+
| userID|artistID|rating|
+-------+--------+------+
|1853593| 1022086|   100|
|1853593| 1022153|   100|
|1853593| 1022226|     0|
|1853593| 1022232|   100|
|1853593| 1023032|   100|
|1853593| 1023157|   100|
|1853593| 1024006|     0|
|1853593| 1024051|   100|
|1853593| 1024379|    90|
|1853593| 1024602|   100|
+-------+--------+------+
only showing top 10 rows
```

```
+-------+------------------+------------------+------------------+
|summary|            userID|          artistID|            rating|
+-------+------------------+------------------+------------------+
|  count|             24114|             24114|             24114|
|   mean|1853787.8376461805|1033359.5978269884| 43.50899892178817|
| stddev| 109.7244459738098|34312.836459266786|40.376834281793464|
|    min|           1853593|             24538|               0.0|
|    max|           1853981|           1101719|             100.0|
+-------+------------------+------------------+------------------+
```

2.3 Data Exploration:

For data exploration ,we performed online analytical processing (OLAP) based on Spark Data frame and Spark SQL on given dataset.

Code snippet:

Data Exploration

Check the number of artists and users. Check the artist that with most ratings and users made most ratings.

```
top_artist = music.groupBy("artistID").count().orderBy('count', ascending=False)
top_user = music.groupBy("userID").count().orderBy('count', ascending=False)
top_artist.createOrReplaceTempView("top_artist")
top_user.createOrReplaceTempView("top_user")
artist.createOrReplaceTempView("artist")
```

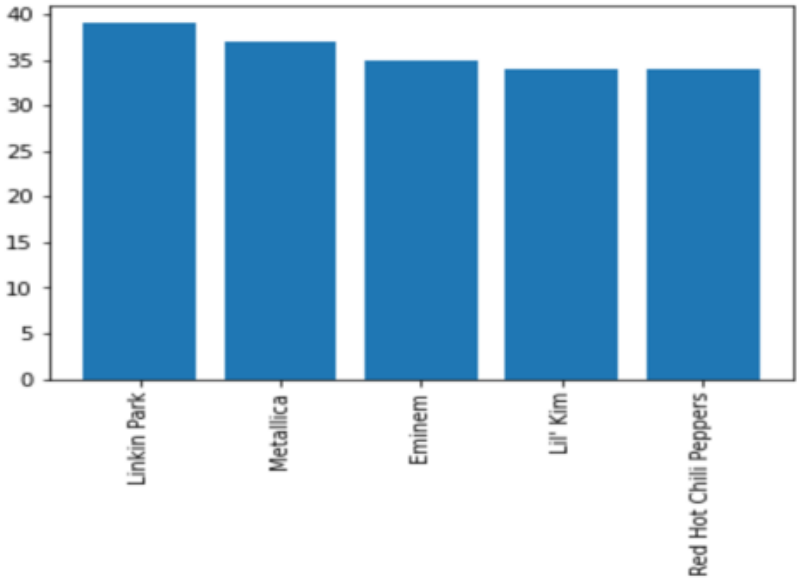
```
pop_artist = spark.sql("select artist,count \
                       from top_artist INNER JOIN artist \
                       on top_artist.artistID = artist.artistID \
                       order by top_artist.count desc")
```

```
Fre_artist = pop_artist.selectExpr("artist as artist","count as no_rating")
```

```
# Show top 20 artist that have most ratings.
pop_artist.show(5)
```

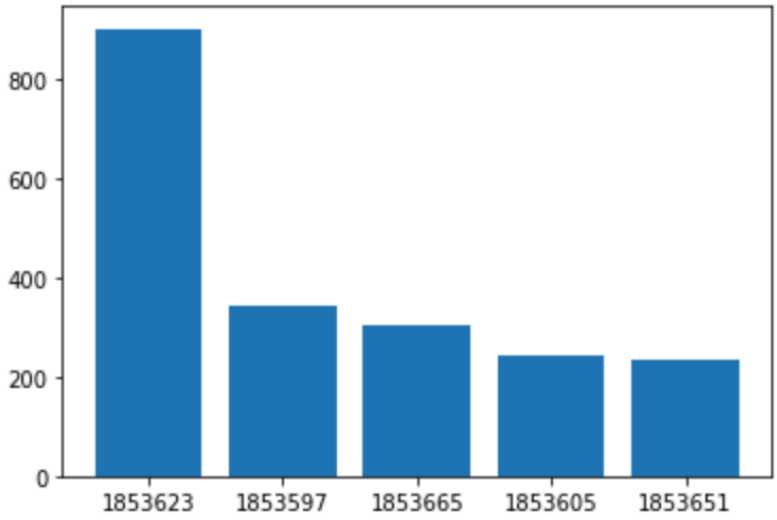
Top pop artist which are frequently rated:

```
+--------------------+---------+
|              artist|no_rating|
+--------------------+---------+
|         Linkin Park|       39|
|           Metallica|       37|
|              Eminem|       35|
|Red Hot Chili Pep...|       34|
|            Lil' Kim|       34|
+--------------------+---------+
only showing top 5 rows
```



Top users frequently giving number of rating:

```
+-------+---------+
| userid|no_rating|
+-------+---------+
|1853623|      901|
|1853597|      344|
|1853665|      304|
|1853605|      244|
|1853651|      234|
+-------+---------+
only showing top 5 rows
```

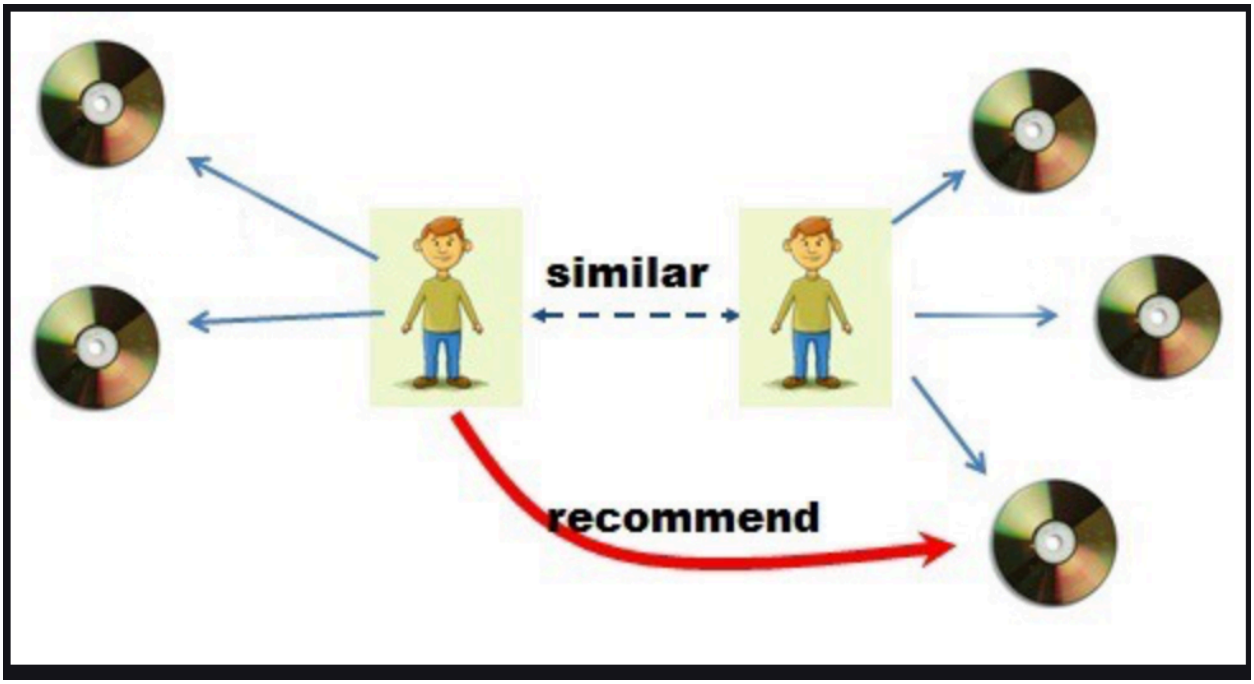


Users rating only one artist and Artist rated once:

```
901 out of 1734 artist with one rating
13 out of 120 user rate one artist
```

3 Algorithm:

We'll often discuss among friends for recommendations to find a new artist to listen for a good music. Naturally, we have more prominent trust in the suggestions from companions who offer tastes like our own. The concept of Collaborative filtering (CF) can be explained by collecting user feedback in the form of ratings for items in a given domain and analyzing similarities in rating behavior amongst several users in determining how to recommend an item. Pictorial representation is given below:



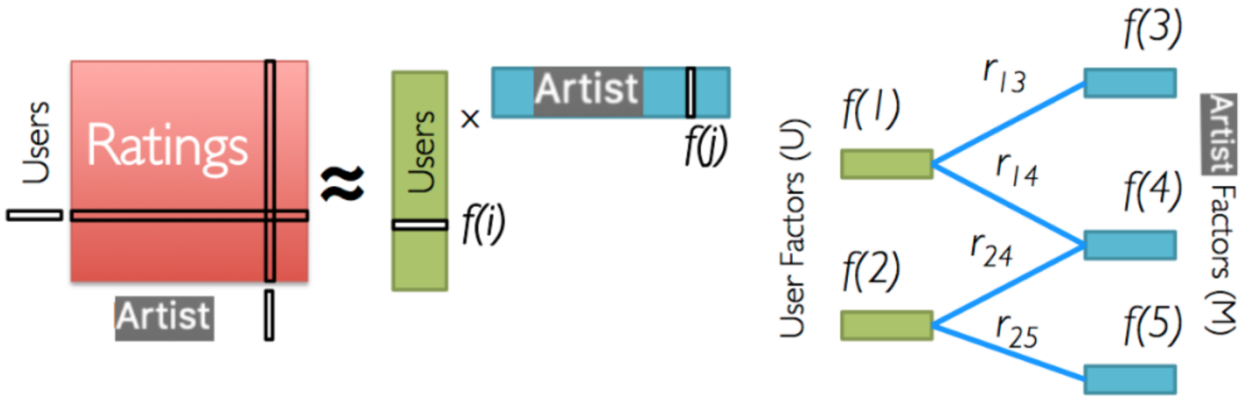
3.1 Implementation of Artist based recommendation using ALS in Spark:

Assume we have a matrix of users and the ratings they accommodated to the artist they have tuned in on the off chance that you'll see a large portion of the matrix is unfilled that is on the grounds that there are extremely numerous artists for anyone to sensibly listen to.

	Artist 1	Artist 2	Artist.....	Artist n
User 1	95			
User 2		5		3
User 3			1	
User 4	2	3	59	
User 5		49		43
User 6	4		5	
User 7		4		
User ...		34		
User m				4

One of the major advantages of alternating least square (**ALS**)is that it works truly well with sparse matrix like this what it tries to do is influence ratings from similar users and fill in all of these clear cells with predictions of how these users would rate these artists. If they somehow happened to watch them the predictions with the most noteworthy qualities at that point become proposals that can be offered to clients.

Suppose we have a matrix our ALS we'll take that matrix (m*n)and factor it into two smaller matrices users(U) and artist(M) where if x together produces an approximation of the original matrix r.



3.2 RMSE:

The **root-mean-square error** (**RMSE**) is a measure of the differences between values predicted by an ALS the values observed under yahoo music dataset. Formulae is given below to calculate:

Root Mean Squared Error (RMSE)	$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}}$

4 Code Snippet to implement ALS and the recommendation for particular user-Id:

4.1 Implementation of ALS model to train the classifier:

Split data into train and test.

```
# The datasets will be repeatedly used, persist them in memory using the cache function
train, test = music.randomSplit([0.8, 0.2], seed=1234)
train.cache()
test.cache()
```

```
DataFrame[userID: int, artistID: int, rating: float]
```

Build the recommender model using ALS

```
# If the rating matrix is derived from another source of information, you can set implicitPrefs to True to get bette
als = ALS(userCol='userID', itemCol='artistID', ratingCol='rating', coldStartStrategy='drop',nonnegative=True)
```

```
# A CrossValidator requires an Estimator, a set of Estimator ParamMaps, and an Evaluator.
# We use a ParamGridBuilder to construct a grid of parameters to search over.
# this grid will have 2 x 5 x 4 = 40 parameter settings for CrossValidator to choose from.
paramGrid = ParamGridBuilder()\
    .addGrid(als.maxIter, [5, 10]) \
    .addGrid(als.regParam, [0.05, 0.1, 0.2, 0.4, 0.8])\
    .addGrid(als.rank, [6, 8, 10, 12])\
    .build()
```

```
crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(metricName="rmse", labelCol="rating",
                          predictionCol="prediction"),
                          numFolds=2)
```

```
import time
start = time.time()
# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(train)
stop = time.time()
print(f"Training time: {stop - start}s")
```

```
Training time: 2620.0461819171906s
```

```
bestModel = cvModel.bestModel
```

```
import time
start = time.time()
# Make predictions on test data. model is the model with combination of parameters that performed best.
predictions = bestModel.transform(test)
predictions.show(5)
stop = time.time()
print(f"Prediction time: {stop - start}s")
```

```
+-------+--------+------+----------+
| userID|artistID|rating|prediction|
+-------+--------+------+----------+
|1854678| 1001043|   0.0|0.31695813|
|1855505| 1001043|  90.0|  7.845177|
|1855920| 1001043|   0.0|0.07847169|
|1855789| 1010495|   0.0| 1.9885976|
|1855794| 1010495|  90.0| 43.255154|
+-------+--------+------+----------+
only showing top 5 rows

Prediction time: 10.357301950454712s
```

```
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse_test = evaluator.evaluate(predictions)
```

```
# RMSE of the testing dataset
rmse_test
```

```
26.86744798955836
```

Make recommendations for a selected user

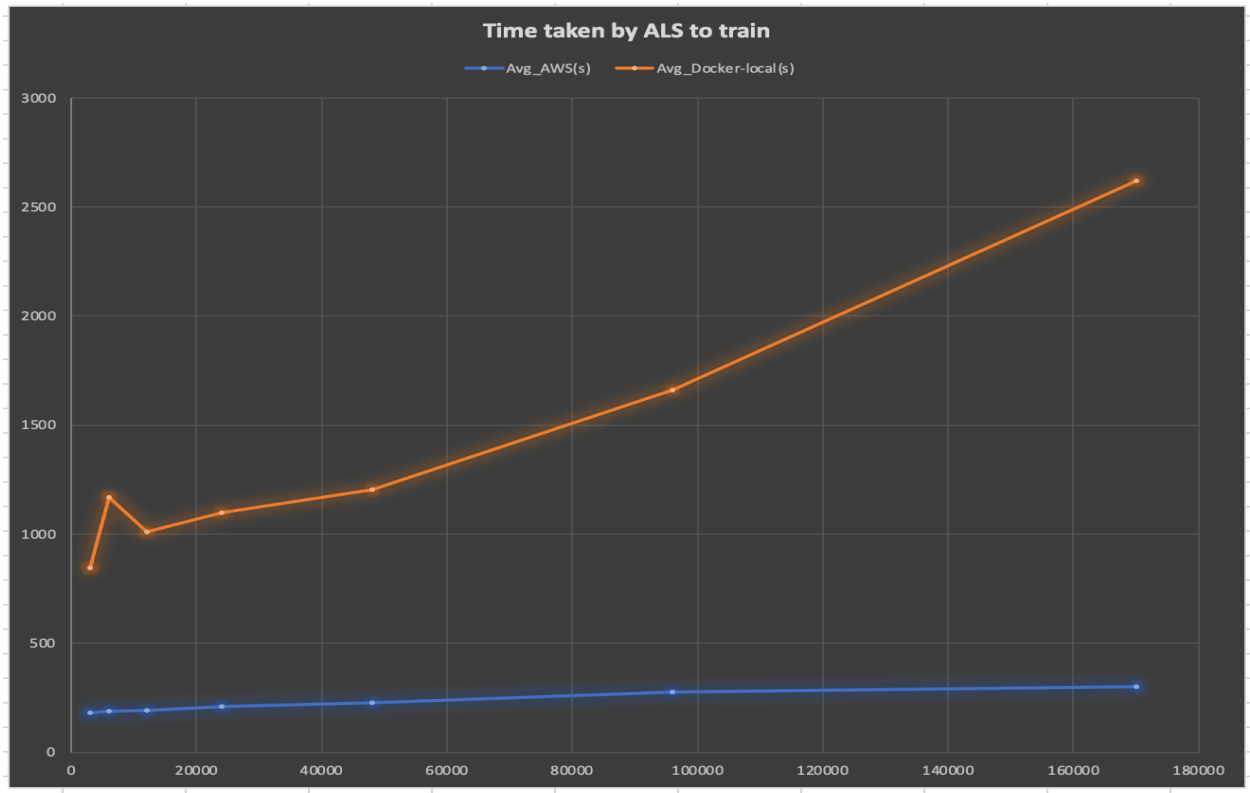
```
# Generate top 10 user recommendations for a specified set of movies
users = music.select(als.getUserCol()).distinct().limit(1)
userSubsetRecs = bestModel.recommendForUserSubset(users, 10)
users.show()
```

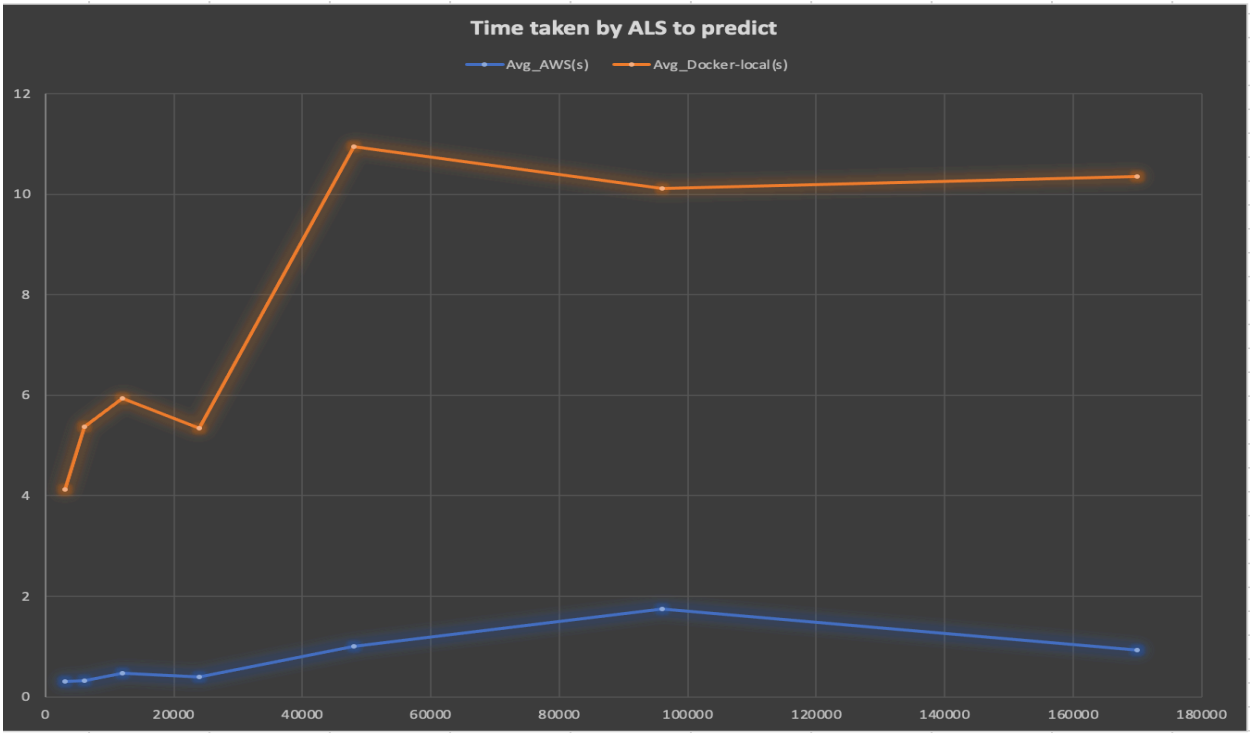
```
+-------+
| userID|
+-------+
|1853815|
+-------+
```

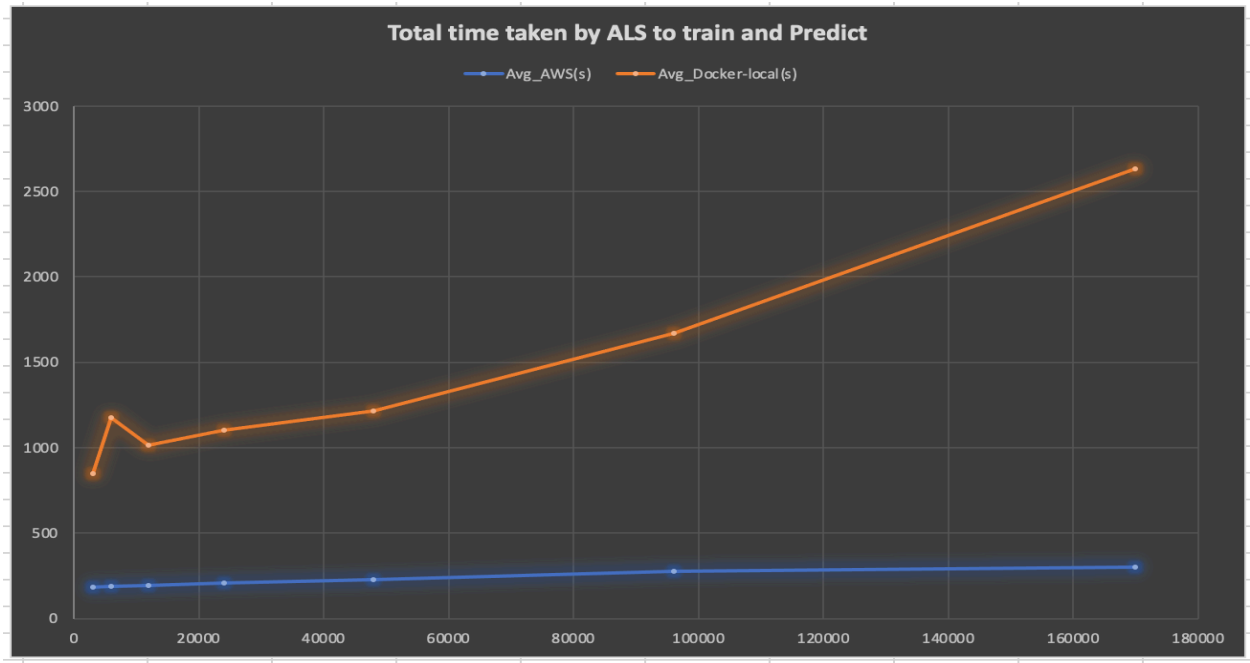
```
userSubsetRecs = userSubsetRecs.toPandas()
ans = []
for i in range(10):
  ans.append(userSubsetRecs[['recommendations']].iloc[0,0][i][0])
artist.where(col("artistID").isin(ans)).show()
```

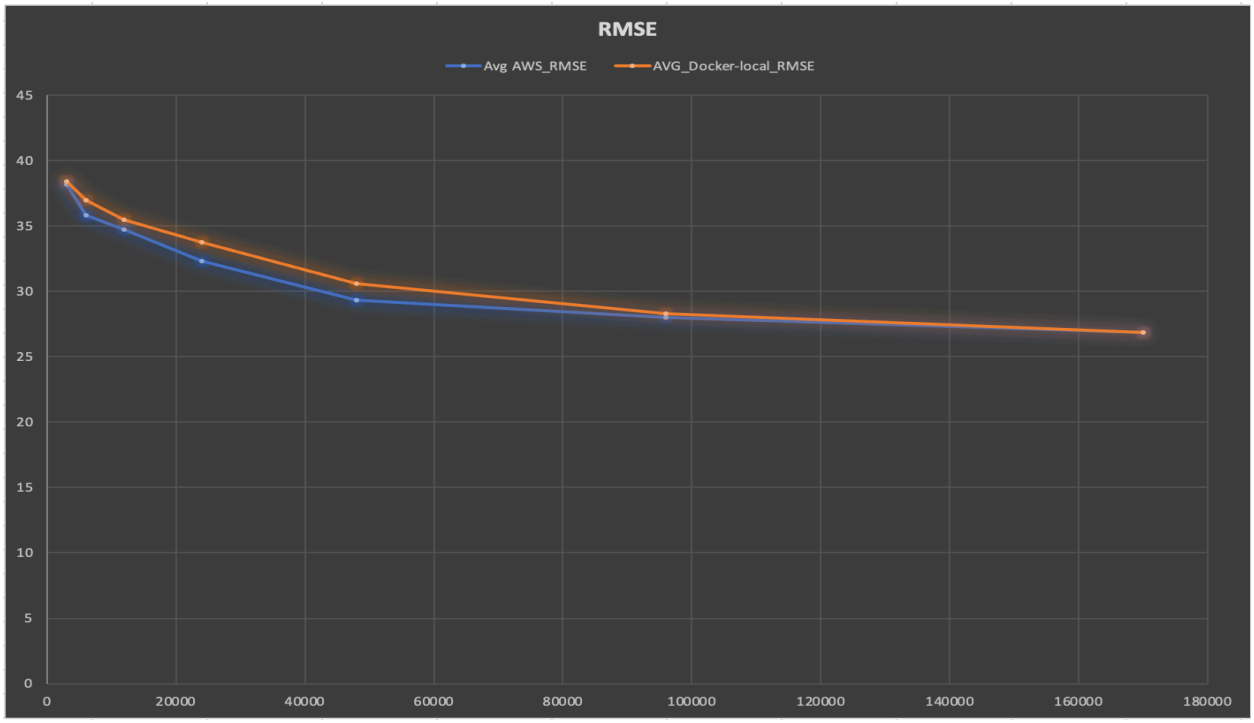
```
+--------+--------------------+
|artistID|              artist|
+--------+--------------------+
| 1001178|       Archer & Park|
| 1002674|        Biggy Smallz|
| 1018845|         The Muppets|
| 1024086|        Shades Apart|
| 1028102|         Vanessa-Mae|
| 1031292|          Animaniacs|
| 1033789|         Sean Lennon|
| 1042975|Hedwig & The Angr...|
| 1069823|          Stillframe|
| 1100636|The Jessica Fletc...|
+--------+--------------------+
```

4.2:Result Analysis:






Total time taken by ALS to train and Predict
Avg_AWS(s)
Avg_Docker-local(s)
3000
2500
2000
1500
1000
500
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000


RMSE
Avg AWS_RMSE
AVG_Docker-local_RMSE
45
40
35
30
25
20
15
10
5
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

4.3 Observation and Records:

Average of Time taken by ALS to train after 5 runs:

Data_scale	Avg_AWS(s)	Avg_Docker-local(s)
3000	183.09	844.57
6000	189.27	1168.86
12000	191.92	1009.69
24000	208.58	1098.78
48000	228.56	1205.65
96000	277.17	1661.66
170000	301.04	2620.23

Average of RMSE values after 5 runs:

Data_scale	Avg AWS_RMSE	AVG_Docker-local_RMSE
3000	38.19	38.39
6000	35.79	36.98
12000	34.71	35.46
24000	32.32	33.75
48000	29.32	30.61
96000	27.99	28.28
170000	26.83	26.87

4.4 Discussion on above recommendations and graphs:

With the assistance of our metrics, we delivered data that speaks to the correlation between artist predictions dependent on rating and comparable users and the accuracy of the artist proposals. Strangely we found the size of the user history influenced the accuracy of various measurements. Test users with smaller rating histories would in general behave better accuracy with measurements dependent on artist similarity, while those with bigger rating accounts would in general have better exactness with measurements dependent on user history. Our outcomes show a for the most part low degree of accuracy delivered by the artist recommendations. The artist

recommendation issue is a troublesome one and can be ascribed to the absence of artist recommendations connection between user's rating history and their rating inclinations in the data set.

We can also see that the cloud platforms AWS performs better as compare to Docker-local from the above time analysis graph.

5 Limitations:

Constraints Because of the subjectivity in music, the suspicion that users with comparable practices may have similar tastes have not been broadly studied. Through collaborative filtering recommender functions admirably, the key issues, for example, cold start, human efforts bias are unavoidable.

5.1 Sparse Data and Lack of additional Information to build-on-From :

In the data set collection, we simply realize the user tuned in to an artist. In any case, we have insufficient data about the user and the artist. It makes it too difficult to even consider improving our presentation. In this way, we think we need to discover more data from different assets or alternate ways. For instance, a number of songs listened to by the same artist, the number of plays, the age, and sexual orientation of the client.

5.2 Cold start:

At the beginning phase, hardly any ratings are given. Because of the absence of these ratings, prediction results are poor. It is otherwise called data sparsity problems.

5.3 Popularity bias :

For the most part, famous music can get more evaluations. The music in the long tail, nonetheless, can seldom get any. Subsequently, collaborative filtering essentially prescribed

famous music to the audience members. Despite the fact that giving well-known things are solid, it is as yet hazardous since user infrequently get enjoyably shocked. Cold beginning It is otherwise called data sparsity issues. At the beginning phase, scarcely any ratings are given. Because of the absence of these evaluations, prediction results are poor.

5.4 Human effort:

An ideal recommender framework ought not to include an excess of human efforts, since users are not continually ready to rate. The ratings may likewise develop towards the individuals who do rate which can led to false prediction.

6 Conclusion:

This Project has given us a sharp knowledge of recommendation frameworks. Our outcomes have indicated how we can utilize explicit data (with a rating for every user artist association). This shows that the artist proposal issue is a troublesome one and can be ascribed to the absence of a high relationship between a users' rating history and their artist inclinations in the dataset and in light of the fact that it was computationally wasteful to acquire all the information to run for the model to find a solid connection. We additionally noticed the low accuracy results from both our own work and those from related work, which constrained us to consider approaches to improve overall precision. At last, we concocted a novel technique to give a recommendation system expected to augment the accuracy of our artist to improve overall precision.

7 Future work:

To work with the implicit dataset and design more precise artist recommender by considering the other factors like number of times user listen to the given artist and the number of hours played by user for given artist. Further design the recommender engine for song recommendation for given user by using Yahoo dataset.

8 References:

1)https://medium.com/hexagondata/collaborative-filtering-how-predictive-modelling-tells-us-what-we-want-37848533ece7

2) https://www.youtube.com/watch?v=FgGjc5oabrA

3) https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html

4) https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2799933550853697/2823893187441173/2202577924924539/latest.html