Project 2: Hot Spot Analysis

Reflection

This project focused on using big data technologies viz. Spark, Scala, and Hadoop for spatial data analysis. I did 2 analysis: Hot zone and Hot cell. Hot zone analysis required to calculate the hotness of the rectangles. Hot cell analysis required to apply spatio-temporal statistics to the given New York monthly Yellow Taxi Dataset from 2009 to 2012 to identify the statistically significant spatial hot spots.

Lessons Learned

I learned to use Scala language and used it to write programs and run them using Spark and Hadoop. I gained a good understanding of the spatial data, how to load it, process it and run queries on the dataset for the analysis. I also learned to use the NVIDIA Rapids library which I used to load the same `.csv` files used in the project. I used the NVIDIA Rapids library since it allows to load the entire dataframe in the GPU memory and it is very fast compared to the usual Pandas Dataframe. I was quickly able to see the columns of my datasets and debug my code easily. I also got a nice introduction to using spatio-temporal statistics by using the Getis-Ord Statistic

Methodology

Hot zone analysis

The range join operation on rectangle and point datasets was already done. I implemented the `ST_Contains` function in `HotzoneUtils.scala`, that returned the number of points located within each rectangle.

```
def ST_Contains(queryRectangle: String, pointString: String): Boolean = {
// Extracting point and rectangle coordinates
val rectangle = queryRectangle.split(",")
val point = pointString.split(",")
val rectangleX1 = rectangle(0).toDouble
val rectangleY1 = rectangle(1).toDouble
val rectangleX2 = rectangle(2).toDouble
val rectangleY2 = rectangle(3).toDouble
val pointX = point(0).toDouble
val pointY = point(1).toDouble

// Checking if a given point falls inside the rectangle
return pointX >= rectangleX1 && pointX <= rectangleX2 && pointY >= rectangleY1 &&
pointY <= rectangleY2
}
```

Then in `HotzoneAnalysis.scala` file, I counted the number of points in each rectangle and sorted the result.

```
// Count the number of points in each rectangle and then sort the result
val resultDf = spark.sql("select rectangle, count(point) as pointCount from joinResult
group by rectangle order by rectangle")
return resultDf
```

Hot spot analysis

I loaded the NYC taxi trip data into Spark dataframe from the given CSV file. By using the functions in `HotcellUtils.scala` , I transformed the data to assign spatial (x, y) and temporal (z) coordinates to each data point.

```
// Calculate number of pickups for each cell
val pickupsByCell = spark.sql("""
  SELECT
    x, y, z,
    COUNT(*) as pickup_count
  FROM pickupInfo
  GROUP BY x, y, z
""")
pickupsByCell.createOrReplaceTempView("pickupsByCell")
```

I wrote the SQL query to calculate the number of pickups for each cell and identified each cell's spatial neighbors.

```
// Find the sum of neighbor values (26), excluding the cell itself
val neighborsSum = spark.sql("""
  SELECT
    a.x as x, a.y as y, a.z as z,
    (SUM(b.pickup_count) - a.pickup_count) as neighbor_count
  FROM pickupsByCell a, pickupsByCell b
  WHERE
    b.x BETWEEN a.x - 1 AND a.x + 1 AND
    b.y BETWEEN a.y - 1 AND a.y + 1 AND
    b.z BETWEEN a.z - 1 AND a.z + 1 AND
    (a.x <> b.x OR a.y <> b.y OR a.z <> b.z)
  GROUP BY a.x, a.y, a.z, a.pickup_count
""")
neighborsSum.createOrReplaceTempView("neighborsSum")
```

Then, I computed the Getis-Ord statistic according to the given formula for each cell to identify spatial hotspots. I considered 26 neighbors for each cell and equal weight (=1) for each cell as mentioned. This involved calculating the global mean and standard deviation of the pickup points and then applying the Getis-Ord formula. Finally, I returned the top 50 hotspots based on the highest Getis-Ord scores.

```
// Calculate global mean and standard deviation and Getis-Ord statistic for each cell
val globalSum = pickupsByCell.agg(sum("pickup_count")).first().getLong(0)
val globalSumSqr = pickupsByCell.agg(sum(pow("pickup_count", 2))).first().getDouble(0)
val meanValue = globalSum.toDouble / numCells
val stdDevValue = math.sqrt((globalSumSqr / numCells) - (meanValue * meanValue))

val getisOrdStats = spark.sql(f"""
  SELECT
    x, y, z,
    ((neighbor_count - (${meanValue} * 26)) / (${stdDevValue} * SQRT((26 * ${numCells}
- 26 * 26) / (${numCells} - 1)))) as getis_ord
  FROM neighborsSum
""").orderBy(desc("getis_ord")).limit(50)

return getisOrdStats
```

Result & Discussion

There were lot of version mismatch in Java and other libraries so I learned the process of troubleshooting and debugging. This also highlighted the importance of aligning development environments and carefully managing file paths, especially in virtualized setups like VirtualBox. The result outputs using the generated `.jar` file, gave the required outputs. The use of Spark and Scala was effective in hangling big data while doing complex spatial-temporal analyses.