databricks Song Recommendation Final

GROUP 4: SONG RECOMMENDATION PROJECT

Introduction

Recommendation systems are probably the most trending data science application today. They can be used to predict users rating or preference for any particular item. All of the major tech companies are using recommendation system in some form or the other. Amazon is using it to suggest "frequently bought together" or "Customers who viewed this item also viewed". YouTube is using it to create an auto playlist based on your preferences. Infact, for companies like Netflix and Spotify, the entire business model and its success revolves around how good their recommandation system is. What's more, Netflix offered a million dollar prize competition in year 2009 to improve its system prediction by 10%.

Problem statement

Build a song recommendation system which can recommend songs to listeners based on information on users behaviors, activities or preferences and predicting what users will like based on their similarity to other users".

Data Source: https://www.kaggle.com/c/msdchallenge/data (https://www.kaggle.com/c/msdchallenge/data)

Loading the data

```
# File location and type
file_song_data = "/FileStore/tables/song_data.csv"
file_triplets_data ="/FileStore/tables/triplets_10000.txt"
```

Uploading the data to the dataframe

```
#Uploading Song CSV file with tab delimited
song_df = spark.read.csv(file_song_data,
                            inferSchema ='true',
                            header = 'true',
                            sep=',')
```

```
song_df.show(5)
```

```
+------------------+-----------------+--------------------+----------------+---
-+
|           song_id|            title|             release|     artist_name|yea
r|
+------------------+-----------------+--------------------+----------------+---
-+
|SOQMMHC12AB0180CB8|     Silent Night|Monster Ballads X...|Faster Pussy cat|200
3|
|SOVFVAK12A8C1350D9|      Tanssi vaan|         Karkuteillä|Karkkiautomaatti|199
5|
|SOGTUKN12AB017F4F1|No One Could Ever|              Butter|  Hudson Mohawke|200
6|
|SOBNYVR12A8C13558C|    Si Vos Querés|             De Culo|     Yerba Brava|200
3|
|SOHSBXH12A8C13B0DF| Tangle Of Aspens|Rene Ablaze Prese...|      Der Mystic|
0|
+------------------+-----------------+--------------------+----------------+---
-+
only showing top 5 rows
```

Defining schema for triplets data

```
from pyspark.sql.types import *

# Creating schema
schema = StructType([StructField('user_id', StringType()),
                     StructField('songid', StringType()),
                     StructField('play_count', IntegerType())])

#Uploading Triplets file with tab delimited
tri_df = spark.read.csv(file_triplets_data,
                        schema= schema,
                          sep='\t')
```

```
tri_df.show(5)
```

```
+--------------------+------------------+----------+
|             user_id|            songid|play_count|
+--------------------+------------------+----------+
|b80344d063b5ccb32...|SOAKIMP12A8C130995|         1|
|b80344d063b5ccb32...|SOBBMDR12A8C13253B|         2|
|b80344d063b5ccb32...|SOBXHDL12A81C204C0|         1|
|b80344d063b5ccb32...|SOBYHAJ12A6701BF1D|         1|
|b80344d063b5ccb32...|SODACBL12A8C13C273|         1|
+--------------------+------------------+----------+
only showing top 5 rows
```

Implicit VS Explicit data:

Explicit data is the data where we have user rating associated with a song or a movie on a fixed scale. For instance: 1 to 5 ratings in the Netflix dataset. From such rating, we can interpret how much a user likes or dislikes a movie, but it is hard to get such data because generally users do not care to rate every movie they see.

Implicit data is the type of data we are using for song recommendation. The data is gathered from the user behavior, with no explicit rating associated with it. It could be how many times a user played a song or watched a movie, how long they have a spent reading a particular article etc. The advantage here is we have a lot of such data but it is usually very noisy and unreliable.

When a user rates a movie 1 on scale of 5 that means that he did not like the movie. But with play count of a song it we can't make any implicit assumption that the user loved the song or hated the song or somewhere-in-between. Also, if they did not play a song does not necessarily mean that they do not like the song. Therefore, we focus on what we know about the users behavior and the confidence we have in whether or not they like any given item. For instance: we can have a higher confidence on a song if the user played it 100 times against a song which he played on 1 time.

Joining the two dataframes to form MSD Data having the playcounts from tri_df

```
MSD = tri_df.join(song_df, tri_df.songid == song_df.song_id,how='left')
MSD.show(5)
```

```
+--------------------+------------------+----------+------------------+-----+--
----------------+----------+----+
|             user_id|            songid|play_count|           song_id|title|
release|artist_name|year|
+--------------------+------------------+----------+------------------+-----+--
----------------+----------+----+
|79f93851e840f9d1f...|SOATHTW12A58A7EDB5|         1|SOATHTW12A58A7EDB5| Mutt|En
ema Of The State|  Blink-182|1998|
|043d81932e75d5749...|SOATHTW12A58A7EDB5|         5|SOATHTW12A58A7EDB5| Mutt|En
ema Of The State|  Blink-182|1998|
|ebacfcb5fa29a601f...|SOATHTW12A58A7EDB5|         1|SOATHTW12A58A7EDB5| Mutt|En
ema Of The State|  Blink-182|1998|
|417c73dd95669d191...|SOATHTW12A58A7EDB5|         1|SOATHTW12A58A7EDB5| Mutt|En
ema Of The State|  Blink-182|1998|
|52ab33fbb2fa3aeb2...|SOATHTW12A58A7EDB5|         1|SOATHTW12A58A7EDB5| Mutt|En
ema Of The State|  Blink-182|1998|
+--------------------+------------------+----------+------------------+-----+--
----------------+----------+----+
only showing top 5 rows
```

Removing the redundant column

```
MSD =
MSD['user_id','song_id','play_count','title','release','artist_name','year']
MSD.show(5)
```

```
+--------------------+------------------+----------+-----+------------------+--
---------+----+
|             user_id|           song_id|play_count|title|           release|ar
tist_name|year|
+--------------------+------------------+----------+-----+------------------+--
---------+----+
|79f93851e840f9d1f...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|1998|
|043d81932e75d5749...|SOATHTW12A58A7EDB5|         5| Mutt|Enema Of The State|
Blink-182|1998|
|ebacfcb5fa29a601f...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|1998|
|417c73dd95669d191...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|1998|
|52ab33fbb2fa3aeb2...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|1998|
+--------------------+------------------+----------+-----+------------------+--
---------+----+
only showing top 5 rows
```

Changing the column name year to release_year

```
MSD = MSD.withColumnRenamed('year', 'release_year')
MSD.show(5)
```

```
+--------------------+------------------+----------+-----+------------------+--
---------+------------+
|             user_id|           song_id|play_count|title|           release|ar
tist_name|release_year|
+--------------------+------------------+----------+-----+------------------+--
---------+------------+
|79f93851e840f9d1f...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|        1998|
|043d81932e75d5749...|SOATHTW12A58A7EDB5|         5| Mutt|Enema Of The State|
Blink-182|        1998|
|ebacfcb5fa29a601f...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|        1998|
|417c73dd95669d191...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|        1998|
|52ab33fbb2fa3aeb2...|SOATHTW12A58A7EDB5|         1| Mutt|Enema Of The State|
Blink-182|        1998|
+--------------------+------------------+----------+-----+------------------+--
---------+------------+
```

```
only showing top 5 rows
```

Identifying total number of distinct songs and users

```
# Number of rows
print(MSD.count())
print(MSD.select("user_id").distinct().count())
print(MSD.select("song_id").distinct().count())
```

```
2086946
76353
10000
```

Data Exploration

```
# Create a view or table

temp_table_name = "user_playlist"

MSD.createOrReplaceTempView(temp_table_name)
```

```
%sql
select * from user_playlist limit 5;
```

user_id	song_id
79f93851e840f9d1faeba586ee18b30fdb0008b6	SOATHTW12A58A7EDB5
043d81932e75d5749ed5758d6420506e7bc457a5	SOATHTW12A58A7EDB5
ebacfcb5fa29a601f596b2d1076d7973177737e1	SOATHTW12A58A7EDB5
417c73dd95669d1919c869ef20fd2d0f7a31403d	SOATHTW12A58A7EDB5
52ab33fbb2fa3aeb2a261734603061e288a2b253	SOATHTW12A58A7EDB5

Most played 10 Songs and there Artists

```
%sql
select artist_name, title, sum(play_count) as number_of_total_play from
user_playlist group by title,artist_name order by sum(play_count) desc limit
10;
```

artist_name	title
Dwight Yoakam	You
Björk	Unc
Kings Of Leon	Rev
Barry Tuckwell/Academy of St Martin-in-the-Fields/Sir Neville Marriner	Hor
Harmonia	Seh
Florence + The Machine	Dog
Kings Of Leon	Use
OneRepublic	Sec
Five Iron Frenzy	Can

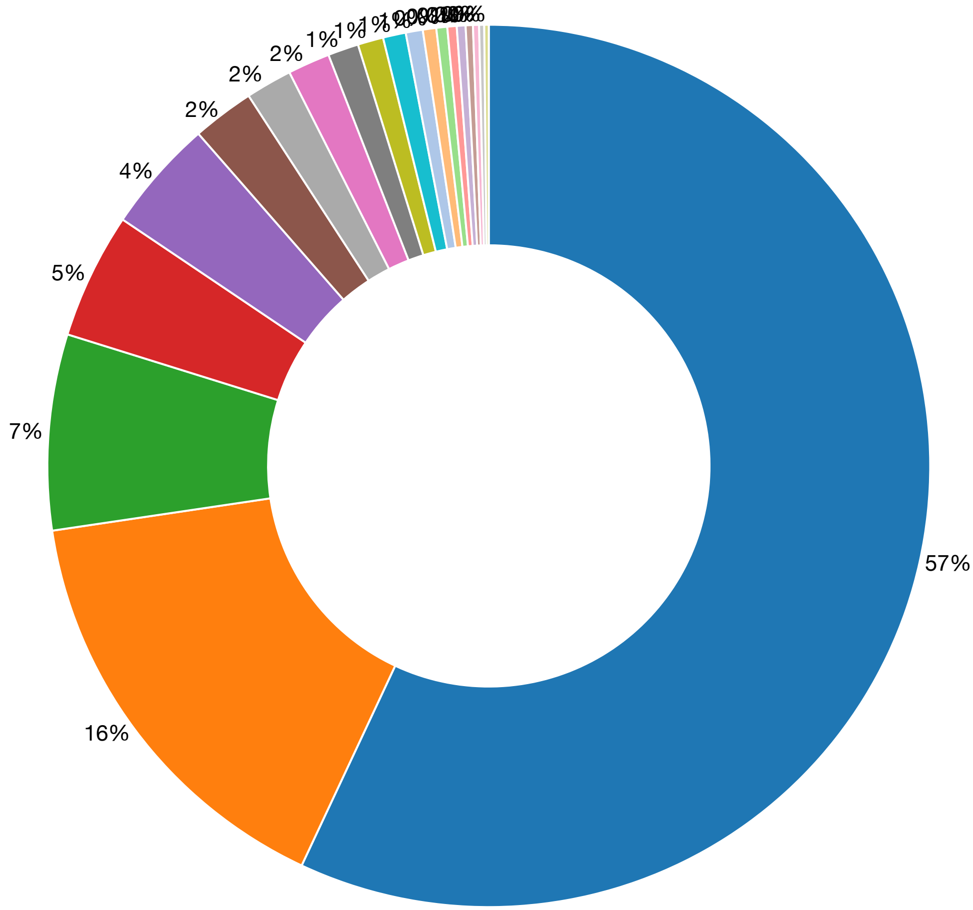
Top 10 listeners

```
%sql
select user_id, sum(play_count) as number_of_total_play from user_playlist
group by user_id order by sum(play_count) desc limit 10;
```

user_id
4be305e02f4e72dad1b8ac78e630403543bab994
6d625c6557df84b60d90426c0116138b617b9449
972cce803aa7beceaa7d0039e4c7c0ff097e4d55
0b19fe0fad7ca85693846f7dad047c449784647e
d13609d62db6df876d3cc388225478618bb7b912
283882c3d18ff2ad0e17124002ec02b847d06e9a
083a2a59603a605275107c00812a811526c2a0af
2231cb435771a1a621ec44e95cdd28b81fad3288
6c944bfe30ae8d6b873139e8305ae131f1607d5f

Distribution of Play count for all songs

```
%sql
select play_count, count(*) as count from user_playlist group by play_count
order by play_count
```



Top played songs based on Artists

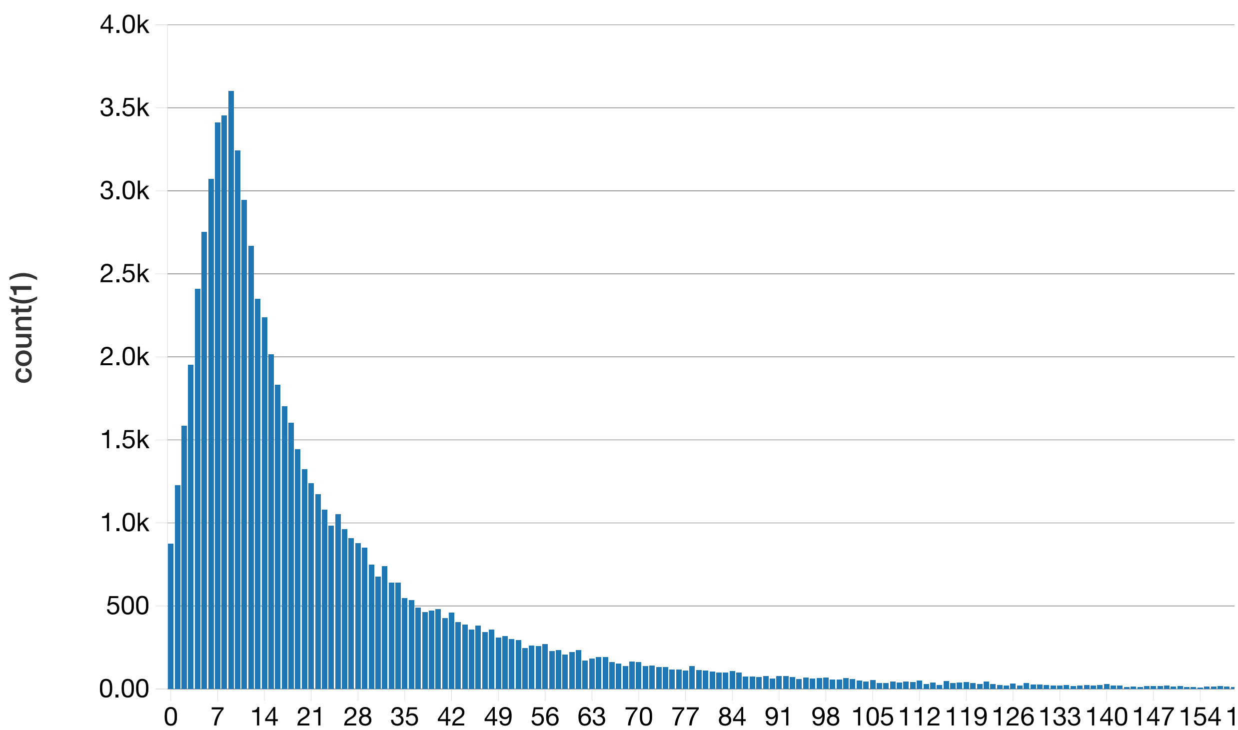
```
%sql
select artist_name, sum(play_count) as number_of_total_play from user_playlist
group by artist_name order by sum(play_count) desc limit 10;
```

artist_name

Kings Of Leon
Coldplay
Florence + The Machine
Dwight Yoakam
Björk
The Black Keys
Muse

Distribution of distinct songs listened by each users

```
%sql
select Number_of_songs,count(*) from (
select user_id,count(distinct song_id) as Number_of_songs from user_playlist
group by user_id order by Number_of_songs) group by Number_of_songs order by
Number_of_songs;
```



Taking 10% of the users from MSD data

```
# number of distinct user_Id
user=MSD.select("user_id").distinct()
user1,user2= user.randomSplit([0.05,0.95], seed=123)
usercount = user1.count()
print("Number of users: ", usercount)
```

```
Number of users:  3783
```

Dataframe of distinct list of songs

```
# number of distinct song_Id
songs= MSD.select("song_id").distinct()
songcount=songs.count()
print("Number of songs: ", songcount)
```

```
Number of songs:  10000
```

Giving incrementing Ids to UserId field to convert them into integer such that it can be accepted by our model

```
from pyspark.sql.functions import monotonically_increasing_id

# Creating new columns of unique integers for user_id and song_id
user_df = user1.withColumn("new_userid", monotonically_increasing_id())
user_df.show()
```

```
+--------------------+----------+
|             user_id|new_userid|
+--------------------+----------+
|126ef5859eb5e96f7...|         0|
|1d7b9780e492c062c...|         1|
|2c0815308dfd33b4b...|         2|
|3d24b9ffed6a82778...|         3|
|3dd54878cb47456b8...|         4|
|459ab8388e7806755...|         5|
|5e7105e485a04bf0b...|         6|
|6bb6fa9a23505dc55...|         7|
|739f32d4c2690554b...|         8|
|7a3943dfa7f83e321...|         9|
|7de4388c64742657d...|        10|
```

```
|9619f405e777e8331...|        11|
|a2c1d795852bd22c6...|        12|
|a8e8fd13a2909af99...|        13|
|aab99ec2a563f732e...|        14|
|c06d794619168b4bf...|        15|
|c2cfc654c54fddc9f...|        16|
|e43304402c7407bc4...|        17|
|e6bf98dccce485c26...|        18|
|e7fc73d0eb0d851bc...|        19|
+--------------------+----------+
only showing top 20 rows
```

Giving incrementing Ids to SongId field to convert them into integer such that it can be accepted by our model

```
songs_df = songs.select("song_id",
monotonically_increasing_id().alias('new_songId'))
songs_df.show()
```

```
+------------------+----------+
|           song_id|new_songId|
+------------------+----------+
|SOATHTW12A58A7EDB5|         0|
|SOAZMXH12AB0186DDE|         1|
|SOBAQTV12A8C142277|         2|
|SOCKUUJ12A6D4FA41C|         3|
|SOCUVKX12A6D4F8ED7|         4|
|SODABFB12A58A81788|         5|
|SODASIJ12A6D4F5D89|         6|
|SODYTRD12A81C2329F|         7|
|SOECOOL12AB0181A2F|         8|
|SOECTGX12A6310E233|         9|
|SOERLLT12AC468DAF3|        10|
|SOGKEGN12AB0185355|        11|
|SOGXQYC12AB0183AE5|        12|
|SOHXDTJ12A81C219C2|        13|
|SOICVFJ12A8AE47FF0|        14|
|SOJGIUN12A6BD55B8E|        15|
|SOKOVZK12A6D4F707F|        16|
|SOKQHXV12AB0185B3D|        17|
|SOKUAGP12A8C133B94|        18|
|SOLIVXX12A6D4F7950|        19|
+------------------+----------+
```

```
only showing top 20 rows
```

Cross joining to map all the users with all the songs such that we have entry for all the songs for each user

```
#Cross Join user and Songs
crossjoin = user_df.crossJoin(songs_df)
crossjoin.show(5)
```

```
+--------------------+----------+------------------+----------+
|             user_id|new_userid|           song_id|new_songId|
+--------------------+----------+------------------+----------+
|126ef5859eb5e96f7...|         0|SOATHTW12A58A7EDB5|         0|
|126ef5859eb5e96f7...|         0|SOAZMXH12AB0186DDE|         1|
|126ef5859eb5e96f7...|         0|SOBAQTV12A8C142277|         2|
|126ef5859eb5e96f7...|         0|SOCKUUJ12A6D4FA41C|         3|
|126ef5859eb5e96f7...|         0|SOCUVKX12A6D4F8ED7|         4|
+--------------------+----------+------------------+----------+
only showing top 5 rows
```

```
crossjoin.count()
```

```
Out[18]: 37830000
```

Joining the crossjoin dataframe with entire data such that we can have play_count populated for all the user and song combination, and replacing the NA's with 0 when there is no match i.e. when user did not listen to that song

```
df = crossjoin.join(MSD, ["user_id", "song_id"], "left").fillna(0)
```

Selecting only numeric columns that we want for Modeling

```
model_df=
df.select(df.new_userid.cast("int"),df.new_songId.cast("int"),df.play_count.cas
t("int"))
```

Alternating Least Squares (ALS)

ALS is an iterative optimization process in which for every run we try to arrive closer and closer to a factorized representation of our original data. Assume that our original matrix M of size U x I, where u are the number of users and i are the number of items. We want to find a way such that we can express our original matrix M into product of two matrix. One matrix of user and hidden features of dimension U * F and second matrix of items and hidden features of dimension F x I. These two matrix have weights for how each user/item relates to each hidden feature. Using gradient descent, we evalaute these two matrix such that their product approximates M as closely as possible.

Defining the Model

```
# Set the ALS hyperparameters
from pyspark.ml.recommendation import ALS

model = ALS(userCol= "new_userid", itemCol= "new_songId", ratingCol=
"play_count", rank = 10, maxIter = 10,alpha = 20, regParam = .05,
coldStartStrategy="drop", nonnegative = True, implicitPrefs = True)
```

Dividing the data into train and test dataset

```
# Split the dataframe into training and test data
(train_data, test_data) =
model_df.select('new_userid','new_songId','play_count').randomSplit([0.7, 0.3],
seed=12345)
```

Rank Ordering Error Metric (ROEM)

The ALS model from Spark ml library contains additional parameter for implicit rating called alpha which is an integer value that tells Spark how much additional song play should add to the confidence of model that the user

actually likes a particular song.

For explicit ratings, we can use RMSE to evaluate the model which makes sense we can match predictions back to a true measure of user ratings. However, in case of implicit rating we don’t have true measure of user ratings we only have the number of times user played a song and a measure of how confident the model is that they like that song and therefore we can’t use RMSE for evaluating our model. However, using test dataset, we can see if our model is giving high predictions to the songs that users have actually listened to.

The logic is if our model is returning a high prediction for a song that the respective user actually listened to, then the model prediction make sense, especially if they’ve listened to it more than once. We can measure this using Rank Ordering Error Metric (ROEM), which checks if songs have higher number of plays have higher predictions.

```
def ROEM(predictions, userCol = "new_userid", itemCol = "new_songId", ratingCol
= "play_count"):
  #Creates table that can be queried
  predictions.createOrReplaceTempView("predictions")

  #Sum of total number of plays of all songs
  denominator = predictions.groupBy().sum(ratingCol).collect()[0][0]

  #Calculating rankings of songs predictions by user
  spark.sql("SELECT " + userCol + " , " + ratingCol + " , PERCENT_RANK() OVER
(PARTITION BY " + userCol + " ORDER BY prediction DESC) AS rank FROM
predictions").createOrReplaceTempView("rankings")

   #Multiplies the rank of each song by the number of plays and adds the
products together
  numerator = spark.sql('SELECT SUM(' + ratingCol + ' * rank) FROM
rankings').collect()[0][0]

  performance = numerator/denominator

  return performance
```

```
train_data.cache()
```

```
Out[24]: DataFrame[new_userid: int, new_songId: int, play_count: int]
```

Fits model to fold within training data

```
fitted_model = model.fit(train_data)
```

Generates predictions using fitted_model on respective CV test data

```
predictions = fitted_model.transform(test_data)
```

Generates and prints a ROEM metric CV test data

```
# Generates and prints a ROEM metric CV test data
validation_performance = ROEM(predictions)
print(validation_performance)
```

```
0.5026878341522527
```

Conclusion

Our model achieved the accuracy of 50.26%. We trained the model on ~4000 users.
Out of 2 two songs recommended, 1 was relevant to the user.
We are able to process 37830000 records (3783 users and 10000 songs).
We tried multiple cloud platforms to increase accuracy and performance of the code.
(Databricks, Google Cloud Platform)
The model performance can be improved if we train it on more data points.

References

https://github.com/jamenlong/ALS_expected_percent_rank_cv/blob/master/ROE
(https://github.com/jamenlong/ALS_expected_percent_rank_cv/blob/master/ROI
https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

(https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe)

IEEE Paper : “Collaborative Filtering for Implicit Feedback Datasets”