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:**

1. DataGenerate.java is generating point data. Each point has X and Y value. If you already have point file, you can skip this step.
2. Pointcellmap.java is used to generate point and cell mapping .You need to change the source file path in the java code to your test point file directory and destination file to your desired file saving location. The generated file contain X, Y and cell number. This file will be used in scala code
3. Neighborcell.java is used to generate cell and neighbor cells mapping. Also change the destination file path in code to your desired path. The generated file contain cell, neighbor cell. This file will be used in scala code
4. Copy the above point cell mapping and cell-neighbor cell mapping file to spark Visual machine.
5. To run scala code, you need to turn on visual machine and follow the readme instruction on desktop to scala shell.
6. Change the file path in the scala code to your corresponding file directory you copied to. Now you can run code sentence by sentence or by block
7. The top 50 Relative density index (RDI) point is showed in the format of cell, RDI , ordered by RDI value desc
8. The last question’s result is showed in format of cell, neighbor cell and RDI, ordered by cell number

Example:

In java

Your test file is at C:\ test.csv. the same apply for destination path, if java need to write to a file

fr = new FileReader("D:\\points.csv") change to fr = new FileReader("C:\\ test.csv ")

In scala, if your point map file is at home/mqp/desktop/pointmaps.csv

val PC = sc.textFile("file:///home/mqp/Documents/pointsMap.csv")

change to

val PC = sc.textFile("file:/// home/mqp/desktop/pointmaps.csv ")

**Explanation:**

It contains 3 separate Java class

DataGenerator class generates point per requirements, export to 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 Scala code to resolve. Execute the Java code will export cell, neighbor cell csv file

We first generated point-cell mapping file and cell-neighbor cell mapping file in java.

Files are copied to virtual machine and later imported as RDDs.

Please note cell-neighbor cell mapping RDD is further filtered to remove the cell number <=0 and >250000. To correct the upper and lower boundary cell's neighbor cell. This new RDD will be used for calculation

In order to get elements ready to calculate relative density index (RDI), we need 3 elements.

(1). each cell's point count; (2). Each cell's neighbor cell count; (3). Each cell's neighbor cell point count total

From Point-cell mapping RDD, we can get cell, point count by countbykey function (1)

From cell-neighbor cell mapping RDD, we can get cell, neighbor cell count by countbykey function (2)

For each cell's neighbor cell point count, we need to join (1) with cell-neighbor cell mapping RDD. The join happened on cell-neighbor cell mapping RDD's neighbor cell column and (1)'s cell column. Therefore, cell-neighbor cell mapping RDD need to adjust the key value position to form a new RDD to do join. The RDD of join product is mapped to a new RDD with two tuple field (cell and point count). The total neighbor cell point count is calculated by reducebykey function by each cell (3)

We can calculate RDI by join (1), (2) and (3) and use their calculated data

The formula is (1)/((3)/(2))

In the formula (1) is current cell point count. (3)/(2) is average cell point count from current cell's neighbor cell

The above result RDD is (4)

We can then take top 50 of RDI cell from (4) by takeordered function. This is the answer for question 2 (5)

For last question, we need to join (5) to cell-neighbor cell mapping RDD so that we get corresponding neighbor cell for those 50 top RDI cell. (6)

Then we use (6) to join (4) to get neighbor cell RDI.

The result is showed as cell, neighbor cell and neighbor cell's RDI, sort by cell number