Lin_Susan_CSE511_Hot Spot Analysis Project Report

Reflection

To approach this project, I first installed the required software—Scala, Spark, and sbt—following the project guidance. From the project overview, I understood the context, which was performing hotspot analysis on NYC taxi trip data. By examining the example data inputs and code, I identified the expected outputs for both hot zone and hot cell analyses, giving me a clear understanding of the project's objectives.

Next, I delved deeper into the Getis-Ord statistics concepts by studying the ACM SIGSpatial Cup 2016 Problem Definition page. This step was particularly challenging, as it involved understanding the mathematical formulation of the G*-statistic and the criteria for defining neighboring cells in a 3D spatial and temporal space. I spent time grasping how neighbors contribute to the statistical evaluation of each cell's "hotness," as well as the implications of spatial and temporal correlations.

After solidifying my understanding of the statistical foundation, I implemented the required analyses in Spark. For the hot zone analysis, I developed a function to parse input data and determine if points were contained within specified rectangular zones. For the hot cell analysis, I structured the workflow to compute trip counts for 3D cells, identify neighbors, and calculate the G*-statistics. Debugging and validating each step of the implementation ensured that the results matched expectations. This iterative approach allowed me to successfully complete the project while enhancing my understanding of both spatial data analysis and distributed computing techniques.

Lessons Learned

This project taught me valuable skills in spatial and spatiotemporal data analysis. By implementing the **ST_Contains** function, I gained a deeper understanding of how to determine whether a point lies within a specific rectangular region. This introduced me to the fundamentals of GIS and its applications in analyzing geographic relationships. Additionally, working with rectangular zones and parsing latitude and longitude boundaries strengthened my ability to process and manipulate coordinate-based data efficiently.

The project also emphasized the importance of aggregating data across spatial and temporal dimensions. By defining 3D cells with latitude, longitude, and time, I learned how to analyze dynamic systems, such as ride-sharing demand, over time and space. Implementing the G*-statistics for hotspot analysis helped me understand how to measure spatial clusters by

leveraging neighbor data, which is crucial for detecting patterns in large datasets. These statistical insights, combined with techniques for identifying neighboring cells and calculating aggregated metrics, provided a solid foundation for spatiotemporal analysis.

Lastly, I enhanced my ability to work with large-scale datasets using SQL and Spark, particularly through operations like Cartesian products and user-defined functions. These tools enabled me to efficiently process data and analyze relationships across multiple dimensions. Beyond technical skills, this project demonstrated the practical applications of hotspot analysis in domains like transportation optimization, urban planning, and disaster management, highlighting the versatility of these techniques for real-world problem-solving.

Implementation

Hot Zone Analysis

In hot zone analysis, we want to find the number of points in each zone, which can be used to decide the zone's hotness. For each point, we rely on the **ST_Contains** function to check whether a given point lies within a rectangular region.

ST_Contains Parameters:

- **queryRectangle**: A string representing the rectangular zone's boundaries. It consists of 4 floating-point numbers separated by commas:
    - The first two numbers represent the latitude and longitude of the lower-left corner of the rectangle.
    - The last two numbers represent the latitude and longitude of the upper-right corner.
    - For example, "1.0,1.0,3.0,3.0" represents a rectangle with: Lower-left corner at (1.0, 1.0); Upper-right corner at (3.0, 3.0)
- **pointString**: A string representing the coordinates of a point in the form of two floating-point numbers separated by a comma. For example, "2.0,2.0" represents a point with: Latitude = 2.0, Longitude = 2.0

ST_Contains Implementation:

1. **Parsing the Rectangle Coordinates**: Split the **queryRectangle** string by the comma delimiter to extract the four bounding coordinates
2. **Parsing the Point Coordinates**: Split the **pointString** string by the comma delimiter to extract the two coordinates of the point.
3. **Containment Check**: A point lies inside the rectangle if:
    - Its **x-coordinate (latitude)** lies between the rectangle's minimum and maximum x-coordinates
    - Its **y-coordinate (longitude)** lies between the rectangle's minimum and maximum y-coordinates
    - The function returns **true** if both conditions are satisfied, otherwise **false**.

Hot Cell Analysis

The **hot cell analysis** implemented here aims to identify the top 50 "hottest" cells for ride pickups within a specified spatial and temporal range. A cell is defined as a 3D unit with dimensions x, y, z, where x and y represent latitude and longitude, and z represents the date. The hotness of a cell is evaluated using the G-statistics, a statistical measure based on the spatial distribution of values across neighboring cells. Here's an expanded and detailed explanation of the implementation:

1. **Defining the Analysis Scope**:
    - A spatial and temporal range is defined by [minX, maxX, minY, maxY, minZ, maxZ]
    - Each cell corresponds to a specific coordinate (x,y,z), with a unit size of 1.
2. **Counting Trips Per Cell**:
    - The first step computes the number of ride pickups **trips_cnt** for each cell using an SQL query.
    - The query filters data based on the defined range and groups it by (x,y,z).
    - Result: A **tripsPerCell** view containing (x,y,z) and their respective trip counts.
3. **Calculating Mean and Standard Deviation**:
    - The mean and standard deviation of the trip counts are calculated across all cells.
    - Note cells with 0 trips are also considered in the calculations to ensure uniformity.
4. **Identifying Neighbor Data**:
    - A Cartesian product of **tripsPerCell** with itself is performed to identify neighbors of each cell.
    - Neighbors are defined as cells that are within a unit distance (±1) in all three dimensions including the same cell.
    - For each cell (x,y,z) the neighbor data includes:
        - The sum of trips in neighboring cells **sum_neighbour_trips**
        - The count of neighboring cells **num_neighbors**
    - Result: A **neighborData** view containing
5. **Computing G-Statistics*:**
    - The G*-statistic for each cell is calculated using the formula:

$$G^* = \frac{\text{sum\_neighbor\_trips} - \mu \cdot \text{num\_neighbors}}{\sigma \cdot \sqrt{\frac{\text{numCells} \cdot \text{num\_neighbors} - (\text{num\_neighbors})^2}{\text{numCells} - 1}}}$$

    - This is implemented as a user-defined function (UDF) in Spark to process the neighbor data efficiently.
6. **Ranking and Output**:
    - The G*-statistics are calculated for each cell, and the cells are ranked in descending order based on their scores.
    - The top 50 cells with the highest G*-scores are selected as the "hottest" cells.
    - The results are displayed and optionally saved for further analysis.