School of Computer Science and Engineering

J Component report

Programme : MIA (Integrated)

Course Title : Big Data Frameworks

Course Code : CSE3120

Slot : F1

Title: Music Recommender System using Apache Spark and Python

Team Members: **Arun Venkat S J - 19MIA1076**
Lokesh Kanna P - 19MIA1014
John Chacko - 19MIA1097

Faculty: Dr. Suganeshwari G

Sign:

Date:

DECLARATION BY THE CANDIDATE

I hereby declare that the report titled "**Music Recommender System using Apache Spark and Python"** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. Suganeshwari G, Associate Professor, SCOPE, Vellore Institute of Technology, Chennai.**

Signature of the Candidate

Acknowledgement

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Suganeshwari G**, School of Computer Science and Engineering for her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to **Dr. R. Ganesan, Dean, School of Computer Science and Engineering (SCOPE)**, Vellore Institute of Technology, Chennai, for extending the facilities of the school towards our project and for his unstinting support.

We express our thanks to our Head of the Department for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Movie recommendation using Apache Spark and Python"** is a bona-fide work of **Arun Venkat S J (19MIA1076), John Chacko (19MIA1097), Lokesh Kanna (19MIA1014),** carried out the "Music Recommender System using Apache Spark and Python" - Project work under my supervision and guidance for Big Data Frameworks - CSE3120.

Dr. Suganeshwari G

SCOPE

TABLE OF CONTENTS

Serial.No	DESCRIPTION	Page no.
1.	INTRODUCTION	6
2.	PROBLEM STATEMENT	7
3.	DATASET	8
4.	METHODOLOGY	12
5.	CODE IMPLEMENTATION	13
6.	RESULT/OUTPUT	23
7.	CONCLUSION	24
8.	REFERENCES	24

INTRODUCTION

- The Big Data framework is a structured approach that consists of six core capabilities that organisations need to take into consideration when setting up their Big Data organization
- Data has become a strategic asset for most organisations. The capability to analyse large data sets and discern pattern in the data can provide organisations with a competitive advantage.
- Netflix, for example, looks at user behaviour in deciding what movies or series to produce. Alibaba, the Chinese sourcing platform, became one of the global giants by identifying which suppliers to loan money and recommend on their platform. Big Data has become Big Business.
- Music is a cross-cultural universal, a ubiquitous activity found in every known human culture. Individuals demonstrate manifestly different preferences in music, and yet relatively little is known about the underlying structure of those preferences.
- As online music streaming becomes the dominant medium for people to listen to their favorite songs, music streaming services are now able to collect large amounts of data on the listening habits of their customers.
- These streaming services, like Spotify, Apple Music or Pandora, are using this data to provide recommendations to their listeners.
- These music recommendation systems are part of a broader class of recommender systems, which filter information to predict a user's preferences when it comes to a certain item.
- In our project we will be using Million song dataset to understand how Spotify and other music companies make these recommendations.

PROBLEM STATEMENT

For this project, we create a recommender system that will recommend new musical artists to a user based on their listening history. Suggesting different songs or musical artists to a user is important to many music streaming services, such as Pandora and Spotify. In addition, this type of recommender system could
also be used as a means of suggesting TV shows or movies to a user (e.g., Netflix).
To create this system, we will be using Spark and the collaborative filtering technique.

Spark: PySpark can significantly accelerate analysis by making it easy to combine local and distributed data transformation operations while keeping control of computing costs. In addition, the language helps data scientists to avoid always having to downsample large sets of data. For tasks such as building a recommendation system or training a machine-learning system, using PySpark is something to consider. It is important for you to take advantage of distributed processing can also make it easier to augment existing data sets with other types of data.

Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user.

DATASET

We used some publicly available song data from Million Song Dataset; The Million Song Dataset is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. However, we modified the original data files so that the code will run in a reasonable time on a single machine. The reduced data files have been suffixed with _small.txt and contains only the information relevant to the top 50 most prolific users (highest artist play counts).

The original data file user_artist_data.txt contained about 141,000 unique users, and 1.6 million unique artists. About 24.2 million users' plays of artists are recorded, along with their count.

Note that when plays are scribbled, the client application submits the name of the artist being played. This name could be misspelled or nonstandard, and this may only be detected later. For example, "The Smiths", "Smiths, The", and "the smiths" may appear as distinct artist IDs in the data set, even though they clearly refer to the same artist. So, the data set includes artist_alias.txt, which maps artist IDs that are known misspellings or variants to the canonical ID of that artist.

The artist_data.txt file then provides a map from the canonical artist ID to the name of the artist

Artist Data:

artist_data.txt consisting of 2 columns
artistid
artist_name

```
1240105     André Visior
1240113     riow arai
1240132     Outkast & Rage Against the Machine
6776115     小松正夫
1030848     Raver's Nature
6671601     Erguner, Kudsi
1106617     Bloque
1240185     Lexy & K. Paul
6671631     Rev. W.M. Mosley
6671632     Labelle, Patti
1240238     the Chinese Stars
1240262     The Gufs
6718605     Bali Music
6828988     Southern Conference Featuring Dr. Ace
1240415     Paul & Paula
1009439     Cinnamon
1018275     School Of Fish
6671680     Armstrong, Louis & His Hot Five
1240508     The Ozark Mountain Daredevils
1240510     The Mercury Program
1240516     Del Close & John Brent
1002584     Nena
6990766     Phil Hendrie - 11/06/98
1240554     Ami Yoshida
1124756     utabi
10023740    Red & Blue feat. Cathy Dennis
1240589     Sebastian Bach & Friends
1240603     The Wake
6748187     Eric Darling
1238620     Juno Reactor, Don Davis
10585028    大友良英ニュー·ジャズ·クインテット
10113150    wouter van veldhoven
3055  Montag
1240848     McFadden & Whitehead
1240853     ORANGE RANGE
10024032    Bobby Sutcliff
1011119     Emergency Broadcast Network
6671734     Alternate Main Title #3
6671738     Dona Walser
```

Artist Alias:

artist_alias.txt consisting of 2 columns
badid
goodid
artist_alias.txt consists of known incorrectly spelt artists and the correct artist id.

```
1027859     1252408
1017615     668
6745885     1268522
1018110     1018110
1014609     1014609
6713071     2976
1014175     1014175
1008798     1008798
1013851     1013851
6696814     1030672
1036747     1239516
1278781     1021980
2035175     1007565
1327067     1308328
2006482     1140837
1314530     1237371
1160800     1345290
1255401     1055061
1307351     1055061
1234249     1005225
6622310     1094137
1261919     6977528
2103190     1002909
9929875     1009048
2118737     1011363
9929864     1000699
6666813     1305683
1172822     1127113
2026635     1001597
6726078     1018408
1039896     1277013
1239168     1266817
6819291     1277876
2030690     2060894
6786886     166
1051692     1307569
1239193     1012079
1291581     78
6642817     1010969
```

User Artist Data:

user_artist_data.txt consisting of 3 columns
-userid
-artistid
-playcount

```
1059637 1000010 238
1059637 1000049 1
1059637 1000056 1
1059637 1000062 11
1059637 1000094 1
1059637 1000112 423
1059637 1000113 5
1059637 1000114 2
1059637 1000123 2
1059637 1000130 19129
1059637 1000139 4
1059637 1000241 188
1059637 1000263 180
1059637 1000289 2
1059637 1000305 1
1059637 1000320 21
1059637 1000340 1
1059637 1000427 20
1059637 1000428 12
1059637 1000433 10
1059637 1000445 88
1059637 1000527 1
1059637 1000617 4
1059637 1000632 250
1059637 1000676 3
1059637 1000790 18
1059637 1000877 1
1059637 1000890 1
1059637 1000926 1
1059637 1000999 22
1059637 1001007 38
1059637 1001027 6
1059637 1001066 3
1059637 1001068 23
1059637 1001107 1
1059637 1001117 1
1059637 1001130 3
1059637 1001198 1
1059637 1001233 3
```

METHODOLOGY

For this project, we will train the model with implicit feedback. This system employs the use of Apache Spark and the collaborative filtering technique. The Alternating Least Squares (ALS) learning algorithm is used for the underlying implementation. To get the best model, we will do a small parameter sweep and choose the model that performs the best on the validation set, we can run a parameter sweep, evaluate each combination of parameters on the validation data, and choose the optimal set of parameters. The parameters then can be used to make predictions on the test data.

Suppose we have a model and some dataset of *true* artist plays for a set of users. This model can be used to predict the top X artist recommendations for a user and these recommendations can be compared the artists that the user actually listened to (here, X will be the number of artists in the dataset of *true* artist plays). Then, the fraction of overlap between the top X predictions of the model and the X artists that the user actually listened to can be calculated. This process can be repeated for all users and an average value returned.

For example, suppose a model predicted [1,2,4,8] as the top X=4 artists for a user. Suppose, that user actually listened to the artists [1,3,7,8]. Then, for this user, the model would have a score of 2/4=0.5. To get the overall score, this would be performed for all users, with the average returned.

Code Implementation

Loading data

Loading the three datasets into RDDs and naming them artistData, artistAlias, and userArtistData.

In [2]:

```
artistData=sc.textFile('/Users/Lokesh/Google Drive/CSC 591  BI/Projects/Project 1-Recommender Systems/01.Apache.Spark.Project-1.RecommenderSystems.FINAL/artist_data_small.txt')
artistAlias=sc.textFile('/Users/Lokesh/Google Drive/CSC 591 BI/Projects/Project 1-Recommender Systems/01.Apache.Spark.Project-1.RecommenderSystems.FINAL/artist_alias_small.txt')
userArtistData=sc.textFile('/Users/Lokesh/Google Drive/CSC 591 BI/Projects/Project 1-Recommender Systems/01.Apache.Spark.Project-1.RecommenderSystems.FINAL/user_artist_data_small.txt')
```

Data Exploration

Here we can explore the dataset with the help of Sparksql which allows us to view our data in a structured form by creating the data as data frames and explore the information. Then we code to find the users' total play counts. Finding three users with the highest number of total play counts (sum of all counters) and printing the user ID, the total play count, and the mean play count (average number of times a user played an artist)

Artist data

```
In [19]: df1 = sqlContext.createDataFrame(artistData)
         df1.printSchema()

         root
          |-- _1: long (nullable = true)
          |-- _2: string (nullable = true)
```

```
In [20]: df1.show() #artist_data.txt consisting of 2 columns
                       #artistid
                       #artist_name

+-------+--------------------+
|     _1|                  _2|
+-------+--------------------+
|1240105|        André Visior|
|1240113|           riow arai|
|1240132|Outkast & Rage Ag...|
|6776115|                小松正夫|
|1030848|       Raver's Nature|
|6671601|      Erguner, Kudsi|
|1106617|              Bloque|
|1240185|       Lexy & K. Paul|
|6671631|     Rev. W.M. Mosley|
|6671632|      Labelle, Patti|
|1240238|   the Chinese Stars|
|1240262|            The Gufs|
|6718605|          Bali Music|
|6828988|Southern Conferen...|
|1240415|        Paul & Paula|
|1009439|            Cinnamon|
|1018275|      School Of Fish|
|6671680|Armstrong, Louis ...|
|1240508|The Ozark Mountai...|
|1240510| The Mercury Program|
+-------+--------------------+
only showing top 20 rows
```

```
In [22]: df1.count()

Out[22]: 30537
```

Artist Alias

In [24]:
```
df2 = sqlContext.createDataFrame(artistAlias)
df2.printSchema()
```

```
root
 |-- _1: long (nullable = true)
 |-- _2: long (nullable = true)
```

In [25]:
```
df2.show()
```

```
+-------+-------+
|     _1|     _2|
+-------+-------+
|1027859|1252408|
|1017615|    668|
|6745885|1268522|
|1018110|1018110|
|1014609|1014609|
|6713071|   2976|
|1014175|1014175|
|1008798|1008798|
|1013851|1013851|
|6696814|1030672|
|1036747|1239516|
|1278781|1021980|
|2035175|1007565|
|1327067|1308328|
|2006482|1140837|
|1314530|1237371|
|1160800|1345290|
|1255401|1055061|
|1307351|1055061|
|1234249|1005225|
+-------+-------+
only showing top 20 rows
```

In [26]:
```
df2.count()
```

Out[26]:
```
587
```

User Artist data

In [30]:
```
df3 = sqlContext.createDataFrame(userArtistData)
df3.printSchema()
```

```
root
 |-- _1: long (nullable = true)
 |-- _2: long (nullable = true)
 |-- _3: long (nullable = true)
```

In [31]:
```
df3.show()
```

```
+-------+-------+-----+
|     _1|     _2|   _3|
+-------+-------+-----+
|1059637|1000010|  238|
|1059637|1000049|    1|
|1059637|1000056|    1|
|1059637|1000062|   11|
|1059637|1000094|    1|
|1059637|1000112|  423|
|1059637|1000113|    5|
|1059637|1000114|    2|
|1059637|1000123|    2|
|1059637|1000130|19129|
|1059637|1000139|    4|
|1059637|1000241|  188|
|1059637|1000263|  180|
|1059637|1000289|    2|
|1059637|1000305|    1|
|1059637|1000320|   21|
|1059637|1000340|    1|
|1059637|1000427|   20|
|1059637|1000428|   12|
|1059637|1000433|   10|
+-------+-------+-----+
only showing top 20 rows
```

```
In [32]: df3.count()

Out[32]: 49481

In [59]: df3.groupby('_2').agg({ '_3':'avg'}).distinct().show()

+-------+------------------+
|     _2|           avg(_3)|
+-------+------------------+
|1001530|1281.7142857142858|
|1002734|128.33333333333334|
|1191501|               1.5|
|   5409|          504.5625|
|1184419|               1.0|
|1000313|             52.75|
|1007802| 19.22222222222222|
|1078674|              21.0|
|1127777|              63.0|
|1301234| 9.333333333333334|
|   2214|              37.0|
|6974425|              21.0|
|     26| 5.285714285714286|
|1027988|            7883.0|
|   5556|1036.3333333333333|
|1009820|             379.5|
|1279485|               1.0|
|1253025|              45.0|
|1007205|28.833333333333332|
|1008678|              19.0|
+-------+------------------+
only showing top 20 rows
```

In [3]:

```
artistData=artistData.map(lambda x: x.split("\t")).map(lambda x: (int(
x[0]), x[1]))
artistAlias=artistAlias.map(lambda x: x.split("\t")).map(lambda x: (in
t(x[0]), int(x[1])))
userArtistData = userArtistData.map(lambda x: x.split(" ")).map(lambda
x: (int(x[0]), int(x[1]), int(x[2])))


artistAliasDict =  dict(artistAlias.collect())



userArtistData = userArtistData.map(lambda x: (x[0], artistAliasDict[x
[1]], x[2]) if x[1] in artistAliasDict.keys() else x)



userPlays=userArtistData.map(lambda x: (x[0],x[2]))


userCount=sc.broadcast(userPlays.countByKey())



topThreeUsers=userPlays.reduceByKey(lambda a,b:  a+b).takeOrdered(3,key
=lambda x:-x[1])
for x in topThreeUsers:
    print('User %d has a total play count of %d and a mean play count
of %d.'%(x[0],x[1],x[1]/userCount.value[x[0]]))
```

```
User 1059637 has a total play count of 674412 and a mean play count
of 1878.
User 2064012 has a total play count of 548427 and a mean play count
of 9455.
User 2069337 has a total play count of 393515 and a mean play count
of 1519.
```

Splitting Data for Testing

Use the randomSplit function to divide the data (userArtistData) into:

- A training set, trainData, that will be used to train the model. This set should constitute 40% of the data.
- A validation set, validationData, used to perform parameter tuning. This set should constitute 40% of the data.
- A test set, testData, used for a final evaluation of the model. This set should constitute 20% of the data.

Using a random seed value of 13. Since these datasets will be repeatedly used, we want to persist them in memory using the cache function.

In addition, printing out the first 3 elements of each set as well as their sizes;

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112,  423)]
[(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139,  4)]
19817
19633
10031
```

In [4]:

```
trainData, validationData, testData = userArtistData.randomSplit([4,4,
2], seed=13)
trainData.cache()
validationData.cache()
testData.cache()



print trainData.take(3)
print  validationData.take(3)
print  testData.take(3)
print trainData.count()
print validationData.count()
print testData.count()
```

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113,  5)
]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637,  1000112,
423)]
[(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637,  1000139
, 4)]
19817
19633
10031
```

Model Evaluation

In [5]:

```
def modelEval(model, dataset):


    allArtists = userArtistData.map(lambda x: x[1]).collect()


    userArtists = set(dataset.map(lambda x: x[0]).collect())



    userArtistsDict = dict(dataset.map(lambda x: (x[0], x[1])).groupByKey().mapValues(set).collect())


    userArtistTrain = dict(trainData.map(lambda x: (x[0],x[1])).groupByKey().mapValues(set).collect())



    total = 0
    for key in userArtists:

        nonTrainArtists = set(allArtists) - userArtistTrain[key]

        origArtists = userArtistsDict[key]

        origArtistsCnt = len(origArtists)

        userArtistTest = sc.parallelize(map(lambda x: (key, x),nonTrainArtists))

        predArtists  =  model.predictAll(userArtistTest).sortBy(ascending=False, keyfunc = lambda x: x[2]).map(lambda x:x[1]).take(origArtistsCnt)

        total +=  (float(len(set(predArtists).intersection(origArtists))) / origArtistsCnt)



    print "The model score for rank %d is %f"%(rank,float(total)/len(userArtists))
```

Model Construction

We built the best model possibly using the validation set of data and the modelEval function. Although, there are a few parameters we could optimize, for the sake of time, we just try a few different values for the rank parameter through the values [2, 10, 20] and figured out which one produces the highest scored based on the model evaluation function.

```
In [6]: rankList = [2,10,20]
        for rank in rankList:
            model = ALS.trainImplicit(trainData, rank , seed=345)
            modelEval(model,validationData)
```

```
The model score for rank 2
is 0.093462 The model score
for rank 10 is 0.097899 The
model score for rank 20 is
0.084259
```

Now, using the bestModel, we will check the results over the test data.

```
In [7]: bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
        modelEval(bestModel, testData)
```

```
The model score for rank 20 is 0.061246
```

Trying Some Artist Recommendations

Using the best model above, we predicted the top 5 artists for user 1059637 using the recommendProducts into the real artist's name using artistAlias.

Result/Output

In [8]:

```
topRating = bestModel.recommendProducts(1059637, 5)
artistRating = map(lambda x: x.product, topRating)
artistDataDict = dict(artistData.collect())
count = 0
for key in artistRating:
    print "Artist " + str(count) + ":", artistDataDict[key]
    count += 1
```

```
Artist 0:
blink-182
Artist 1:
Elliott
Smith
Artist 2: Taking Back
Sunday Artist 3:
Incubus
Artist 4: Death Cab for Cutie
```

CONCLUSION

In this experiment, we were able to make a music recommendation system using basic metadata-based model and popular music recommender approach: collaborative filtering technique.

Though it has achieved great success, its drawbacks such as popularity bias and human efforts are obvious.

The model score for rank 20 is 0.061246 and each user's personal preference has been studied and their top 5 favorite artists is presented to respective user.

References

- [1] Everyone listens to music, but how we listen is changing. [online] Available at: http://www.nielsen.com/us/en/insights/news/2015/everyonelistens-to-music-but-how- we-listen-is-changing.html [Accessed 10 Oct. 2017].
- [2] Labrosa.ee.columbia.edu. (2017). Million Song Dataset | scaling MIR research. [online] Available at: https://labrosa.ee.columbia.edu/millionsong/ [Accessed 10 Oct. 2017].
- [3] en.wikipedia.org. (2017). Netflix Prize. [online] Available at: https://en.wikipedia.org/wiki/Netflix_Prize [Accessed 11 Oct. 2017].
- [4] Linden, G., Smith, B. and York, J. (2003). Amazon.com Recommendations Item-to- Item Collaborative Filtering. [ebook] Available at: https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf [Accessed 10 Oct. 2017].

- [5] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW '17). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173- 182. DOI: https://doi.org/10.1145/3038912.3052569
- [6] Gershman, A. and Meisels, A. (2015). A Decision Tree Based Recommender System. [ebook] IEEE Xplore, pp.170- 179. Available at: https://subs.emis.de/LNI/Proceedings/Proceedings165/170.p df [Accessed 9 Oct. 2017].
- [7] Min SH., Han I. (2005) Recommender Systems Using Support Vector Machines. In: Lowe D., Gaedke M. (eds) Web Engineering. ICWE 2005. Lecture Notes in Computer Science, vol 3579. Springer, Berlin, Heidelberg
- [8] R-bloggers. (2017). Hybrid content-based and collaborative filtering recommendations with {ordinal} logistic regression (2): Recommendation as discrete choice. [online] Available at: https://www.r-bloggers.com/hybrid- content-based-and- collaborative-filteringrecommendations-with-ordinal-logistic-regression-2-recommendation-as-discrete-choice/ [Accessed 7 Oct. 2017].