Big Data Management Project 3

Spring 2017

Team 18 Caitlin Kuhlman, Wenlei Cao

Code Organization:

* Project3
  + Problem1
  + Problem2

**Problem 1. Spark SQL (transaction data processing)**

**To run:**

The code is in Scala.

It first create a RDD from a local file. Therefore, you need to change the path to your test file location.

The code can be run sentence by sentence, or by block

Example:

If your file located at home/tmp/transactions.csv

You need to change

val T = spark.sparkContext.textFile("file:///home/mqp/Documents/transactions.csv")

to

val T = spark.sparkContext.textFile("file:/// home/tmp/transactions.csv ")

Then you can run the rest of code

**Explanation:**

First, create a RDD from transaction.csv.

Next, in order to be able to use Spark SQL query, we add schema (column name) to the RDD and transform it into a dataframe. Function, createOrReplaceTempView, is used to create a view for Spark sql query.

Last, we apply SQL query per question’s requirement for filter, aggregation, join.

**Problem 2. Grid cell of high Relative Density index**

**To run:**

The main code is in Scala.

It first create a RDD from a local file. Therefore, you need to change the path to your test file location.

The code can be run sentence by sentence, or by block

Example:

Change local file path, the same as previous example

**Explanation:**

It contains 3 separate Java class

DataGenerator class generates point per requirements, export a csv file

Pointcellmap class take the point generated from DataGenerator and create a mapping between point and cell number, export a point x, y, cell csv file

Neigborcells class create a mapping between each cell and its neighbor cells. Because the 4 boundary’s neighbor is different, this Java code take care of left side and right side difference. The upside and downside difference was left later to Spark SQL to resolve. Execute the Java code will export cell, neighbor cell csv file

In scala code,Pointcellmap and NeighborCells csv files were converted to RDD and further converted to dataframe.NeighborCells were filter out number < =0 or number >250000 to take care of upside and downside neighbor cell number. In the code, we call RealNeighborCellMap

To calculate the relative density index

We need x.count/ avg (y1.count, y2.count…yn.count)

Based on pointcellmap, we are able to calculate point count in each cell. Final table schema: cell, pointcount. (1)

Joining pointcellmap and neighborcells, we are able to calculate each cell, its neighbor cell point count and neighbor cell number. (2)

Now, by joining (1) and (2), we can calculate the relative density index for each cell, we can order it by index value descendently. Output only top 50. This is the answer for step 2. (3)

From (3), we know the top50 cell number, we can join RealNeighborCellMap to know what neighbor cell number they have, then the neighbor cell number further join (1) and (2) to calculate neighbor cell relative density index. This is the answer for step 3