Programming for Big Data - Spark Lab 3

1. Introduction

Please use your databricks for this lab. We will build a recommender and then explore a dataset with Spark Sql.
https://community.cloud.databricks.com/

2. Building a Recommender using Matrix Factorisation

We will use the MovieLens dataset to build a movie-based recommender using collaborative filtering.

The data is available here: ceadar.dit.ie/bojan.bozic/SPARK/data/ml-100k.zip and unzip the files onto you desktop. Then complete the following steps:

1. Start a new cluster called MovieRecommender.
2. Click the tables tab and import the u.data from the folder.
    a. Save the file string name
3. Create a new notebook called MovieRecommenderNotebook

2.2. Extract the features

```
val rawData = sc.textFile("FILE_FROM_DB/u.data")
rawData.first()

val rawRatings = rawData.map(_.split("\t").take(3))
rawRatings.first()
```

Now load ALS Class and the Rating Class into your environment.

```
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.Rating
```

The ALS class is an implementation of Alternating Least Squares - an approach to predicting recommendations using user-item matrices. The rating class is required for this implementation to specify the rating for each movie by each user. We map the rawRatings and build a ratings collection using a Scala class.

```
val ratings = rawRatings.map { case Array(user, movie, rating) => Rating(user.toInt,
movie.toInt, rating.toDouble) }
ratings.first()
```

2.3. Training the recommendation model

ALS takes the following set of parameters:

rank: This refers to the number of factors in our ALS model, that is, the number of hidden features in our low-rank approximation matrices. Generally, the greater the number of factors, the better, but this has a direct impact on memory usage, both for computation and to store models for serving, particularly for large number of users or items. Hence, this is often a trade-off in real-world use cases. A rank in the range of 10 to 200 is usually reasonable.

iterations: This refers to the number of iterations to run. While each iteration in ALS is guaranteed to decrease the reconstruction error of the ratings matrix, ALS models will converge to a reasonably good solution after relatively few iterations. So, we don't need to run for too many iterations in most cases (around 10 is often a good default).

lambda: This parameter controls the regularization of our model. Thus, lambda controls over tting. The higher the value of lambda, the more is the regularization applied. What constitutes a sensible value is very dependent on the size, nature, and sparsity of the underlying data, and as with almost all machine learning models, the regularization parameter is something that should be tuned using out-of-sample test data and cross-validation approaches.

We'll use rank of 50, 10 iterations, and a lambda parameter of 0.01 to illustrate how to train our model:

```
val model = ALS.train(ratings, 50, 10, 0.01)
```

This returns a MatrixFactorizationModel object, which contains the user and item factors in the form of an RDD of (id, factor) pairs. These are called userFeatures and productFeatures, respectively. For example:

```
model.userFeatures.count
model.productFeatures.count
```

As expected, we have a factor array for each user (943 factors) and movie (1682 factors). This is a model built on explicit feedback. To create a model on implicit feedback you must use the trainImplicit method.

2.4. Using the model - User Recommendations

This model enables us to make predictions of a user's ratings for a particular movie we can:

```
val predictedRating = model.predict(789, 123)
```

Which returns the following:

```
predictedRating: Double = 3.7620191400934666
```

This means that the user 789 will give the movie 123 a rating of 3.7.

```
val userId = 789
val K = 10
val topKRecs = model.recommendProducts(userId, K)
println(topKRecs.mkString("\n"))
```

QUESTION:

What are the top 2 recommendations for the following users:

User 269:

User 286:

User 257:

User 117:

User 268:

3. Exploring a dataset using Spark SQL

Download the following dataset and import it into a table in the databricks platform.

ceadar.dit.ie/bojan.bozic/SPARK/data/students.csv

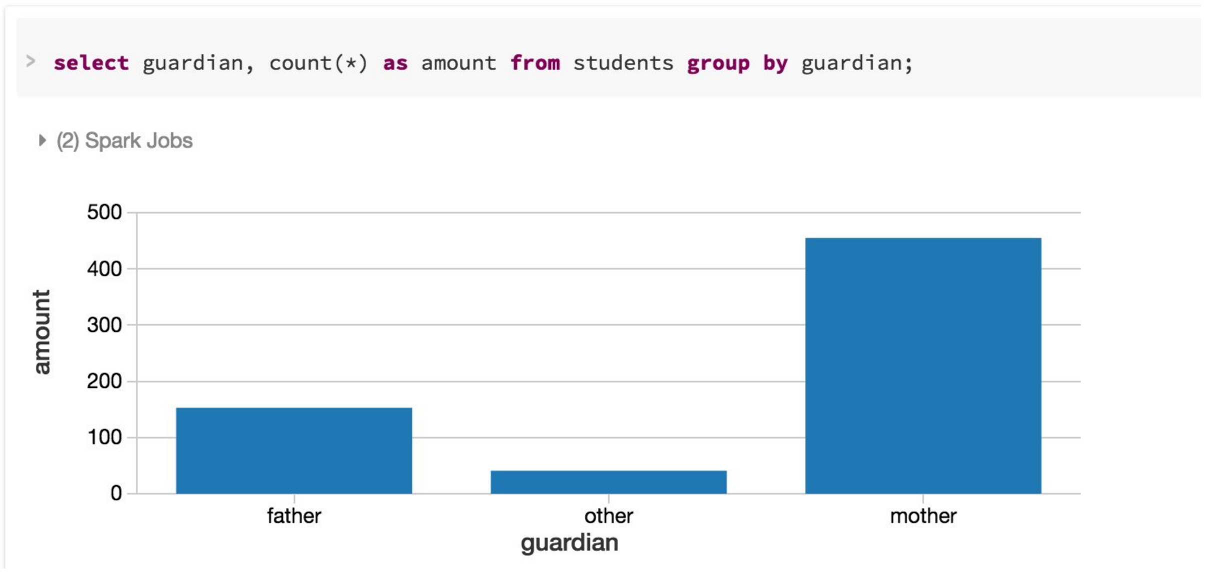
This is a dataset of topics is taken from The UCI Machine Learning Repository. We will explore this dataset and then answer a number of questions. Import the data and create a table in the “Data” menu (select the file, click “create table with UI” and follow the process).

Create a new notebook called SQLNotebook. Specify SQL as the language.

To run a query, type the query and execute it, you can see the results as either a table or graph. For example, to see the split on guardians we can type:

Select guardian, count(*) as amount from students group by guardian;

Which gives us:



```
> select guardian, count(*) as amount from students group by guardian;
```

More here:

https://docs.databricks.com/user-guide/visualizations/charts-and-graphs-scala.html

Please provide a visualisation as an answer for each question:

Question 1: What is the difference in study time between students that have guardian’s that are their mother versus their father?

Question 2: Is there a relationship between guardian and failure?

Question 3: How is failures distributed between male and female (including age as a variable of the graph)

Question 4: Do parents jobs have an impact on how absent a student is from school?

Question 5: Is there a relationship between failure and activities?

Question 6: What is average ages of both male and female students?

Question 7: How does having the internet impact the final grade on average?

4. Accessing data from a different notebook.

You can access the table from another notebook - using scala or python for example. Using the Scala movie recommender notebook and the sqlContext, type the following:

```
val df = sqlContext.sql("SELECT * from students")
df.first
```

This means you can chop and change approaches, illustrating the flexibility of spark.

5. Summary

In this lab we have looked into building a movie recommender and used Spark SQL to analyse a dataset.