IBM COURSERA ADVANCE DATA SCIENCE CAPSTION PROJECT

Instructor

Romeo Kienzler
Chief Data Scientist, Course Lead
IBM Watson IoT

53,524 Learners
5 Courses

Offered by

IBM

IBM offers a wide range of technology and consulting services; a broad portfolio of middleware for collaboration, predictive analytics, software development and systems management; and the world's most advanced servers and supercomputers. Utilizing its business consulting, technology and R&D expertise, IBM helps clients become "smarter" as the planet becomes more digitally interconnected. IBM invests more than $6 billion a year in R&D, just completing its 21st year of patent leadership. IBM Research has received recognition beyond any commercial technology research organization and is home to 5 Nobel

SHOW ALL

Submitted by:-

Ankit Raj

OUTLINES:-

I. DATASET _ USE CASE
II. DATA EXPLORATION
III. DATA CLEANING AND TRAINING
IV. MODEL DEFINITION AND TRAINING
V. MODEL EVALUATION – HYPERPARAMETERS TUNING

Data Set:-

Dataset published by Audioscrobbler – a music recommendation system for last.fm

Contains:

- 140,000 unique users
- 1.6 million unique artists

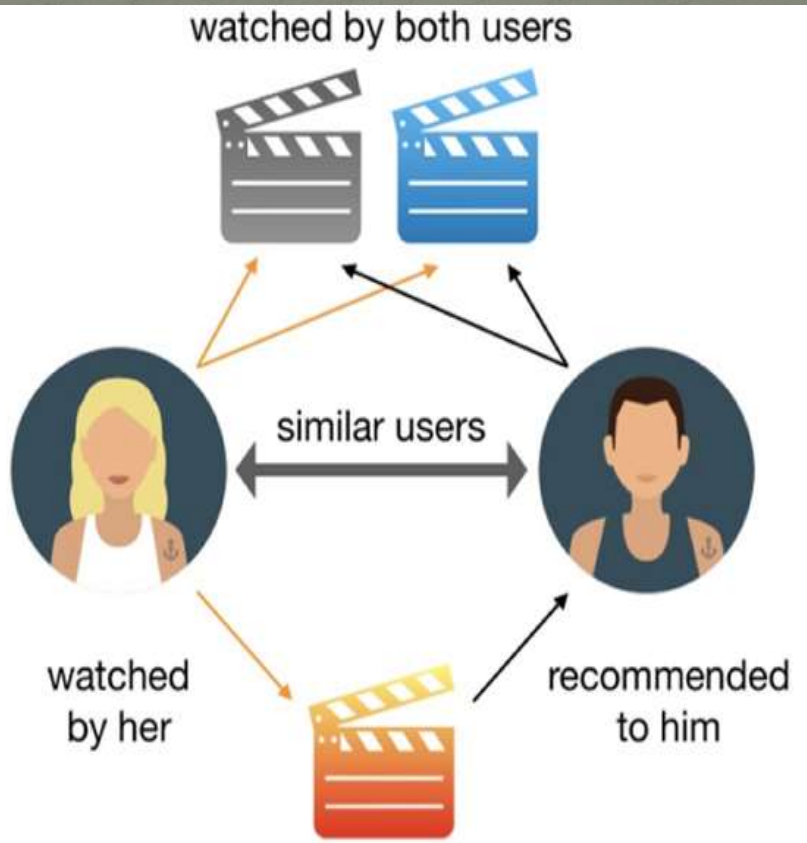
userID	artistID	playCount
113186	46843	56
745456	84646	118
…	…	…
…	…	…

misspelledID	standardID
84646	184528
148328	435846
…	…

artistID	artist_name
84646	Ed Sheeran
148328	Coldplay
…	…

USE CASE:-

Develop music recommendation system



DATA EXPLORATION:-

How many distinct users do we have in our data?

In [5]:
```
allusers = userArtistDF.count()
print("All rows in database: ", allusers )
uniqueUsers = userArtistDF.select('userID').distinct().count()
print("Total n. of distinct users: ", uniqueUsers)
```

```
All rows in database:  24296858
Total n. of distinct users:  148111
```

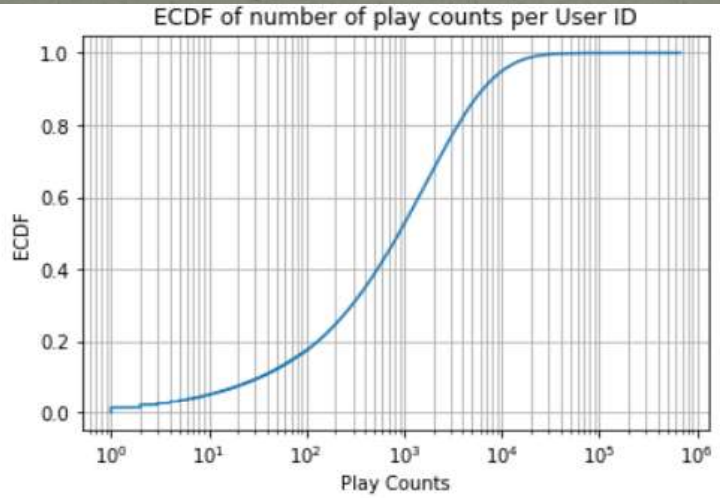
How many distinct artists do we have in our data ?

In [6]:
```
uniqueArtists = userArtistDF.select('artistID').distinct().count()
print("Total n. of artists: ", uniqueArtists)
```

```
Total n. of artists:  1631028
```

DATA EXPLORATION:-

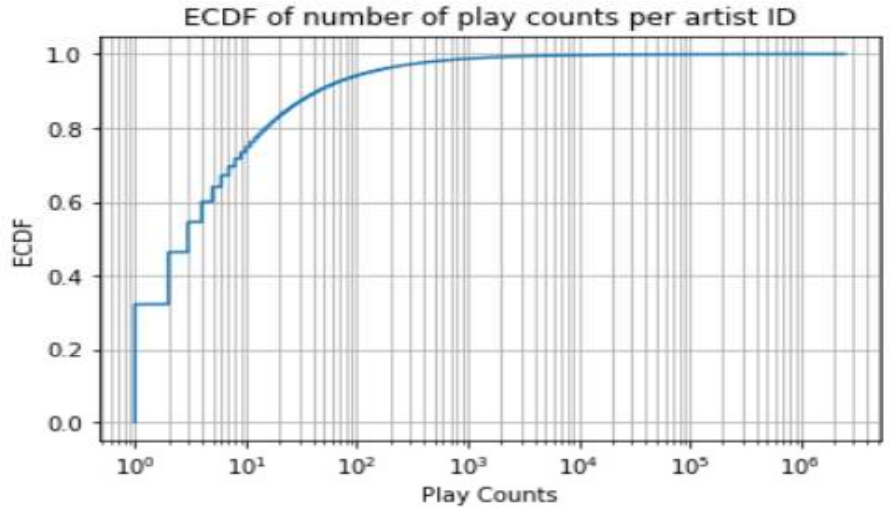
We have total 371638969 play counts.
In average, every user plays 2509 times.
25% of the users have the play counts less than or equal to (<=) 204 times.
50% of the users have the play counts less than or equal to (<=) 892times.
75% of the users have the play counts less than or equal to (<=) 2800 times.
95% of the users have the play counts less than or equal to (<=) 10120 times.
99% of the users have the play counts less than or equal to (<=) 21569 times.
The result is plausible with the figure above.
About 7746 users (5.23%) have the play counts less than or equal to (<=) 10 times. These users have very little interaction with the system, so there is more difficult for recommending for these users than other users creating more impact in the system (have a certain number of playCount).




```
Total = 371638969
Mean = 2509.1922207
Min = 1
Max = 674412
Percentile 25% :204.0
Percentile 50% :892.0
Percentile 75% :2800.0
Percentile 90% :6484.0
Percentile 95% :10120.0
Percentile 99% :21569.2
The percentage of user playing less than 10 times P(Y<=10) = 0.05228511049145573
```

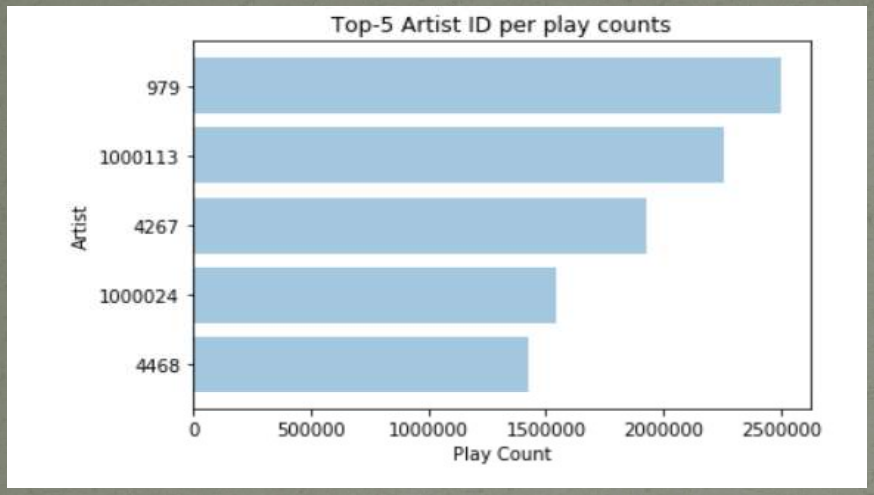
DATA EXPLORATION:-

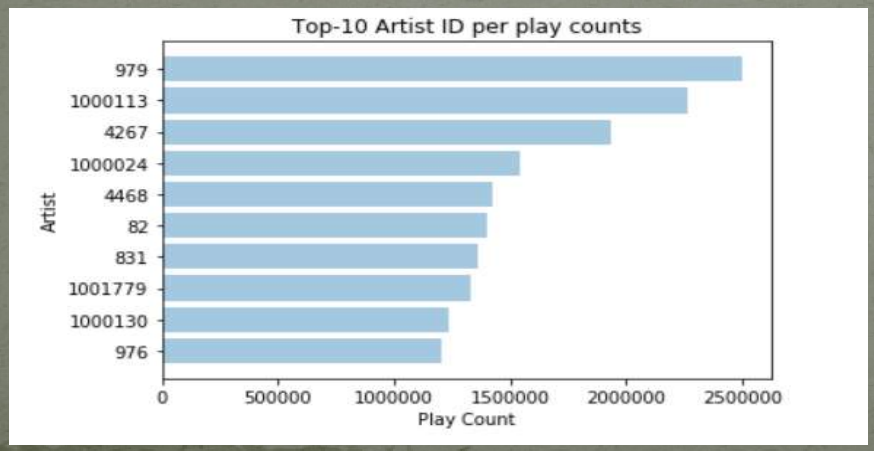
Total 371638969 play counts. In average, playCount per artist is 227 times
Only 74.87% of the artists is played less than or equal to (<=) 10 times.
And 98.74% of the artists is played less than or equal to (<=) 1000 times.
In the other hand, we have top 5 artist play counts: [1425942 1542806 1930592 2259185 2502130]. This accounts for 2.6% on overall number of playCount (5 out of 1631028 artists). Moreover, the play count of top 5 artist is much higher than the mean. So we can implie that we can recommend most-played artists to every user with this top 5 artirst, and still get high performance.

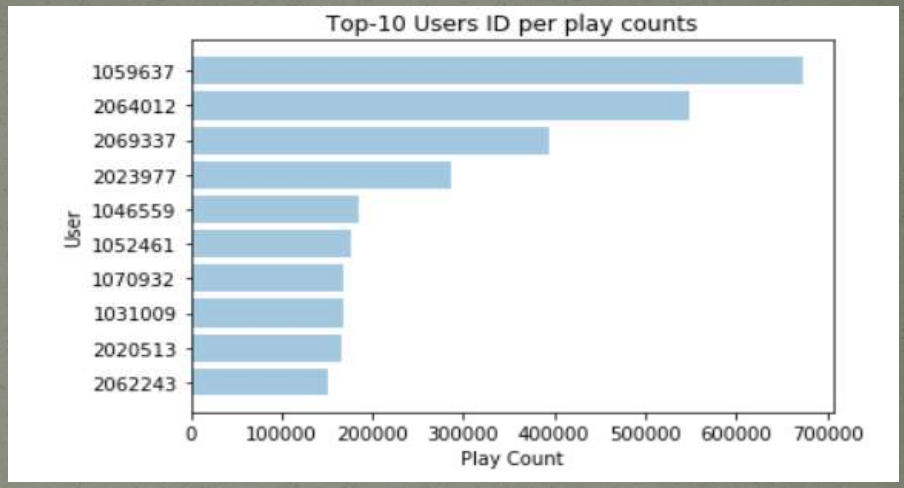


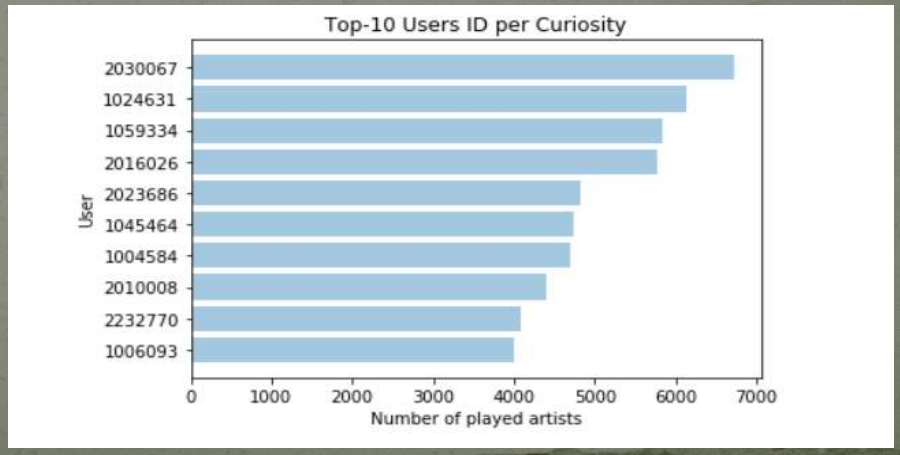

```
Sum = 371638969
Mean = 227.855664648
Min = 1
Max = 2502130
Top 5 play counts: [1425942 1542806 1930592 2259185 2502130]
Sum top 5 artist play counts: 9660655
Percentage of top 5 artist play counts: 0.0259947309239
P(playCount<=10) = 0.7486793605014751
P(playCount<=1000) = 0.987435531456235
```

DATA EXPLORATION BY BAR CHART





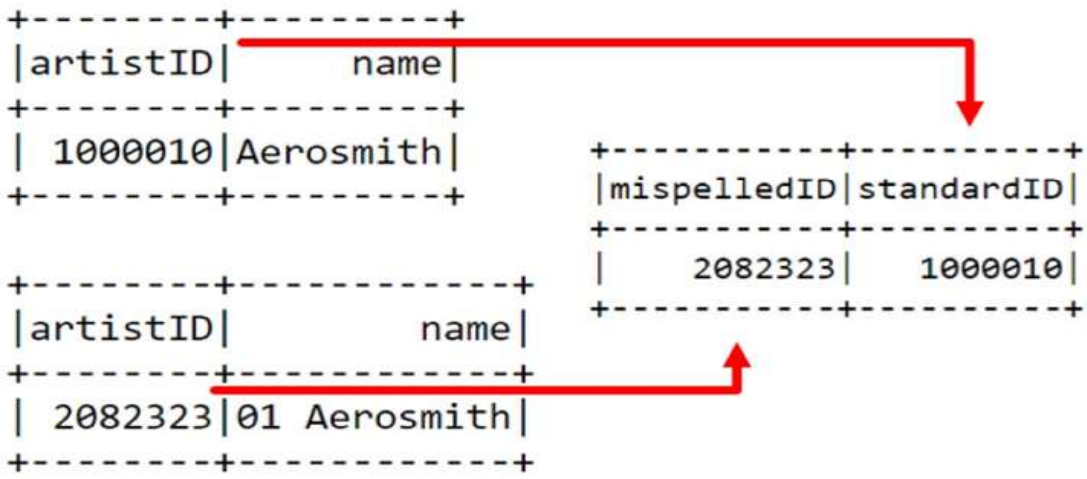




DATA CLEANSING:-

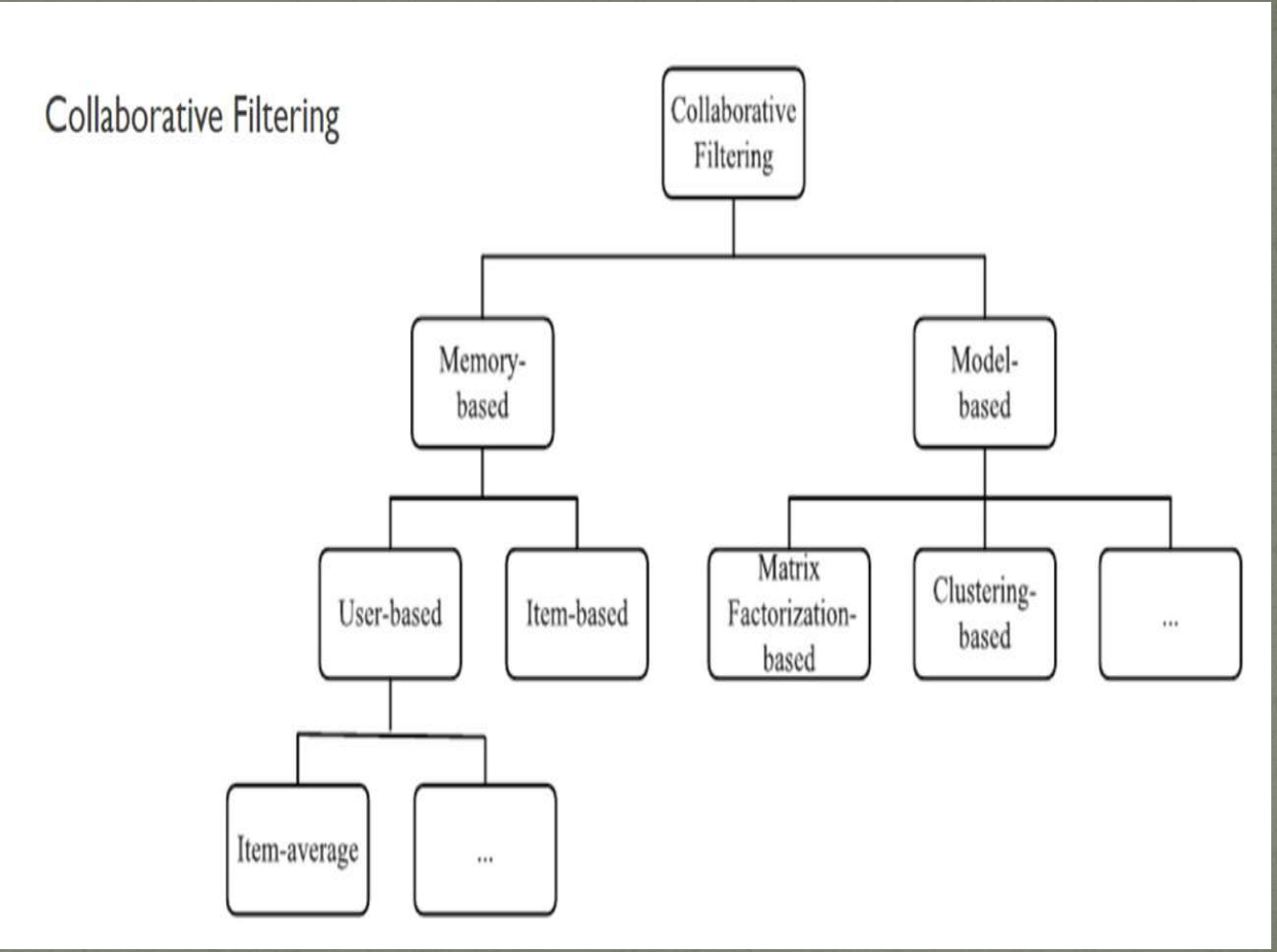
DATA CLEANSING

Same artist but different IDs =>



Solved: Replace misspelledID by StandardID

MODEL DEFINITION:-



Model definition:-

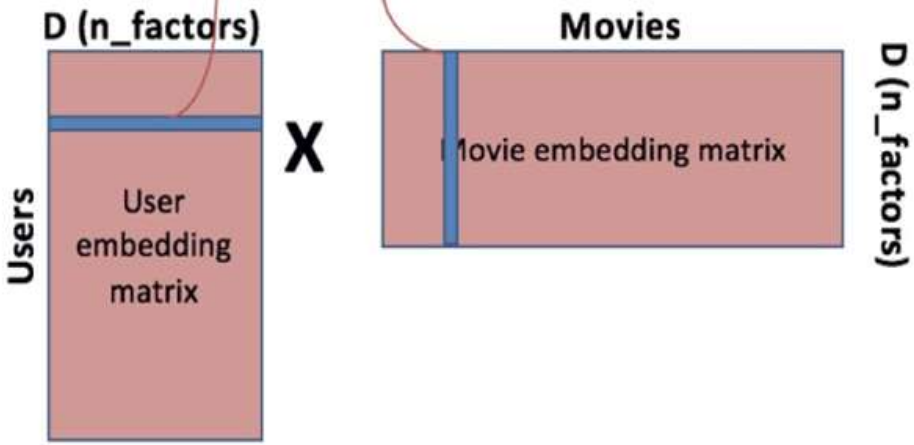
Matrix Factorization

Dot product of Movie-A with User-X
gives prediction for Movie-A by User-X

Movies

Users	*(Sparse matrix)*	Waking Life	Boyhood	Before Sunset
	Jesse	4.5		4.0
	Celine		3.5	5.0

≈



Model Training:-

A model can be trained by using ALS.trainImplicit(<training data>, <rank>)

- Split data to 70% for training and 30% for testing
- We can also use some additional parameters to adjust the quality of the model. Currently, let's set:

Param	Value
Rank	10
Iterations	5
Lambda	0.01
Alpha	1.0

```
#setting parameters
rank=10
iterations=5
lambda_=0.01
alpha=1.0

#training
t0 = time()
model = ALS.trainImplicit(allData, rank)
t1 = time()
print("finish training model in %f secs" % (t1 - t0))
```

```
finish training model in 83.877863 secs
```

Model Training:-

Pridict Top 5 Artish which User has ID = 2093760 , may find Interesting.

In [45]:
```
model = ALS.trainImplicit(ratings=trainData, rank=10, iterations=5, lambda_=1.0, alpha=40.0)
allData.unpersist()

userID = 2093760
recommendations = model.recommendProducts(userID,5)

recArtist = set(rating[1] for rating in recommendations)

# Filter in those artists, get just artist, and print
def artistNames(line):
#     [artistID, name]
    if (line[0] in recArtist):
        return True
    else:
        return False

recList = artistByID.filter(artistNames).values().collect()
print(recList)

unpersist(model)
```

```
['Ankit', 'John', 'kapllar', 'Obama', 'michael Jackson']
```

Evaluation Model:-

```
t0 = time()
auc = calculateAUC( cvData,bAllItemIDs, model.predictAll)
t1 = time()
print("auc=",auc)
print("finish in %f seconds" % (t1 - t0))
```

```
auc= 0.960706689415 73
finish in 79.103230 seconds
```

HYPERPARAMETERS TUNNING:-

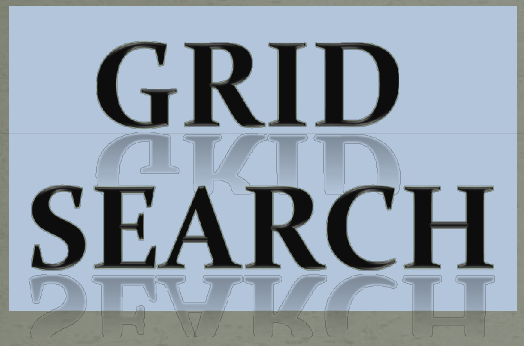
In [65]:
```
evaluations = []

for rank in [10, 50]:
    for lambda_ in [1.0, 0.0001]:
        for alpha in [1.0, 40.0]:
            print("Train model with rank=%d lambda_=%f alpha=%f" % (rank, lambda_, alpha))
            # with each combination of params, we should run multiple times and get avg
            # for simple, we only run one time.
            model = ALS.trainImplicit(ratings=trainData,rank=rank,iterations=5,lambda_=lambda_ ,alpha=alph
a)
            auc = calculateAUC(cvData,bListenCount,model.predictAll)

            evaluations.append(((rank, lambda_, alpha), auc))

            unpersist(model)
```

HYPERPARAMETERS TUNNING:-



Rank	Lambda	Alpha	AUC
10	1.0	40.0	0.9738
10	0.0001	40.0	0.9718
50	1.0	40.0	0.9715
50	0.0001	40.0	0.97
10	1.0	1.0	0.9644
50	1.0	1.0	0.9592
10	0.0001	1.0	0.9584
50	0.0001	1.0	0.9427

The combination of parameters that gets the highest AUC is: rank = 10 ; lambda = 1.0 ; alpha = 40

Conclusion:-

- The model is highly biased on the artists who have large number of play counts
- *we may not need to include lambda and alpha paramerers to the model if we only retrieve the top result less than 10 artists.

Thanking You:-

Taught by: Romeo Kienzler, Chief Data Scientist, Course Lead
IBM Watson IoT

Taught by: Niketan Pansare, Senior Software Engineer
IBM Research

Taught by: Tom Hanlon, Training Director
Skymind

Taught by: Max Pumperla, Deep Learning Engineer

Taught by: Ilja Rasin, Data Scientist
IBM Watson Health

THANKS TUTORS
THANKS COURSERA