Team project: Yfitops analytics

Deadline and answers: we will be discussing the solutions of each exercise, as announced in the courses' schedule.

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, (4) preparing you for the final exam.

Team formation: We recommend that you form teams of 4 persons. You are free to form your own teams.

Submission details: You do not need to submit your work. The project will **not** be graded and no solutions will be uploaded. The points in the questions are presented only as an indication of the difficulty of each question. However, we will assume that you have solved (or know how to solve) the project. Similar exercises may end up in the final exam. You can ask questions about the project and possible directions to solve it during the class.

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.

Things to consider and try out:
You should think of the following techniques/questions/concerns:
- Broadcasting – what does it mean, and how/when does it help/harm scalability and efficiency? What makes sense to broadcast and what doesn't.
- SparkSQL and datasets/dataframes Vs pure Spark with RDDs
- Setting the parallelism and number of partitions?
- Checking whether a solution/code indeed has a high degree of parallelism and it exploits all available CPUs? What can go wrong in terms of parallelism? It is expected that your code achieves the maximum degree of parallelism and what is executed centrally, on the master node, is of constant complexity.
- Finding where the bottlenecks are.
- When does it make sense to collect and what does collect mean for scalability?
- Sparse vs dense representations, e.g., within Spark's Mllib?
For the final exam you are expected to know how to answer/reason about these questions.

Question 1 - Loading the data (5 points)

Write code in Spark to import the dataset: (a) as a data frame/dataset, and (b) as an RDD. 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 2 - Basic transformations (5 points)

Write code in Spark to find the id of the user that gave at least 10 ratings, and has the highest average rating. If more than one such users exist with equal average rating, you can break the ties arbitrarily (i.e., return any one of them). You are not allowed to use SparkSQL or

dataframes/datasets for this exercise. You are only allowed to use RDDs. The format of the answer should be a pair of <id, averageRating>
Solution: reducebykey sum, where value is <cnt,sumofratings>, and then reduce

Question 3 - Basic transformations (7 points)

Write code in Spark to find the id of the user that gave at least 10 ratings, and has the highest average rating. If more than one such users exist with equal rating, you need to return the user with the **smallest id** among them**.** You are not allowed to use SparkSQL or dataframes/datasets for this exercise. You are only allowed to use RDDs. The format of the answer should be a pair of <id, averageRating>
Solution: same as Q2 but possibly need to store the userid in the values (maybe not even necessary, there are other approaches)

Question 4 - Drawing of Spark diagrams (5 points)

Draw the Spark diagram for your solution for Question 3.

Question 5 - Using Spark for ML (8 points)

You want to build a recommendation model with Spark, for recommending new songs to users based on their ratings on past songs. There are many ways to answer this question, as well as many definitions on what would constitute a good recommendation. You are free to choose, and experiment with different ones. One possible approach is to model the users/songs ratings as a sparse matrix, and then run classical data mining/machine learning algorithms, such as K-Means or K-nearest neighbor, to find possible answers. Mllib is a Spark library that can be used for this exercise.

Question 6 - Find the most frequently played song (10 points)

Write code in Spark to find the id of the song that was played the most times in the dataset. If more than one such song exists with an equal number of plays, return all of them.

Question 7 - Building a Count-min sketch in Spark (10 points)

Build a Count-min sketch that allows you to summarise the dataset, and based on it, be able to estimate the **number of times** each song was played. The sketch should be constructed in a distributed fashion, within Spark. The sketch can be subsequently collected and queried at the driver machine, i.e., the querying part does not need to be distributed. You are expected to build the Count-min sketch from scratch.
Indicative questions to think about for this question: (a) how does the size of the sketch influence efficiency and accuracy, (b) how can this sketch be represented, in order to increase

efficiency, (c) what properties should the hash functions of the sketch adhere to, (d) how would you change your code, if you needed to estimate the number of times each song was played by each user?

Question 8 - Building a Count-min sketch in Spark Streaming (10 points)

Change your answer at question 7 such that the dataset is read from a stream. You can simulate a streaming input by using the provided code in streamALargeFile.java.

Question 9 - Experimenting with Count-min sketches that are built with different (ε,δ) parameters (5 points)

Re-run your solution of Question 6 with Count-min sketches configured with the following (ε,δ) values:
Case 1: (ε,δ)=(0.1, 0.1)
Case 2: (ε,δ)=(0.01, 0.1)
Case 3: (ε,δ)=(0.1, 0.01)
Use the sketches to estimate some queries (i.e., number of times each song was played). Try with the ids of the songs that were played many times (the answers of question 6) but also with other random song ids. How does (ε,δ) affect the accuracy, for each case?

Question 10 - Finding the most frequently played song (15 points)

Think how you could modify your solution of Question 7 such that it allows you to find the most frequently played song, again using a Count-min sketch. Draw the Spark diagram. You do not need to implement it in code if you are confident that you can do it, but it is useful to think about it.

Question 11 - Examining all subsets of 3 users (20 points)

You are asked to examine all combinations of 3 users, to find the subset of users that listened to the most distinct songs. You are free to use a sketch, if you think this will help.
Hint 1: This is a difficult problem to solve efficiently, because the search space (the number of combinations of users) is fairly high on this dataset. It is fine if your solution can handle a smaller part of the dataset.
Hint 2: If you want to (think how to) make your solution more efficient: how many combinations of three users exist? Is it manageable to examine all these combinations exhaustively, or do you need methods to prune some combinations early? Is there an easier way to do so? Try to optimize your solution (either by changing your algorithm, or by optimizing your code) such that

it can handle as many users as possible. If you come up with good ideas, do discuss them in the class when we discuss this problem (even if you did not fully implement or test them)!
Hint 3: An approximation would also be sufficient.

Use of ChatGPT and other assistants: As discussed during the first week, and according to the course's policy. You are allowed to use ChatGPT and similar tools, in the same way you would use a search engine, e.g., to find the reason for an exception. These tools can be used to facilitate your learning process, not to answer your homework. As a sanity check, you can think of the following question: if I ask this question to my teacher, or to a teaching assistant, will they answer? The teaching assistant and your teacher will never give you an answer that 'solves' the exercise, as this would defeat the learning goals of the exercise. They can help you by providing a different perspective, or by explaining to you why your code is very slow, or why it fails. You can use these tools in the same way. Another way to use these tools is to see alternative solutions, **after** you solve and submit your exercise (Keep in mind that ChatGPT often errs, but in a very convincing way. Therefore, the answers you get might as well be wrong.). In all cases, you are expected to fully understand the solution, to the point that you will be able to apply it in a different context (and in the exam), without the help of such tools.

An example: Your teacher and teaching assistants would never answer a question: "Write me code that does XYZ", as this answer will not help you learn – it will only help you evade the necessary hands-on practice. They would, however, look into the following question, and try to help you: "I am running Spark on my computer, but even the word-count demo code that I downloaded from the Spark webpage does not seem to use more than one core. What might be the problem?"