Collaborative_Filtering

September 6, 2023

[0]:
```
# Importing the necessary libraries
import pandas as pd
from pyspark.sql.functions import collect_list, col
from pyspark.sql import SparkSession
```

[0]:
```
# For reading the files from MongoDB
mongo_uri = "mongodb+srv://rmaxseiner:s7wUCv7q7Xkji8P@cluster0.opg6m.mongodb.
 ↪net"
database_name = "AIT_614"
collection_name = "orders"

spark = SparkSession.builder \
    .appName("MongoDBAtlasConnector") \
    .config("spark.mongodb.input.uri", f"{mongo_uri}/{database_name}.
 ↪{collection_name}") \
    .config("spark.mongodb.output.uri", f"{mongo_uri}/{database_name}.
 ↪{collection_name}") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.
 ↪12:3.0.1") \
    .getOrCreate()


collection_name = "order_product"

order_product_train_df = spark.read.format("com.mongodb.spark.sql.
 ↪DefaultSource") \
    .option("uri", f"{mongo_uri}/{database_name}.{collection_name}") \
    .option("pipeline", "[{ $match: { data_set: 'train' } }]") \
    .load()
print("Then number of records in the " + collection_name + " dataframe train␣
 ↪set is " + str(order_product_train_df.count()))

print(order_product_train_df.describe())
```

```
order_product_prior_df = spark.read.format("com.mongodb.spark.sql.
 ↪DefaultSource") \
    .option("uri", f"{mongo_uri}/{database_name}.{collection_name}") \
    .option("pipeline", "[{ $match: { data_set: 'prior' } }]") \
    .load()

print("Then number of records in the " + collection_name + " dataframe prior␣
 ↪set is " + str(order_product_prior_df.count()))
```

```
Then number of records in the order_product dataframe train set is 1384617
DataFrame[summary: string, add_to_cart_order: string, data_set: string,
order_id: string, product_id: string, reordered: string]
Then number of records in the order_product dataframe prior set is 32434489
```

[0]:
```
# Reading another file from Mongo DB
collection_name = "orders"
orders_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", f"{mongo_uri}/{database_name}.{collection_name}") \
    .load()

print("Then number of records in the " + collection_name + " dataframe is " +␣
 ↪str(orders_df.count()))
```

```
Then number of records in the orders dataframe is 3421083
```

[0]:
```
# Selecting only the necessary columns to reduce the load as this is a big␣
 ↪dataset
temp1 = order_product_prior_df.select('order_id', 'product_id')
temp2 = order_product_train_df.select('order_id', 'product_id')
```

[0]:
```
# Merging the above pyspark dataframes
merge_temp = temp1.union(temp2)
```

[0]:
```
# Displaying the merged dataframe
merge_temp.show(5)
```

```
+--------+----------+
|order_id|product_id|
+--------+----------+
|       2|     33120|
|       2|     28985|
|       2|      9327|
|       2|     45918|
|       2|     30035|
+--------+----------+
only showing top 5 rows
```

[0]:
```
# Choosing only the necessary columns from the orders dataframe
user_temp = orders_df.select('order_id', 'user_id')
```

[0]:
```
# Merging the orders with the train&prior merged dataframe to get the user_id's␣
 ↪for each order
df4 = merge_temp.join(user_temp, merge_temp.order_id == user_temp.order_id,␣
 ↪'inner').select(merge_temp["*"], user_temp['user_id'])
```

[0]:
```
# Displaying the combined dataframe
df4.show(5)
```

```
+--------+----------+-------+
|order_id|product_id|user_id|
+--------+----------+-------+
|      26|     35951| 153404|
|      26|     24852| 153404|
|      26|     46206| 153404|
|      26|     25890| 153404|
|      26|     33120| 153404|
+--------+----------+-------+
only showing top 5 rows
```

[0]:
```
# To get the user_id, product_id and the number of times the user has purchased␣
 ↪that particular product
df5 = df4.groupBy('user_id', 'product_id').count().sort("count" , ascending =␣
 ↪False)
```

[0]:
```
df5.show(5)
```

```
+-------+----------+-----+
|user_id|product_id|count|
+-------+----------+-----+
|  41356|      6583|  100|
|  41356|     14366|  100|
|  41356|     38652|  100|
|  17997|      4210|   99|
| 141736|     25133|   99|
+-------+----------+-----+
only showing top 5 rows
```

[0]:
```
# Ensuring that the columns are in the right datatype as they have to be in␣
 ↪integer format for the ALS model
final_df = df5.withColumn("user_id", df5["user_id"].cast('int')).
 ↪withColumn("prod_id", df5["product_id"].cast('int')).
 ↪withColumn("prod_count", df5["count"].cast('int'))
```

[0]:
```
# Checking the datatypes of each column
final_df.printSchema()
```

```
root
 |-- user_id: integer (nullable = true)
 |-- product_id: integer (nullable = true)
 |-- count: long (nullable = false)
 |-- prod_id: integer (nullable = true)
 |-- prod_count: integer (nullable = false)
```

[0]:
```
# Removing duplicate columns
final_df = final_df.select('user_id', 'prod_id','prod_count')
final_df.show(10)
```

```
+-------+-------+----------+
|user_id|prod_id|prod_count|
+-------+-------+----------+
|  41356|   6583|       100|
|  41356|  14366|       100|
|  41356|  38652|       100|
|  17997|   4210|        99|
| 141736|  25133|        99|
|  41356|  29671|        99|
| 103593|  28204|        99|
|  99707|  24852|        98|
| 120897|  12013|        98|
|  84478|  31981|        97|
+-------+-------+----------+
only showing top 10 rows
```

[0]:
```
# Checking for one sample user
display(final_df.filter(final_df.user_id == 1))
```

[0]:
```
# Running the ALS model for collaborative filtering
from pyspark.ml.recommendation import ALS

# Splitting the dataset into test and train for evalution purposes
train_df, test_df = final_df.randomSplit([0.8, 0.2])
```

[0]:
```
# Building the ALS model
als_obj = ALS(maxIter=5, rank=10, regParam=0.1, userCol="user_id",␣
 ↪itemCol="prod_id", ratingCol= "prod_count", coldStartStrategy="drop",␣
 ↪implicitPrefs=False)

als_model = als_obj.fit(train_df)
```

[0]:
```
# Testing the predictions
pred_res = als_model.transform(test_df)

# Measuring the RMSE ( root mean square error)
from pyspark.ml.evaluation import RegressionEvaluator

reg_evaluator = RegressionEvaluator(metricName="rmse", labelCol="prod_count",␣
 ↪predictionCol="prediction")
rmse = reg_evaluator.evaluate(pred_res)

print(str(rmse))
```

```
2.6983857476651947
```

It was a real challenge to have the ALS model running. Preprocessing the code took the longest time. We tried running the Cross Validation but it kept shutting down the kernel due the large size of the dataset and the combinations of the parameters to be run for them.

[0]:
```
# Generate top 3 product recommendations for each user ( Top 3 recommendations␣
 ↪only - due to the large size of the dataset we chose to have only the top 3 )
userRecs = als_model.recommendForAllUsers(3)
```

[0]:
```
# Displaying the recommendations for one user
# This returns the top 3 product_id's and the rating for each of them
userRecs.where(userRecs.user_id==1).show(truncate = False)
```

```
+-------+-----------------------------------------------------------+
|user_id|recommendations                                            |
+-------+-----------------------------------------------------------+
|1      |[{19907, 38.025448}, {5997, 35.425365}, {43532, 32.658146}]|
+-------+-----------------------------------------------------------+
```

[0]:
```
# Generating recommendations for each item ( Top 3 recommendations only - due␣
 ↪to the large size of the dataset we chose to have only the top 3 )
rec_for_prod = als_model.recommendForAllItems(3)
```

[0]:
```
# Displaying the recommendations for one product
# This gives us the top 3 user_id's and rating for each of them
rec_for_prod.where(rec_for_prod.prod_id==10).show(truncate = False)
```

```
+-------+-----------------------------------------------------------+
|prod_id|recommendations                                            |
+-------+-----------------------------------------------------------+
|10     |[{16397, 46.7334}, {82414, 33.8878}, {26489, 32.607334}]|
+-------+-----------------------------------------------------------+
```

References:

1. [1] Dr. Liao's lab tutorials and code examples on blackboard for the AIT614 course
2. Collaborative Filtering - Pyspark - https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html