BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Introduction to the MovieLens dataset

Jamen Long
Data Scientist

MOVIELENS DATASET:

F. Maxwell Harper and Joseph A. Konstan. 2015
The MovieLens Datasets: History and Context.
ACM Transitions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 Pages. DOI=http://dx.doi.org/10.1145/2827872

MOVIELENS DATASET:

F. Maxwell Harper and Joseph A. Konstan. 2015
The MovieLens Datasets: History and Context.
ACM Transitions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 Pages. DOI=http://dx.doi.org/10.1145/2827872

Ratings: 20,00263
Users: 138,493
Movies: 27,278

Explore the Data

```
df.show()
df.columns()
```

MovieLens Sparsity

$$Sparsity = \frac{Number\ of\ Ratings\ in\ Matrix}{(Number\ of\ Users)\ \mathrm{x}\ (Number\ of\ Movies)}$$

Sparsity: Numerator

```
# Number of ratings in matrix
numerator = ratings.count()
```

Sparsity: Users and Movies

```
# Distinct users and movies
users = ratings.select("userId").distinct().count()
movies = ratings.select("movieId").distinct().count()
```

Sparsity: Denominator

```
# Number of ratings in matrix
numerator = ratings.count()

# Distinct users and movies
users = ratings.select("userId").distinct().count()
movies = ratings.select("movieId").distinct().count()

# Number of ratings matrix could contain if no empty cells
denominator = users * movies
```

Sparsity

```
# Number of ratings in matrix
numerator = ratings.count()

# Distinct users and movies
users = ratings.select("userId").distinct().count()
movies = ratings.select("movieId").distinct().count()

# Number of ratings matrix could contain if no empty cells
denominator = users * movies

#Calculating sparsity
sparsity = 1 - (numerator*1.0 / denominator)
print ("Sparsity: "), sparsity
```

```
Sparsity: .998
```

The .distinct() Method

```
ratings.select("userId").distinct().count()
```

```
671
```

GroupBy Method

```
# Group by userId
ratings.groupBy("userId")
```

GroupBy Method

```
# Num of song plays by userId
ratings.groupBy("userId").count().show()
```

```
+------+-----+
|userId|count|
+------+-----+
|   148|   76|
|   243|   12|
|    31|  232|
|   137|   16|
|   251|   19|
|    85|  752|
|    65|  737|
|   255|    9|
|    53|  190|
|   133|  302|
|   296|   74|
|    78|  301|
|   108|  136|
|   155|    3|
|   193|  174|
|   101|    1|
+------+-----+
```

GroupBy Method Min

```
from pyspark.sql.functions import min, max, avg

# Min num of song plays by userId
msd.groupBy("userId").count()
                .select(min("count")).show()
```

```
+----------+
|min(count)|
+----------+
|         1|
+----------+
```

GroupBy Method Max

```
from pyspark.sql.functions import min, max, avg

# Min num of song plays by userId
ratings.groupBy("userId").count()
                .select(min("count")).show()
```

```
+----------+
|min(count)|
+----------+
|        56|
+----------+
```

```
# Max num of song plays by userId
ratings.groupBy("userId").count()
                .select(max("count")).show()
```

```
+----------+
|max(count)|
+----------+
|      1162|
+----------+
```

GroupBy Method Avg

```
# Avg num of song plays by userId
ratings.groupBy("userId").count()
               .select(avg("count")).show()
```

```
+-----------+
|avg(count)|
+-----------+
|  233.34579|
+-----------+
```

Filter Method

```
# Removes users with less than 20 ratings
ratings.groupBy("userId").count().filter(col("count") >= 20).show()
```

```
+------+-----+
|userId|count|
+------+-----+
|   148|   76|
|    31|  232|
|    85|  752|
|    65|  737|
|    53|  190|
|   133|  302|
|   296|   74|
|    78|  301|
|   108|  136|
|   193|  174|
+------+-----+
```

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

ALS model buildout on MovieLens Data

Jamen Long
Data Scientist

Fitting a Basic Model

```
# Split data
(training_data, test_data) = movie_ratings.randomSplit([0.8, 0.2])

# Build ALS model
from pyspark.ml.recommendation import ALS

als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
            rank=25, maxIter=100, regParam=.05, nonnegative=True,
            coldStartStrategy="drop", implicitPrefs=False)

# Fit model to training data
model = als.fit(training_data)

# Generate predictions on test_data
predictions = model.transform(test_data)

# Tell Spark how to evaluate predictions
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
# Obtain and print RMSE
rmse = evaluator.evaluate(predictions)
print ("RMSE: "), rmse
```

```
RMSE: 1.45
```

Intro to ParamGridBuilder and CrossValidator

```
ParamGridBuilder()

CrossValidator()
```

ParamGridBuilder

```
# Imports ParamGridBuilder package
from pyspark.ml.tuning import ParamGridBuilder

# Creates a ParamGridBuilder
param_grid = ParamGridBuilder()
```

Adding Hyperparameters to the ParamGridBuilder

```
# Imports ParamGridBuilder package
from pyspark.ml.tuning import ParamGridBuilder

# Creates a ParamGridBuilder, and adds hyperparameters
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [])
                    .addGrid(als.maxIter, [])
                    .addGrid(als.regParam, [])
```

Adding Hyperparameter Values to the ParamGridBuilder

```
# Imports ParamGridBuilder package
from pyspark.ml.tuning import ParamGridBuilder

# Creates a ParamGridBuilder, and adds hyperparameters and values
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [5, 40, 80, 120])
                    .addGrid(als.maxIter, [5, 100, 250, 500])
                    .addGrid(als.regParam, [.05, .1, 1.5])
                    .build()
```

CrossValidator

```
# Imports CrossValidator package
from pyspark.ml.tuning import CrossValidator

# Creates cross validator and tells Spark what to use when training # and evalua
cv = CrossValidator(estimator = als,
                    estimatorParamMaps = param_grid,
                    evaluator = evaluator,
                    numFolds = 5)
```

Cross Validator Instantiation and Estimator

```
# Imports CrossValidator package
from pyspark.ml.tuning import CrossValidator

# Instantiates a cross validator
cv = CrossValidator()
```

Cross Validator ParamMaps

```
# Imports CrossValidator package
from pyspark.ml.tuning import CrossValidator

# Tells Spark what to use when training a model
cv = CrossValidator(estimator = als,
                    estimatorParamMaps = param_grid,
                    )
```

Cross Validator

```
# Imports CrossValidator package
from pyspark.ml.tuning import CrossValidator

# Tells Spark what alg, hyperparameter values, how to evaluate
# each model and number of folds to use during training
cv = CrossValidator(estimator = als,
                    estimatorParamMaps = param_grid,
                    evaluator = evaluator,
                    numFolds = 5)
```

Random Split

```
# Create training and test set (80/20 split)
(training, test) = movie_ratings.randomSplit([0.8, 0.2])

# Build generic ALS model without hyperparameters
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
            coldStartStrategy="drop", nonnegative = True,
            implicitPrefs = False)
```

ParamGridBiulder

```
# Create training and test set (80/20 split)
(training, test) = movie_ratings.randomSplit([0.8, 0.2])

# Build generic ALS model without hyperparameters
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
            coldStartStrategy="drop", nonnegative = True,
            implicitPrefs = False)

# Tell Spark what values to try for each hyperparameter
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [5, 40, 80, 120])
                    .addGrid(als.maxIter, [5, 100, 250, 500])
                    .addGrid(als.regParam, [.05, .1, 1.5])
                    .build()
```

Evaluator

```
# Create training and test set (80/20 split)
(training, test) = movie_ratings.randomSplit([0.8, 0.2])

# Build generic ALS model without hyperparameters
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
            coldStartStrategy="drop", nonnegative = True,
            implicitPrefs = False)

# Tell Spark what values to try for each hyperparameter
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [5, 40, 80, 120])
                    .addGrid(als.maxIter, [5, 100, 250, 500])
                    .addGrid(als.regParam, [.05, .1, 1.5])
                    .build()

# Tell Spark how to evaluate model performance
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
            predictionCol="prediction")
```

Cross Validator

```
# Build generic ALS model without hyperparameters
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
            coldStartStrategy="drop", nonnegative = True,
            implicitPrefs = False)

# Tell Spark what values to try for each hyperparameter
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [5, 40, 80, 120])
                    .addGrid(als.maxIter, [5, 100, 250, 500])
                    .addGrid(als.regParam, [.05, .1, 1.5])
                    .build()

# Tell Spark how to evaluate model performance
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
            predictionCol="prediction")

# Build cross validation step using CrossValidator
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator = als,
                    estimatorParamMaps = param_grid,
                    evaluator = evaluator,
                    numFolds = 5)
```

Best Model

```
# Tell Spark what values to try for each hyperparameter
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder()
                    .addGrid(als.rank, [5, 40, 80, 120])
                    .addGrid(als.maxIter, [5, 100, 250, 500])
                    .addGrid(als.regParam, [.05, .1, 1.5])
                    .build()

# Tell Spark how to evaluate model performance
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
            predictionCol="prediction")

# Build cross validation step using CrossValidator
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator = als,
                    estimatorParamMaps = param_grid,
                    evaluator = evaluator,
                    numFolds = 5)

# Run the cv on the training data
model = cv.fit(training)

# Extract best combination of values from cross validation
best_model = model.bestModel
```

Predictions and Performance Evaluation

```
# Extract best combination of values from cross validation
best_model = model.bestModel

# Generate test set predictions and evaluate using RMSE
predictions = best_model.transform(test)
rmse = evaluator.evaluate(predictions)

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

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Model Performance Evaluation and Output Cleanup

Jamen Long
Data Scientist

Root Mean Squared Error

$$\text{RMSE} = \sqrt{\frac{\Sigma(y_{\text{pred}} - y_{\text{actual}})^2}{N}}$$

Pred vs Actual

```
+----+------+
|pred|actual|
+----+------+
|   5|   4.5|
|   3|   3.5|
|   4|     4|
|   2|     1|
+----+------+
```

Pred vs Actual: Difference

```
+----+------+----+
|pred|actual|diff|
+----+------+----+
|   5|   4.5| 0.5|
|   3|   3.5|-0.5|
|   4|     4| 0.0|
|   2|     1| 1.0|
+----+------+----+
```

Difference Squared

```
+----+------+----+-------+
|pred|actual|diff|diff_sq|
+----+------+----+-------+
|   5|   4.5| 0.5|   0.25|
|   3|   3.5|-0.5|   0.25|
|   4|     4| 0.0|   0.00|
|   2|     1| 1.0|   1.00|
+----+------+----+-------+
```

Sum of Difference Squared

```
+----+------+----+-------+
|pred|actual|diff|diff_sq|
+----+------+----+-------+
|   5|   4.5| 0.5|   0.25|
|   3|   3.5|-0.5|   0.25|
|   4|     4| 0.0|   0.00|
|   2|     1| 1.0|   1.00|
+----+------+----+-------+

sum of diff_sq = 1.5
```

Average of Difference Squared

```
+----+------+----+-------+
|pred|actual|diff|diff_sq|
+----+------+----+-------+
|   5|   4.5| 0.5|   0.25|
|   3|   3.5|-0.5|   0.25|
|   4|     4| 0.0|   0.00|
|   2|     1| 1.0|   1.00|
+----+------+----+-------+

sum of diff_sq = 1.5
avg of diff_sq = 1.5 / 4 = 0.375
```

RMSE

```
+----+------+----+-------+
|pred|actual|diff|diff_sq|
+----+------+----+-------+
|   5|   4.5| 0.5|   0.25|
|   3|   3.5|-0.5|   0.25|
|   4|     4| 0.0|   0.00|
|   2|     1| 1.0|   1.00|
+----+------+----+-------+

sum of diff_sq = 1.5
avg of diff_sq = 1.5 / 4 = 0.375
RMSE = sq root of avg of diff_sq = 0.61
```

Recommend for all users

```
# Generate n recommendations for all users
recommendForAllUsers(n) # n is an integer
```

Unclean Recommendation Output

```
ALS_recommendations.show()
```

```
+------+---------------------+
|userId|      recommendations|
+------+---------------------+
|   360|[[65037, 4.491346]...|
|   246|[[3414, 4.8967672]...|
|   346|[[4565, 4.9247236]...|
|   476|[[83318,4.9556283]...|
|   367|[[4632, 4.7018986]...|
|   539|[[1172, 5.2528191]...|
|   599|[[6413, 4.7284415]...|
|   220|[[80,   4.4857406]...|
|   301|[[66665, 5.190159]...|
|   173|[[65037, 4.316745]...|
+------+---------------------+
```

Cleaning Up Recommendation Output

```
ALS_recommendations.registerTempTable("ALS_recs_temp")

clean_recs = spark.sql("SELECT userId,
                                movieIds_and_ratings.movieId AS movieId,
                                movieIds_and_ratings.rating AS prediction
                      FROM ALS_recs_temp
                      LATERAL VIEW explode(recommendations) exploded_table
                      AS movieIds_and_ratings")
```

Explode Function

```
exploded_recs = spark.sql("SELECT uderId,
                                  explode(recommendations) AS MovieRec
                           FROM ALS_recs_temp")
exploded_recs.show()
```

```
+------+----------------------------------------+
|userId|                                MovieRec|
+------+----------------------------------------+
|   360|{"movieId": 65037, "rating": 4.4913464}|
|   360|{"movieId": 59684, "rating": 4.4832921}|
|   360|{"movieId": 31435, "rating": 4.4822811}|
|   360|{"movieId": 593, "rating": 4.456215}    |
|   360|{"movieId": 67504, "rating": 4.4028492}|
|   360|{"movieId": 83411, "rating": 4.3391834}|
|   360|{"movieId": 83318, "rating": 4.3199939}|
|   360|{"movieId": 83359, "rating": 4.3000213}|
|   360|{"movieId": 76170, "rating": 4.2987138}|
|   360|{"movieId": 17, "rating": 4.2539403}   |
|   360|{"movieId": 2112, "rating": 4.11893843}|
+------+----------------------------------------+
```

Adding Lateral View

```
ALS_recommendations.registerTempTable("ALS_recs_temp")

clean_recs = spark.sql("SELECT userId,
                                movieIds_and_ratings.movieId AS movieId,
                                movieIds_and_ratings.rating AS prediction
                       FROM ALS_recs_temp
                       LATERAL VIEW explode(recommendations) exploded_table
                       AS movieIds_and_ratings")
```

Explode and Lateral View Together

```
ALS_recommendations.registerTempTable("ALS_recs_temp")

clean_recs = spark.sql("SELECT userId,
                                movieIds_and_ratings.movieId AS movieId,
                                movieIds_and_ratings.rating AS prediction
                      FROM ALS_recs_temp
                      LATERAL VIEW explode(recommendations) exploded_table
                      AS movieIds_and_ratings")
clean_recs.show()
```

```
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|   360|  65037|  4.491346|
|   360|  59684|  4.491346|
|   360|  34135|  4.491346|
|   360|    593|  4.453185|
|   360|  67504|  4.389951|
|   360|  83411|  4.389944|
|   360|  83318|  4.389938|
|   360|  83359|  4.373281|
|   360|  76173|  4.190159|
|   360|   5114|  4.116745|
+------+-------+----------+
```

```
clean_recs.join(movie_info, ["movieId"], "left").show()
```

```
+------+-----------------+--------------------+
|userId|movieId|prediction|               title|
+------+-----------------+--------------------+
|   360|  65037|  4.491346|        Ben X (2007)|
|   360|  59684|  4.491346| Lake of Fire (2006)|
|   360|  34135|  4.491346|Rory O Shea Was H...|
|   360|    593|  4.453185|Silence of the La...|
|   360|  67504|  4.389951|Land of Silence a...|
|   360|  83411|  4.389944|         Cops (1922)|
|   360|  83318|  4.389938|    Goat, The (1921)|
|   360|  83359|  4.373281| Play House, The(...|
|   360|  76173|  4.190159| Micmacs (Micmacs...|
|   360|   5114|  4.116745|Bad and the Beaut...|
+------+-----------------+--------------------+
```

Filtering Recommendations

```
clean_recs.join(movie_ratings, ["userId", "movieId"], "left")
```

```
clean_recs.join(movie_ratings, ["userId", "movieId"], "left").show()
```

```
+------+-------+----------+------+
|userId|movieId|prediction|rating|
+------+-------+----------+------+
|   173|    318|  4.947126|  null|
|   150|    318|  4.066513|   5.0|
|   369|    318|  4.514297|   5.0|
|    27|    318|  4.523860|  null|
|    42|    318|  4.568357|   5.0|
|   662|    318|  4.242076|   5.0|
|   250|    318|  5.042126|   5.0|
|    94|    318|  4.291757|   5.0|
|   515|    318|  5.165822|  null|
|   109|    318|  4.885314|   5.0|
+------+-------+----------+------+
```

```
clean_recs.join(movie_ratings, ["userId", "movieId"], "left")
            .filter(movie_ratings.rating.isNull()).show()
```

```
+------+-----------------+------+
|userId|movieId|prediction|rating|
+------+-----------------+------+
|   173|     318|  4.947126|  null|
|    27|     318|  4.523860|  null|
|   515|     318|  5.165822|  null|
|   275|     318|  5.171431|  null|
|   503|     318|  4.308533|  null|
|   106|     318|  4.688634|  null|
|   249|     318|  4.759836|  null|
|   368|     318|  3.589334|  null|
|   581|     318|  4.717382|  null|
|   208|     318|  3.920525|  null|
+------+-----------------+------+
```

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!