

**A Project Report  
on  
ARTIST RECOMMENDATION SYSTEM USING  
COLLABORATIVE FILTERING**

**submitted in partial fulfillment of the requirements for the award of the degree  
of  
BACHELOR OF TECHNOLOGY  
in  
COMPUTER SCIENCE AND ENGINEERING**

**by**

**17WH1A05A9**

**Ms. N. HARSHINI**

**17WH1A05A7**

**Ms. MNS.YAMINI**

**17WH1A0568**

**Ms. M. AKSHITHA**

**under the esteemed guidance of**

**Ms. A. KRANTHI  
Assistant Professor**



**Department of Computer Science and Engineering  
BVRIT HYDERABAD  
College of Engineering for Women  
(NBA Accredited – EEE, ECE, CSE and IT)  
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)  
Bachupally, Hyderabad – 500090**

**May, 2021**

## **DECLARATION**

We hereby declare that the work presented in this project entitled “**ARTIST RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING**” submitted towards completion of Project Work in IV year of B.Tech., CSE at ‘BVRIT HYDERABAD College of Engineering for Women’, Hyderabad is an authentic record of our original work carried out under the guidance of Ms. A.Kranthi, Assistant Professor, Department of CSE.

Sign. with date:

**Ms. N.HARSHINI**  
**(17WH1A05A9)**

Sign. with date:

**Ms. MNS.YAMINI**  
**(17WH1A05A7)**

Sign. with date:

**Ms. M.AKSHITHA**  
**(17WH1A0568)**

**BVRIT HYDERABAD**  
**College of Engineering for Women**  
**(NBA Accredited – EEE, ECE, CSE and IT)**  
**(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)**  
**Bachupally, Hyderabad – 500090**

**Department of Computer Science and Engineering**



**Certificate**

This is to certify that the Project Work report on “**ARTIST RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING**” is a bona fide work carried out by Ms. N.HARSHINI (17WH1A05A9) ; Ms. MNS.YAMINI (17WH1A05A7) ; Ms. M.AKSHITHA (17WH1A0568) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

**Head of the Department**  
**Dr. K. Srinivasa Reddy**  
**Professor and HoD ,**  
**Department of CSE**

**Guide**  
**Ms. A. Kranthi**  
**Assistant Professor**

**External Examiner**

## **Acknowledgements**

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K. Srinivasa Reddy, Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. A. Kranthi, Assistant Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for her constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE Department** who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

**Ms. N.HARSHINI**  
**(17WH1A05A9)**

**Ms. MNS. YAMINI**  
**(17WH1A05A7)**

**Ms. M.AKSHITHA**  
**(17WH1A0568)**

## **ABSTRACT**

With commercial music streaming service which can be accessed from mobile devices, the availability of digital music currently is abundant compared to previous era. Sorting out all this digital music is a very time-consuming and causes information fatigue. Therefore, it is very useful to develop a music recommender system that can search in the music libraries automatically and suggest suitable songs to users. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously. The project develops a music recommender system that can give recommendations based on similarity of features on audio listened by the user. This study uses matrix factorization and collaborative filtering to look similarity between features. The results of this study indicate that users prefer recommendations that consider music genres compared to recommendations based solely on similarity.

## LIST OF FIGURES

S.No.	Topic	Page No.
	Abstract	i
	List of Figures	ii
1.	Introduction	1
	1.1 Objectives	1
	1.2 Methodology	1
	1.2.1 Dataset	1
	1.2.2 Matrix Factorization	2
	1.2.3 Collaborative Filtering	2
	1.3 Organization of Project	3
2.	Theoretical Analysis of the proposed project	5
	2.1 Requirements Gathering	5
	2.1.1 Software Requirements	5
	2.1.2 Hardware Requirements	5
	2.2 Technologies Description	5
3.	Design	7
	3.1 Data Flow Diagram	7
4.	4.1 Coding	8
	4.2 Test Cases	13
	4.3 Training Dataset Screenshots	13
	4.4 Output Screenshots	16
5.	Conclusion and Future Scope	18
6.	References	19

## 1. INTRODUCTION

With commercial music streaming service which can be accessed from mobile devices, the availability of digital music currently is abundant compared to previous era. Sorting out all this digital music is a very time-consuming and causes information fatigue. Therefore, it is very useful to develop a music recommender system that can search in the music libraries automatically and suggest suitable songs to users. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously.

### 1.1 Objectives

The project develops a music recommender system that can give recommendations based on similarity of features on audio listened by the user. This study uses matrix factorization and collaborative filtering to look similarity between features. The results of this study indicate that users prefer recommendations that consider music genres compared to recommendations based solely on similarity.

### 1.2 Methodology

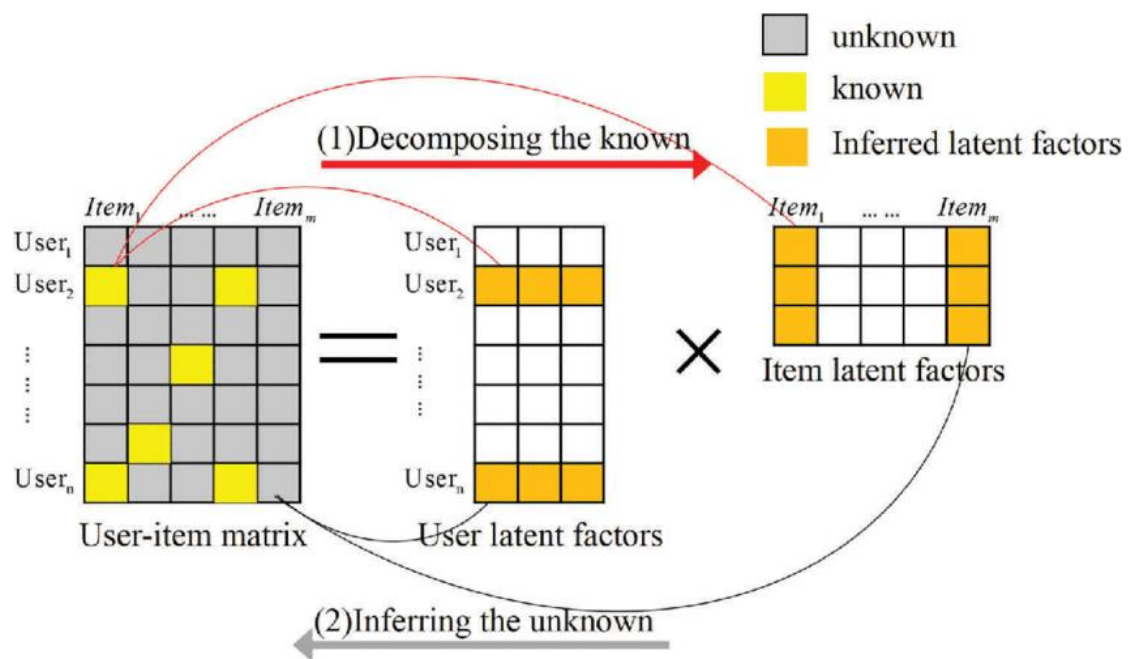
To provide artist recommendations to the users, large amount of song data is required. The songs are downloaded from the user\_artist\_data database. In this section the methodology followed is discussed in detail.

#### 1.2.1 Dataset

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.

### 1.2.2 Matrix Factorization

Matrix factorization is a common and effective way to implement a recommendation system. Matrix factorization attempts to learn ways to quantitatively characterize both users and items in a lower-dimensional space (as opposed to looking at every item the user has ever rated), such that these characterizations can be used to predict user behavior and preferences for a known set of possible items. In recent years, matrix factorization has become increasingly popular due to its accuracy and scalability.



### 1.2.3 Collaborative Filtering

Collaborative Filtering is one of the most widely used and successful technologies in recommendation systems. Collaborative filtering-based recommendation techniques have achieved great success, and have a wide range of application prospects in many fields such as e-commerce and social networks. However, as big data arise, the Collaborative Filtering-based approach often suffers from several shortcomings, such as data sparsity, cold start, and scalability issues, which seriously affect the recommended quality of recommendation systems. To tackle the aforementioned problems, many data mining and machine learning techniques such as clustering, singular value decomposition (SVD), probability matrix factorization (PMF), and non-negative matrix factorization (NMF) are proposed to improve the performance of recommendation systems.



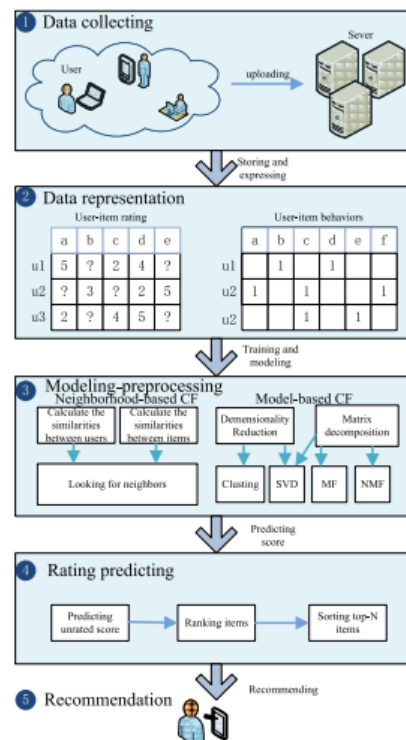


Fig 1.2.3.1 Framework of Collaborative Filtering based recommendation system

## 1.3 Organization of Project

The technique which is developed is taking input as a user data and train the uploaded data from the dataset using convolutional recurrent neural network and collaborative filtering. Based on user's maximum play count, then recommends artists as a result.

We have three modules in our project.

- Model Evaluation
- Model Construction
- Trying Some Artist Recommendations

### Necessary Package Imports

```
In [ ] : from pyspark.mllib.recommendation
import * import random from operator import *
```

### Loading data

Load the three datasets into RDDs and name them artistData, artistAlias, and userArtistData. View the README, or the files themselves, to see how this data is

formatted. Some of the files have tab delimiters while some have space delimiters. Make sure that `userArtistDataRDD` contains only the canonical artist IDs.

### Data Exploration

Finding the 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).

The output should look as follows:

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

Using the `randomSplit` function to divide the data (`userArtistData`) into: A training set, `trainData`, that will be used to train the model. This set constitutes 40% of the data. A validation set, `validationData`, used to perform parameter tuning. This set constitutes 40% of the data. A test set, `testData`, is used for a final evaluation of the model. This set constitutes 20% of the data. Using a random seed value of 13. Since these datasets will be repeatedly used this will probably needed to persist them in memory using the `cache` function.

In addition, print out the first 3 elements of each set as well as their sizes; if you created these sets correctly, your output should look as follows:

```
[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000113, 5)] [(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000112, 4 23)] [(1059637, 1000094, 1), (1059637, 1000130, 19129), (1059637, 1000139, 4)] 19817 19633 10031
```

## **2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT**

### **2.1 Requirements Gathering**

#### **2.1.1 Software Requirements**

Programming Language : Python 3.6

Dataset : user\_artist\_data.txt, artist\_alias.txt, artist\_data.txt Datasets

Packages : pyspark

Tool : Google Colaboratory

#### **2.1.2 Hardware Requirements**

Operating System: Windows 10

Processor : Intel Core i5-8265U

CPU Speed : 1.80 GHz

Memory : 4 GB (RAM)

### **2.2 Technologies Description**

#### **Python**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

- Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive – you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or

understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

## **Datasets**

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.

## **PySpark**

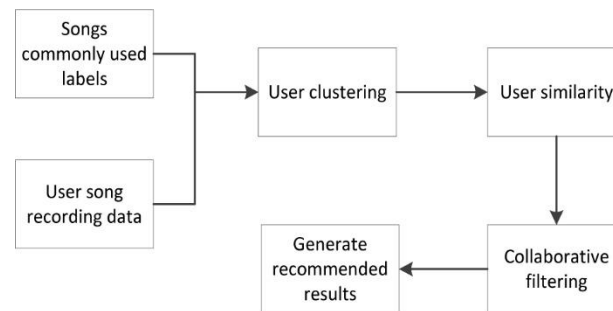
PySpark is the Python API written in python to support Apache Spark. ... Apache Spark is written in Scala and can be integrated with Python, Scala, Java, R, SQL languages. Spark is basically a computational engine, that works with huge sets of data by processing them in parallel and batch systems.

## **Google Colab**

Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded into notebook.

### 3. DESIGN

#### 3.1 Data Flow Diagram



**Fig 3.1.1: Data Flow Diagram**

The above image is the data flow of artist recommendation system using collaborative filtering. From the database user\_artist\_data.txt artist labels and user mean play count is considered, then user clustering is performed, and on considering user similarity in artist labels, collaborative filtering is done where top artists are recommended based on the play history of the user.

## 4. IMPLEMENTATION

### 4.1 Coding

```

!pip install pyspark
from pyspark.mllib.recommendation import *
import random
import pandas as pd
from operator import *
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
artistData = sc.textFile("/content/sample_data/artist_data_small.txt").map(lambda l:
l.split("\t")).map(lambda l: (int(l[0]), l[1]))
artistAlias = sc.textFile("/content/sample_data/artist_alias_small.txt").map(lambda l:
l.split("\t")).map(lambda l: (int(l[0]), int(l[1])))
artistAliasDict = artistAlias.collectAsMap()
def canonical(ele):
    if ele in artistAliasDict:
        return artistAliasDict.get(ele)
    return int(ele)
userArtistData =
sc.textFile("/content/sample_data/user_artist_data_small.txt").map(lambda line:
line.split(" ").map(lambda e: (int(e[0]), canonical(e[1]), int(e[2]))))
import array
from collections import namedtuple
from pyspark import SparkContext, since
from pyspark.rdd import RDD
from pyspark.mllib.common import JavaModelWrapper, callMLlibFunc, inherit_doc
from pyspark.mllib.util import JavaLoader, JavaSaveable
from pyspark.sql import DataFrame
__all__ = ['MatrixFactorizationModel', 'ALS', 'Rating']
class Rating(namedtuple("Rating", ["user", "product", "rating"])):
    @since("0.9.0")
    def predict(self, user, product):
        """ Predicts rating for the given user and product. """

```

```

        return self._java_model.predict(int(user), int(product))

@since("0.9.0")
def predictAll(self, user_product):
    """ Returns a list of predicted ratings for input user and product pairs. """
    assert isinstance(user_product, RDD), "user_product should be RDD of (user,
product)"
    first = user_product.first()
    assert len(first) == 2, "user_product should be RDD of (user, product)"
    user_product = user_product.map(lambda u_p: (int(u_p[0]), int(u_p[1])))
    return self.call("predict", user_product)

@since("1.2.0")
def userFeatures(self):
    """ Returns a paired RDD, where the first element is the user and the second is an
array of features corresponding to that user. """
    return self.call("getUserFeatures").mapValues(lambda v: array.array('d', v))

@since("1.2.0")
def productFeatures(self):
    """ Returns a paired RDD, where the first element is the product and the second is an
array of features corresponding to that product. """
    return self.call("getProductFeatures").mapValues(lambda v: array.array('d', v))

@since("1.4.0")
def recommendUsers(self, product, num):
    """ Recommends the top "num" number of users for a given product and returns a list
of Rating objects sorted by the predicted rating in descending order. """
    return list(self.call("recommendUsers", product, num))

@since("1.4.0")
def recommendProducts(self, user, num):

```

""""Recommends the top "num" number of products for a given user and returns a list of Rating objects sorted by the predicted rating in descending order.""""

```
return list(self.call("recommendProducts", user, num))
```

```
def recommendProductsForUsers(self, num):
```

"""" Recommends the top "num" number of products for all users. The number of recommendations returned per user may be less than "num". """"

```
return self.call("wrappedRecommendProductsForUsers", num)
```

```
temp1 = pd.read_csv("/content/sample_data/artist_alias_small.csv")
```

```
temp1.head()
```

```
temp1.describe()
```

```
temp2= pd.read_csv("/content/sample_data/artist_data_small.csv")
```

```
temp2.head()
```

```
temp2.describe()
```

```
temp3 = pd.read_csv("/content/sample_data/user_artist_data_small.csv")
```

```
temp3.head()
```

```
temp3.describe()
```

```
userPlayList = userArtistData.map(lambda x: (x[0], (x[1],
```

```
x[2]))).aggregateByKey((0,0),\
```

```
(lambda x,y: (x[0] + 1, x[1] + y[1])),\
```

```
(lambda rdd1, rdd2: (rdd1[0]+rdd2[0], rdd1[1]+rdd2[1]))).map(lambda t:
```

```
(t[0], t[1][1], t[1][1]/t[1][0]))
```

```
top3Users = userPlayList.sortBy(ascending=False, keyfunc=(lambda x: x[1])).take(3)
```

```
for t in top3Users:
```

```
    print("User " + str(t[0]) + " has a total play count of " + str(t[1]) + " and a mean play  
count of " + str(t[2]) + ".")
```

```
trainData, validationData, testData = userArtistData.randomSplit([0.4,0.4,0.2], 13)
```



```

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

def modelEval(model, dataset):

    #get all users and artists
    allUser = set(userArtistData.map(lambda x: (x[0])).collect())
    allArtist = set(userArtistData.map(lambda x: (x[1])).collect())

    #get dictionary of user and their artists for traindata and validation data
    trainUserArtist = trainData.map(lambda x: (x[0], {x[1]})).reduceByKey(lambda
x,y: x.union(y)).collectAsMap()
    datasetUserArtist = validationData.map(lambda x: (x[0],
{x[1]})).reduceByKey(lambda x,y: x.union(y)).collectAsMap()

    overAllScore = 0
    for user in allUser:
        # all actual user preferred artists
        actualUserArtist = datasetUserArtist[user]
        # all artist except that from traindata
        nonTrainDataArtist = allArtist - trainUserArtist[user]
        # prepare the test data consisting of tuples (user, product) => (user, artist)
        test = map(lambda x: (user, x), nonTrainDataArtist)
        # convert the set into rdd as rdd is required by predictAll method
        testRDD = sc.parallelize(test)
        # predict the ratings

```

```

predictedUserArtist = model.predictAll(testRDD)
# sort the ratings in descending order of ratings
sortedPrediction = predictedUserArtist.sortBy(ascending=False,
keyfunc=lambda x: x.rating)
# get number of top artists equal to number of actual artist for the user
c = len(actualUserArtist)
predictedSet = set(sortedPrediction.map(lambda x: x.product).take(c))
# take the intersection of actual artist and predicted artist for a user and calculate
the score
correctPrediction = predictedSet.intersection(actualUserArtist)
overAllScore += float(len(correctPrediction)) / c

return "The model score for rank "+str(model.rank)+" is
"+str(overAllScore/len(allUser))

model = ALS.trainImplicit(trainData, rank=2, seed=345)
print(modelEval(model, validationData))
model = ALS.trainImplicit(trainData, rank=10, seed=345)
print(modelEval(model, validationData))
model = ALS.trainImplicit(trainData, rank=20, seed=345)
print(modelEval(model, validationData))

bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
modelEval(bestModel, testData)
artistDict = artistData.collectAsMap()
userid=2010008
ratings = bestModel.recommendProducts(userid, 5)
ratingsArtists = map(lambda r : r.product, ratings)
i=1
print("Artist recommendations for user with ",userid," are:")
for artist in ratingsArtists:
    print("Artist "+ str(i)+ ":",artistDict[artist])
    i+=1

```

## 4.2 TEST CASES

Integrated and regression testing strategies are used in this application for testing.

Test Case Id	Test Scenario	Expected Result	Actual Result	Pass/Fail
TC01	Check whether packages are imported	Should import without errors	As Expected	Pass
TC02	Check whether datasets are defined and described	Show datasets	As Expected	Pass
TC03	Check whether model evaluation is working	Creation of model	As Expected	Pass
TC04	Check training, testing and validation of datasets	Divide the data for training, testing and validation	As Expected	Pass
TC05	Check artist recommendations	Show top artists for a user	As Expected	Pass

**Fig 4.2.1: Test Cases**

## 4.3 DATASET TRAINING SCREENSHOTS

```
temp1 = pd.read_csv("/content/sample_data/artist_alias_small.csv")
temp1.head()
```

	badid	goodid
0	1027859	1252408
1	1017615	668
2	6745885	1268522
3	1018110	1018110
4	1014609	1014609

```
[ ] temp1.describe()
```

	badid	goodid
count	5.870000e+02	5.870000e+02
mean	3.348950e+06	1.469856e+06
std	3.223688e+06	1.830129e+06
min	1.500000e+01	1.500000e+01
25%	1.060312e+06	1.006480e+06
50%	1.291110e+06	1.034635e+06
75%	6.664274e+06	1.259498e+06

**Fig 4.3.1: Running artist\_alias\_small.csv dataset**

```
[ ] temp2= pd.read_csv("/content/sample_data/artist_data_small.csv")
temp2.head()
```

	artistid	artist_name
0	1240113	riow arai
1	1240132	Outkast & Rage Against the Machine
2	6776115	小松正夫
3	1030848	Raver's Nature
4	6671601	Erguner, Kudsi

```
[ ] temp2.describe()
```

	artistid
count	3.053700e+04
mean	2.723610e+06
std	2.875962e+06
min	1.000000e+00
25%	1.033180e+06
50%	1.238800e+06
75%	2.164639e+06

Fig 4.3.2: Running artist\_data\_small.csv

```
[ ] temp3 = pd.read_csv("/content/sample_data/user_artist_data_small.csv")
temp3.head()
```

	userid	artistid	playcount
0	1059637	1000010	238
1	1059637	1000049	1
2	1059637	1000056	1
3	1059637	1000062	11
4	1059637	1000094	1

```
[ ] temp3.describe()
```

	userid
count	4.948100e+04
mean	1.328420e+06
std	4.529913e+05
min	1.000647e+06
25%	1.024631e+06
50%	1.059245e+06
75%	2.010008e+06
max	2.288164e+06

Fig 4.3.3: Running user\_artist\_data\_small dataset

```
[ ] trainData, validationData, testData = userArtistData.randomSplit([0.4,0.4,0.2], 13)
print(trainData.take(3))
print(validationData.take(3))
print(testData.take(3))
print(trainData.count())
print(validationData.count())
print(testData.count())
trainData.cache()
validationData.cache()
testData.cache()

[(1059637, 1000049, 1), (1059637, 1000056, 1), (1059637, 1000114, 2)]
[(1059637, 1000010, 238), (1059637, 1000062, 11), (1059637, 1000123, 2)]
[(1059637, 1000094, 1), (1059637, 1000112, 423), (1059637, 1000113, 5)]
19769
19690
10022
PythonRDD[26] at RDD at PythonRDD.scala:53
```

**Fig 4.3.4: Command to train the dataset**

```
[ ] model = ALS.trainImplicit(trainData, rank=2, seed=345)
print(modelEval(model, validationData))
model = ALS.trainImplicit(trainData, rank=10, seed=345)
print(modelEval(model, validationData))
model = ALS.trainImplicit(trainData, rank=20, seed=345)
print(modelEval(model, validationData))
```

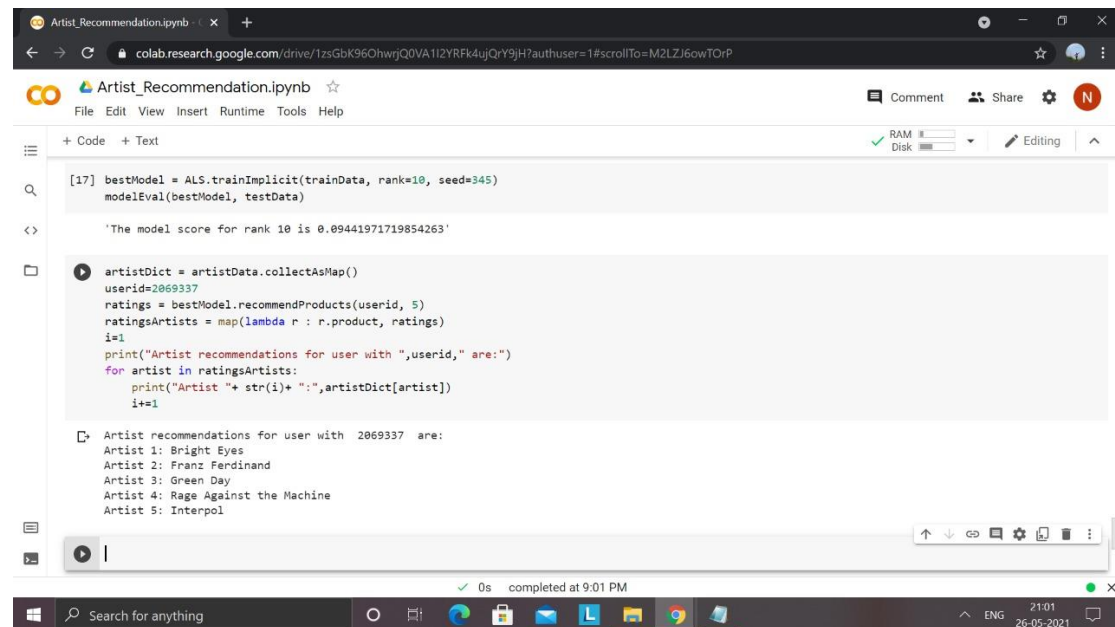
```
The model score for rank 2 is 0.08616827592156559
The model score for rank 10 is 0.09441971719854263
The model score for rank 20 is 0.08408995233356337
```

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

```
'The model score for rank 10 is 0.09441971719854263'
```

**Fig 4.3.5: Model Evaluation**

## 4.4 OUTPUT SCREENSHOTS



The screenshot shows a Jupyter Notebook interface with the following code and output:

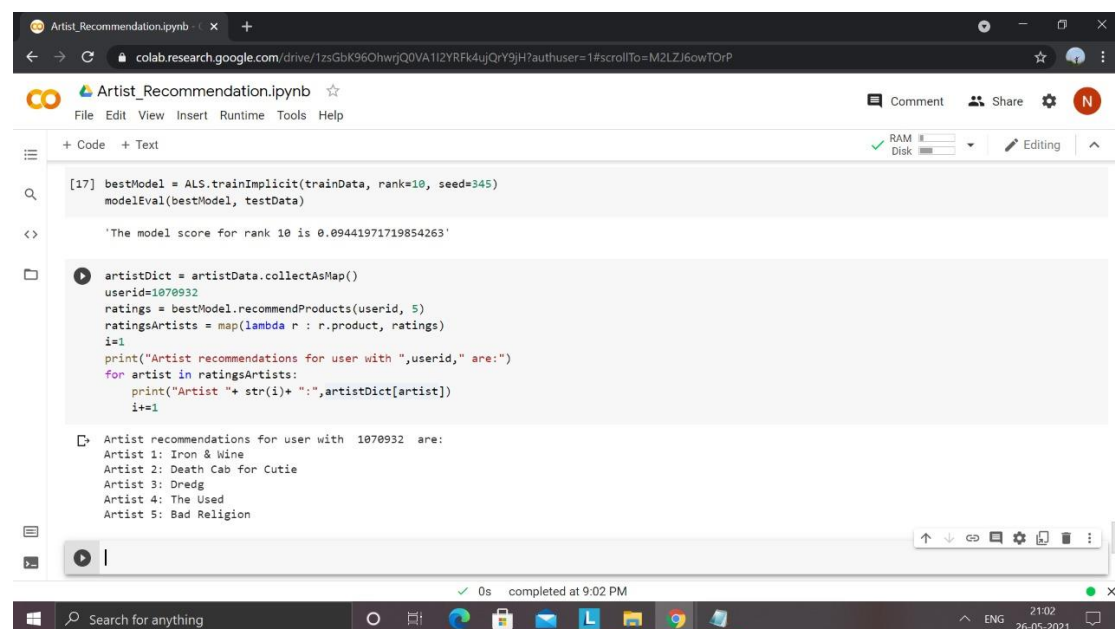
```
[17] bestModel = ALS.trainImplicit(trainData, rank=10, seed=345)
      modelEval(bestModel, testData)
```

The model score for rank 10 is 0.09441971719854263'

```
artistDict = artistData.collectAsMap()
userid=2069337
ratings = bestModel.recommendProducts(userid, 5)
ratingsArtists = map(lambda r : r.product, ratings)
i=1
print("Artist recommendations for user with ",userid," are:")
for artist in ratingsArtists:
    print("Artist "+ str(i)+ ":",artistDict[artist])
    i+=1
```

Artist recommendations for user with 2069337 are:  
 Artist 1: Bright Eyes  
 Artist 2: Franz Ferdinand  
 Artist 3: Green Day  
 Artist 4: Rage Against the Machine  
 Artist 5: Interpol

Fig 4.4.1: Output showing artist recommendations



The screenshot shows a Jupyter Notebook interface with the following code and output:

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

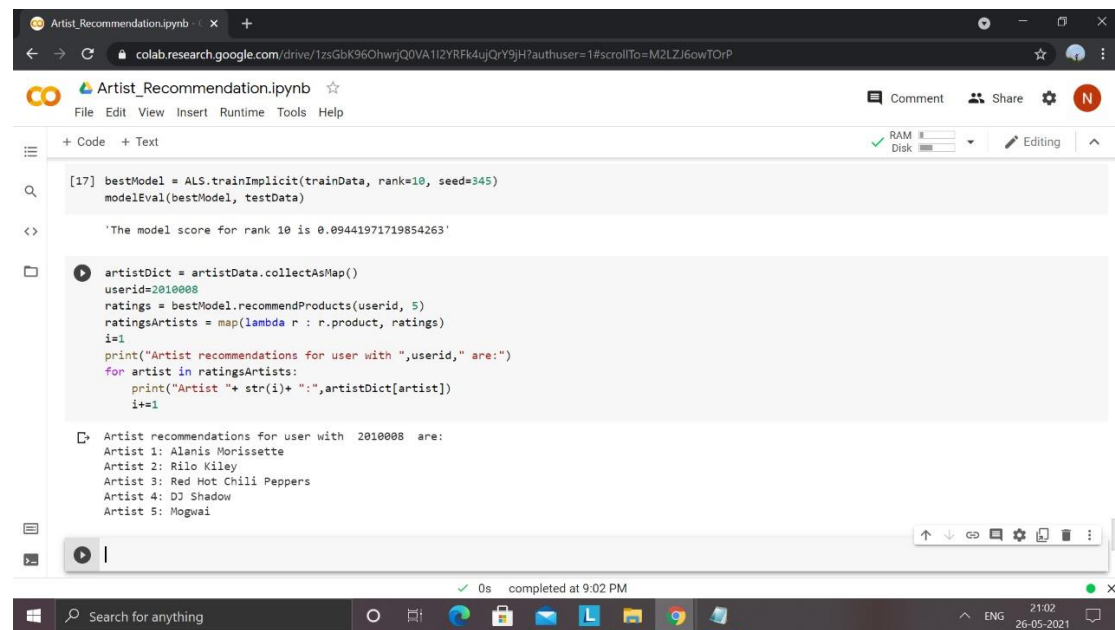
The model score for rank 10 is 0.09441971719854263'

```
artistDict = artistData.collectAsMap()
userid=1070932
ratings = bestModel.recommendProducts(userid, 5)
ratingsArtists = map(lambda r : r.product, ratings)
i=1
print("Artist recommendations for user with ",userid," are:")
for artist in ratingsArtists:
    print("Artist "+ str(i)+ ":",artistDict[artist])
    i+=1
```

Artist recommendations for user with 1070932 are:  
 Artist 1: Iron & Wine  
 Artist 2: Death Cab for Cutie  
 Artist 3: Dredg  
 Artist 4: The Used  
 Artist 5: Bad Religion

Fig 4.4.2: Output showing artist recommendations

## Artist Recommendation System using Collaborative Filtering



The screenshot shows a Google Colab notebook titled 'Artist\_Recommendation.ipynb'. The code in the notebook performs the following steps:

- Trains an ALS model with rank=10 and seed=345.
- Evaluates the model on test data, resulting in a score of 0.09441971719854263.
- Collects artist data into a dictionary.
- Recommends 5 products for user 2010008.
- Prints the artist recommendations for user 2010008.

The output of the notebook shows the following artist recommendations for user 2010008:

```
Artist recommendations for user with 2010008 are:  
Artist 1: Alanis Morissette  
Artist 2: Rilo Kiley  
Artist 3: Red Hot Chili Peppers  
Artist 4: DJ Shadow  
Artist 5: Mogwai
```

**Fig 4.4.3: Output showing artist recommendations**

## **5. CONCLUSION AND FUTURE SCOPE**

The application helps users to get artist recommendations from their playing history using collaborative filtering. With commercial music streaming service which can be accessed from mobile devices, the availability of digital music currently is abundant compared to previous era. Sorting out all this digital music is a very time-consuming and causes information fatigue. Therefore, it is very useful to develop a music recommender system that can search in the music libraries automatically and suggest suitable songs to users. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously.

The future enhancement of this application is

- To increase the accuracy.
- To display recommendations dynamically.



## 6. REFERENCES

- [1] Rui Chen, Qingyi Huai , Yan-Shuo Chang, Bo Wang, Lei Zhang, and Xiangjie Kong, (Senior Member, IEEE)), Collaborative Filtering-Based Recommender Systems, **IEEE Conference in 2018**. (Base paper)
- [2] Deldjoo Y, Cremonesi P, Schedl M, Quadrana M The effect of different video summarization models on the quality of video recommendation based on low-level visual features. (2017).
- [3] Lee JH, Wishkoski R, Aase L, Meas P, Hubbles C Understanding users of cloud music services: selection factors, management and access behavior, and perceptions. (2017)
- [4] Schäfer T, Mehlhorn C Can personality traits predict musical style preferences? A meta-analysis. (2017).
- [5] I. Portugal, P. Alencar, and D. Cowan, “The use of machine learning algorithms in recommender systems: A systematic review,” *Expert Syst. Appl.*, vol. 97, pp. 205–227, Dec(2018).