Reference

DataCamp course

Course Description

This course will show you how to build recommendation engines using Alternating Least Squares in PySpark. Using the popular MovieLens dataset and the Million Songs dataset, this course will take you step by step through the intuition of the Alternating Least Squares algorithm as well as the code to train, test and implement ALS models on various types of customer data.

Recommendations Are Everywhere

This chapter will show you how powerful recommendations engines can be, and provide important distinctions between collaborative-filtering engines and content-based engines as well as the different types of implicit and explicit data that recommendation engines can use. You will also learn a very powerful way to uncover hidden features (latent features) that you may not even know exist in customer datasets.

See the power of a recommendation engine

Taylor and Jane both like watching movies. Taylor only likes dramas, comedies and romances. Jane likes only action, adventure and otherwise exciting films. One of the greatest benefits of ALS-based recommendation engines is that they can identify movies or items that users will like, even if they themselves think that they might not like them. Take a look at the movie ratings that Taylor and Jane have provided below. It would stand to reason that their different preferences would generate different recommendations.

In [ ]:
```
TJ_ratings.show()

# Generate recommendations for users
get_ALS_recs(["Jane", "Taylor"])
```

```
+---------+--------------------+------+
|user_name|          movie_name|rating|
+---------+--------------------+------+
|   Taylor|            Twilight|   4.9|
|   Taylor|  A Walk to Remember|   4.5|
|   Taylor|        The Notebook|   5.0|
|   Taylor|Raiders of the Lo...|   1.2|
|   Taylor|      The Terminator|   1.0|
|   Taylor|      Mrs. Doubtfire|   1.0|
|     Jane|            Iron Man|   4.8|
|     Jane|Raiders of the Lo...|   4.9|
|     Jane|      The Terminator|   4.6|
```

```
|     Jane|          Anchorman|   1.2|
|     Jane|       Pretty Woman|   1.0|
|     Jane|          Toy Story|   1.2|
+---------+-------------------+------+

    userId       ...              genres
0   Taylor       ...               Drama
1   Taylor       ...               Drama
2   Taylor       ...              Comedy
3   Taylor       ...      Comedy|Romance
4   Taylor       ...      Comedy|Romance
5   Taylor       ...      Comedy|Drama|R
6     Jane       ...            Thriller
7     Jane       ...      Adventure|Fant
8     Jane       ...      Adventure|Fant
9     Jane       ...              Action
10    Jane       ...      Action|Adventu
11    Jane       ...      Action|Drama|W
12    Jane       ...      Action|Sci-Fi|
```

It seems that this recommendation engine produces meaningful results.

Collaborative vs Content-Based Filtering

Below are statements that are often used when providing recommendations, select the one that DOES NOT indicate collaborative filtering.

Answer the question 50 XP Possible Answers "Because you liked that product, we think you'll like this product" press 1 "Users that bought that also bought this" press 2 "Other people like you also liked this movie" press 3 "80% of your friends liked this movie, we think you'll like it too." press 4 "Here are top choices from similar users" press

Answer 1

Collaborative vs Content-Based Filtering Part II

Look at the df dataframe using the .show() method and/or the .columns method, and determine whether it is suited for "collaborative filtering", "content-based filtering", or "both".

+------+-------+-----------------+--------+--------+-------------+------+ |UserId|MovieId| Movie_Title| Genre|Language|Year_Produced|rating| +------+-------+-----------------+--------+--------+-------------+------+ | User1| 2112| Finding Nemo|Animated| English| 2003| 3| | User1| 2113| The Terminator| Action| English| 1984| 0| | User1| 2114| Spinal Tap| Satire| English| 1984| 4| | User1| 2115|Life Is Beautiful| Drama| Italian| 1998| 4| | User2| 2112| Finding Nemo|Animated| English| 2003| 4| | User2| 2113| The Terminator| Action| English| 1984| 0| | User2| 2114| Spinal Tap| Satire| English| 1984| 0| | User2| 2115|Life Is Beautiful| Drama| Italian| 1998| 4| | User3| 2112| Finding Nemo|Animated| English| 2003| 1| | User3| 2113| The Terminator| Action| English| 1984| 2| | User3| 2114| Spinal Tap| Satire| English| 1984| 1| | User3| 2115|Life Is Beautiful| Drama| Italian| 1998| 0| | User4| 2112| Finding

Nemo|Animated| English| 2003| 3| | User4| 2113| The Terminator| Action| English| 1984| 1| | User4| 2114| Spinal Tap| Satire| English| 1984| 0| | User4| 2115|Life Is Beautiful| Drama| Italian| 1998| 0| +------+-------+-----------------+--------+--------+-------------+------+

Possible Answers Collaborative filtering Content-based filtering Both collaborative and content-based filtering

Answer Because this dataset includes descriptive tags like genre and language, as well as user ratings, it is suited for both collaborative and content-based filtering.

Implicit vs Explicit Data

Recall the differences between implicit and explicit ratings. Take a look at the df1 dataframe to understand whether the data includes implicit or explicit ratings data.

+--------------------+------------------+---------+ | Movie_Title| Genre|Num_Views| +--------------------+------------------+---------+ | Finding Nemo|Animated Childrens| 12| | Toy Story|Animated Childrens| 6| | Iron Man| Action| 1| | Captain America| Action| 1| | The Incredibles|Animated Childrens| 9| | Frozen|Animated Childrens| 22| |The Shawshank Red...| Drama| 2| | Rabbit Proof Fence| Drama| 2| |Searching for Sug...| Documentary| 3| | Powder| Drama| 1| | The Fugitive| Action| 2| +--------------------+------------------+---------+

In [ ]:
```
# Type "implicit" or "explicit"
answer = "implicit"
```

Ratings data types

Markus watches a lot of movies, from documentaries to superhero movies, classics and dramas. Drawing on your previous experience with Spark, use the markus_ratings dataframe, which contains data on the number of movie views Markus has seen in various genres, think about whether these are implicit or explicit ratings, and use the groupBy() method to determine which genre has the highest rating, which could likely influence what recommendations ALS would generate for Markus.

In [ ]:
```
# Group the data by "Genre"
markus_ratings.groupBy("Genre").sum().show()
```

```
+------------------+--------------+
|             Genre|sum(Num_Views)|
+------------------+--------------+
|             Drama|             5|
|       Documentary|             3|
|            Action|             4|
|Animated Childrens|            49|
+------------------+--------------+
```

Alternate uses of recommendation engines.

Select the best definition of "latent features".

Answer the question 50 XP Possible Answers Features or tags that have manually been attached to items that categorize those items press 1 Features that are contained in data, but that aren't directly observable press 2 Features that show up "later" in the machine learning process press 3 Features that are added by human beings press 4

Answer 2: Latent features aren't directly observable by humans, and need mathematical operations to uncover them.

Confirm understanding of latent features

Matrix P is provided here. It's columns represent movies and it's rows represent several latent features. Use your understanding of Spark commands to view matrix P, and see if you can determine what some of the latent features might represent. After examining the matrix, look at the dataframe Pi which contains a rough approximation of what these latent features could represent. See if you weren't far off.

In [ ]:
```
# Examine matrix P using the .show() method
P.show()

# Examine matrix Pi using the .show() method
Pi.show()
```

```
+--------+------------+--------+---------+------------+------+----------+
|Iron Man|Finding Nemo|Avengers|Toy Story|Forrest Gump|Wall-E|Green Mile|
+--------+------------+--------+---------+------------+------+----------+
|     0.2|         2.4|     0.1|      2.4|           0|   2.5|         0|
|     1.5|         1.4|     1.4|      1.3|         1.8|   1.8|       2.5|
|     2.5|         1.1|     2.4|      0.9|         0.2|   0.9|      0.09|
|     1.9|           2|     1.5|      2.2|         1.2|   0.3|      0.01|
|       0|           0|       0|      2.3|         2.2|     0|       2.5|
+--------+------------+--------+---------+------------+------+----------+

+---------+--------+------------+--------+---------+------------+------+-
---------+
| Lat Feat|Iron Man|Finding Nemo|Avengers|Toy Story|Forrest Gump|Wall-E|G
reen Mile|
+---------+--------+------------+--------+---------+------------+------+-
---------+
| Animated|     0.2|         2.4|     0.1|      2.4|           0|   2.5|
0|
|    Drama|     1.5|         1.4|     1.4|      1.3|         1.8|   1.8|
2.5|
|Superhero|     2.5|         1.1|     2.4|      0.9|         0.2|   0.9|
0.09|
```

```
|   Comedy|     1.9|           2|     1.5|      2.2|          1.2|   0.3|
0.01|
|Tom Hanks|       0|           0|       0|      1.8|          2.2|     0|
2.5|
+---------+--------+------------+--------+---------+-----------+------+-
---------+
```

```
Did you notice how some movies cross genres pretty easily?
```

How does ALS work?

In this chapter you will review basic concepts of matrix multiplication and matrix factorization, and dive into how the Alternating Least Squares algorithm works and what arguments and hyperparameters it uses to return the best recommendations possible. You will also learn important techniques for properly preparing your data for ALS in Spark.

Matrix Multiplication

To understand matrix multiplication more directly, let's do some matrix operations manually.

In [ ]:
```
# Use the .head() method to view the contents of matrices a and b
print("Matrix A:")
print (a.head())

print("Matrix B:")
print (b.head())

# Complete the matrix called "product" with the product of matrices a and b.
product = np.array([[10,12], [15,18]])

# Run this validation to see how your estimate performs
product == np.dot(a,b)
```

```
Matrix A:
     0  1
One  2  2
Two  3  3
Matrix B:
     0  1
One  1  2
Two  4  4
```

Matrix Multiplication Part II

Let's put your matrix multiplication skills to test.

In [ ]:
```
# Print the dimensions of C
print(C.shape)

# Print the dimensions of D
print(D.shape)

# Can C and D be multiplied together?
C_times_D = None
```

```
(4, 5)
(3, 2)
```

Matrix Factorization

Matrix G is provided here as a Pandas dataframe. View it to understand what it looks like. Look at the possible factor matrices H, I, and J (also Pandas dataframes), and determine which two matrices will produce the matrix G when mutliplied together.

In [ ]:
```
# Take a look at Matrix G using the following print function
print("Matrix G:")
print(G)

# Take a look at the matrices H, I, and J and determine which pair of those matri
print("Matrix H:")
print(H)
print("Matrix I:")
print(I)
print("Matrix J:")
print(J)

# Multiply the two matrices that are factors of the matrix G
prod = np.matmul(H,J)
print(G == prod)
```

```
Matrix G:
   0  1
0  6  6
1  3  3
Matrix H:
   0  1
0  2  2
1  1  1
Matrix I:
   0  1
0  3  3
1  3  3
Matrix J:
   0  1
```

```
   0  1
0  1  1
1  2  2
      0     1
0  True  True
1  True  True
```

Non-Negative Matrix Factorization

It's possible for one matrix to have two equally close factorizations where one has all positive values, and the other has some negative values. The matrix M has been factored twice using two different factorizations. Take a look at each pair of factor matrices L and U, and W and H to see the differences. Then use their products to see that they produce essentially the same product.

In [ ]:
```
# View the L, U, W, and H matrices.
print("Matrices L and U:")
print(L)
print(U)

print("Matrices W and H:")
print(W)
print(H)

# Calculate RMSE for LU
print ("RMSE of LU: ", getRMSE(LU, M))

# Calculate RMSE for WH
print ("RMSE of WH: ", getRMSE(WH, M))
```

```
Matrices L and U:
      0         1         2  3
0  1.00  0.000000  0.000000  0
1  0.01 -0.421053  0.098316  1
2  1.00  0.000000  1.000000  0
3  0.10  1.000000  0.000000  0
   0     1      2         3
0  1  2.00  1.000  2.000000
1  0 -0.19 -0.099 -0.198000
2  0  0.00  1.000 -1.000000
3  0  0.00  0.000  0.194947
Matrices W and H:
      0     1     2     3
0  2.61  0.24  0.00  0.12
1  0.00  0.05  0.02  0.17
2  1.97  0.00  0.58  0.83
3  0.05  0.00  0.00  0.00
      0     1     2     3
0  0.38  0.65  0.34  0.41
1  0.00  1.20  0.15  3.72
```

```
2  0.42  1.09  1.38  0.07
3  0.00  0.11  0.65  0.17
RMSE of LU:  0.072
RMSE of WH:  0.072
```

Did you notice that LU and WH essentailly created the same product despite LU having some negative values and WH having all positive values?

Estimating Recommendations

Use your knowledge of matrix multiplication to determine which movie will have the highest recommendation for User_3. The ratings matrix has been factorized into U and P with ALS.

In [ ]:
```python
# View left factor matrix
print (U)
```

```
        U_LF_1   ...    U_LF_4
User_1    0.80   ...       0.8
User_2    0.40   ...       0.2
User_3    0.05   ...       2.2
User_4    0.30   ...       0.2
User_5    0.10   ...       0.0
User_6    0.00   ...       0.5
User_7    0.01   ...       0.4
User_8    0.90   ...       1.0
User_9    1.00   ...       0.2
```

In [ ]:
```python
# View right factor matrix
print (P)
```

```
        Movie_1   ...    Movie_4
P_LF_1      0.5   ...       1.10
P_LF_2      0.2   ...       0.01
P_LF_3      0.3   ...       0.90
P_LF_4      1.0   ...       0.89
```

In [ ]:
```python
# Multiply factor matrices
UP = np.matmul(U,P)

# Convert to pandas DataFrame
print (pd.DataFrame(UP, columns = P.columns, index = U.index))
```

```
        Movie_1   ...    Movie_4
User_1    1.292   ...     1.8621
```

```
User_2    0.420   ...      0.6721
User_3    2.648   ...      2.0430
User_4    0.412   ...      0.6881
User_5    0.620   ...      0.9350
User_6    0.626   ...      0.8053
User_7    0.607   ...      0.9612
User_8    1.590   ...      1.8870
User_9    1.112   ...      1.3340

Did you guess Movie 2? It has the highest predicted rating at 4.664 out o
f 5.
```

RMSE As ALS Alternates

As you know, ALS will alternate between the two factor matrices, adjusting their values each time to iteratively come closer and closer to approximating the original ratings matrix. This exercise is intended to illustrate this to you. Matrix T is a ratings matrix, and matrices F1, F2, F3, F4, F5, and F6 are the respective products of ALS after iterating 2, 3, 4, 5, and 6 times respectively. Follow the instructions below to see how the RMSE changes as ALS iterates.

In [ ]:
```
# Use the getRMSE(preds, actuals) function to calculate the RMSE of matrices T an
getRMSE(F1, T)

# Create list of F2, F3, F4, F5, and F6
Fs = [F2, F3, F4, F5, F6]

# Calculate RMSEs for F2 - F6
getRMSEs(Fs, T)
```

```
F1:  2.4791263858912522
F2: 0.4389326310548279
F3: 0.17555006757053257
F4: 0.15154042416388636
F5: 0.13191130368008455
F6: 0.04533823201006271
```

Correct format and distinct users

Take a look at the R dataframe. Notice that it is in conventional or "wide" format with a different movie in each column. Also notice that the User's and movie names are not in integer format. Follow the steps to properly prepare this data for ALS.

In [ ]:
```python
from pyspark.sql.functions import monotonically_increasing_id
R.show()

# Use the to_long() function to convert the dataframe to the "long" format.
ratings = to_long(R)
ratings.show()

# Get unique users and repartition to 1 partition
users = ratings.select("User").distinct().coalesce(1)

# Create a new column of unique integers called "userId" in the users dataframe.
users = users.withColumn("userId", monotonically_increasing_id()).persist()
users.show()
```

```
+----------------+-----+----+----------+--------+
|            User|Shrek|Coco|Swing Kids|Sneakers|
+----------------+-----+----+----------+--------+
|    James Alking|    3|   4|         4|       3|
|Elvira Marroquin|    4|   5|      null|       2|
|       Jack Bauer| null|   2|         2|       5|
|     Julia James|    5|null|         2|       2|
+----------------+-----+----+----------+--------+

+----------------+----------+------+
|            User|     Movie|Rating|
+----------------+----------+------+
|    James Alking|     Shrek|     3|
|    James Alking|      Coco|     4|
|    James Alking|Swing Kids|     4|
|    James Alking|  Sneakers|     3|
|Elvira Marroquin|     Shrek|     4|
|Elvira Marroquin|      Coco|     5|
|Elvira Marroquin|  Sneakers|     2|
|      Jack Bauer|      Coco|     2|
|      Jack Bauer|Swing Kids|     2|
|      Jack Bauer|  Sneakers|     5|
|     Julia James|     Shrek|     5|
|     Julia James|Swing Kids|     2|
|     Julia James|  Sneakers|     2|
+----------------+----------+------+

+----------------+------+
|            User|userId|
+----------------+------+
|Elvira Marroquin|     0|
|      Jack Bauer|     1|
|    James Alking|     2|
|     Julia James|     3|
+----------------+------+
```

Each user now has a unique integer assigned to it.

Assigning integer id's to movies

Let's now do the same thing to the movies. Then let's join the new user id's and movie id's into one dataframe.

In [ ]:
```
# Extract the distinct movie id's
movies = ratings.select("Movie").distinct()

# Repartition the data to have only one partition.
movies = movies.coalesce(1)

# Create a new column of movieId integers.
movies = movies.withColumn("movieId", monotonically_increasing_id()).persist()

# Join the ratings, users and movies dataframes
movie_ratings = ratings.join(users, "User", "left").join(movies, "Movie", "left")
movie_ratings.show()
```

```
+----------+---------------+------+------+-------+
|     Movie|           User|Rating|userId|movieId|
+----------+---------------+------+------+-------+
|     Shrek|   James Alking|     3|     2|      3|
|      Coco|   James Alking|     4|     2|      1|
|Swing Kids|   James Alking|     4|     2|      2|
|   Sneakers|   James Alking|     3|     2|      0|
|     Shrek|Elvira Marroquin|     4|     0|      3|
|      Coco|Elvira Marroquin|     5|     0|      1|
|   Sneakers|Elvira Marroquin|     2|     0|      0|
|      Coco|     Jack Bauer|     2|     1|      1|
|Swing Kids|     Jack Bauer|     2|     1|      2|
|   Sneakers|     Jack Bauer|     5|     1|      0|
|     Shrek|    Julia James|     5|     3|      3|
|Swing Kids|    Julia James|     2|     3|      2|
|   Sneakers|    Julia James|     2|     3|      0|
+----------+---------------+------+------+-------+
```

You now have distinct userId's and movieId's that are integer data types.

Build Out An ALS Model

Let's specify your first ALS model. Complete the code below to build your first ALS model.

```
In [ ]: # Split the ratings dataframe into training and test data
(training_data, test_data) = ratings.randomSplit([0.8, 0.2], seed=1234)

# Set the ALS hyperparameters
from pyspark.ml.recommendation import ALS
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", rank = 10, max
          coldStartStrategy="drop", nonnegative = True, implicitPrefs = False)

# Fit the mdoel to the training_data
model = als.fit(training_data)

# Generate predictions on the test_data
test_predictions = model.transform(test_data)
test_predictions.show()
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
+------+-------+------+----------+
```

ou just built our your first ALS model and generated some test predictions. It's a toy dataset, so the results aren't particularly meaningful, but you now know how to do this.

Build RMSE Evaluator

Now that you know how to fit a model to training data and generate test predicitons, you need a way to evaluate how well your model performs. For this we'll build an evaluator. Evaluators in Spark can be built out in various ways. For our purposes here, we want a regressionEvaluator that caclucates the RMSE. After we build our our regressionEvaluator, we can fit the model to our data and generate predictions.

```
In [ ]: # Import RegressionEvaluator
from pyspark.ml.evaluation import RegressionEvaluator

# Complete the evaluator code
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionC

# Extract the 3 parameters
print (evaluator.getMetricName())
print (evaluator.getLabelCol())
print (evaluator.getPredictionCol())
```

```
rmse
rating
prediction
```

You now know how to build a model, generate predictions, and build an evaluator to tell you how well the model predicted the test values.

Get RMSE

Now that you know how to build a model and generate predictions, and have an evaluator to tell us how well it predicts ratings, we can calculate the RMSE to see how well an ALS model performed. We'll use the evaluator that we built in the previous exercise to calculate and print the rmse.

In [ ]:
```
# Evaluate the "predictions" dataframe
RMSE = evaluator.evaluate(test_predictions)

# Print the RMSE
print (RMSE)
```

```
0.16853197489754093

This RMSE means that on average, the model's test predictions are about .
16 off from the true values.
```

Recommending Movies

In this chapter you will be introduced to the MovieLens dataset. You will walk through how to assess it's use for ALS, build out a full cross-validated ALS model on it, and learn how to evaluate it's performance. This will be the foundation for all subsequent ALS models you build using Pyspark.

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 (http://dx.doi.org/10.1145/2827872)

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

Viewing the MovieLens Data

Familiarize yourself with the ratings dataset provided here. Would you consider the data to be implicit or explicit ratings?

In [ ]:
```
# Look at the column names
print (ratings.columns)

# Look at the first few rows of data
print (ratings.show())
```

```
['userId', 'movieId', 'rating', 'timestamp']
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
|     1|     31|   2.5|1260759144|
|     1|   1029|   3.0|1260759179|
|     1|   1061|   3.0|1260759182|
```

```
|     1|   1129|   2.0|1260759185|
|     1|   1172|   4.0|1260759205|
|     1|   1263|   2.0|1260759151|
|     1|   1287|   2.0|1260759187|
|     1|   1293|   2.0|1260759148|
|     1|   1339|   3.5|1260759125|
|     1|   1343|   2.0|1260759131|
|     1|   1371|   2.5|1260759135|
|     1|   1405|   1.0|1260759203|
|     1|   1953|   4.0|1260759191|
|     1|   2105|   4.0|1260759139|
|     1|   2150|   3.0|1260759194|
|     1|   2193|   2.0|1260759198|
|     1|   2294|   2.0|1260759108|
|     1|   2455|   2.5|1260759113|
|     1|   2968|   1.0|1260759200|
|     1|   3671|   3.0|1260759117|
+------+-------+------+----------+
only showing top 20 rows

None
```

This dataset includes ratings from the customers. This indicates that these are explicit ratings.

Calculate sparsity

As you know, ALS works well with sparse datasets. Let's see how much of the ratings matrix is actually empty. Remember that sparsity is calculated by the number of cells in a matrix that contain a rating divided by the total number of values that matrix could hold given the number of users and items (movies). In other words, dividing the number of ratings present in the matrix by the product of users and movies in the matrix and subtracting that from 1 will give us the sparsity or the percentage of the ratings matrix that is empty.

In [ ]:

```
# Count the total number of ratings in the dataset
numerator = ratings.select("rating").count()

# Count the number of distinct users.
num_users = ratings.select("userId").distinct().count()

# Count the number of distinct movies.
num_movies = ratings.select("movieId").distinct().count()

# Set the denominator equal to the number of users multiplied by the number of mo
denominator = num_users * num_movies

# Divide the numerator by the denominator
sparsity = (1.0 - (numerator *1.0)/denominator)*100
print ("The ratings dataframe is ", "%.2f" % sparsity + "% empty.")
```

The ratings dataframe is 98.36% empty.

The GroupBy and Filter Methods

Now that we know a little more about the dataset, let's look at some general summary metrics of the ratings dataset and see how many ratings the movies have and how many ratings the users have provided. Two common methods that will be helpful to you as you perform summary statistics in Spark are the .filter() and the .groupBy() methods. The .filter() method allows you to filter out any data that doesn't meet your specified criteria.

In [ ]:
```python
# Import the requisite packages
from pyspark.sql.functions import col

# View the ratings dataset
ratings.show()

# Filter out all userIds greater than 100
ratings.filter(col("userId") < 100).show()

# Group data by userId, count song plays
ratings.groupBy("userId").count().show()
```

```
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
|     1|     31|   2.5|1260759144|
|     1|   1029|   3.0|1260759179|
|     1|   1061|   3.0|1260759182|
|     1|   1129|   2.0|1260759185|
|     1|   1172|   4.0|1260759205|
|     1|   1263|   2.0|1260759151|
|     1|   1287|   2.0|1260759187|
|     1|   1293|   2.0|1260759148|
|     1|   1339|   3.5|1260759125|
|     1|   1343|   2.0|1260759131|
|     1|   1371|   2.5|1260759135|
|     1|   1405|   1.0|1260759203|
|     1|   1953|   4.0|1260759191|
|     1|   2105|   4.0|1260759139|
|     1|   2150|   3.0|1260759194|
|     1|   2193|   2.0|1260759198|
|     1|   2294|   2.0|1260759108|
|     1|   2455|   2.5|1260759113|
|     1|   2968|   1.0|1260759200|
|     1|   3671|   3.0|1260759117|
+------+-------+------+----------+
only showing top 20 rows

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

```
|     1|     31|   2.5|1260759144|
|     1|   1029|   3.0|1260759179|
|     1|   1061|   3.0|1260759182|
|     1|   1129|   2.0|1260759185|
|     1|   1172|   4.0|1260759205|
|     1|   1263|   2.0|1260759151|
|     1|   1287|   2.0|1260759187|
|     1|   1293|   2.0|1260759148|
|     1|   1339|   3.5|1260759125|
|     1|   1343|   2.0|1260759131|
|     1|   1371|   2.5|1260759135|
|     1|   1405|   1.0|1260759203|
|     1|   1953|   4.0|1260759191|
|     1|   2105|   4.0|1260759139|
|     1|   2150|   3.0|1260759194|
|     1|   2193|   2.0|1260759198|
|     1|   2294|   2.0|1260759108|
|     1|   2455|   2.5|1260759113|
|     1|   2968|   1.0|1260759200|
|     1|   3671|   3.0|1260759117|
+------+-------+------+----------+
only showing top 20 rows

+------+-----+
|userId|count|
+------+-----+
|   296|   20|
|   467|   64|
|   125|  210|
|   451|   52|
|   666|   40|
|     7|   88|
|    51|   31|
|   124|   85|
|   447|   87|
|   591|   30|
|   307|   72|
|   475|  655|
|   574|  342|
|   613|   53|
|   169|  113|
|   205|  206|
|   334|   34|
|   544|  268|
|   577|  279|
|   581|   49|
+------+-----+
only showing top 20 rows
```

Now you know how to groupBy() and filter() pyspark dataframes. In the next exercise we are going to combine these two methods. If you want to apply two filters, you can do so like this: ratings.filter((col('userId') < 100) & (col('userId') > 50)).show().

MovieLens Summary Statistics

Let's take the groupBy() method a bit firther. Once you've applied the .groupBy() method to a dataframe, you can subsequently run aggregate functions such as .sum(), .avg(), .min() and have the results grouped. This exercise will walk you through how this is done. The min and avg functions have been imported from pyspark.sql.functions for you.

In [ ]:
```python
# Min num ratings for movies
print("Movie with the fewest ratings: ")
ratings.groupBy("movieId").count().select(min("count")).show()

# Avg num ratings per movie
print("Avg num ratings per movie: ")
ratings.groupBy("movieId").count().select(avg("count")).show()

# Min num ratings for user
print("User with the fewest ratings: ")
ratings.groupBy("userId").count().select(min("count")).show()

# Avg num ratings per users
print("Avg num ratings per user: ")
ratings.groupBy("userId").count().select(avg("count")).show()
```

```
Movie with the fewest ratings:
+----------+
|min(count)|
+----------+
|         1|
+----------+

Avg num ratings per movie:
+------------------+
|        avg(count)|
+------------------+
|11.030664019413193|
+------------------+

User with the fewest ratings:
+----------+
|min(count)|
+----------+
|        20|
+----------+

Avg num ratings per user:
```

```
+------------------+
|        avg(count)|
+------------------+
|149.03725782414307|
+------------------+
```

Users have at least 20 ratings and on average of 149 ratings. And movies have at least 1 rating with an average of 11 ratings.

View Schema

As you know from previous chapters, Spark's implementaiton of ALS requires that movieIds and userIds be provided as integer datatypes. Many datasets need to be prepared accordingly in order for them to function properly with Spark. A common issue is that Spark thinks numbers are strings, and vice versa. Here you'll use the .cast() method to address these types of problems. Let's take a look at the schema of the dataset to ensure it's in the correct format.

In [ ]:
```
# Use the .printSchema() method to see the datatypes of the ratings dataset.
ratings.printSchema()

# Tell Spark to convert the columns to the proper data types.
ratings = ratings.select(ratings.userId.cast("integer"), ratings.movieId.cast("in

# Call .printSchema() again to confirm the columns are now in the correct format
ratings.printSchema()
```

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

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

Create test/train splits and build your ALS model

You already know how to build an ALS model having done it in the previous chapter. We will do that again here, but we'll take some additional steps to fully build out a cross-validated model. First let's import the requisite packages and create our train and test data sets in preparation for the cross validation step.

In [ ]:
```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 1234)

# Create ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", nonnegative = 

# Confirm that a model called "als" was created
type(als)
```

Tell Spark how to tune your ALS model

Now we'll need to create a ParamGrid to tell Spark what hyperparameters we want it to tune, how to tune them, and then build out an evaluator so Spark can know how to measure the algorithm's performance.

In [ ]:
```
# Import the requisite items
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 75, 100]) \
            .addGrid(als.maxIter, [5, 50, 100, 200]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()

# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionC
print ("Num models to be tested: ", len(param_grid))
```

Num models to be tested: 64

Now Spark knows what combinations of hyperparameters to try and how to evaluate them.

Build your cross validation pipeline

Now that we have our data, our train/test splits, our model and our hyperparameter values, let's tell Spark how to cross validate our model so it can find the best combination of hyperparameters and return it to us.

In [ ]:
```
# Build cross validation using CrossValidaator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evalu

# Confirm cv was built
print (cv)
```

CrossValidator_41b5ad2de40a70630b63
You now have a working CrossValidator

Best Model and Best Model Parameters

Now we have our cross validator ("cv") built out. We can now tell Spark to take our data, fit the ALS algorithm to it, and try the different combinations of hyperparameter values from our param_grid so it can identify what values provide the smallest rmse. Unfortunately, this takes too long to complete here, but for your reference, this is how it is done:

#Fit cross validator to the 'train' dataset model = cv.fit(train)

#Extract best model from the cv model above best_model = model.bestModel This code has been run separately, and the best_model has been identified. Use the commands given to extract the parameters of the model.

In [ ]:
```
# Print best_model
print(type(best_model))

# Complete the code below to extract the ALS model parameters
print("**Best Model**")

# Print "Rank"
print("  Rank:", best_model.getRank())

# Print "MaxIter"
print("  MaxIter:", best_model.getMaxIter())

# Print "RegParam"
print("  RegParam:", best_model.getRegParam())
```

Best Model Rank: 50 MaxIter: 100 RegParam: 0.1 If you'll notice, the best hyperparameter values were all in the middle of the ranges we provided. If they were on the low or high end, we would simply adjust our ranges accordingly. Given that they were in the middle, we could tune even further by narrowing our range to get even more precise hyperparameter values.

Generate predictions and calculate RMSE

Now that we have a model that is trained on our data and tuned through cross validation, we can see how it performs on the test dataframe. To do this, we'll calculate the RMSE. As a side note, the generation of test predictions takes more than a few minutes with this dataset. For this reason, the test predictions have been generated already and are provided here as a dataframe called test_predictions. For your reference, they are generated using this code: test_predictions = best_model.transform(test).

In [ ]:
```
# View the predictions
test_predictions.show()

# Calculate the RMSE of the test_predictions
RMSE = evaluator.evaluate(test_predictions)
print (RMSE)
```

+------+-------+------+------------------+ |userId|movieId|rating| prediction| +------+-------+------+------------------+ | 380| 463| 3.0| 4.093334993256898| | 460| 471| 5.0| 4.789751482535894| | 440| 471| 3.0| 2.440344619907418| | 306| 471| 3.0|3.3247629567900976| | 19| 471| 3.0| 3.067333162723295| | 299| 471| 4.5| 5.218491499885204| | 537| 471| 5.0| 5.69083471617962| | 241| 471| 4.0| 3.816546176254299| | 23| 471| 3.5| 2.539020466532909| | 195| 471| 3.0| 3.355342979133588| | 487| 471| 4.0| 3.105186392315445| | 242| 471| 5.0| 5.893115933597325| | 30| 471| 4.0| 4.017221049024606| | 516| 1088| 3.0|3.3911144131643005| | 111| 1088| 3.5| 4.504826156475481| | 57| 1088| 4.0| 3.024549429857915| | 54| 1088| 5.0| 5.235519746597422| | 19| 1088| 3.0| 3.70884171609874| | 387| 1088| 4.0| 4.070842875474657| | 514| 1088| 3.0| 2.313176685047038| +------+-------+------+------------------+ only showing top 20 rows

```
0.6332304339145925
```

Remember that the RMSE is a rather subjective metric. Would you say that the RMSE in this case is sufficient to make meaningful recommendations?

Interpreting the RMSE

This model was able to achieve an RMSE of 0.633. Click on the best interpretation of what this means.

Answer the question 50 XP Possible Answers An RMSE of 0.633 means that the predictions are 6.33% off from the original values of the ratings matrix. press 1 An RMSE of 0.633 means that 6.33% of the time, the recommendations generated by our ALS model will be wrong. press 2 AN RMSE of 0.633 means that 6.33% of our users will receive recommendations that they won't take. press 3 An RMSE of 0.633 means that on average the model predicts 0.633 above or below values of the original ratings matrix. press 4

Answer 4. However, is it really 6.33% but not 63.3%?

Do Recommendations Make Sense

Now that we have an understanding of how well our model performed, and can take some confidence that it will provide recommendations that are relevant to users, let's actually look at recommendations made to a user and see if they make sense.

The original ratings data is provided here as original_ratings. Take a look at user 60 and user 63's original ratings, and compare them to what ALS recommended for him/her. In your opinion, are the recommendations consistent with his/her original preferences?

In [ ]:

```
# Look at user 60's ratings
print ("User 60's Ratings:")
original_ratings.filter(col("userId") == 60).sort("rating", ascending = False).sh

# Look at the movies recommended to user 60
print ("User 60s Recommendations:")
recommendations.filter(col("userId") == 60).show()

# Look at user 63's ratings
print ("User 63's Ratings:")
original_ratings.filter(col("userId") == 63).sort("rating", ascending = False).sh

# Look at the movies recommended to user 63
print ("User 63's Recommendations:")
recommendations.filter(col("userId") == 63).show()
```

```
User 60's Ratings:
+------+-------+------+--------------------+--------------------+
|userId|movieId|rating|               title|              genres|
+------+-------+------+--------------------+--------------------+
|    60|    858|     5|   GodfatherThe(1972)|         Crime|Drama|
|    60|    235|     5|        EdWood(1994)|        Comedy|Drama|
|    60|   1732|     5|BigLebowskiThe(1998)|        Comedy|Crime|
|    60|   2324|     5|LifeIsBeautiful(L...|Comedy|Drama|Roma...|
|    60|   3949|     5|RequiemforaDream(...|               Drama|
|    60|    541|     5|    BladeRunner(1982)|Action|Sci-Fi|Thr...|
|    60|   5995|     5|     PianistThe(2002)|           Drama|War|
|    60|   6350|     5|Laputa:Castleinth...|Action|Adventure|...|
|    60|   7361|     5|EternalSunshineof...|Drama|Romance|Sci-Fi|
|    60|   8638|     5|   BeforeSunset(2004)|       Drama|Romance|
|    60|   8981|     5|         Closer(2004)|       Drama|Romance|
|    60|  27803|     5|SeaInsideThe(Mara...|               Drama|
|    60|  30749|     5|    HotelRwanda(2004)|           Drama|War|
|    60|   5060|     5|M*A*S*H(a.k.a.MAS...|    Comedy|Drama|War|
|    60|   1221|     5|Godfather:PartIIT...|         Crime|Drama|
|    60|   5690|     5|GraveoftheFirefli...| Animation|Drama|War|
|    60|   1653|     5|        Gattaca(1997)|Drama|Sci-Fi|Thri...|
|    60|     16|   4.5|         Casino(1995)|         Crime|Drama|
|    60|    111|   4.5|     TaxiDriver(1976)|Crime|Drama|Thriller|
|    60|   1080|   4.5|MontyPython'sLife...|              Comedy|
+------+-------+------+--------------------+--------------------+
only showing top 20 rows

User 60s Recommendations:
+------+-------+----------+--------------------+--------------------+
|userId|movieId|prediction|               title|              genres|
+------+-------+----------+--------------------+--------------------+
|    60|  83318|  5.810963|        GoatThe(1921)|              Comedy|
|    60|  83411|  5.810963|           Cops(1922)|              Comedy|
```

```
|    60|  73344|  5.315315|ProphetA(UnProphÃ...|         Crime|Drama|
|    60|   3309| 5.2298656|    Dog'sLifeA(1918)|               Comedy|
|    60|   8609| 5.2298656|OurHospitality(1923)|               Comedy|
|    60|  72647| 5.2298656|   Zorn'sLemma(1970)|                Drama|
|    60|   5059| 5.2298656|LittleDieterNeeds...|          Documentary|
|    60|   8797| 5.2298656|      Salesman(1969)|          Documentary|
|    60|  25764| 5.2298656|  CameramanThe(1928)|Comedy|Drama|Romance|
|    60|   7074| 5.2298656|  NavigatorThe(1924)|               Comedy|
|    60|  31547| 5.2298656|LessonsofDarkness...|      Documentary|War|
|    60|   4405| 5.2298656|LastLaughThe(Letz...|                Drama|
|    60|  26400| 5.2298656| GatesofHeaven(1978)|          Documentary|
|    60|  80599| 5.2298656|BusterKeaton:AHar...|          Documentary|
|    60|  92494| 5.1418443|DylanMoran:Monste...|   Comedy|Documentary|
|    60|   3216| 5.1418443|VampyrosLesbos(Va...|Fantasy|Horror|Th...|
|    60|   6918| 5.1184077|UnvanquishedThe(A...|                Drama|
|    60|  40412| 5.0673676|DeadMan'sShoes(2004)|       Crime|Thriller|
|    60|  52767|  5.043912|          21Up(1977)|          Documentary|
|    60|   8955| 5.0317564|      Undertow(2004)|Crime|Drama|Thriller|
+------+-------+----------+--------------------+---------------------+

User 63's Ratings:
+------+-------+------+--------------------+--------------------+
|userId|movieId|rating|               title|              genres|
+------+-------+------+--------------------+--------------------+
|    63|      1|     5|      ToyStory(1995)|Adventure|Animati...|
|    63|     16|     5|        Casino(1995)|         Crime|Drama|
|    63|    260|     5|StarWars:EpisodeI...|Action|Adventure|...|
|    63|    318|     5|ShawshankRedempti...|         Crime|Drama|
|    63|    592|     5|        Batman(1989)|Action|Crime|Thri...|
|    63|   1193|     5|OneFlewOvertheCuc...|               Drama|
|    63|   1198|     5|RaidersoftheLostA...|    Action|Adventure|
|    63|   1214|     5|         Alien(1979)|       Horror|Sci-Fi|
|    63|   1221|     5|Godfather:PartIIT...|         Crime|Drama|
|    63|   1259|     5|     StandbyMe(1986)|     Adventure|Drama|
|    63|   1356|     5|StarTrek:FirstCon...|Action|Adventure|...|
|    63|   1639|     5|    ChasingAmy(1997)|Comedy|Drama|Romance|
|    63|   2797|     5|           Big(1988)|Comedy|Drama|Fant...|
|    63|   2858|     5|AmericanBeauty(1999)|       Drama|Romance|
|    63|   2918|     5|FerrisBueller'sDa...|              Comedy|
|    63|   3114|     5|     ToyStory2(1999)|Adventure|Animati...|
|    63|   3176|     5|TalentedMr.Ripley...|Drama|Mystery|Thr...|
|    63|   3481|     5|  HighFidelity(2000)|Comedy|Drama|Romance|
|    63|   3578|     5|     Gladiator(2000)|Action|Adventure|...|
|    63|   4306|     5|         Shrek(2001)|Adventure|Animati...|
+------+-------+------+--------------------+--------------------+
only showing top 20 rows

User 63's Recommendations:
+------+-------+----------+--------------------+---------------------+
```

```
|userId|movieId|prediction|               title|              genres|
+------+-------+----------+--------------------+--------------------+
|    63|  92210| 4.8674645|DisappearanceofHa...|Adventure|Animati...|
|    63| 110873| 4.8674645|CentenarianWhoCli...|Adventure|Comedy|...|
|    63|   9010| 4.8588977|LoveMeIfYouDare(J...|       Drama|Romance|
|    63| 108583|  4.836118|FawltyTowers(1975...|              Comedy|
|    63|   8530| 4.8189244|   DearFrankie(2004)|       Drama|Romance|
|    63|  83318|  4.813581|       GoatThe(1921)|              Comedy|
|    63|  83411|  4.813581|          Cops(1922)|              Comedy|
|    63|  65037| 4.7906556|          BenX(2007)|               Drama|
|    63|  54328|  4.688013|MyBestFriend(Monm...|              Comedy|
|    63|   3437|  4.678849|     CoolasIce(1991)|               Drama|
|    63|   2924|  4.675808|DrunkenMaster(Jui...|       Action|Comedy|
|    63|   1196| 4.6633716|StarWars:EpisodeV...|Action|Adventure|...|
|    63|  27156| 4.6382804|NeonGenesisEvange...|Action|Animation|...|
|    63|  26865| 4.6308517|FistofLegend(Jing...|        Action|Drama|
|    63|   5244| 4.6302986|ShogunAssassin(1980)|    Action|Adventure|
|    63|  93320| 4.6302986|TrailerParkBoys(1...|        Comedy|Crime|
|    63|  50641| 4.6302986|  House(Hausu)(1977)|Comedy|Fantasy|Ho...|
|    63|   6598|  4.624031|StepIntoLiquid(2002)|         Documentary|
|    63|   7502| 4.6232696|BandofBrothers(2001)|    Action|Drama|War|
|    63|  73344|  4.609774|ProphetA(UnProphÃ...|         Crime|Drama|
+------+-------+----------+--------------------+--------------------+
```

Does it look like the model picked up on user 60's preference for drama, crime and comedy or user 63's preference for action, adventure and drama?

What if you don't have customer ratings?

In most real-life situations, you won't not have "perfect" customer data available to build an ALS model. This chapter will teach you how to use your customer behavior data to "infer" customer ratings and use those inferred ratings to build an ALS recommendation engine. Using the Million Songs Dataset as well as another version of the MovieLens dataset, this chapter will show you how to use the data available to you to build a recommendation engine using ALS and evaluate it's performance.

Confirm understanding of implicit rating concepts

What is the difference between "implicit" ratings and "explicit" ratings?

Answer the question 50 XP Possible Answers Users agree to give explicit ratings data and do not agree to give implicit ratings data. press 1 Explicit ratings are values that users have given to explicitly rate their preferences. Implicit ratings are "implied" from user behavior. press 2 Implicit ratings can't be used for recommendation engines whereas explicit ratings can. press 3 Implicit ratings don't have latent features whereas explicit ratings do. press

Answer 2: Explicit ratings are true ratings that users have explicitly provided. Implicit ratings are data that are used to estimate user preference.

MSD summary statistics

Let's get familiar with the Million Songs Echo Nest Taste Profile data subset. For purposes of this course, we'll just call it the Million Songs dataset or msd. Let's get the number of users and the number of songs. Let's also see which songs have the most plays from this subset.

In [ ]:
```
# Look at the data
msd.show()

# Count the number of distinct userId's
user_count = msd.select("userId").distinct().count()
print("Number of users: ", user_count)

# Count the number of distinct songId's
song_count = msd.select("songId").distinct().count()
print("Number of songs: ", song_count)
```

+------+------+---------+ |userId|songId|num_plays| +------+------+---------+ | 148| 148| 0| | 243| 496| 0| | 31| 471| 0| | 137| 463| 0| | 251| 623| 0| | 85| 392| 0| | 65| 540| 0| | 255| 243| 0| | 53| 516| 0| | 133| 31| 0| | 296| 85| 0| | 78| 451| 0| | 108| 580| 0| | 155| 137| 0| | 193| 251| 0| | 211| 65| 0| | 34| 458| 0| | 115| 53| 0| | 126| 255| 0| | 101| 588| 0| +------+------+---------+ only showing top 20 rows

```
Number of users:  321
Number of songs:  729
```

Grouped summary statistics

In this exercise, we are going to combine the .groupBy() and .filter() methods that you've used previously to calculate the min and avg number of users that have rated each song, and the minand avg number of songs that each user has rated. Because our data now includes 0's for items not yet consumed, we'll need to .filter() them out when doing grouped summary statistics like this. The msd dataset is provided for you here.

In [ ]:
```
# Min num implicit ratings for a song
print("Fewest implicit ratings for a song: ")
msd.filter(col("num_plays") > 0).groupBy("songId").count().select(min("count")).s

# Avg num implicit ratings per songs
print("Avg num implicit ratings per song: ")
msd.filter(col("num_plays") > 0).groupBy("songId").count().select(avg("count")).s

# Min num implicit ratings from a user
print("Fewest implicit ratings from a user: ")
msd.filter(col("num_plays") > 0).groupBy("userId").count().select(min("count")).s

# Avg num implicit ratings from users
print("Avg num implicit ratings per user: ")
msd.filter(col("num_plays") > 0).groupBy("userId").count().select(avg("count")).s
```

```
Fewest implicit ratings for a song:
```

```
+----------+
|min(count)|
+----------+
|         3|
+----------+

Avg num implicit ratings per song:
+------------------+
|        avg(count)|
+------------------+
|35.251063829787235|
+------------------+

Fewest implicit ratings from a user:
+----------+
|min(count)|
+----------+
|        21|
+----------+

Avg num implicit ratings per user:
+-----------------+
|       avg(count)|
+-----------------+
|77.42056074766356|
+-----------------+
```

Users have at least 21 implicit ratings with an average of 77 and each song has at least 3 implicit ratings with an average of 35.

Add Zeros

Many recommendation engines use implicit ratings. In many cases these datasets don't include behavior counts for items that a user has never purchased. In these cases, you'll need to add them and include zeros. The dataframe Z is provided for you. It contains userId's, productId's and num_purchases which is the number of times a user has purchased a specific product.

In [ ]:

```
# View the data
Z.show()

# Extract distinct userIds
users = Z.select("userId").distinct()
products = Z.select("productId").distinct()

# Cross join users and products
cj = users.crossJoin(products)

# Join cross_join and Z
Z_expanded = cj.join(Z, ["userId", "productId"], "left").fillna(0)

# View Z_expanded
Z_expanded.show()
```

```
+------+---------+-------------+
|userId|productId|num_purchases|
+------+---------+-------------+
|  2112|      777|            1|
|     7|       44|           23|
|  1132|      227|            9|
|   686|     1981|            2|
|    42|     2390|            5|
|    13|     1662|           21|
|  2112|     1492|            8|
|    22|     1811|           96|
+------+---------+-------------+

+------+---------+-------------+
|userId|productId|num_purchases|
+------+---------+-------------+
|    22|       44|            0|
|    22|      777|            0|
|    22|     1811|           96|
|    22|      227|            0|
|    22|     1662|            0|
|    22|     1492|            0|
|    22|     2390|            0|
|    22|     1981|            0|
|   686|       44|            0|
|   686|      777|            0|
|   686|     1811|            0|
|   686|      227|            0|
|   686|     1662|            0|
|   686|     1492|            0|
|   686|     2390|            0|
|   686|     1981|            2|
|    13|       44|            0|
```

```
|    13|     777|           0|
|    13|    1811|           0|
|    13|     227|           0|
+------+--------+------------+
only showing top 20 rows
```

Notice how the dataset expanded significantly as a result of adding zeros.

Specify ALS Hyperparameters

You're now going to build your first implicit rating recommendation engine using ALS. To do this, you will first tell Spark what values you want it to try when finding the best model. Four empty lists are provided below. You will fill them with specific values that Spark can use to build several different ALS models. In the next exercise, you'll tell Spark to build out these models using the lists below.

In [ ]:

```
# Complete the lists below
ranks = [10, 20, 30, 40]
maxIters = [10, 20, 30, 40]
regParams = [.05, .1, .15]
alphas = [20, 40, 60, 80]
```

Build Implicit Models

Now that you have all of your hyperparameter values specified, let's have Spark build enough models to test each combination. To facilitate this, a for loop is provided here. Follow the instructions below to automatically create these ALS models. In subsequent exercises you will run these models on test datasets to see which one performs the best. The ALS algorithm is already imported for you.

In [ ]:

```
# For loop will automatically create and store ALS models
for r in ranks:
    for mi in maxIters:
        for rp in regParams:
            for a in alphas:
                model_list.append(ALS(userCol= "userId", itemCol= "songId", ratin

# Print the model list, and the length of model_list
print (model_list, "Length of model_list: ", len(model_list))

# Validate
len(model_list) == (len(ranks)*len(maxIters)*len(regParams)*len(alphas))
```

[ALS_411487e4bfabac19086b, ALS_4d4880a73bce1741d61c, ALS_4dc7b9393eb2b8166887, ALS_445595f3fe0cdcfebc01, ALS_44548b813a4b453c05a7, ALS_44ba87f5c58a66e75405, ALS_41b8afdce360a3bc11fa, ALS_4471a747fd5f0a750913, ALS_46038f2314bfd8c6bd82, ALS_43f18dbbe6a1fa1ac82d, ALS_424ebbd3f1723f388a4a, ALS_455691ed821df646dc03, ALS_4d5b93e10f1c4799504f, ALS_4b379e2ba8db5506949b, ALS_4e8880ccbf93fa110733, ALS_43e0aac82c7eaedb8dd3, ALS_49f7835cb48254cd0bee, ALS_477d80d02e294e051c82, ALS_4346a303f4c2d5831615, ALS_475d9888c46ac397847c, ALS_41c5b0dd8103d0509763, ALS_44c8b4c137bcf9705c36, ALS_4f91b9e0853ae2a33da9, ALS_4f438159cf98293fa67a, ALS_4991bc7b453153d8191e, ALS_44238fa0758a9cd2111c, ALS_4b4aba9623a95f32551a, ALS_4821a3cf3ee0ce23cd68, ALS_45b58e2a389736d82e7a, ALS_445da022b908f1064c56, ALS_4a5798eea6f6a373b36e, ALS_45e59dc3723b773f28bf, ALS_4b848bfccdd7ac26304d, ALS_49aea3e50107f480780d, ALS_4f81996baba51b00a13d, ALS_430cadd3d33c1697d922, ALS_44909be672b8b0dcee7f, ALS_4838ad8165ac30a65970, ALS_42a682b9fc879dfe4dc3, ALS_45eea16f00ca182d1248, ALS_4baf81d845d8a45d22ab, ALS_44d6aab0d5441bd915c5, ALS_4bad8038c7545edf71e6, ALS_44d39ba7ce20038d965b, ALS_4467b58ba6b145e0c00a, ALS_465dbb6a9ea115175a6b, ALS_4c608094de4d5dc293a5, ALS_4761aabbcea301d4b681, ALS_41c59d6daf6422d7ad45, ALS_491b9a95a3cea9c3ae19, ALS_496da3a3aa69ab6f274f, ALS_424e908d17e624a929ba, ALS_425287e9fb93f99a1b91, ALS_45a3bb7fc49b610a95f0, ALS_4296b99d8c633dc4052e, ALS_44999a7e352a2f1c170b, ALS_44d6b6e22e4cdf7e76ec, ALS_495a983bb28d382540d3, ALS_4cc7845aae8daad32db0, ALS_4e22a170ea1f47672545, ALS_449cbb09d686af82be55, ALS_41ec9cb8addaa2d71380, ALS_45d59272d9d51cb63387, ALS_42f395ebb6d0bfac8394, ALS_4972a4401760323d9e6c, ALS_42beacbed5a90507d7a9, ALS_481fa8479192d4b717dd, ALS_4dd88151a358cb364412, ALS_453cb8f624c4702a6e7c, ALS_4061a61a82ea1261c305, ALS_4efd814388207bd7661b, ALS_4f189cc300f1445bfc21, ALS_48e4a039597dd1a9d68d, ALS_405d9de4a2651a8670b8, ALS_49949d907dd05f3925da, ALS_44e3bd7778520f19606f, ALS_4e5398a78cf6ba6445ac, ALS_4b9e8a812f031beb6146, ALS_4368910b9418ab340370, ALS_479c920dbede9b4cb0d0, ALS_48adbe5ac48b22ab7804, ALS_449aa7312555a8fc6027, ALS_4752a16aec370025d44f, ALS_421297fdeb89147adc4c, ALS_4f60aaa6f846278435de, ALS_45fbb4f5ec1983ac4214, ALS_42979c3560b351f2a3bb, ALS_41b599914a6a2e67518d, ALS_4c05bdb9823ccf06d17d, ALS_4b75bcbc1c421ab0fc56, ALS_406bb5a95a3be31ab2c8, ALS_4d23a940c2d2aa99b06f, ALS_45d4b892bfb5dc60d034, ALS_40068c5f0aebd57b99d0, ALS_4cb688881b30f8e290c5, ALS_4cb59a24ccb89e29693a, ALS_47eea3d82747aa1b1a53, ALS_491cabf3aeeba4461cc4, ALS_421faa360839f3303545, ALS_443ba44574b77c30b4f6, ALS_42f7beb15c90d4d0e5f9, ALS_44bdac2905fb49e50251, ALS_4d66b016d3a245da818f, ALS_4a0fb86db24eff1f5a99, ALS_4ab19da46ef80c55f061, ALS_4ffdb5e10245fe2506fe, ALS_46888145e6b0181b30f9, ALS_4f418c8e227757d62db9, ALS_47a78bcdda7e79068aec, ALS_449fb95657918a1104d4, ALS_4259bda263bbca0c471e, ALS_4141b9698e195474ab45, ALS_4daba596de5671573f2a, ALS_47858e9e9d52e6eb7ad3, ALS_4e22bced2b850bd8d972, ALS_4a13ae89565bef700c50, ALS_4bb482b0a93a1661398c, ALS_4031941f184068f6a6ca, ALS_4de998992384f1343620, ALS_4985a4f0b4f5c6e73868, ALS_4812ac51e0181cf92ad5, ALS_49919a0be76d7bcc56e0, ALS_42eda30a018bd1468d11, ALS_46668cc26defa1b38d03, ALS_48c18581d736aae6cde9, ALS_4c48b544fab9239fe543, ALS_411795bd654c1faf8f9c, ALS_457f94216dc71243bf92, ALS_4e36b86e3e0c5bcf04b8, ALS_4f8ab2c15fedaed72a28, ALS_415bacd2d346ac141726, ALS_43b79c9347f029b341cd, ALS_4f86930fc65dbff6e769, ALS_44bbbe1c4a9b6432e09e, ALS_4d4784006ae7971f6489, ALS_4e6787168f3062bef9c3, ALS_42e0b163cad52631dc41, ALS_4359bce955a6531186a9, ALS_42c4ac8e32b3a117ba43, ALS_42108c906f9f30748a18, ALS_477f841ebddd7e187645, ALS_4df98187f8c16b7c0539, ALS_446b8172cd350504223d, ALS_4e6ebf8e9fd235406927, ALS_4725a77f715ba747daa2, ALS_4ee182bd2d0af93be0ea, ALS_4ef1ab50989eaba19842, ALS_4e3fbb961450f3a9d7cd, ALS_4b08ab5829306ceac4b4, ALS_438e8dc736deae1b731f, ALS_445b93050497db684b3d, ALS_4ebc83111171e58a6f92, ALS_4839a8530e53341ec598, ALS_4f3390c9d69b073c6f36, ALS_4fed9fe9d2d7fea96d70, ALS_46d281a9f5bcdb8aebfd, ALS_43abad7d8d04e47f2979, ALS_4419841fc32142457e20, ALS_466b8798b135ea1379bc, ALS_4e2fa02640fef06a96c3, ALS_42ec99f032cfb6f1c53c, ALS_4be8aac415409c03f872, ALS_43649a8fabf92cbc707c, ALS_42fea0f2a94062e26fee, ALS_40f1b120f1631bfd001f, ALS_455d99fc91fe7c492abc, ALS_48e3956f40c07e399ca6, ALS_4fabbcee907b02e2c3b5, ALS_44488bfac704c7b0e33a, ALS_419c88692eda8a3a0da3, ALS_4d5fbff164921f9de392, ALS_484d8969d09baf87154c, ALS_4881809a857c4c9f1ebd, ALS_497b865c60d42c7b68e6, ALS_4f53a8068bd7e2eaaa3f, ALS_451db3f33ff74f97c611, ALS_4d23b1e068da25b93d42, ALS_4ee78f418938440645ae, ALS_487c8a18f8f5beb2a333, ALS_4839832e86a9b0f29b47, ALS_49c7ad53a473a80f813e, ALS_40f1acb29000afb4af25, ALS_4fd591c1eb916300d67a, ALS_4ad1be5c9a78ef009680, ALS_4a39b49b18d25d668eac, ALS_468f96081cbc465e80bc, ALS_46c091e3c3e217a5f5eb, ALS_415c8f5f45ddb0ea4fc7, ALS_4e14bd7999cd6bf5ccf0, ALS_40d0967c5a5146940fea, ALS_49e090f74b5793f46fdf, ALS_4cb2b9907eb6bde3df26] Length of model_list: 192

You've built your first 192 implicit rating models. Now let's see how they performed.

Running a Cross-Validated Implicit ALS Model

Now that we have several ALS models, each with a different set of hyperparameter values, we can train them on a training portion of the msd dataset using cross validation, and then run them on a test set of data and evaluate how well each one performs using the ROEM function discussed earlier. Unfortunately this takes too much time for this exercise, so it has been done separately. But for your reference you can evaluate your model_list using the following loop (we are using the msd dataset in this case):

#Split the data into training and test sets (training, test) = msd.randomSplit([0.8, 0.2])

#Building 5 folds within the training set. train1, train2, train3, train4, train5 = training.randomSplit([0.2, 0.2, 0.2, 0.2, 0.2], seed = 1) fold1 = train2.union(train3).union(train4).union(train5) fold2 = train3.union(train4).union(train5).union(train1) fold3 = train4.union(train5).union(train1).union(train2) fold4 = train5.union(train1).union(train2).union(train3) fold5 = train1.union(train2).union(train3).union(train4)

foldlist = [(fold1, train1), (fold2, train2), (fold3, train3), (fold4, train4), (fold5, train5)]

#Empty list to fill with ROEMs from each model ROEMS = []

#Loops through all models and all folds for model in model_list: for ft_pair in foldlist:

```
        # Fits model to fold within training data
        fitted_model = model.fit(ft_pair[0])

        # Generates predictions using fitted_model on respective CV test data
        predictions = fitted_model.transform(ft_pair[1])

        # Generates and prints a ROEM metric CV test data
        r = ROEM(predictions)
        print ("ROEM: ", r)

    #Fits model to all of training data and generates preds for test data
    v_fitted_model = model.fit(training)
    v_predictions = v_fitted_model.transform(test)
    v_ROEM = ROEM(v_predictions)

    #Adds validation ROEM to ROEM list
    ROEMS.append(v_ROEM)
    print ("Validation ROEM: ", v_ROEM)
```

For purposes of walking you through the steps, the test predictions for 192 models have already been generated, and their ROEM has been calculated. They are found in the ROEMS list provided. Because a list isn't unique to Pyspark, and because numpy works really well with lists, we're going to use numpy here. Follow the instructions below to find the best ROEM and the model that provided it.

In [ ]:
```
# Import numpy
import numpy

# Find the index of the smallest ROEM
i = numpy.argmin(ROEMS)
print ("Index of smallest ROEM:", i)

# Find ith element of ROEMS
print ("Smallest ROEM: ", ROEMS[i])
```

Index of smallest ROEM: 38 Smallest ROEM: 0.01980198019801982 Looks like Model 38 has the lowest ROEM of 0.019. Next we'll extract the hyperparameters.

Extracting Parameters

You've now tested 192 different models on the msd dataset, and you found the best ROEM and it's respective model (model 38). You now need to exctract the hyperparameters. The model_list you created previously is provided here. It contains all 192 models you generated. Use the instructions below to extract the hyperparameters.

In [ ]:
```
# Extract the best_model
best_model = model_list[38]

# Extract the Rank
print ("Rank: ", best_model.getRank())

# Extract the MaxIter value
print ("MaxIter: ", best_model.getMaxIter())

# Extract the RegParam value
print ("RegParam: ", best_model.getRegParam())

# Extract the Alpha value
print ("Alpha: ", best_model.getAlpha())
```

```
Rank:  10
MaxIter:  40
RegParam:  0.05
Alpha:  60.0
```

Looks like a low rank, a higher maxIter, a low regParam and a medium-high alpha is keeping the ROEM low. Because some of these values are on the high and low ends of the values we tried, it would be worth adding some additional values to test in our hyperparameter values, and doing this step again, but for right now, you should understand the process.

Binary Model Performance

You've already built several ALS models by now, so we won't do that again. An implicit ALS model has already been fitted to the binary ratings of the MovieLens dataset. Let's look at the binary_test_predictions from this model to see what we can learn.

The ROEM function has been defined for you. Feel free to run help(ROEM) in the console if you want more details on how to execute it!

In [ ]:
```
# Import the col function
from pyspark.sql.functions import col

# Look at the test predictions
binary_test_predictions.show()

# Evaluate ROEM on test predictions
ROEM(binary_test_predictions)

# Look at user 42's test predictions
binary_test_predictions.filter(col("userId") == 42).show()
```

```
+------+-------+------+-----------+
|userId|movieId|viewed| prediction|
+------+-------+------+-----------+
```

```
|    91|    148|     0|        0.0|
|   601|    148|     0|        0.0|
|   545|    148|     0|0.060729448|
|   505|    148|     0|  0.2972868|
|   526|    148|     0|        0.0|
|   478|    148|     0|        0.0|
|   106|    148|     0|        0.0|
|   135|    148|     0|        0.0|
|    78|    463|     0|0.050237626|
|   259|    463|     0|0.027345931|
|   127|    463|     0|0.016793307|
|   502|    463|     0|0.019936942|
|   441|    463|     0|0.002274946|
|   664|    463|     0| 0.15109323|
|   418|    463|     0|0.011756073|
|   390|    463|     0|  0.3210355|
|    32|    463|     0|0.018041197|
|    58|    463|     0| 0.24846576|
|   311|    463|     1|  1.0126673|
|   471|    471|     0|  0.7693843|
+------+-------+------+-----------+
only showing top 20 rows

ROEM: 0.07436376290899886
+------+-------+------+-----------+
|userId|movieId|viewed| prediction|
+------+-------+------+-----------+
|    42|    858|     0|  0.9915983|
|    42|   3703|     0|  0.5134803|
|    42|   2606|     0|        0.0|
|    42|   6213|     0|0.008813023|
|    42|   2342|     0|        0.0|
|    42|  58107|     0|0.033887785|
|    42|   6953|     0| 0.29286233|
|    42|  41716|     0|        0.0|
|    42|  49394|     0|0.021620607|
|    42|   6509|     0|        0.0|
|    42|   3512|     0|        0.0|
|    42|   6810|     0|        0.0|
|    42|  30749|     0| 0.60764235|
|    42|  74282|     0|0.042640854|
|    42|   2255|     0|        0.0|
|    42|   3891|     0|        0.0|
|    42|  31116|     0|        0.0|
|    42|   2013|     0|  0.4043246|
|    42|   3390|     0|        0.0|
|    42|   1488|     0|        0.0|
+------+-------+------+-----------+
only showing top 20 rows
```

he model has a pretty low ROEM. Did you notice that the model predicted some high numbers for unseen movies? This indicates that the model is creating recommendations from the movies that users have not seen.

Recommendations From Binary Data

So you see from the ROEM, these models can still generate meaningful test predictions. Let's look at the actual recommendations now. The col function from the pyspark.sql.functions class has been imported for you.

In [ ]:
```
# View user 26's original ratings
print ("User 26 Original Ratings:")
original_ratings.filter(col("userId") == 26).show()

# View user 26's recommendations
print ("User 26 Recommendations:")
binary_recs.filter(col("userId") == 26).show()

# View user 99's original ratings
print ("User 99 Original Ratings:")
original_ratings.filter(col("userId") == 99).show()

# View user 99's recommendations
print ("User 99 Recommendations:")
binary_recs.filter(col("userId") == 99).show()
```

User 26 Original Ratings: +------+-------+------+--------------------+--------------------+ |userId|movieId|rating| title| genres| +------+-------+------+--------------------+--------------------+ | 26| 1| 5| ToyStory(1995)|Adventure|Animati...| | 26| 2542| 5|LockStock&TwoSmok...|Comedy|Crime|Thri...| | 26| 2571| 5| MatrixThe(1999)|Action|Sci-Fi|Thr...| | 26| 4011| 5| Snatch(2000)|Comedy|Crime|Thri...| | 26| 6016| 5|CityofGod(Cidaded...|Action|Adventure|...| | 26| 8798| 5| Collateral(2004)|Action|Crime|Dram...| | 26| 27831| 5| LayerCake(2004)|Crime|Drama|Thriller| | 26| 44191| 5| VforVendetta(2006)|Action|Sci-Fi|Thr...| | 26| 44195| 5|ThankYouforSmokin...| Comedy|Drama| | 26| 50872| 5| Ratatouille(2007)|Animation|Childre...| | 26| 54286| 5|BourneUltimatumTh...|Action|Crime|Thri...| | 26| 58559| 5| DarkKnightThe(2008)|Action|Crime|Dram...| | 26| 59369| 5| Taken(2008)|Action|Crime|Dram...| | 26| 59784| 5| KungFuPanda(2008)|Action|Animation|...| | 26| 60684| 5| Watchmen(2009)|Action|Drama|Myst...| | 26| 65514| 5| IpMan(2008)| Action|Drama|War| | 26| 32| 4|TwelveMonkeys(a.k...|Mystery|Sci-Fi|Th...| | 26| 47| 4|Seven(a.k.a.Se7en...| Mystery|Thriller| | 26| 50| 4|UsualSuspectsThe(...|Crime|Mystery|Thr...| | 26| 69| 4| Friday(1995)| Comedy| +------+-------+------+--------------------+--------------------+ only showing top 20 rows

```
User 26 Recommendations:
+------+-------+----------+--------------------+--------------------+
|userId|movieId|prediction|               title|              genres|
+------+-------+----------+--------------------+--------------------+
|    26|  30707| 1.1401137|Million Dollar Ba...|               Drama|
|    26|    293| 1.1154407|Léon: The Profess...|Action|Crime|Dram...|
|    26|    111| 1.0985317|  Taxi Driver (1976)|Crime|Drama|Thriller|
|    26|  81845| 1.0974996|King's Speech, Th...|               Drama|
|    26|   5971| 1.0956558|My Neighbor Totor...|Animation|Childre...|
|    26|  70286| 1.0950022|   District 9 (2009)|Mystery|Sci-Fi|Th...|
|    26|  48394| 1.0917767|Pan's Labyrinth (...|Drama|Fantasy|Thr...|
|    26|  46578| 1.0879191|Little Miss Sunsh...|Adventure|Comedy|...|
+------+-------+----------+--------------------+--------------------+

User 99 Original Ratings:
+------+-------+------+--------------------+--------------------+
|userId|movieId|rating|               title|              genres|
+------+-------+------+--------------------+--------------------+
|    99|    318|     5|ShawshankRedempti...|         Crime|Drama|
|    99|    356|     5|   ForrestGump(1994)|Comedy|Drama|Roma...|
|    99|    357|     5|FourWeddingsandaF...|      Comedy|Romance|
|    99|    509|     5|      PianoThe(1993)|       Drama|Romance|
|    99|    595|     5|BeautyandtheBeast...|Animation|Childre...|
|    99|    608|     5|         Fargo(1996)|Comedy|Crime|Dram...|
|    99|    720|     5|Wallace&Gromit:Th...|Adventure|Animati...|
|    99|    903|     5|       Vertigo(1958)|Drama|Mystery|Rom...|
|    99|    912|     5|    Casablanca(1942)|       Drama|Romance|
|    99|    918|     5|MeetMeinSt.Louis(...|             Musical|
|    99|    953|     5|It'saWonderfulLif...|Children|Drama|Fa...|
|    99|    969|     5|AfricanQueenThe(1...|Adventure|Comedy|...|
|    99|   1028|     5|   MaryPoppins(1964)|Children|Comedy|F...|
|    99|   1204|     5|LawrenceofArabia(...| Adventure|Drama|War|
|    99|   1233|     5|BootDas(BoatThe)(...|    Action|Drama|War|
|    99|   1304|     5|ButchCassidyandth...|      Action|Western|
|    99|   1617|     5|L.A.Confidential(...|Crime|Film-Noir|M...|
|    99|   1619|     5|SevenYearsinTibet...| Adventure|Drama|War|
|    99|   1721|     5|       Titanic(1997)|       Drama|Romance|
|    99|   2067|     5| DoctorZhivago(1965)|   Drama|Romance|War|
+------+-------+------+--------------------+--------------------+
only showing top 20 rows

User 99 Recommendations:
+------+-------+----------------+--------------------+----------------
---+
|userId|movieId|      prediction|               title|             gen
res|
+------+-------+----------------+--------------------+----------------
---+
|    99|   3148|       1.1828514|Cider House Rules...|              Dr
```

```
ama|
|    99|    111|1.1700658000000002|  Taxi Driver (1976)|Crime|Drama|Thril
ler|
|    99|    920|         1.1438243|Gone with the Win...|   Drama|Romance|
War|
|    99|    265|         1.1368732|Like Water for Ch...|Drama|Fantasy|Ro
m...|
|    99|   1959|         1.1332427|Out of Africa (1985)|       Drama|Roma
nce|
|    99|   1188|         1.1314342|Strictly Ballroom...|      Comedy|Roma
nce|
|    99|     34|         1.1144139|         Babe (1995)|      Children|Dr
ama|
|    99|    910|         1.1108267|Some Like It Hot ...|        Comedy|Cr
ime|
|    99|    589|          1.096542|Terminator 2: Jud...|       Action|Sci
-Fi|
|    99|   1225|1.0958363000000002|      Amadeus (1984)|             Dr
ama|
+------+-------+------------------+--------------------+-----------------
---+
```

ALS seems to have picked up on the fact that user 26 likes thrillers, crime movies, action and adventure, and that user 99 likes dramas and romances. Do these look like good recommendations to you?