Project_BigData

December 11, 2019

[89]:

```
from pyspark.sql import functions as F
from pyspark.sql import DataFrameNaFunctions as DFna
from pyspark.sql.functions import udf, col, when
import matplotlib.pyplot as plt
import pyspark as ps
import os, sys, requests, json
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import Row
import numpy as np
import math
import pandas as pd
from pandas import Series, DataFrame
```

[90]:

```
spark = ps.sql.SparkSession.builder \
            .master("local[4]") \
            .appName("building recommender") \
            .getOrCreate() # create a spark session

sc = spark.sparkContext
```

[36]:

```
# read movies CSV
movies = spark.read.option("header", "true").csv("/Users/vkoushikmuthyapu/
 ↪desktop/ml-latest-small/movies.csv",inferSchema=True)
movies.printSchema()
movies.show()
```

```
root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)

+-------+--------------------+--------------------+
|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|
|      6|         Heat (1995)|Action|Crime|Thri…|
|      7|      Sabrina (1995)|      Comedy|Romance|
|      8| Tom and Huck (1995)|  Adventure|Children|
|      9| Sudden Death (1995)|              Action|
|     10|    GoldenEye (1995)|Action|Adventure|…|
|     11|American Presiden…|Comedy|Drama|Romance|
|     12|Dracula: Dead and…|       Comedy|Horror|
|     13|        Balto (1995)|Adventure|Animati…|
|     14|        Nixon (1995)|               Drama|
|     15|Cutthroat Island …|Action|Adventure|…|
|     16|       Casino (1995)|         Crime|Drama|
|     17|Sense and Sensibi…|       Drama|Romance|
|     18|   Four Rooms (1995)|              Comedy|
|     19|Ace Ventura: When…|              Comedy|
|     20|  Money Train (1995)|Action|Comedy|Cri…|
+-------+--------------------+--------------------+
only showing top 20 rows
```

```
[37]: ratings = spark.read.option("header", "true").csv("/Users/vkoushikmuthyapu/
       ↪desktop/ml-latest-small/ratings.csv",inferSchema=True)
      ratings.printSchema()
      ratings.show()
```

```
root
 |-- userId: integer (nullable = true)
 |-- movieId: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- timestamp: integer (nullable = true)

+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
|     1|      1|   4.0|964982703|
|     1|      3|   4.0|964981247|
|     1|      6|   4.0|964982224|
|     1|     47|   5.0|964983815|
|     1|     50|   5.0|964982931|
|     1|     70|   3.0|964982400|
|     1|    101|   5.0|964980868|
|     1|    110|   4.0|964982176|
|     1|    151|   5.0|964984041|
|     1|    157|   5.0|964984100|
|     1|    163|   5.0|964983650|
|     1|    216|   5.0|964981208|
|     1|    223|   3.0|964980985|
|     1|    231|   5.0|964981179|
|     1|    235|   4.0|964980908|
|     1|    260|   5.0|964981680|
|     1|    296|   3.0|964982967|
|     1|    316|   3.0|964982310|
|     1|    333|   5.0|964981179|
|     1|    349|   4.0|964982563|
+------+-------+------+---------+
only showing top 20 rows
```

[38]:

```
newrating = ratings.select(['userId', 'movieId', 'rating'])
newrating.show()
```

```
+------+-------+------+
|userId|movieId|rating|
+------+-------+------+
|     1|      1|   4.0|
|     1|      3|   4.0|
|     1|      6|   4.0|
|     1|     47|   5.0|
|     1|     50|   5.0|
|     1|     70|   3.0|
|     1|    101|   5.0|
|     1|    110|   4.0|
|     1|    151|   5.0|
|     1|    157|   5.0|
|     1|    163|   5.0|
|     1|    216|   5.0|
|     1|    223|   3.0|
|     1|    231|   5.0|
|     1|    235|   4.0|
|     1|    260|   5.0|
|     1|    296|   3.0|
|     1|    316|   3.0|
|     1|    333|   5.0|
|     1|    349|   4.0|
+------+-------+------+
only showing top 20 rows
```

[39]:

```
newerratings = newrating.rdd
newerratings
```

[39]:

```
MapPartitionsRDD[706] at javaToPython at NativeMethodAccessorImpl.java:0
```

[40]:
```
training_df, validation_df, test_df = newrating.randomSplit([.6, .2, .2],␣
 ↪seed=0)
#training_RDD = training_df.rdd.map(lambda x: (x[0], x[1])).cache()
#validation_for_predict_RDD = validation_df.rdd
#test_for_predict_RDD = test_df.rdd.map(lambda x: (x[0], x[1])).cache()
training_df
```

[40]:
```
DataFrame[userId: int, movieId: int, rating: double]
```

[132]:
```
als = ALS(maxIter=10, regParam=0.05, rank=18, userCol="userId",␣
 ↪itemCol="movieId", ratingCol="rating", coldStartStrategy="drop",␣
 ↪nonnegative= True)          # regularization param)
model = als.fit(training_df)
# make prediction
predictions = model.transform(validation_df)
new_predictions = predictions.filter(col('prediction') != np.nan)
rmse = evaluator.evaluate(new_predictions)
print ("For rank =",18, "reg =", 0.05 ," the RMSE= " ,rmse)
```

```
For rank = 18 reg = 0.05  the RMSE=  0.9803222190248909
```

[41]:
```
iterations = 10
regularization_parameter = [0.001, 0.01, 0.05, 0.1, 0.2]
ranks = [8, 10, 12, 14, 16, 18, 20]
errors = []
err = 0
#tolerance = 0.02
```

[80]:
```
min_error = float('inf')
best_rank = -1
best_iteration = -1

for rank in ranks:
    for reg in regularization_parameter:
        # train ALS model
        als = ALS(maxIter=iterations, regParam=reg, rank=rank,␣
 ↪userCol="userId", itemCol="movieId", ratingCol="rating",␣
 ↪coldStartStrategy="drop", nonnegative= True)          # regularization param)
        #model = als.fit(training_df)
        # make prediction
        #predictions = model.transform(validation_df)
        #new_predictions = predictions.filter(col('prediction') != np.nan)
        param_grid= ParamGridBuilder().addGrid(als.rank,[rank]).addGrid(als.
 ↪maxIter,[10]).addGrid(als.regParam,[reg]).build()
        evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",␣
 ↪predictionCol="prediction")
        crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=param_grid,
                          evaluator=evaluator,
                          numFolds=5)
        cvModel = crossval.fit(training_df)
        cvModel_pred = cvModel.transform(validation_df)
        cvModel_pred = cvModel_pred.filter(col('prediction') != np.nan)
        rmse = evaluator.evaluate(cvModel_pred)
        errors.append(rmse)


        print ("For rank =",rank, "reg =", reg ," the RMSE= " ,rmse)
        if rmse < min_error:
            min_error = rmse
            best_rank = rank
            best_reg = reg
print ("The best model was trained with rank= ", best_rank, "With reg= ",␣
 ↪best_reg)
```

```
For rank = 8 reg = 0.001  the RMSE=  1.4467448634372828
For rank = 8 reg = 0.01  the RMSE=  1.141916955112322
For rank = 8 reg = 0.05  the RMSE=  0.9856996845615988
For rank = 8 reg = 0.1  the RMSE=  0.9141869244440859
For rank = 8 reg = 0.2  the RMSE=  0.8966076281249638
For rank = 10 reg = 0.001  the RMSE=  1.456039625308575
For rank = 10 reg = 0.01  the RMSE=  1.2036612472694024
For rank = 10 reg = 0.05  the RMSE=  0.9956121741381565
For rank = 10 reg = 0.1  the RMSE=  0.9158487323974703
For rank = 10 reg = 0.2  the RMSE=  0.8964889415116521
For rank = 12 reg = 0.001  the RMSE=  1.5242743892266397
For rank = 12 reg = 0.01  the RMSE=  1.235516337012286
For rank = 12 reg = 0.05  the RMSE=  1.0054946510104066
For rank = 12 reg = 0.1  the RMSE=  0.9178090013903915
For rank = 12 reg = 0.2  the RMSE=  0.8985851115100361
For rank = 14 reg = 0.001  the RMSE=  1.5311798036079438
For rank = 14 reg = 0.01  the RMSE=  1.2574262798813995
For rank = 14 reg = 0.05  the RMSE=  1.004432265748177
For rank = 14 reg = 0.1  the RMSE=  0.9163648847902192
For rank = 14 reg = 0.2  the RMSE=  0.8990094563068959
For rank = 16 reg = 0.001  the RMSE=  1.5977755203925776
For rank = 16 reg = 0.01  the RMSE=  1.2945525589067843
For rank = 16 reg = 0.05  the RMSE=  1.004848074487897
For rank = 16 reg = 0.1  the RMSE=  0.9149908997725777
For rank = 16 reg = 0.2  the RMSE=  0.8988097775919107
For rank = 18 reg = 0.001  the RMSE=  1.64173527955219
For rank = 18 reg = 0.01  the RMSE=  1.3049537155223991
For rank = 18 reg = 0.05  the RMSE=  1.008261040265912
For rank = 18 reg = 0.1  the RMSE=  0.915377249956471
For rank = 18 reg = 0.2  the RMSE=  0.8988576731676955
For rank = 20 reg = 0.001  the RMSE=  1.7042923866197208
For rank = 20 reg = 0.01  the RMSE=  1.3538113042962867
For rank = 20 reg = 0.05  the RMSE=  1.0154965096385316
For rank = 20 reg = 0.1  the RMSE=  0.9160800093109257
For rank = 20 reg = 0.2  the RMSE=  0.8999188491754373
The best model was trained with rank=  10 With reg=  0.2
```

```
      ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-80-738ecb0fa263> in <module>
         30             best_reg = reg
         31 print ("The best model was trained with rank= ", best_rank, "With␣
↪reg= ", best_reg)
    ---> 32 best = model.bestModel


        NameError: name 'model' is not defined
```

[82]:

```
best = cvModel.bestModel
```

```
best = ALS_42b8a21f89f7ed1ddbfd
```

[102]:

```
#here testing with new test data
als = ALS(maxIter=iterations, regParam=0.2, rank=10, userCol="userId",␣
 ↪itemCol="movieId", ratingCol="rating")
#param_grid= ParamGridBuilder().addGrid(als.rank,[10]).addGrid(als.
 ↪maxIter,[10]).addGrid(als.regParam,[0.2]).build()
cvModel = als.fit(training_df)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",␣
 ↪predictionCol="prediction")
cvModel_pred = cvModel.transform(test_df)
cvModel_pred = cvModel_pred.filter(col('prediction') != np.nan)
rmseT = evaluator.evaluate(cvModel_pred)
print("test data Rmse= ", rmseT)
#display(cvModel_pred.sort("userID","rattings"))
```

```
test data Rmse=  0.8935807766860681
```

[103]:

```
prediction = (cvModel_pred.sort(newrating["userID"]))
prediction.show()
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|     1|   2143|   4.0| 3.4857223|
|     1|   2959|   5.0| 4.8959446|
|     1|    736|   3.0|  3.546459|
|     1|   3273|   5.0|  2.521188|
|     1|   2078|   5.0| 4.5469894|
|     1|    553|   5.0| 4.2281413|
|     1|    733|   4.0|  4.186975|
|     1|   2141|   5.0| 3.6555433|
|     1|    423|   3.0| 3.2964041|
|     1|   2654|   5.0| 3.6944335|
|     1|   1270|   5.0|  4.675041|
|     1|     47|   5.0|  4.542998|
|     1|   1136|   5.0| 4.7466288|
|     1|   1226|   5.0| 3.6889038|
|     1|   3527|   4.0| 4.1616406|
|     1|    367|   4.0| 3.7773545|
|     1|      1|   4.0| 4.5714073|
|     1|    500|   3.0| 3.9910703|
|     1|   2987|   5.0|  4.081792|
|     1|   1025|   5.0| 4.2760987|
+------+-------+------+----------+
only showing top 20 rows
```

[104]:

```
cvModel_pred = cvModel_pred.na.drop()
cvModel_pred.describe().show()
```

```
+-------+------------------+------------------+------------------+--------------
----+
|summary|            userId|           movieId|            rating|
prediction|
+-------+------------------+------------------+------------------+--------------
----+
|  count|             19140|             19140|             19140|
19140|
|   mean| 322.9807732497388|17542.324503657263|3.5158307210031348|
3.272393845381408|
| stddev|181.02445275760041|32812.525669396804|1.0359594289983285|0.669760686352
2036|
|    min|                 1|                 1|               0.5|
0.28364804|
|    max|               610|            189333|               5.0|
5.4728813|
+-------+------------------+------------------+------------------+--------------
----+
[125]: user_recs = best.recommendForAllUsers(10)
       user_recs
```

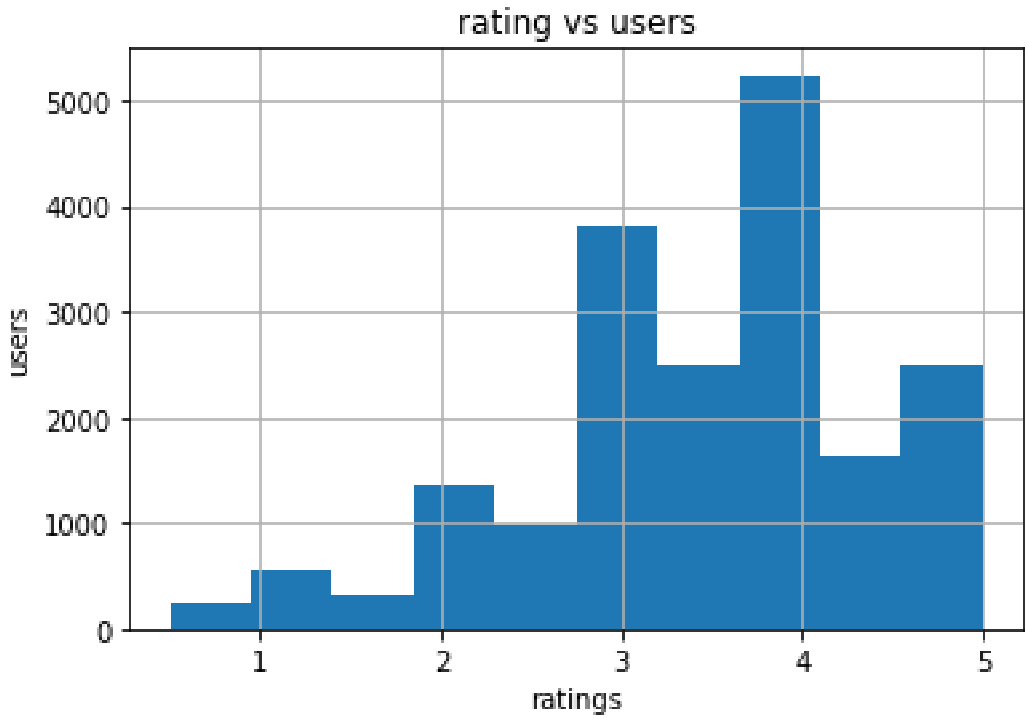
```
[125]: DataFrame[userId: int, recommendations: array<struct<movieId:int,rating:float>>]
```

```
[126]: def recs_users(recs):
           recs = recs.select("recommendations.movieId", "recommendations.rating")
           movies = recs.select("movieId").toPandas().iloc[0,0]
           ratings = recs.select("rating").toPandas().iloc[0,0]
           ratings_matrix = pd.DataFrame(movies, columns = ["movieId"])
           ratings_matrix["ratings"] = ratings
           ratings_matrix_ps = ratings_matrix
           return ratings_matrix_ps
```

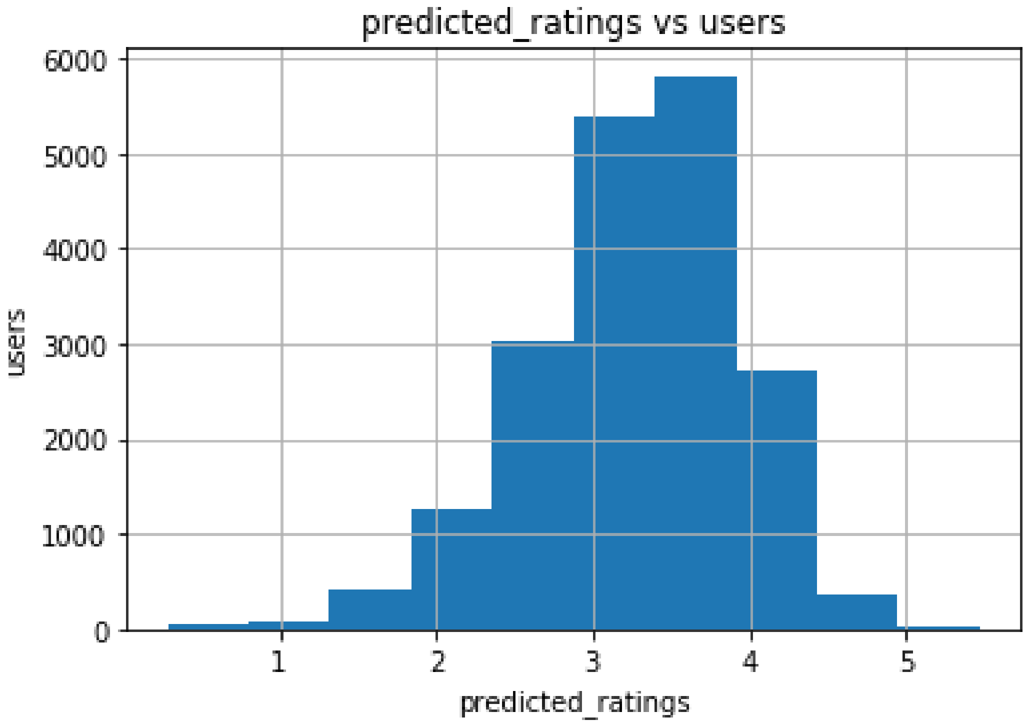
```
[127]: test = recs_users(user_recs)
       test.join(movies)
```

```
[127]:    movieId   ratings                               movieId,title,genres
       0     3379  4.701992  1,Toy Story (1995),Adventure|Animation|Childre…
       1     6818  4.486014        2,Jumanji (1995),Adventure|Children|Fantasy
       2     3358  4.478576           3,Grumpier Old Men (1995),Comedy|Romance
       3     5915  4.441676    4,Waiting to Exhale (1995),Comedy|Drama|Romance
       4     5490  4.441676        5,Father of the Bride Part II (1995),Comedy
       5    99764  4.402958                6,Heat (1995),Action|Crime|Thriller
       6   148881  4.402958                    7,Sabrina (1995),Comedy|Romance
       7    40491  4.402958           8,Tom and Huck (1995),Adventure|Children
       8     8477  4.402958                       9,Sudden Death (1995),Action
       9     3153  4.383376      10,GoldenEye (1995),Action|Adventure|Thriller
```

```
[135]: cvModel_pred.toPandas()['rating'].hist()
       plt.xlabel('ratings')
       plt.ylabel('users')
       plt.title('rating vs users')
       plt.show()
```



[136]:
```
cvModel_pred.toPandas()['prediction'].hist()
plt.xlabel('predicted_ratings')
plt.ylabel('users')
plt.title('predicted_ratings vs users')
plt.show()
```


predicted_ratings vs users
6000
5000
4000
3000
2000
1000
0
users
1
2
3
4
5
predicted_ratings

[ ]:

```
```

[ ]:

```
```