

PKA-MovieRecommender-DataFrame

August 22, 2024

```
[1]: from pyspark.sql import SparkSession
from pyspark.sql import functions as f

spark = SparkSession.builder.appName("MovieRecommender").config("spark.driver.
↪memory", "32g").getOrCreate()
```

```
[2]: ratings = (
    spark.read.json(
        path="data/movies.json"
    )
    # .withColumn("timestamp", f.to_timestamp(f.from_unixtime("timestamp")))
)
ratings.printSchema()
```

```
root
|-- helpfulness: string (nullable = true)
|-- product_id: string (nullable = true)
|-- profile_name: string (nullable = true)
|-- review: string (nullable = true)
|-- score: double (nullable = true)
|-- summary: string (nullable = true)
|-- time: long (nullable = true)
|-- user_id: string (nullable = true)
```

```
[3]: import pyspark.sql.functions as F
from pyspark.sql.functions import col, regexp_extract, sha1
from pyspark.sql.types import IntegerType, LongType
from numpy import array
import hashlib
import math
hex_to_bigint_udf = F.udf(lambda x: int(x, 16) % (10 ** 8), LongType())
ratings = ratings.withColumn('user_id', hex_to_bigint_udf(sha1(col('user_id')).
↪cast('string')))) \
    .withColumn('product_id',
↪hex_to_bigint_udf(sha1(col('product_id')).cast('string')))) \
```

```

        .select('user_id', 'product_id', col('score').cast('int'))
#ratings = ratings.withColumn("user_id", regexp_extract(col("user_id"), r"\d+",
    ↳0).cast(IntegerType()))
#ratings = ratings.withColumn("product_id", regexp_extract(col("product_id"),
    ↳r"\d+", 0).cast(IntegerType()))
print(ratings.count())
#ratings=ratings.dropna()
#print(ratings.count())
ratings.show(1)

#ratings.show(1)

```

50000

```

+-----+-----+-----+
|user_id|product_id|score|
+-----+-----+-----+
|5460385| 51259877|    3|
+-----+-----+-----+
only showing top 1 row

```

The ALS class has this signature:

```

class pyspark.ml.recommendation.ALS(
    rank=10,
    maxIter=10,
    regParam=0.1,
    numUserBlocks=10,
    numItemBlocks=10,
    implicitPrefs=False,
    alpha=1.0,
    userCol="user",
    itemCol="item",
    seed=None,
    ratingCol="rating",
    nonnegative=False,
    checkpointInterval=10,
    intermediateStorageLevel="MEMORY_AND_DISK",
    finalStorageLevel="MEMORY_AND_DISK",
    coldStartStrategy="nan",
)

```

```

[4]: from pyspark.ml.recommendation import ALS
     from pyspark.ml.evaluation import RegressionEvaluator

```

```
[5]: als = ALS(
      userCol="user_id",
      itemCol="product_id",
      ratingCol="score",
    )

    (training_data, validation_data) = ratings.randomSplit([8.0, 2.0])

    evaluator = RegressionEvaluator(
        metricName="rmse", labelCol="score", predictionCol="prediction"
    )

    model = als.fit(training_data)
    predictions = model.transform(validation_data)
```

```
[6]: predictions.show(3)
```

```
+-----+-----+-----+-----+
|user_id|product_id|score|prediction|
+-----+-----+-----+-----+
|  56939|  34703051|    4|         NaN|
|  26869|  51749818|    5|         NaN|
|  26869|  53328528|    5|         NaN|
+-----+-----+-----+-----+
only showing top 3 rows
```

```
[7]: rmse = evaluator.evaluate(predictions.na.drop())
```

```
[8]: print(rmse)
```

```
1.8792253469747913
```

```
[9]: from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

    parameter_grid = (
        ParamGridBuilder()
        .addGrid(als.rank, [1, 5])
        .addGrid(als.maxIter, [20])
        .addGrid(als.regParam, [0.05])
        .addGrid(als.alpha, [1])
        .build()
    )
```

```
[10]: type(parameter_grid)
```

```
[10]: list
```

```
[11]: from pprint import pprint
```

```
pprint(parameter_grid)
```

```
[{Param(parent='ALS_3ce3f0d6d02e', name='alpha', doc='alpha for implicit
preference'): 1.0,
  Param(parent='ALS_3ce3f0d6d02e', name='maxIter', doc='max number of iterations
(>= 0).'): 20,
  Param(parent='ALS_3ce3f0d6d02e', name='regParam', doc='regularization
parameter (>= 0).'): 0.05,
  Param(parent='ALS_3ce3f0d6d02e', name='rank', doc='rank of the
factorization'): 1},
 {Param(parent='ALS_3ce3f0d6d02e', name='alpha', doc='alpha for implicit
preference'): 1.0,
  Param(parent='ALS_3ce3f0d6d02e', name='maxIter', doc='max number of iterations
(>= 0).'): 20,
  Param(parent='ALS_3ce3f0d6d02e', name='regParam', doc='regularization
parameter (>= 0).'): 0.05,
  Param(parent='ALS_3ce3f0d6d02e', name='rank', doc='rank of the
factorization'): 5}]
```

```
[12]: crossvalidator = CrossValidator(
    estimator=als,
    estimatorParamMaps=parameter_grid,
    evaluator=evaluator,
    numFolds=2,
)
```

```
crossval_model = crossvalidator.fit(training_data)
predictions = crossval_model.transform(validation_data)
```

```
[13]: rmse = evaluator.evaluate(predictions.na.drop())
print(rmse)
```

```
4.77205709228304
```

```
[14]: model = crossval_model.bestModel
```

```
[15]: print(model)
```

```
ALSModel: uid=ALS_3ce3f0d6d02e, rank=1
```

```
[ ]:
```