Algoritmos para Big Data

2022/23

Recommender Systems

This lecture is about recommender systems (or recommendation systems). In the meantime, we highlight the usefulness of Spark SQL, particularly when it relates to persistent tables.

Spark SQL

As mentioned in the initial lectures, Spark SQL is a Spark module for structured data processing. It works alongside the APIs of DataFrame and Dataset and it is responsible for performing extra optimizations. We can also execute SQL queries and reading data from various files formats an Hive tables. (Apache Hive can manage large datasets residing in distributed storage using SQL)

Further details can be found in https://spark.apache.org/docs/latest/sql-programming-guide.html and https://spark.apache.org/docs/latest/api/sql/index.html

We can check the reference guide for Structured Query Language (SQL) which includes syntax, semantics, keywords, and examples for common SQL usage.

Problem formulation

This exercise aims to build a recommender system of books, with focus on the recommendation model itself. The functional requirements for the Spark program we want to create are as follows:

1. To load the dataset and perform Exploratory Data Analysis (EDA), then store the information properly cleaned, including as SQL tables.

2. To create a recommendation model supported by the ALS algorithm provided by Spark MLlib.
3. To pre-compute recommendations and store them in SQL tables.
4. To show recommendations.

Dataset

The data we are processing is from the dataset **Book-Crossing**. As stated in the website from where it can be downloaded, http://www2.informatik.uni-freiburg.de/~cziegler/BX/ , the BookCrossing (BX) dataset was collected by Cai-Nicolas Ziegler in a 4-week crawl (August / September 2004) from the Book-Crossing community with kind permission from Ron Hornbaker, CTO of Humankind Systems. It contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit / implicit) about 271,379 books.

Alternatively, we can use the command *wget* from the Terminal to download the dataset:

```
wget http://www2.informatik.uni-freiburg.de/~cziegler/BX/BX-CSV-Dump.zip
```

The dataset comprises 3 tables, as follows:

- **BX-Users**. Contains the users. Note that user IDs (User-ID) have been anonymized and map to integers. Demographic data is provided (Location , Age) if available. Otherwise, these fields contain NULL-values.
- **BX-Books**. Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (Book-Title , Book-Author , Year-Of-Publication , Publisher), obtained from Amazon Web Services. Note that in case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavours (Image-URL-S , Image-URL-M , Image-URL-L), i.e., small, medium, large. These URLs point to the Amazon web site.
- **BX-Book-Ratings**. Contains the book rating information. Ratings (Book-Rating) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

The columns are separated by ; and all files contain the correspondent header.

In [1]:
```
# If we need to install some packages, e.g. matplotlib

# ! pip3 install matplotlib
# ! pip3 install seaborn
```

Initial settings

Additional packages and imports

In [2]:
```
# Some imports

import os
import sys

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Useful visualization functions

Some functions that we can use to plot data but as Python dataframes.

Disclaimer: these functions are broadly distributed among users. Further adjustments are needed and/or advisable. Feel free to use your own plotting functions.

Remember: *"A picture is worth a thousand words"*

def plot(df, xcol, ycol): sns.lineplot(data=df, x=xcol, y=ycol)

In [3]:
```
def plotBar(df, xcol, ycol, huecol=None):
    sns.barplot(data=df, x=xcol, y=ycol, hue=huecol)
```

In [4]:
```
def plotHistogram(df, xcol, huecol=None):
    sns.histplot(data=df, x=xcol, hue=huecol, multiple="stack")
```

def plotScatter(df, xcol, ycol, huecol): sns.set_theme(style="white") sns.scatterplot(data=df, x=xcol, y=ycol, hue=huecol)
def plotScatterMatrix(df, huecol): sns.pairplot(data=df, hue=huecol)

In [5]:
```
def plotCorrelationMatrix(corr, annot=False):
    # generate a mask for the upper triangle
    mask = np.triu(np.ones_like(corr, dtype=bool))

    # set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))

    # generate a custom diverging colormap
    cmap = sns.diverging_palette(230, 20, as_cmap=True)
    #cmap='coolwarm'

    # draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, annot=annot,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Collect and label data

Checking working diretory and data files

In [ ]:
```
! pwd
! ls -la
```

In [7]:
```
! head -n 2 BX-Users.csv
! tail -n 2 BX-Users.csv
```

```
"User-ID";"Location";"Age"
"1";"nyc, new york, usa";NULL
"278857";"knoxville, tennessee, usa";NULL
"278858";"dublin, n/a, ireland";NULL
```

In [8]:
```
! head -n 2 BX-Books.csv
! tail -n 2 BX-Books.csv
```

```
"ISBN";"Book-Title";"Book-Author";"Year-Of-Publication";"Publisher";"Image-URL-S";"Image-URL-M";"Image-URL-L"
"0195153448";"Classical Mythology";"Mark P. O. Morford";"2002";"Oxford University Press";"http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg";"http://images.amazon.com/images/P/0195153448.01.MZZZZZZZ.jpg";"http://images.amazon.com/images/P/0195153448.01.LZZZZZZZ.jpg"
"0192126040";"Republic (World's Classics)";"Plato";"1996";"Oxford University Press";"http://images.amazon.com/images/P/0192126040.01.THUMBZZZ.jpg";"http://images.amazon.com/images/P/0192126040.01.MZZZZZZZ.jpg";"http://images.amazon.com/images/P/0192126040.01.LZZZZZZZ.jpg"
"0767409752";"A Guided Tour of Rene Descartes' Meditations on First Philosophy with Complete Translations of the Meditations by Ronald Rubin";"Christopher  Biffle";"2000";"McGraw-Hill Humanities/Social Sciences/Languages";"http://images.amazon.com/images/P/0767409752.01.THUMBZZZ.jpg";"http://images.amazon.com/images/P/0767409752.01.MZZZZZZZ.jpg";"http://images.amazon.com/images/P/0767409752.01.LZZZZZZZ.jpg"
```

In [9]:
```
! head -n 2 BX-Book-Ratings.csv
! tail -n 2 BX-Book-Ratings.csv
```

```
"User-ID";"ISBN";"Book-Rating"
"276725";"034545104X";"0"
"276721";"0590442449";"10"
"276723";"05162443314";"8"
```

In [10]:
```
# some Spark related imports we will use hereafter

from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import *

from pyspark.ml import Pipeline

from pyspark.ml.feature import StringIndexer
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
```

Build a SparkSession instance if one does not exist. Notice that we can only have one per JVM spark = SparkSession\ .builder\ .appName("Recommender")\ .config("spark.sql.shuffle.partitions",6)\ .config("spark.sql.repl.eagerEval.enabled", True)\ .getOrCreate()

Reading datasets

In [11]:
```
df_users = spark.read.csv("BX-Users.csv", header="true",
                                inferSchema="true", sep=";")
```

In [12]:
```
df_books =
```

In [13]:
```
df_ratings =
```

Checking data

Schema, show and count

Users

In [14]:
```
df_users.printSchema()
df_users.show(5, truncate=False)
num_users = df_users.count()
num_users
```

```
root
 |-- User-ID: string (nullable = true)
 |-- Location: string (nullable = true)
 |-- Age: string (nullable = true)

+-------+---------------------------------+----+
|User-ID|Location                         |Age |
+-------+---------------------------------+----+
|1      |nyc, new york, usa               |NULL|
|2      |stockton, california, usa        |18  |
|3      |moscow, yukon territory, russia  |NULL|
|4      |porto, v.n.gaia, portugal        |17  |
|5      |farnborough, hants, united kingdom|NULL|
+-------+---------------------------------+----+
only showing top 5 rows
```

Out[14]: 278859

Books

In [15]:

```
root
 |-- ISBN: string (nullable = true)
 |-- Book-Title: string (nullable = true)
 |-- Book-Author: string (nullable = true)
 |-- Year-Of-Publication: integer (nullable = true)
 |-- Publisher: string (nullable = true)
 |-- Image-URL-S: string (nullable = true)
 |-- Image-URL-M: string (nullable = true)
 |-- Image-URL-L: string (nullable = true)

-RECORD 0-----------------------------------------------------------------------
 ISBN                | 0195153448
 Book-Title          | Classical Mythology
 Book-Author         | Mark P. O. Morford
 Year-Of-Publication | 2002
 Publisher           | Oxford University Press
 Image-URL-S         | http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg
 Image-URL-M         | http://images.amazon.com/images/P/0195153448.01.MZZZZZZZ.jpg
 Image-URL-L         | http://images.amazon.com/images/P/0195153448.01.LZZZZZZZ.jpg
-RECORD 1-----------------------------------------------------------------------
 ISBN                | 0002005018
 Book-Title          | Clara Callan
 Book-Author         | Richard Bruce Wright
 Year-Of-Publication | 2001
 Publisher           | HarperFlamingo Canada
 Image-URL-S         | http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg
 Image-URL-M         | http://images.amazon.com/images/P/0002005018.01.MZZZZZZZ.jpg
 Image-URL-L         | http://images.amazon.com/images/P/0002005018.01.LZZZZZZZ.jpg
only showing top 2 rows
```

Out[15]: 271379

Ratings

In [16]:

```
root
 |-- User-ID: integer (nullable = true)
 |-- ISBN: string (nullable = true)
 |-- Book-Rating: integer (nullable = true)

+-------+----------+-----------+
|User-ID|ISBN      |Book-Rating|
+-------+----------+-----------+
|276725 |034545104X|0          |
|276726 |0155061224|5          |
|276727 |0446520802|0          |
|276729 |052165615X|3          |
|276729 |0521795028|6          |
+-------+----------+-----------+
only showing top 5 rows
```

Out[16]:
```
1149780
```

Evaluate data

Let us get some data insight, with some EDA based on descriptive statistics and visualizations.

Datatypes

Problems:

- In Users, User-ID is string but should be integer. (See Ratings)
- In Users, Age is string but should be integer.

In [17]:
```
df_users = ( df_users
                .withColumn('User-ID-new',
                .withColumn('Age-new',
            )
```

In [18]:
```
# Check the changes made
```

```
root
 |-- User-ID: string (nullable = true)
 |-- Location: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- User-ID-new: integer (nullable = true)
 |-- Age-new: integer (nullable = true)

+-------+---------------------------------+----+-----------+-------+
|User-ID|Location                         |Age |User-ID-new|Age-new|
+-------+---------------------------------+----+-----------+-------+
|1      |nyc, new york, usa               |NULL|1          |null   |
|2      |stockton, california, usa        |18  |2          |18     |
|3      |moscow, yukon territory, russia  |NULL|3          |null   |
|4      |porto, v.n.gaia, portugal        |17  |4          |17     |
|5      |farnborough, hants, united kingdom|NULL|5          |null   |
+-------+---------------------------------+----+-----------+-------+
only showing top 5 rows
```

Nulls

Identify number of nulls or NaN in columns.

In [19]:
```
[num_users, df_users.dropna().count()]
```

Out[19]:
```
[278859, 168096]
```

In [20]:
```
[num_books, df_books.dropna().count()]
```

Out[20]:
```
[271379, 271379]
```

In [21]:
```
[num_ratings, df_ratings.dropna().count()]
```

Out[21]:
```
[1149780, 1149780]
```

In [22]:
```
print('\nNulls in Users:')
cols_to_forget = ['User-ID', 'Age']
users_cols_interest = [x for x in df_users.columns if x not in cols_to_forget]
for cl in users_cols_interest:
    k = df_users.select(cl).filter(F.col(cl).isNull() | F.isnan(cl)).count()
    if k > 0:
        print(f'Column {cl} with {k} nulls or NaN, out of {num_users} records ({k*100/num_users:.2f}%)')
```

```
Nulls in Users:
Column User-ID-new with 1 nulls or NaN, out of 278859 records (0.00%)
Column Age-new with 110763 nulls or NaN, out of 278859 records (39.72%)
```

In [23]:
```
df_users.filter(F.col('User-ID-new').isNull() | F.isnan(cl)).show()
```

```
+---------------+--------+----+-----------+-------+
|        User-ID|Location| Age|User-ID-new|Age-new|
+---------------+--------+----+-----------+-------+
|, milan, italy"|    NULL|null|       null|   null|
+---------------+--------+----+-----------+-------+
```

In [24]:
```
# Get rid of that wrong record, and we may avoid using Age-new column

df_users = df_users.dropna(subset=['User-ID-new'])
num_users = df_users.count()
num_users
```

Out[24]: 278858

Important:

Recall that if we delete an observation in one table, still consistency among tables has to be preserved.

We leave the checking as an exercise.

Duplicates

In [25]:
```
[num_users, df_users.dropDuplicates().count()]
```

Out[25]:
```
[278858, 278858]
```

In [26]:
```
[num_books, df_books.dropDuplicates().count()]
```

Out[26]:
```
[271379, 271379]
```

In [27]:
```
[num_ratings, df_ratings.dropDuplicates().count()]
```

Out[27]:
```
[1149780, 1149780]
```

Uniqueness

In [28]:
```
print('\nUniqueness in Users:')
for cl in users_cols_interest:
    k = df_users.select(cl).distinct().count()
    print(f'Column {cl} with {k} distinct values, out of {num_users} records ({k*100/num_users:.2f}%)')
```

```
Uniqueness in Users:
Column Location with 57309 distinct values, out of 278858 records (20.55%)
Column User-ID-new with 278858 distinct values, out of 278858 records (100.00%)
Column Age-new with 166 distinct values, out of 278858 records (0.06%)
```

In [29]:
```
print('\nUniqueness in Books:')
```

```
Uniqueness in Books:
Column ISBN with 271379 distinct values, out of 271379 records (100.00%)
Column Book-Title with 242154 distinct values, out of 271379 records (89.23%)
Column Book-Author with 102028 distinct values, out of 271379 records (37.60%)
Column Year-Of-Publication with 116 distinct values, out of 271379 records (0.04%)
Column Publisher with 16807 distinct values, out of 271379 records (6.19%)
Column Image-URL-S with 271063 distinct values, out of 271379 records (99.88%)
Column Image-URL-M with 271063 distinct values, out of 271379 records (99.88%)
Column Image-URL-L with 271063 distinct values, out of 271379 records (99.88%)
```

In [30]:
```
print('\nUniqueness in Ratings:')
```

```
Uniqueness in Ratings:
Column User-ID with 105283 distinct values, out of 1149780 records (9.16%)
Column ISBN with 340556 distinct values, out of 1149780 records (29.62%)
Column Book-Rating with 11 distinct values, out of 1149780 records (0.00%)
```

Outliers

Summary of values for numeric columns of interest, one by one. Use of describe() or summary()

In [31]:
```
cols = ['User-ID-new', 'Age-new']
for cl in cols:

```

```
+-------+-----------------+
|summary|      User-ID-new|
+-------+-----------------+
|  count|           278858|
|   mean|         139429.5|
| stddev|80499.51502027822|
|    min|                1|
|    max|           278858|
+-------+-----------------+

+-------+------------------+
|summary|          Age-new|
+-------+------------------+
|  count|            168096|
|   mean| 34.75143370454978|
| stddev|14.428097382455421|
|    min|                 0|
|    max|               244|
+-------+------------------+
```

In [32]:
```
df_books.
```

```
+-------+------------------+
|summary|Year-Of-Publication|
+-------+------------------+
|  count|            271379|
|   mean| 1959.7560496574902|
| stddev|  258.0113625638112|
|    min|                 0|
|    max|              2050|
+-------+------------------+
```

In [33]:
```
cols = ['User-ID', 'Book-Rating']
for cl in cols:

```

```
+-------+------------------+
|summary|           User-ID|
+-------+------------------+
|  count|           1149780|
|   mean|140386.39512602412|
| stddev| 80562.27771851176|
|    min|                 2|
|    max|            278854|
+-------+------------------+

+-------+------------------+
|summary|       Book-Rating|
+-------+------------------+
|  count|           1149780|
|   mean|2.8669501991685364|
| stddev| 3.854183859201656|
|    min|                 0|
|    max|                10|
+-------+------------------+
```

Problems:

- In Users, 'Age-new' ranges from 0 to 244.
- In Books, Year-Of-Publication ranges from 0 to 2050.

Clearly, some data is wrong. We should be are of that.

Visualizations

Some visualizations to better understand the data. We leave it as an exercise!

Saving clean data

Saving data for further use if needed.

In [35]:
```
cols = ['User-ID-new', 'Location']
df_clean_users = df_users.select(cols)
```

In [36]:
```
cols = ['ISBN', 'Book-Title', 'Book-Author', 'Publisher']
df_clean_books = df_books.select(cols)
```

In [37]:
```
# No need to duplicate variables here but ... for the sake of better understanding

df_clean_ratings = df_ratings
```

Context:

For the recommendation model, ratings data is critical.

As usual, we may want to have a smaller dataset just for the purpose of testing locally. But as mentioned before, consistency among the three tables has to be guaranteed.

We can also use a smaller ratings dataset, but **keeping** the complete users and books datasets.

In [38]:
```
# from counting of ratings = 1149780

fraction = 0.5 # reduce to 50%

seed = 5
with_replacement = False
df_clean_ratings_small = df_clean_ratings.sample(withReplacement=with_replacement,
                                                 fraction=fraction, seed=seed)
df_clean_ratings_small.count()
```

Out[38]:
```
574185
```

In [39]:
```
# Delete memory consuming variables that are no longer needed
```

In [40]:
```
# Saving users

output_users = "users.parquet"
df_clean_users.write.mode("overwrite").parquet(output_users)
```

In [41]:
```
# Saving books
```

```
23/04/22 17:37:10 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
```

In [42]:
```
# Saving ratings

output_ratings = "ratings.parquet"
df_clean_ratings.

output_ratings = "ratings_small.parquet"
df_clean_ratings_small.
```

```
23/04/22 17:37:11 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
```

In [ ]:
```
# Check in the running directory if that was accomplished

! ls -la
```

Also, save them as persistent tables into Hive metastore

Notes:

- An existing Hive deployment is not necessary to use this feature. Spark will take care of it.
- We can create a SQL table from a DataFrame with createOrReplaceTempView command, valid for the session. (there is also the option of global temporary views, to be shared among all sessions till the Spark application terminates)
- But with saveAsTable, there will be a pointer to the data in the Hive metastore. So persistent tables will exist even after the Spark program has restarted, as long as connection is maintained to the same metastore.

See details in http://spark.apache.org/docs/latest/sql-data-sources.html

In [ ]:
```
# Persistent tables into Hive metastore

df_clean_users.write.mode("overwrite").saveAsTable("UsersTable")
df_clean_books.
df_clean_ratings.
```

Feature engineering

Data to be used

In [45]:
```
df_clean_ratings_to_use = df_clean_ratings
# df_ratings_to_use = df_clean_ratings_small
```

Overview

After establishing the clean data to be used, we should get an overview about what we have achieved, with some statistics and visualizations.

But

we leave it as it is now, because so far there are no significant changes. Eventually, we could check the ratings and draw some plots, as it is the critical part of the system. You can have a go in that regard.

Features transformation

As mentioned, ratings are critial here. Recall that, in the dataframe, the schema is User-ID (integer), ISBN (string) and Book-Rating (integer). ISBN poses a problem as the ML algorithm requires numbers to process. Hence, we have to convert it to numbers - we will use `StringIndexer` to do so.

In [47]:
```
# StringerIndexer for ISBN

indexer = StringIndexer(inputCol="ISBN", outputCol="ISBN-Index", handleInvalid="keep")
```

In [48]:
```
# Columns from ratings that are going to be considered in the model

user_col = "User-ID"
item_col = "ISBN-Index"
rating_col = "Book-Rating"
```

Select and train model

In order to create the recommendation model, we will use the Alternating Least Squares (ALS) algorithm provided by Spark MLlib. See details in http://spark.apache.org/docs/latest/ml-collaborative-filtering.html , as we advise to check the main assumptions the implemented algorithm relies upon. For example, notice that:

- it underlies a collaborative filtering strategy;
- it aims to fill in the missing entries of a user-item association matrix, in which users and items are described by a small set of latent factors that can be used to predict missing entries. The latent factors are learned by the ALS algorithm.

Again, as for data to train the model, the focus is on ratings.

Train/test split

We will use the standard split 80/20, for the reasons explained in previous lectures.

In [49]:
```
# train/test ratings split

df_train, df_test = df_clean_ratings.

# caching data ... but just the training part and if we want to (check the implications)
# df_train.cache()

# print the number of rows in each part
print(f"There are {df_train.count()} rows in the training set and {df_test.count()} in the test set.")
```

```
There are 919970 rows in the training set and 229810 in the test set.
```

Note:

As we did with clean data, we may consider storing the data split into files, should we want to use it elsewhere. This relates to the need of guaranteeing unicity in a different environment. We leave it as it is now.

ALS model

Using the `ALS` estimator (the algorithm) to learn from the training data and consequently to build the model.

In [50]:
```
# Build the recommendation model using ALS on the training data
# note that we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics


als = ALS(maxIter=5, regParam=0.01,
          userCol=user_col,
          itemCol=item_col,
          ratingCol=rating_col,
          coldStartStrategy="drop",
          implicitPrefs=True
         )

# if the rating matrix is derived from another source of information
# (i.e. it is inferred from other signals), we may set implicitPrefs
# to True to get better results (see ALS reference)
```

ML pipeline configuration

In [51]:
```
# The pipeline holds two stages set above

# As we will see below, we are going to use it just for evaluation purposes

pipeline = Pipeline(stages=
```

Model fitting

Get the model (as transformer) by fitting the pipeline to training data. It may take time!

In [ ]:
```
pipeline_model =
```

Evaluate model

Let us evaluate the ALS model.

Testing the model

It is time to apply the model built to test data. Again, we will use the pipeline set above. Notice that, since the pipeline model is a transformer, we can easily apply it to test data.

In [53]:
```
# Make predictions on test data and show values of columns of interest

df_prediction =
```

In [54]:
```
# Checking its schema and content

df_prediction.printSchema()
df_prediction.show(truncate=False)
df_prediction.count()
```

```
root
 |-- User-ID: integer (nullable = true)
 |-- ISBN: string (nullable = true)
 |-- Book-Rating: integer (nullable = true)
 |-- ISBN-Index: double (nullable = false)
 |-- prediction: float (nullable = false)

23/04/22 17:39:13 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 296:>                 (0 + 0) / 8][Stage 310:>                (0 + 0) / 10]
23/04/22 17:39:14 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 296:=> (5 + 3) / 8][Stage 310:> (1 + 5) / 10][Stage 311:> (0 + 0) / 10]
23/04/22 17:39:15 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
```

```
23/04/22 17:39:17 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/04/22 17:39:18 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB
+-------+----------+-----------+----------+------------+
|User-ID|ISBN      |Book-Rating|ISBN-Index|prediction  |
+-------+----------+-----------+----------+------------+
|28177  |0375727345|10         |12.0      |0.62694347  |
|33862  |0375727345|5          |12.0      |0.032191776 |
|32440  |0375727345|8          |12.0      |0.21301384  |
|22605  |0375727345|7          |12.0      |0.003980553 |
|29424  |0375727345|6          |12.0      |0.009848104 |
|16634  |0375727345|0          |12.0      |-0.19294487 |
|16795  |0375727345|9          |12.0      |0.8389824   |
|35424  |0375727345|0          |12.0      |0.02602458  |
|29168  |0375727345|0          |12.0      |0.042082306 |
|36606  |0375727345|0          |12.0      |0.25273326  |
|21576  |0375727345|8          |12.0      |-0.095143445|
|14849  |0375727345|0          |12.0      |0.0         |
|6251   |0375727345|6          |12.0      |-0.17332862 |
|6575   |0375727345|0          |12.0      |1.0489374   |
|35857  |0375727345|5          |12.0      |0.43122125  |
|12285  |0375727345|9          |12.0      |0.007032776 |
|37979  |0375727345|9          |12.0      |-0.00791256 |
|41841  |0375727345|8          |12.0      |0.1919349   |
|74728  |0375727345|8          |12.0      |-0.072210416|
|75428  |0375727345|9          |12.0      |0.021363957 |
+-------+----------+-----------+----------+------------+
only showing top 20 rows

23/04/22 17:39:18 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 333:>                 (0 + 0) / 8][Stage 347:>                 (0 + 0) / 10]
23/04/22 17:39:19 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 333:>  (0 + 0) / 8][Stage 347:> (0 + 0) / 10][Stage 348:> (0 + 0) / 10]
23/04/22 17:39:20 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB


23/04/22 17:39:22 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/04/22 17:39:23 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB

```

Out[54]: 174536

In [55]:
```
df_prediction.orderBy("User-ID").show(truncate=False)
```

```
23/04/22 17:39:25 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 387:>                (0 + 0) / 8][Stage 401:>                (0 + 0) / 10]
23/04/22 17:39:26 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 387:>  (0 + 0) / 8][Stage 401:> (0 + 0) / 10][Stage 402:> (0 + 0) / 10]
23/04/22 17:39:27 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB

23/04/22 17:39:28 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/04/22 17:39:29 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB
[Stage 423:>                                                        (0 + 8) / 9]
```

```
+-------+----------+-----------+----------+-------------+
|User-ID|ISBN      |Book-Rating|ISBN-Index|prediction   |
+-------+----------+-----------+----------+-------------+
|8      |0399135782|0          |4077.0    |-1.6880228E-9|
|8      |0002005018|5          |16189.0   |9.2019986E-10|
|8      |0671870432|0          |193498.0  |5.643216E-10 |
|8      |1881320189|7          |110296.0  |2.4213542E-10|
|10     |1841721522|0          |3434.0    |1.0123705E-12|
|14     |0971880107|0          |0.0       |-0.0044853445|
|17     |0553278398|0          |1829.0    |0.0014182599 |
|32     |038078243X|0          |19957.0   |2.4111976E-6 |
|39     |0553582909|8          |4642.0    |0.0018880817 |
|44     |0842342702|0          |516.0     |9.4112195E-4 |
|67     |042511774X|0          |323.0     |4.95722E-11  |
|99     |0451166892|3          |141.0     |0.0035735366 |
|99     |0446677450|10         |1463.0    |7.2222704E-4 |
|99     |0312261594|8          |5372.0    |0.0011581918 |
|99     |0553347594|9          |41050.0   |3.0804033E-4 |
|114    |0446612618|8          |7905.0    |0.009146694  |
|114    |0446608653|9          |1056.0    |0.016526029  |
|160    |9728579225|8          |67796.0   |1.6077525E-10|
|165    |0061099325|4          |1251.0    |0.0049548014 |
|183    |9724129411|9          |296718.0  |4.3810675E-10|
+-------+----------+-----------+----------+-------------+
only showing top 20 rows
```

In [56]:
```
df_prediction.orderBy("ISBN-Index").show(truncate=False)
```

```
23/04/22 17:39:30 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 424:>                 (0 + 0) / 8][Stage 438:>                 (0 + 0) / 10]
23/04/22 17:39:31 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 424:>  (0 + 0) / 8][Stage 438:> (0 + 0) / 10][Stage 439:> (0 + 0) / 10]
23/04/22 17:39:32 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
```

```
23/04/22 17:39:33 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/04/22 17:39:34 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB
+-------+----------+-----------+----------+-------------+
|User-ID|ISBN      |Book-Rating|ISBN-Index|prediction   |
+-------+----------+-----------+----------+-------------+
|31826  |0971880107|1          |0.0       |-0.27701813  |
|10338  |0971880107|0          |0.0       |0.0012092944 |
|35513  |0971880107|8          |0.0       |-8.5264915E-8|
|35823  |0971880107|0          |0.0       |0.0          |
|7286   |0971880107|0          |0.0       |0.33245176   |
|27740  |0971880107|0          |0.0       |0.05949734   |
|1435   |0971880107|5          |0.0       |0.48156643   |
|21484  |0971880107|0          |0.0       |-0.063117385 |
|26525  |0971880107|0          |0.0       |-0.014014728 |
|30144  |0971880107|0          |0.0       |0.009675688  |
|14079  |0971880107|0          |0.0       |-0.115478896 |
|26443  |0971880107|0          |0.0       |0.008276263  |
|4781   |0971880107|0          |0.0       |-1.8554585E-8|
|35859  |0971880107|0          |0.0       |0.5671146    |
|13722  |0971880107|0          |0.0       |0.010174753  |
|1025   |0971880107|0          |0.0       |0.3754246    |
|16999  |0971880107|0          |0.0       |0.09323458   |
|33816  |0971880107|0          |0.0       |0.31047556   |
|15053  |0971880107|0          |0.0       |-9.665739E-7 |
|32329  |0971880107|6          |0.0       |-0.11149354  |
+-------+----------+-----------+----------+-------------+
only showing top 20 rows
```

Evaluation metrics

Let us use an evaluator.

In [57]:
```
# Evaluate the model by computing the RMSE on the test data

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

rmse = evaluator.evaluate(df_prediction)
print("Root-mean-square error = " + str(rmse))
```

```
23/04/22 17:39:35 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
23/04/22 17:39:35 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 461:>  (0 + 0) / 8][Stage 475:> (0 + 0) / 10][Stage 476:> (0 + 0) / 10]0]
23/04/22 17:39:36 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB


23/04/22 17:39:38 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/04/22 17:39:39 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB
[Stage 497:>                                                        (0 + 8) / 8]
23/04/22 17:39:40 WARN DAGScheduler: Broadcasting large task binary with size 12.4 MiB
Root-mean-square error = 4.651106603145104


```

Saving the pipeline

In [58]:
```
# We can save the pipeline for further use should it be required

pipeline.save("pipeline-ALS")

# later on, it can be loaded anywhere

# check the content of created directories
```

In [ ]:
```

```

Tune model

We can improve the model. For example, by carrying out better data cleasing operations and take into consideration efficiency issues.

Deploy model

Pre-computing recommendations

The `ALS` algorithm provides some functions to get recommendations directly.

Although we can achieve results if working with predictions after the pipeline set (see below), we will take advantage of such methods directly.

We should emphasize that, as it stands, we will not be using the pipeline for this task.

+-------+----------+-----------+----------+-------------+ |User-ID|ISBN |Book-Rating|ISBN-Index|prediction | +-------+----------+-----------+----------+-------------+ |30261 |0385504209|0 |2.0 |0.04549066 | |3363 |0385504209|0 |2.0 |0.03916065 |

In [ ]:
```
# Checking with training data for the sake of example

df_train_indexed = indexer.fit(df_train).transform(df_train)
model = als.fit(df_train_indexed)
```

In [62]:
```
# Get all distinct users and books

#user_col = "User-ID"
#item_col = "ISBN-Index"
#rating_col = "Book-Rating"

users = df_train_indexed.select(als.getUserCol()).distinct()

books = df_train_indexed.select(als.getItemCol()).distinct()
```

In [63]:
```
users.show()
```

```
+-------+
|User-ID|
+-------+
|    463|
|    496|
|   1238|
|   1591|
|   1829|
|   2366|
|   3175|
|   3918|
|   4900|
|   5300|
|   5803|
|   6336|
|   6357|
|   6466|
|   6654|
|   7253|
|   7340|
|   7754|
|   7982|
|   8086|
+-------+
only showing top 20 rows
```

In [64]:
```
books.show()
```

```
23/04/22 17:40:10 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
[Stage 626:=================================>                     (5 + 3) / 8]
23/04/22 17:40:12 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
+----------+
|ISBN-Index|
+----------+
|   15893.0|
|     596.0|
|  280485.0|
|    7313.0|
|    2862.0|
|     558.0|
|     299.0|
|  100406.0|
|  234747.0|
|  270298.0|
|  191645.0|
|  289931.0|
|     305.0|
|  213226.0|
|   28481.0|
|   80005.0|
|   87780.0|
|   82979.0|
|   25813.0|
|   13607.0|
+----------+
only showing top 20 rows
```

In [65]:
```
[users.count(), books.count()]
```

```
23/04/22 17:40:13 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
```

```
[Stage 635:=======>                                                 (1 + 7) / 8]
23/04/22 17:40:14 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
```

Out[65]:
```
[92938, 298820]
```

In [66]:
```
# Generate top book recommendations for users

top_n_books = 2
user_recs = model.recommendForAllUsers(top_n_books)

# Generate top book recommendations for a specified set of users

# subset_users = users.limit(5)
# user_subset_recs = model.recommendForUserSubset(subset_users, top_n_books)
```

In [67]:
```
user_recs.show(truncate=False)

# user_subset_recs.show(truncate=False)
```

```
23/04/22 17:40:15 WARN DAGScheduler: Broadcasting large task binary with size 11.0 MiB
[Stage 654:=================================================>(99 + 1) / 100]
```

```
23/04/22 17:42:58 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
+-------+----------------------------------------+
|User-ID|recommendations                         |
+-------+----------------------------------------+
|12     |[{13, 2.2562064E-8}, {45, 1.9955959E-8}]|
|44     |[{55, 0.020076418}, {45, 0.018091321}]  |
|53     |[{16, 0.014748665}, {13, 0.009437016}]  |
|78     |[{18, 0.011642931}, {13, 0.011133369}]  |
|81     |[{5, 0.008602302}, {39, 0.006890223}]   |
|85     |[{1, 0.008010264}, {8, 0.0037765228}]   |
|133    |[{1, 0.016934456}, {22, 0.008972366}]   |
|137    |[{2, 1.7402932E-10}, {45, 1.6306474E-10}]|
|183    |[{5, 0.03477192}, {16, 0.026116533}]    |
|193    |[{40960, 0.0}, {40970, 0.0}]            |
|236    |[{40960, 0.0}, {40970, 0.0}]            |
|243    |[{8, 0.4884001}, {12, 0.43862292}]      |
|300    |[{15, 0.014299512}, {42, 0.012803585}]  |
|362    |[{77, 0.0053710025}, {0, 0.0047543505}] |
|384    |[{40960, 0.0}, {40970, 0.0}]            |
|392    |[{0, 8.4980944E-5}, {93, 6.320027E-5}]  |
|406    |[{1, 1.5957012E-8}, {2, 1.21170505E-8}] |
|460    |[{10, 0.0030535515}, {0, 0.0029589625}] |
|463    |[{25, 0.0032482832}, {77, 0.0030188842}]|
|472    |[{55, 0.0011070749}, {33, 0.001098036}] |
+-------+----------------------------------------+
only showing top 20 rows
```

In [68]:
```python
# Generate top user recommendations for each book

top_n_users = 2
book_recs = model.recommendForAllItems(top_n_users)

# Generate top user recommendations for a specified set of books

# subset_books = books.limit(5)
# book_subset_recs = model.recommendForItemSubset(subset_books, top_n_users)
```

In [69]:
```
book_recs.show(truncate=False)

# book_subset_recs.show(truncate=False)
```

```
23/04/22 17:42:59 WARN DAGScheduler: Broadcasting large task binary with size 11.0 MiB
[Stage 683:=====================================================>(99 + 1) / 100]
23/04/22 17:45:37 WARN DAGScheduler: Broadcasting large task binary with size 10.9 MiB
+----------+----------------------------------------+
|ISBN-Index|recommendations                         |
+----------+----------------------------------------+
|26        |[{235282, 1.3388069}, {4017, 1.2779285}]   |
|27        |[{214786, 1.4063966}, {7346, 1.2597617}]   |
|28        |[{235282, 1.6009766}, {254465, 1.3653027}] |
|31        |[{76499, 1.4286722}, {31826, 1.0966935}]   |
|34        |[{153662, 2.2438536}, {11676, 1.3453243}]  |
|44        |[{76499, 1.6846067}, {98391, 1.5944945}]   |
|53        |[{55490, 1.1888067}, {98391, 1.1182915}]   |
|65        |[{142524, 0.91407824}, {232131, 0.8715034}]|
|76        |[{153662, 1.6012522}, {11676, 1.2615935}]  |
|78        |[{98391, 1.6920795}, {56399, 1.529717}]    |
|81        |[{11676, 0.8763598}, {104636, 0.8072574}]  |
|85        |[{11676, 1.2336705}, {204864, 0.86159515}] |
|101       |[{11676, 1.1785723}, {204864, 1.1030658}]  |
|103       |[{98391, 1.1639751}, {56399, 1.1351087}]   |
|108       |[{230522, 0.916898}, {60244, 0.8593741}]   |
|115       |[{95359, 0.7740643}, {60244, 0.7203063}]   |
|126       |[{11676, 0.8900601}, {258534, 0.7349734}]  |
|133       |[{153662, 0.9926629}, {104636, 0.955925}]  |
|137       |[{69078, 0.9394564}, {56959, 0.913174}]    |
|148       |[{258534, 0.7546577}, {104636, 0.71062905}]|
+----------+----------------------------------------+
only showing top 20 rows
```

Storing recommendations as persistent tables

Save the recommendations as persistent tables into the Hive metastore.

In [ ]:
```
user_recs.write.mode("overwrite").saveAsTable("UserRecommendationsTable")
```

In [ ]:
```
book_recs.
```

It looks like we had a problem storing array> Exercise: how can we sort it out?

In [ ]:
```
# Check in the running directory
```

Exploring results

1. Given a user, shows the recommended list of books.
2. Given a book, shows the list of users who might be interested on.

We are going to use Spark SQL tables.

In [73]:
```
# user to explore

user = 0
```

In [74]:
```
# book to explore

book = 0
```

First, let us check the SQL tables.

In [75]:
```
# Register information about users as a SQL temporary view

df_clean_users.createOrReplaceTempView("users")
```

In [76]:
```
# Register information about books as a SQL temporary view

df_clean_books.createOrReplaceTempView("books")
```

In [77]:
```
print(spark.catalog.listDatabases())
```

```
[Database(name='default', description='Default Hive database', locationUri='file:/Users/adriano/Documents/Academia/ISCTE/Teaching/2021-22/Praticas/Notebooks/Solutions/spark-warehouse')]
```

In [78]:
```
 spark.catalog.listTables(dbName="default")
```

Out[78]:
```
[Table(name='bookrecommendationstable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='bookstable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='ratingstable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='scoretable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='userrecommendationstable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='userstable', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='books', database=None, description=None, tableType='TEMPORARY', isTemporary=True),
 Table(name='users', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]
```

In [79]:
```
# Use managed tables

spark.sql("USE default")
```

Out[79]:
```
DataFrame[]
```

In [80]:
```
spark.catalog.listColumns('bookstable')
```

Out[80]:
```
[Column(name='ISBN', description=None, dataType='string', nullable=True, isPartition=False, isBucket=False),
 Column(name='Book-Title', description=None, dataType='string', nullable=True, isPartition=False, isBucket=False),
 Column(name='Book-Author', description=None, dataType='string', nullable=True, isPartition=False, isBucket=False),
 Column(name='Publisher', description=None, dataType='string', nullable=True, isPartition=False, isBucket=False)]
```

In [81]:
```
 spark.sql("SELECT * FROM users").show(10, truncate=False)
```

```
+-----------+--------------------------------+
|User-ID-new|Location                        |
+-----------+--------------------------------+
|1          |nyc, new york, usa              |
|2          |stockton, california, usa       |
|3          |moscow, yukon territory, russia |
|4          |porto, v.n.gaia, portugal       |
|5          |farnborough, hants, united kingdom|
|6          |santa monica, california, usa   |
|7          |washington, dc, usa             |
|8          |timmins, ontario, canada        |
|9          |germantown, tennessee, usa      |
|10         |albacete, wisconsin, spain      |
+-----------+--------------------------------+
only showing top 10 rows
```

In [82]:
```
spark.sql("SELECT * FROM books").show(10, vertical=True, truncate=False)
```

```
-RECORD 0-----------------------------------------------------------------------------------------------------
 ISBN        | 0195153448
 Book-Title  | Classical Mythology
 Book-Author | Mark P. O. Morford
 Publisher   | Oxford University Press
-RECORD 1-----------------------------------------------------------------------------------------------------
 ISBN        | 0002005018
 Book-Title  | Clara Callan
 Book-Author | Richard Bruce Wright
 Publisher   | HarperFlamingo Canada
-RECORD 2-----------------------------------------------------------------------------------------------------
 ISBN        | 0060973129
 Book-Title  | Decision in Normandy
 Book-Author | Carlo D'Este
 Publisher   | HarperPerennial
-RECORD 3-----------------------------------------------------------------------------------------------------
```

```
 ISBN        | 0374157065
 Book-Title  | Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused I
t
 Book-Author | Gina Bari Kolata
 Publisher   | Farrar Straus Giroux
-RECORD 4-------------------------------------------------------------------------------------------------
--
 ISBN        | 0393045218
 Book-Title  | The Mummies of Urumchi
 Book-Author | E. J. W. Barber
 Publisher   | W. W. Norton &amp;amp; Company
-RECORD 5-------------------------------------------------------------------------------------------------
--
 ISBN        | 0399135782
 Book-Title  | The Kitchen God's Wife
 Book-Author | Amy Tan
 Publisher   | Putnam Pub Group
-RECORD 6-------------------------------------------------------------------------------------------------
--
 ISBN        | 0425176428
 Book-Title  | What If?: The World's Foremost Military Historians Imagine What Might Have Been
 Book-Author | Robert Cowley
 Publisher   | Berkley Publishing Group
-RECORD 7-------------------------------------------------------------------------------------------------
--
 ISBN        | 0671870432
 Book-Title  | PLEADING GUILTY
 Book-Author | Scott Turow
 Publisher   | Audioworks
-RECORD 8-------------------------------------------------------------------------------------------------
--
 ISBN        | 0679425608
 Book-Title  | Under the Black Flag: The Romance and the Reality of Life Among the Pirates
 Book-Author | David Cordingly
 Publisher   | Random House
-RECORD 9-------------------------------------------------------------------------------------------------
--
 ISBN        | 074322678X
 Book-Title  | Where You'll Find Me: And Other Stories
```

```
 Book-Author | Ann Beattie
 Publisher   | Scribner
only showing top 10 rows
```

In [83]:
```
print("The recommended books for user " + str(user) + " are: ")
```

```
The recommended books for user 0 are:
```

We leave it as exercise!

In [84]:
```
print("The users who might be interested on the book " + str(book) + " are: ")
```

```
The users who might be interested on the book 0 are:
```

We leave it as exercise!

Additional exercise

Given the current status of this notebook, redo its content such that major tasks are split into various notebooks, or Python modules. The purpose is to modularize code having in mind the setup of a real recommender system. That is:

- A downloader module, with focus on downloading data, cleasing it, and then storing it in a data store.
- A recommender building module, to create a recommendation model
- A recommender running module, to pre-compute recommendations and to save them in a data store.
- A recommender server, to retrieve recommendations upon queries made to the data store.

References

- Learning Spark - Lightning-Fast Data Analytics, 2nd Ed. J. Damji, B. Wenig, T. Das, and D. Lee. O'Reilly, 2020
- Spark: The Definitive Guide - Big Data Processing Made Simple, 1st Ed. B. Chambers and M. Zaharia. O'Reilly, 2018
- http://spark.apache.org/docs/latest/ml-guide.html
- https://docs.python.org/3/

In [ ]: