Recommendation system

October 9, 2021

#

Final project

Lê Vũ Lợi - Trần Thị Uyên

1 Spark Movie Recommendation

In this notebook, we will use an Alternating Least Squares (ALS) algorithm with Spark APIs to predict the ratings for the movies in MovieLens-25M dataset

The dataset is MovieLens-25M dataset, which includes 20000264 user ratings on movies from idbm website. There are four csv files in the dataset, including movies.csv, ratings.csv, tags.csv, links.csv

In this notebook, we only use movies.csv and ratings.csv as they provides the information needed in constructing recommendation system, such as, movie id, user id, ratings

Outline of this notebook: - Step 1: Read data from HDFS - Step 2: Data statistics - Step 3: Build recommendation model - Step 4: Evaluate and test the model

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
%matplotlib inline
```

[2]:
```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
# Test the spark
df = spark.createDataFrame([{"hello": "world"} for x in range(1000)])
df.show(3, False)
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform
(file:/home/levuloi/spark-3.1.2-bin-hadoop3.2/jars/spark-unsafe_2.12-3.1.2.jar)
to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of
org.apache.spark.unsafe.Platform
```

```
WARNING: Use --illegal-access=warn to enable warnings of further illegal
reflective access operations
WARNING: All illegal access operations will be denied in a future release
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
[Stage 0:>                                                          (0 + 1) / 1]

+-----+
|hello|
+-----+
|world|
|world|
|world|
+-----+
only showing top 3 rows
```

[3]:
```python
import os
os.environ["PYSPARK_PYTHON"] = "python3"
```

1.1 Part 1: Read .csv files using SparkSQL

[3]:
```python
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Movie analysis") \
    .getOrCreate()
```

[4]:
```python
# if using google colab, using this
root = "movielens/"
# if using databrick
# databrick_root = "/FileStore/tables/movielen_small/"


movies_df = spark.read.load(root+"movies.csv", format='csv', header = True)
ratings_df = spark.read.load(root+ "ratings.csv", format='csv', header = True)
links_df = spark.read.load(root+"links.csv", format='csv', header = True)
tags_df = spark.read.load(root+"tags.csv", format='csv', header = True)
```

[5]:
```python
movies_df.show(5)
```

```
+-------+--------------------+--------------------+
|movieId|               title|              genres|
+-------+--------------------+--------------------+
|      1|    Toy Story (1995)|Adventure|Animati…|
```

```
|      2|      Jumanji (1995)|Adventure|Childre…|
|      3|Grumpier Old Men …|      Comedy|Romance|
|      4|Waiting to Exhale…|Comedy|Drama|Romance|
|      5|Father of the Bri…|              Comedy|
+-------+--------------------+--------------------+
only showing top 5 rows
```

[6]:
```
ratings_df.show(5)
```

```
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
|     1|      2|   3.5|1112486027|
|     1|     29|   3.5|1112484676|
|     1|     32|   3.5|1112484819|
|     1|     47|   3.5|1112484727|
|     1|     50|   3.5|1112484580|
+------+-------+------+----------+
only showing top 5 rows
```

[7]:
```
links_df.show(5)
```

```
+-------+-------+------+
|movieId| imdbId|tmdbId|
+-------+-------+------+
|      1|0114709|   862|
|      2|0113497|  8844|
|      3|0113228| 15602|
|      4|0114885| 31357|
|      5|0113041| 11862|
+-------+-------+------+
only showing top 5 rows
```

[8]:
```
tags_df.show(5)
```

```
+------+-------+-------------+----------+
|userId|movieId|          tag| timestamp|
+------+-------+-------------+----------+
|    18|   4141|  Mark Waters|1240597180|
|    65|    208|    dark hero|1368150078|
|    65|    353|    dark hero|1368150079|
|    65|    521|noir thriller|1368149983|
|    65|    592|    dark hero|1368150078|
+------+-------+-------------+----------+
only showing top 5 rows
```

[9]:
```python
tmp1 = ratings_df.groupBy("userID").count().toPandas()['count'].min()
tmp2 = ratings_df.groupBy("movieId").count().toPandas()['count'].min()
print('For the users that rated movies and the movies that were rated:')
print('Minimum number of ratings per user is {}'.format(tmp1))
print('Minimum number of ratings per movie is {}'.format(tmp2))
```

```
For the users that rated movies and the movies that were rated:
Minimum number of ratings per user is 20
Minimum number of ratings per movie is 1
```

[10]:
```python
tmp1 = sum(ratings_df.groupBy("movieId").count().toPandas()['count'] == 1)
tmp2 = ratings_df.select('movieId').distinct().count()
print('{} out of {} movies are rated by only one user'.format(tmp1, tmp2))
```

```
[Stage 16:====================================================>  (191 + 2) / 200]
```

```
3972 out of 26744 movies are rated by only one user
```

1.2 Part 2: Data statistics

---

[12]:
```python
movies_df.registerTempTable("movies")
ratings_df.registerTempTable("ratings")
links_df.registerTempTable("links")
tags_df.registerTempTable("tags")
```

1.2.1 Q1: The number of Users

[13]:
```python
# Using SQL + Spark method
user_count = spark.sql("select count(distinct userId) as user_count from␣
↪ratings")
# Using pyspark
# user_count = ratings_df.select(["userId"]).distinct().count()
user_count.show()
```

```
[Stage 19:================================================>      (177 + 2) / 200]
```

```
+----------+
|user_count|
+----------+
|    138493|
+----------+
```

1.2.2 Q2: The number of Movies

[14]:
```
# %sql
# Using SQL
movie_count = spark.sql("select count(distinct movieId) as movie_count from␣
↪movies")
movie_count.show()
```

```
[Stage 22:=========================================>             (154 + 2) / 200]

+-----------+
|movie_count|
+-----------+
|      27278|
+-----------+
```

1.2.3 Q3: How many movies are rated by users? List movies not rated before

3.1 show the number of movie rated

[15]:
```
from pyspark.sql.functions import col
# %sql
# Show number of movies rated
movie_rated = spark.sql("select count(distinct movieId) from movies where␣
↪movieId in (select distinct movieId from ratings)")
movie_rated.show()
```

```
[Stage 26:================================================>    (185 + 2) / 200]

+----------------------+
|count(DISTINCT movieId)|
+----------------------+
|                 26744|
+----------------------+
```

3.2 list movies that are not rated before

[16]:
```
# %sql
# Show movies that are not rated
movie_not_rated = spark.sql("select distinct movieId, title from movies where␣
↪movieId not in (select distinct movieId from ratings)")
movie_not_rated.show()
```

```
+-------+--------------------+
|movieId|               title|
+-------+--------------------+
| 115907|    Crew, The (2008)|
| 120380| The Fountain (1989)|
| 121703|Take a Giant Step…|
| 127208|      Results (2015)|
| 115545|Star for Two, A (…|
| 122371|Let the Good Time…|
| 122373|The Vanishing Ame…|
| 128870|    Yesterday (1988)|
| 118025|Upstairs and Down…|
| 122015|       Sergio (2009)|
|  65078|Jane Austen in Ma…|
| 120751|Home of the Brave…|
| 121741|    The Cheat (1950)|
|  31797|White Banners (1938)|
|  92845|Untamed Youth (1957)|
| 116590|Death In Love (2008)|
| 121632|     Merlusse (1938)|
| 128866|Queen of the Moun…|
| 111667|Toward the Unknow…|
| 115376|Black Sleep, The …|
+-------+--------------------+
only showing top 20 rows
```

The amount of movies that are not counted before

[17]:
```
count_not_rated = movie_not_rated.count()
count_not_rated
```

[17]: 534

Missing values in rating

[18]:
```
rating_missing_value = spark.sql("select * from ratings where rating = NULL or␣
↪timestamp = NULL or userId=NULL or movieId=NULL")
rating_missing_value.show()
```

```
+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
+------+-------+------+---------+
```

Check null values in the table

```
[19]: # movie_missing_value = spark.sql("select movieId from movies where title =␣
      ↪NULL or genres = NULL")
      movie_missing_value= movies_df.where(col('title').isNull() | col('genres').
      ↪isNull())
      movie_missing_value.show()
```

```
+-------+-----+------+
|movieId|title|genres|
+-------+-----+------+
+-------+-----+------+
```

1.2.4 Q4: List Movie Genres

Directly list movie genres

```
[20]: # %sql
      #Using SQL + PySpark method
      # directly list movie genres
      movie_genres_df = spark.sql("select distinct genres as genres_count from␣
      ↪movies")
      movie_genres_df.show()
```

```
+--------------------+
|        genres_count|
+--------------------+
|Comedy|Horror|Thr...|
|Adventure|Sci-Fi|...|
|Action|Adventure|...|
| Action|Drama|Horror|
|Comedy|Drama|Horr...|
|Action|Animation|...|
|Fantasy|Musical|M...|
|Adventure|Mystery...|
|Animation|Childre...|
|Action|Adventure|...|
| Adventure|Animation|
|    Adventure|Sci-Fi|
|Documentary|Music...|
|  Documentary|Sci-Fi|
|Adventure|Childre...|
| Musical|Romance|War|
|Action|Adventure|...|
|Adventure|Childre...|
|Comedy|Crime|Horr...|
|Crime|Drama|Fanta...|
+--------------------+
only showing top 20 rows
```

Split genres and then list movie genres

[25]:
```
# splite movie genres and then count
from pyspark.sql.functions import col, explode, split
movie_genres = movies_df.withColumn("splited_genres",␣
↪explode(split(col("genres"),"[|]")))
splited_genres_df = movie_genres.select(["splited_genres"]).distinct().
↪orderBy("splited_genres",ascending=True)
splited_genres_df.show()
```

```
[Stage 62:========================================>              (147 + 3) / 200]

+------------------+
|    splited_genres|
+------------------+
|(no genres listed)|
|            Action|
|         Adventure|
|         Animation|
|          Children|
|            Comedy|
|             Crime|
|       Documentary|
|             Drama|
|           Fantasy|
|         Film-Noir|
|            Horror|
|              IMAX|
|           Musical|
|           Mystery|
|           Romance|
|            Sci-Fi|
|          Thriller|
|               War|
|           Western|
+------------------+
```

1.2.5 Q5: Movie for Each Category

[21]:
```
# %sql
# Using SQL + Pyspark method
category_movie_count = spark.sql(" with movie_category as (select distinct␣
↪(explode(split(genres, '[|]'))) as category, * from movies) \
```

```
                  select category, count(category) as category_count from␣
↪movie_category group by category order by category_count desc")
category_movie_count.show()
print("Number of categories: ",len(category_movie_count.collect()))
```

```
+------------------+--------------+
|          category|category_count|
+------------------+--------------+
|             Drama|         13344|
|            Comedy|          8374|
|          Thriller|          4178|
|           Romance|          4127|
|            Action|          3520|
|             Crime|          2939|
|            Horror|          2611|
|       Documentary|          2471|
|         Adventure|          2329|
|            Sci-Fi|          1743|
|           Mystery|          1514|
|           Fantasy|          1412|
|               War|          1194|
|          Children|          1139|
|           Musical|          1036|
|         Animation|          1027|
|           Western|           676|
|         Film-Noir|           330|
|(no genres listed)|           246|
|              IMAX|           196|
+------------------+--------------+

[Stage 54:====================================================>  (189 + 2) / 200]

Number of categories:  20
```

1.3 Analysis:

It seems like the dataset is actually incomplete, despite there is no null values, since there are 34 items labeled as **(no genres listed)**.

However, Since we will use Non-negative Matrix Factorization method for movie recommendation system, which only cares the userId and movieId, without considering categories, **I don't drop the movies that are labeled as (no genres listed)**

[22]:
```
#Using PySpark method
from pyspark.sql.functions import col, explode, split
```

```
movie_genres = movies_df.withColumn("splited_genres",␣
 ↪explode(split(col("genres"),"[|]")))
splited_genres_df = movie_genres.groupBy(["splited_genres"]).count().
 ↪orderBy("count",ascending=False)
splited_genres_df = splited_genres_df.
 ↪withColumnRenamed('count','category_count')
splited_genres_df.show()
```

```
[Stage 57:===================================================>     (182 + 2) / 200]

+------------------+--------------+
|    splited_genres|category_count|
+------------------+--------------+
|             Drama|         13344|
|            Comedy|          8374|
|          Thriller|          4178|
|           Romance|          4127|
|            Action|          3520|
|             Crime|          2939|
|            Horror|          2611|
|       Documentary|          2471|
|         Adventure|          2329|
|            Sci-Fi|          1743|
|           Mystery|          1514|
|           Fantasy|          1412|
|               War|          1194|
|          Children|          1139|
|           Musical|          1036|
|         Animation|          1027|
|           Western|           676|
|         Film-Noir|           330|
|(no genres listed)|           246|
|              IMAX|           196|
+------------------+--------------+
```

show the movie with genres labeled as "(no genres listed)"

[23]:
```
movie_genres.where(col("splited_genres").isin(["(no genres listed)"])).
 ↪join(movies_df,"movieId","left").show()
```

```
[Stage 59:>                                                         (0 + 1) / 1]

+-------+--------------------+------------------+------------------+------------
--------+------------------+
|movieId|               title|            genres|    splited_genres|
title|            genres|
```

```
+-------+--------------------+------------------+------------------+------------
--------+------------------+
|  83773|Away with Words (…|(no genres listed)|(no genres listed)|Away with
Words (…|(no genres listed)|
|  83829|Scorpio Rising (1…|(no genres listed)|(no genres listed)|Scorpio
Rising (1…|(no genres listed)|
|  84768|   Glitterbug (1994)|(no genres listed)|(no genres listed)|
Glitterbug (1994)|(no genres listed)|
|  86493|Age of the Earth,…|(no genres listed)|(no genres listed)|Age of the
Earth,…|(no genres listed)|
|  87061|Trails (Veredas) …|(no genres listed)|(no genres listed)|Trails
(Veredas) …|(no genres listed)|
|  91246|Milky Way (Tejút)…|(no genres listed)|(no genres listed)|Milky Way
(Tejút)…|(no genres listed)|
|  92435|Dancing Hawk, The…|(no genres listed)|(no genres listed)|Dancing
Hawk, The…|(no genres listed)|
|  92641|Warsaw Bridge (Po…|(no genres listed)|(no genres listed)|Warsaw
Bridge (Po…|(no genres listed)|
|  94431|Ella Lola, a la T…|(no genres listed)|(no genres listed)|Ella Lola, a
la T…|(no genres listed)|
|  94657|Turkish Dance, El…|(no genres listed)|(no genres listed)|Turkish
Dance, El…|(no genres listed)|
|  95541|Blacksmith Scene …|(no genres listed)|(no genres listed)|Blacksmith
Scene …|(no genres listed)|
|  95750|Promise of the Fl…|(no genres listed)|(no genres listed)|Promise of
the Fl…|(no genres listed)|
|  96479|  Nocturno 29 (1968)|(no genres listed)|(no genres listed)|  Nocturno
29 (1968)|(no genres listed)|
|  96651|Les hautes solitu…|(no genres listed)|(no genres listed)|Les hautes
solitu…|(no genres listed)|
| 100294|       Désiré (1992)|(no genres listed)|(no genres listed)|
Désiré (1992)|(no genres listed)|
| 113472|Direct from Brook…|(no genres listed)|(no genres listed)|Direct from
Brook…|(no genres listed)|
| 113545|Primus Hallucino-…|(no genres listed)|(no genres listed)|Primus
Hallucino-…|(no genres listed)|
| 114335|   La cravate (1957)|(no genres listed)|(no genres listed)|   La
cravate (1957)|(no genres listed)|
| 114587|Glumov's Diary (D…|(no genres listed)|(no genres listed)|Glumov's
Diary (D…|(no genres listed)|
| 114723|      At Land (1944)|(no genres listed)|(no genres listed)|      At
Land (1944)|(no genres listed)|
+-------+--------------------+------------------+------------------+------------
--------+------------------+
only showing top 20 rows
```

1.4 Part 3: Spark ALS based approach for training model

We will use an Spark ML to predict the ratings `sc.textFile` and then convert it to the form of (user, item, rating) tuples.

[24]:
```
ratings_df.show()
```

```
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
|     1|      2|   3.5|1112486027|
|     1|     29|   3.5|1112484676|
|     1|     32|   3.5|1112484819|
|     1|     47|   3.5|1112484727|
|     1|     50|   3.5|1112484580|
|     1|    112|   3.5|1094785740|
|     1|    151|   4.0|1094785734|
|     1|    223|   4.0|1112485573|
|     1|    253|   4.0|1112484940|
|     1|    260|   4.0|1112484826|
|     1|    293|   4.0|1112484703|
|     1|    296|   4.0|1112484767|
|     1|    318|   4.0|1112484798|
|     1|    337|   3.5|1094785709|
|     1|    367|   3.5|1112485980|
|     1|    541|   4.0|1112484603|
|     1|    589|   3.5|1112485557|
|     1|    593|   3.5|1112484661|
|     1|    653|   3.0|1094785691|
|     1|    919|   3.5|1094785621|
+------+-------+------+----------+
only showing top 20 rows
```

[26]:
```
movie_ratings=ratings_df.drop('timestamp')
```

[27]:
```
# Data type convert
from pyspark.sql.types import IntegerType, FloatType
movie_ratings = movie_ratings.withColumn("userId", movie_ratings["userId"].
 ↪cast(IntegerType()))
movie_ratings = movie_ratings.withColumn("movieId", movie_ratings["movieId"].
 ↪cast(IntegerType()))
movie_ratings = movie_ratings.withColumn("rating", movie_ratings["rating"].
 ↪cast(FloatType()))
```

[28]:
```
movie_ratings.show()
```

```
+------+-------+------+
```

```
|userId|movieId|rating|
+------+-------+------+
|     1|      2|   3.5|
|     1|     29|   3.5|
|     1|     32|   3.5|
|     1|     47|   3.5|
|     1|     50|   3.5|
|     1|    112|   3.5|
|     1|    151|   4.0|
|     1|    223|   4.0|
|     1|    253|   4.0|
|     1|    260|   4.0|
|     1|    293|   4.0|
|     1|    296|   4.0|
|     1|    318|   4.0|
|     1|    337|   3.5|
|     1|    367|   3.5|
|     1|    541|   4.0|
|     1|    589|   3.5|
|     1|    593|   3.5|
|     1|    653|   3.0|
|     1|    919|   3.5|
+------+-------+------+
only showing top 20 rows
```

1.5 ALS Model Selection and Evaluation

With the ALS model, we can use a grid search to find the optimal hyperparameters.

[30]:
```python
# import package
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator,ParamGridBuilder
```

[36]:
```python
#Create test and train set
(training,test)=movie_ratings.randomSplit([0.8,0.2])
```

[37]:
```python
#Create ALS model
len(movie_ratings.select(["userId"]).distinct().collect()),len(movie_ratings.
 ↪select(["movieId"]).distinct().collect())
```

[37]:
```
(610, 9724)
```

1.5.1 Apply ALS (alternative least square) algorithm for matrix factorization

[38]:
```
als_model = ALS(rank=10, maxIter=10,coldStartStrategy='drop' )
als_model.setUserCol("userId")
als_model.setItemCol("movieId")
als_model.setRatingCol("rating")
als_model.setPredictionCol("Prediction")
als_model.getUserCol(),als_model.getItemCol(),als_model.
 ↪getRatingCol(),als_model.getPredictionCol(),
```

[38]:
```
('userId', 'movieId', 'rating', 'Prediction')
```

1.5.2 Model selection

We use grid search to select the best hyperparameters for the model

[39]:
```
#Tune model using ParamGridBuilder
grid = ParamGridBuilder()\
   .baseOn({als_model.predictionCol:"Prediction"})\
   .addGrid(als_model.regParam,[0.1,0.5,0.8])\
   .addGrid(als_model.rank,[5,10,15])\
   .build()
```

1.5.3 Cross-validation

We use Root Mean Square Error (RMSE) to compute the error rate of ALS model,between rating scores and prediction score

[40]:
```
# Define evaluator as RMSE
evaluator=␣
 ↪RegressionEvaluator(predictionCol="Prediction",labelCol="rating",metricName="rmse")
```

[41]:
```
# Build Cross validation
cv = CrossValidator(estimator=als_model,
            estimatorParamMaps=grid,
            evaluator=evaluator,
            numFolds=5,seed=2020,parallelism=2)
```

[42]:
```
#Fit ALS model to training data
cvModel = cv.fit(training)
```

```
2021-10-09 00:45:24,932 WARN storage.BlockManager: Block rdd_289_0 already
exists on this machine; not re-adding it
2021-10-09 00:45:29,196 WARN netlib.BLAS: Failed to load implementation from:
com.github.fommil.netlib.NativeSystemBLAS
2021-10-09 00:45:29,198 WARN netlib.BLAS: Failed to load implementation from:
com.github.fommil.netlib.NativeRefBLAS
2021-10-09 00:45:29,429 WARN netlib.LAPACK: Failed to load implementation from:
```

```
com.github.fommil.netlib.NativeSystemLAPACK
2021-10-09 00:45:29,429 WARN netlib.LAPACK: Failed to load implementation from:
com.github.fommil.netlib.NativeRefLAPACK
```

[43]:
```python
#Extract best model from the tuning exercise using ParamGridBuilder
best_model = cvModel.bestModel
best_model
```

[43]:
```
ALSModel: uid=ALS_b2d9b8924e83, rank=10
```

[33]:
```python
# cvModel = ALS.load('/user/levuloi/model/recsys')
```

1.5.4 Model testing

And finally, make a prediction and check the testing error.

[44]:
```python
#Generate predictions and evaluate using RMSE
predictions=best_model.transform(test)
rmse = evaluator.evaluate(predictions)
```

[45]:
```python
#Print evaluation metrics and model parameters
print ("RMSE = "+str(rmse))
print ("**Best Model**")
print (" Rank:",best_model.rank),
print (" MaxIter:", best_model._java_obj.parent().getMaxIter()),
print (" RegParam:",best_model._java_obj.parent().getRegParam()),
```

```
RMSE = 0.8788339548183068
**Best Model**
 Rank: 10
 MaxIter: 10
 RegParam: 0.1
```

[45]:
```
(None,)
```

[46]:
```python
predictions.show()
```

```
[Stage 3926:===========>(198 + 2) / 200][Stage 3927:=============> (9 + 0) / 10]

+------+-------+------+----------+
|userId|movieId|rating|Prediction|
+------+-------+------+----------+
|   597|    471|   2.0| 4.3538284|
|   182|    471|   4.5| 3.7673397|
|   500|    471|   1.0| 3.5942218|
|   387|    471|   3.0|  3.406635|
|   520|    471|   5.0| 3.1891556|
```

```
|   171|    471|   3.0|   4.11799|
|   104|    471|   4.5| 3.2904468|
|   463|   1088|   3.5| 3.1465316|
|   177|   1088|   3.5|  3.918715|
|   169|   1088|   4.5|  4.339727|
|   286|   1088|   3.5| 3.3241692|
|   594|   1088|   4.5| 4.7292356|
|   307|   1088|   3.0| 2.5752902|
|    84|   1088|   3.0| 3.6741903|
|   391|   1088|   1.0| 3.3877826|
|    10|   1088|   3.0| 3.4025044|
|   226|   1088|   1.0| 3.2794697|
|    68|   1088|   3.5| 3.4670625|
|   517|   1088|   1.0| 3.1021283|
|   587|   1238|   4.0| 2.8353624|
+------+-------+------+----------+
only showing top 20 rows
```

1.5.5 Model apply and see the performance

[47]:
```python
alldata=best_model.transform(movie_ratings)
rmse = evaluator.evaluate(alldata)
print ("RMSE = "+str(rmse))
```

```
[Stage 3956:====================================================>(199 + 1) / 200]

RMSE = 0.636452150262987
```

[48]:
```python
alldata.registerTempTable("alldata")
```

[49]:
```python
result = spark.sql("select * from alldata")
result.show()
```

```
[Stage 3983:===============================================>    (184 + 2) / 200]

+------+-------+------+----------+
|userId|movieId|rating|Prediction|
+------+-------+------+----------+
|   133|    471|   4.0| 3.3777323|
|   597|    471|   2.0| 4.3538284|
|   385|    471|   4.0| 3.7240336|
|   436|    471|   3.0| 3.4426775|
|   602|    471|   4.0|  3.338972|
|    91|    471|   1.0|  2.388136|
|   409|    471|   3.0| 3.5924802|
```

```
|   372|    471|   3.0| 3.0086079|
|   599|    471|   2.5|  2.781713|
|   603|    471|   4.0| 2.9310842|
|   182|    471|   4.5| 3.7673397|
|   218|    471|   4.0|  3.646721|
|   474|    471|   3.0| 3.3155231|
|   500|    471|   1.0| 3.5942218|
|    57|    471|   3.0| 3.5005147|
|   462|    471|   2.5| 2.2502546|
|   387|    471|   3.0|  3.406635|
|   610|    471|   4.0|  3.526957|
|   217|    471|   2.0| 2.4717174|
|   555|    471|   3.0| 3.7069519|
+------+-------+------+----------+
only showing top 20 rows
```

[50]:
```
result = spark.sql("select * from movies join alldata on movies.movieId=alldata.
↪movieId")
result.show()
```

```
[Stage 4012:=================================================>   (185 + 2) / 200]

+-------+--------------------+------+------+-------+------+----------+
|movieId|               title|genres|userId|movieId|rating|Prediction|
+-------+--------------------+------+------+-------+------+----------+
|    471|Hudsucker Proxy, …|Comedy|   133|    471|   4.0| 3.3777323|
|    471|Hudsucker Proxy, …|Comedy|   597|    471|   2.0| 4.3538284|
|    471|Hudsucker Proxy, …|Comedy|   385|    471|   4.0| 3.7240336|
|    471|Hudsucker Proxy, …|Comedy|   436|    471|   3.0| 3.4426775|
|    471|Hudsucker Proxy, …|Comedy|   602|    471|   4.0|  3.338972|
|    471|Hudsucker Proxy, …|Comedy|    91|    471|   1.0|  2.388136|
|    471|Hudsucker Proxy, …|Comedy|   409|    471|   3.0| 3.5924802|
|    471|Hudsucker Proxy, …|Comedy|   372|    471|   3.0| 3.0086079|
|    471|Hudsucker Proxy, …|Comedy|   599|    471|   2.5|  2.781713|
|    471|Hudsucker Proxy, …|Comedy|   603|    471|   4.0| 2.9310842|
|    471|Hudsucker Proxy, …|Comedy|   182|    471|   4.5| 3.7673397|
|    471|Hudsucker Proxy, …|Comedy|   218|    471|   4.0|  3.646721|
|    471|Hudsucker Proxy, …|Comedy|   474|    471|   3.0| 3.3155231|
|    471|Hudsucker Proxy, …|Comedy|   500|    471|   1.0| 3.5942218|
|    471|Hudsucker Proxy, …|Comedy|    57|    471|   3.0| 3.5005147|
|    471|Hudsucker Proxy, …|Comedy|   462|    471|   2.5| 2.2502546|
|    471|Hudsucker Proxy, …|Comedy|   387|    471|   3.0|  3.406635|
|    471|Hudsucker Proxy, …|Comedy|   610|    471|   4.0|  3.526957|
|    471|Hudsucker Proxy, …|Comedy|   217|    471|   2.0| 2.4717174|
|    471|Hudsucker Proxy, …|Comedy|   555|    471|   3.0| 3.7069519|
+-------+--------------------+------+------+-------+------+----------+
```

```
only showing top 20 rows
```

[35]:
```
# best_model.save("/user/levuloi/model/recsys")
```

1.6 Recommend moive to users with id: 575, 232.

you can choose some users to recommend the moives

[54]:
```
from pyspark.sql.types import *
def get_recommendation(id_type="userId",id =None, numItems = 5):
  """
  id_type: type of id to input:  userId, movieId
  id: movie/ user id to query, either string or integer type
  numItems: number of Items to recommend in each query
  """
  if id_type =="userId" :
    # User-Id based
    recommendation = best_model.recommendForAllUsers(numItems)
    recommended_movies_df = recommendation.where(col("userId")==int(id)).
↪toPandas()
  elif id_type =="movieId" :
    # Movie-Id based
    recommendation = best_model.recommendForAllItems(numItems)
    recommended_movies_df = recommendation.where(col("movieId")==int(id)).
↪toPandas()
  else:
    print("id_type should be either 'userId' or 'movieId'")
    print("id should integer")
    return None

  #make sure there are movies recommended
  if len(recommended_movies_df) >0:
    movie_recommended = recommended_movies_df.iloc[0].loc["recommendations"]

    schema = StructType([
          StructField('movieId', IntegerType(), False),
          StructField('Prediction', FloatType(), False)
      ])
    movies = spark.createDataFrame(movie_recommended,schema)
    print("Movies recommended to User:%d"%int(id))
    movies = movies.join(movies_df,'movieId','left').toPandas()
  else:
    print("No movies for "+id_type+ ": %d"%int(id))
    movies = None
  return movies
```

[55]:
```
#Query User Id: 575
get_recommendation(id_type="userId", id =575)
```

```
Movies recommended to User:575
```

[55]:
```
   movieId  Prediction                                              title  \
0    26810    5.112532                                Bad Boy Bubby (1993)
1     6666    5.087752  Discreet Charm of the Bourgeoisie, The (Charme…
2     3153    5.014952                    7th Voyage of Sinbad, The (1958)
3    59814    4.990246                                  Ex Drummer (2007)
4   104879    4.979079                                   Prisoners (2013)

                     genres
0                     Drama
1       Comedy|Drama|Fantasy
2    Action|Adventure|Fantasy
3  Comedy|Crime|Drama|Horror
4     Drama|Mystery|Thriller
```

[56]:
```
#Query User Id: 232
get_recommendation(id_type="userId", id =232)
```

```
Movies recommended to User:232
```

[56]:
```
   movieId  Prediction                                              title  \
0    26073    4.646534  Human Condition III, The (Ningen no joken III)…
1   179135    4.646534                              Blue Planet II (2017)
2    84273    4.646534                   Zeitgeist: Moving Forward (2011)
3     7071    4.646534                 Woman Under the Influence, A (1974)
4   184245    4.646534                             De platte jungle (1978)

        genres
0    Drama|War
1  Documentary
2  Documentary
3        Drama
4  Documentary
```

1.7 Find the similar moives for moive with id: 463, 471

You can find the similar moives based on the ALS results

[59]:
```
get_recommendation(id_type="movieId", id =463)
```

```
[Stage 4189:=================================================>   (185 + 2) / 200]
```

```
No movies for movieId: 463
```

We can see that there is a movie with title and genres equal to "None". The reason is that this movie id from rating dataframe doesn't exist in movie dataframe from movie.csv.

[ ]:

```

```