Team project: yfitopS analytics

Deadline: as announced in canvas

Procedural stuff

Learning outcomes: The learning outcomes of the project include: (1) understanding the practical relevance of big data platforms in real-world problems, (2) being able to quickly master and make practical use of big data platforms, and (3) being able to implement and exploit synopses.

Team formation: Form teams of 6 persons each. You are free to form your own teams. Each member of the team is expected to fully agree with the team's submission, and to be able to describe/explain and support the team's submission.

Submission details: Submit all relevant material for the milestone in Canvas before the deadline. Each team should submit exactly one submission. No late submissions will be accepted.

General constraints: (1) Your code should not use external libraries (i.e., anything else than the ones included with Java 11/Python and Spark version 3.3.4). If you are in doubt, ask us. (2) Your code should run correctly on our server, with the provided submission system. (3) You should not change the signatures (input/output parameters) of the provided template methods. You should fill your code only in the provided space, as mentioned in the template.

Project details

Abstract: yfitopS is an online music subscription service. Registered users can play songs, search for songs, and get recommendations for songs that they may like based on their previous activity. You just joined yfitopS as a data analyst and it is your task to propose solutions for some of these problems. Good luck!

Data: The key data for this project is a file called plays.csv, containing all the songs that each user listened to, and the user's rating for each song. The lines in the file follow the format: <userid, songid, rating>. Both user id and song id are positive integers (max. value 2^{31}). When a rating exists, it is an integer from 1 to 10, with higher ratings meaning that the user liked the song more.

For example, line:
1, 44782, 3
means that user 1 listened to song 44782, and gave rating 3.

Since users do not always provide ratings, many of the lines do not include ratings. In that case, the line format becomes <userid, songid>. For example, line:
2, 44782
means that user 2 listened to song 44782 but did not provide a rating for the song.

It is allowed that a user listens to the same song multiple times, in which case, the same <userid, songid> pair appears many times in the file. However, each user can rate each song **at most once**, i.e., for each userid and songid combination, there is at most one <userid, songid, rating> triple.

Downloading the data:

We provide two data files in canvas (as zip files). The first one is small, for local execution. The second is large, and is similar in size to the one used at the server – but not identical in contents. Remember that you need to unzip the files before running your code.

Question 0 (0 points)

Write code in Spark to import the dataset: (a) as a data frame/dataset, and (b) as an RDD. Even though this question receives 0 points, you will rely on this code for the following questions. Therefore, if your code for this question is wrong, this will most likely lead to wrong answers for the following questions.

Question 1 (20 points)

The average rating of a user is the mean value of all ratings provided by that user. For example, assuming that we have the following information for user 1:
1, 44, 3
1, 45, 5
1, 56
1, 53
1, 46, 1

The average rating for the user is (3+5+1)/3. The definition of average rating does not account for the lines without a rating.

Write code **that uses SparkSQL** to find the total number of all users that provided at least 100 ratings, and have an average rating below 2.

Your answer should be printed as follows:
>> Q1: number

For example,
>> Q1: 4

Constraints:

a) Your code should achieve the maximum degree of parallelism on the available hardware. The code that is executed centrally (on the master node) should be of constant complexity. This also means that the code executed on the master should contain no loops.
b) The actual computation of your answer should be with **a single standard SQL command, i.e., SQL that could also run in a relational database.** Besides creating a view or a temporary table (which may use the dataframe API) you should not use the dataframe API for the actual computation of the answer.

For example, the following command:

```
spark.sql("SELECT name FROM people WHERE age>10").show()
```

satisfies this constraint (albeit, it does not correctly answer the question), whereas

```
df.filter(col("age").gt(10)).select("name");
```

does not satisfy this constraint because it uses the dataframe API on something else than creating a view or a temporary table (in this example, for filtering and for projection).

Report/deliver the following information/code:

a) Fill in the missing code in question1.java or question1.py for answering the question.
b) The SQL query [Report and poster]
c) The output of the query plan [Report]
d) A paragraph (or pseudocode) describing how you would implement the same functionality in Spark, but without using SparkSQL. Notice that you do not need to actually implement it. [Report and short discussion in the poster]
e) A paragraph where you compare your Spark implementation (point d above) and the SparkSQL implementation, in terms of efficiency. The paragraph should solely rely on the query plan of SparkSQL, and on your answer on point d above, i.e., without actually implementing point d [Report and short discussion in poster]

The expected length of this question in the report is 1 page. Sub-questions a, b, c receive 10 points total, and points d and e receive 10 points total (assuming that the previous sub-questions are correct).

Hint: For all questions, you can test that your SQL answers are correct on the small dataset, for which we provide the expected answers. Keep in mind that correct answers on the small dataset do not guarantee that your code is correct. However, wrong answers on the small dataset mean that your code is wrong.

Answer on the small dataset:
>> Q1: 4

Question 2 (20 points)

You are asked to write a Spark program (not using SparkSQL) for finding the grumpiest user. The formal definition of the grumpiest user is as follows:

a) The user provided at least 100 ratings, and,
b) The user has the minimum average rating, compared to all users that provided at least 100 ratings.

You can assume that there is exactly one grumpiest user, i.e., no two users have an identical minimum average rating!
Notice that the definition of the grumpiest user only considers the users with at least 100 ratings! Failing to take this into account will lead to wrong results.

Your answer should be printed as follows:
>> Q2: userid, averagerating

For example,
>> Q2: 861, 1.0

Constraints:

a) Your code should have a high (but configurable) degree of parallelism. The code that is executed centrally (on the master node) should be of constant complexity. This also means that the code executed on the master should contain no loops.
b) Your answer should not use SQL or the dataframe API. This also means that you should not use the groupBy function. Think of how this functionality could be done with reduceByKey instead!

Report/deliver the following information/code:

a) Fill in the missing method in question2.java or question2.py.
b) Brief description of your solution [Report and poster]
c) Investigate how parallelism helps on the performance of your code in Question 2. In particular, test how fast is your code for different degrees of parallelism (indicatively, 1, 5, 10, 20, 40, 80). Plot, discuss, and explain your results. [Report and poster]

The expected length of this question in the report is 1 page. Points a and b receive 10 points total, and point c receives 10 points total (assuming that the previous points are correct).

Answer on the small dataset:
>> Q2: 861, 1.0

Question 3 (20 points)

yfitopS is losing clients. Allegedly, its users get easily bored listening to the same songs again and again. Therefore, yfitopS wants to be able to recommend new songs to each user, based on user ratings from similar users. Intuitively, two users are considered similar when they have similar ratings on the songs. A similarity of two users is measured using cosine similarity. Formally, for any user X, let Ratings(X) represent the vector that contains all ratings that this user has provided. For example, assume that the system contains only 5 songs (with ids from 1 to 5), and the only contents in plays.csv for users 8 and 9 are the following lines:

```
8, 1, 3
8, 5
8, 2, 7
9, 2, 5
9, 4
9, 5, 10
```

Then Ratings(8) is as follows [3, 7, 0, 0, 0] and Ratings(9) is [0, 5, 0, 0, 10].

Finally, the similarity of the two users is computed by the cosine similarity.[1] In this example:

$$\text{Sim}(8, 9) = \frac{\sum_{s=1}^{5} Ratings(8)[s] * Ratings(9)[s]}{\sqrt{\sum_{s=1}^{5} (Ratings(8)[s])^2} * \sqrt{\sum_{s=1}^{5} (Ratings(9)[s])^2}} = \frac{3*0+7*5+0*0+0*0+0*10}{\sqrt{3^2+7^2+0+0+0} * \sqrt{0+5^2+0+0+10^2}} \approx 0.41$$

You need to write Spark code that, for each user with id A, it finds the user with id B, with $A < B$, that satisfies both following conditions:

a) has the maximum Sim(A,B). In case of a tie (i.e., for a user A, many users have the same maximum similarity), you need to return the user with the smallest ID
b) has Sim(A,B) greater than 0.95. For any user A, if there is no user B with similarity greater than this threshold, then no output should be returned for A.

Your answer should be printed as follows:

```
>> Q3: <userid1, userid2, similarity>
>> Q3: <userid3, userid4, similarity>
…
```

(for all pairs of users that satisfy the above conditions). Similarity should be rounded to 2 decimals points.

For example,

```
>> Q3: <8, 11, 0.96>
```

[1] https://en.wikipedia.org/wiki/Cosine_similarity

```
>> Q3: <3, 10, 0.97>
…
```

Constraints:

a) Your code should achieve the maximum degree of parallelism on the available hardware. The code that is executed centrally (on the master node) should be of constant complexity. This also means that the code executed on the master should contain no loops.

b) Your answer should not use SQL or the dataframe API. This also means that you should not use the groupBy function. Think of how this functionality could be done with reduceByKey instead!

c) You are allowed (but not forced) to use the mllib library.

d) You should not perform redundant computations. Exploit the commutative property of the cosine similarity.

Hint: If you want, you can use sparse vectors to reduce RAM (either the ones included in mllib or your own implementation).

Report/deliver the following information/code:

a) Fill the missing method in question3.java or question3.py.

b) Brief description of your solution [Report and poster]

The expected length of this question in the report is 1 page. Points a and b receive 20 points total.

Answer on the small dataset:

```
>> Q3: <373, 1233, 0.99>
>> Q3: <72, 932, 0.99>
>> Q3: <1233, 2094, 0.99>
>> Q3: <227, 1087, 0.99>
>> Q3: <373, 2094, 0.99>
>> Q3: <227, 1948, 0.99>
>> Q3: <72, 1793, 0.99>
>> Q3: <1087, 1948, 0.99>
>> Q3: <932, 1793, 0.99>
```

Question 4 (20% of the project)

yfitopS contains the largest song collection in the planet – some say also in the universe, but there is no conclusive proof for this yet. Indicatively, the current song count is close to 2^{31}. However, perhaps not surprisingly, the users mostly listen to a very small percentage of these songs. You are asked to estimate the total number of distinct songs that the users listened to (even if they did not provide a rating). Since yfitopS is trying to cut costs, you need a memory-efficient solution, relying on sketches.

In particular, in terms of code, you need to:

a) Implement FM sketches in Spark (Hint: check the pseudocode[2]). The sketch should be configured with ε=0.1 and δ=0.1.
b) Write Spark code that parses the input file and constructs an FM sketch (remember that an FM sketch will contain multiple repetitions[3])
c) Have the master node collect the FM sketch and do the final estimate

Constraints:

a) Your code should achieve the maximum degree of parallelism on the available hardware. The code that is executed centrally (on the master node) should be only the code that produces the final estimate based on a sketch produced in a distributed fashion.
b) Your answer should not use SQL or the dataframe API. This also means that you should not use the groupBy function.

Report/deliver the following information/code:

a) Fill the missing method in question4.java or question4.py.
b) Experiment on the accuracy and efficiency of the sketch, when varying the ε and δ values [Report and poster].
c) Brief description of your solution and your observations from the experiment [Report and poster]

The expected length of this question in the report is 1 page. Points a,b,c receive 20 points total.

Your answer should be printed as follows:
>> Q4: number

For example,
>> Q4: 100

Answer on the small dataset:
>> Q4: 57310

Question 5 (20% of the project)

Your boss just had a weird request. She wants to find all triples of users that have cumulatively rated at most 8000 distinct songs, with userid1 < userid2 < userid3. Nobody knows why, and your boss does not care about more explanations. Still, you need to implement it. Precisely:
In particular, in terms of code, you need to:

a) Implement this functionality in Spark

[2] Ganguly, S., Garofalakis, M. & Rastogi, R. Tracking set-expression cardinalities over continuous update streams. *VLDB* **13**, 354–369 (2004). https://doi.org/10.1007/s00778-004-0135-3 (Fig. 2)
[3] Graham Cormode; Minos Garofalakis; Peter J. Haas; Chris Jermaine, *Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches* , now, 2011, doi: 10.1561/1900000004. (see Section 5.4.2)

b) Write Spark code that parses the input file and prints out the output. The output lines should follow this format: <a,b,c,numberOfRatings>, where a,b,c are user ids, with a < b < c, and numberOfRatings is the total number of ratings.

Your answer should be printed as follows:
>> Q5: <userid1, userid2, userid3, totalRatings>
>> Q5: <userid1, userid2, userid3, totalRatings>
…
(for all pairs of users that satisfy the above conditions).

For example,
>> Q5: <1, 3, 12, 9>
>> Q5: <1, 3, 20, 10>

Constraints:

a) Your code should achieve the maximum degree of parallelism on the available hardware. The code that is executed centrally (on the master node) should be of constant complexity, or for printing the final answer. This also means that the code executed on the master should contain no loops.
b) You are allowed to use a sketch, as long as you thoroughly explain the details, and
c) Each combination of users that satisfy the constraints should be printed exactly once. The order of the user ids is irrelevant. For example, tuples <a,b,c,numberOfRatings> <b,a,c,numberOfRatings> are considered equal. Only one of these tuples should be printed.
d) Your code should be optimized to finish quickly, and not use a lot of resources and energy. Even though the search space of a naive solution for the described problem is huge, there exist some easy early pruning techniques that can drastically reduce the search space.
e) Your answer should not use SQL or the dataframe API. This also means that you should not use the groupBy function.

Report/deliver the following information/code:

a) Fill the missing method in question5.java or question5.py.
b) Brief description of your solution [Report and poster]

The expected length of this question in the report is 1 page. Points a and b receive 20 points total.

Answer on the small dataset:
>> Q5: <72,861,932,6086>
>> Q5: <72,861,1793,6086>
>> Q5: <72,932,1793,5663>
>> Q5: <112,563,861,7029>
>> Q5: <112,773,861,6973>
>> Q5: <112,845,861,6959>

```
>> Q5: <227,861,1087,6158>
>> Q5: <227,861,1948,6158>
>> Q5: <227,1087,1948,5725>
>> Q5: <373,861,2094,6204>
>> Q5: <373,861,1233,6204>
>> Q5: <373,1233,2094,5772>
>> Q5: <563,773,861,6957>
>> Q5: <563,845,861,6939>
>> Q5: <773,845,861,6887>
>> Q5: <861,932,1793,6086>
>> Q5: <861,1087,1948,6158>
>> Q5: <861,1233,2094,6204>
```

Deliverables:
You are expected to submit:

a) For java solutions
   i) An executable file app.jar. The file needs to include all dependencies/libraries (a so-called "fat-jar"). It should follow the provided template, such that it can be submitted to the submission system without changes.
   ii) A zip file containing the code that was used to generate the jar. Make sure that the code compiles normally.
b) For Python solutions
   i) A zip containing the source code. It should follow the provided template, such that it can be submitted to the submission system without changes.
c) A pdf report of around 4 pages (not more than 5 pages), containing:
   i) The answers to the above questions
d) An excel file, containing your peer evaluations (optional). These will be used to decide on your individual project grades. An equal contribution will be assumed if this table is not present. More details (and the excel template) will be posted in canvas. It is **not** possible to revise/update this table after the project's deadline, or after you receive your project grade.
e) A 1-page poster outlining your contribution and results. This should be the exact same poster that you will present during the poster session. You are **not** allowed to change/update it after the deadline.

General instructions:

a) Aim for compactness. Feel free to use tables, in order to summarize the results in a succinct way and save space.

b) References are not included in the page limit of the report. However, references should be from other authors, and not technical reports/extensions of the submission
c) Include all submission files in a single zip. Upload the zip in Canvas before the deadline.