CSE 511: Data Processing at Scale
Report on Project 1- NoSQL and Project 2- Hot Spot Analysis

Rohan Reddy Sambidi
School of Computing and Augmented Intelligence
Arizona State University
Tempe, USA
rsambidi@asu.edu

***Abstract*—This document is a report on course projects from CSE 511 course. This course comprised two projects: Project 1- NoSQL, and Project 2- Hot Spot Analysis. Project 1 involved implementing Python functions to perform specific operations on a provided NoSQL database. Project 2 involved Apache Spark and Scala to employ spatial analysis techniques to assess and quantify hot spots in spatio-temporal big data on New York City taxi trips. These projects have been completed individually without any collaboration.**

***Index Terms*—NoSQL database, Hot Spot Analysis, Haversine formula, Getis-Ord Gi* statistic, space-time cube**

I. Introduction

The CSE 511 course comprised two projects- NoSQL and Hot Spot Analysis- that enhance skills in handling, analyzing, and interpreting real-world data. The NoSQL project aims for utilizing existing NoSQL database tools to perform task-specific data retrieval. The Hot Spot Analysis project aims at using cluster computing systems, such as Spark, to carry out database operations on large-scale data.

A. Project-1: NoSQL

The objective of this project is to perform query operations on NoSQL databases, and retrieve the results. A NoSQL database, which contains data of various businesses, is provided. The task involved in this project was to implement two Python functions that can each execute a specific query on the database. The first function, `FindBusinessBasedOnCity(...)`, returns all businesses in a given city. The second function, `FindBusinessBasedOnLocation(...)`, returns all businesses that lie within a certain radius of a given location coordinates.

1) `FindBusinessBasedOnCity`: This function takes three parameters— `cityToSearch`, `saveLocation1`, `collection`— and retrieves all the businesses from the data (`collection`) that are located in a certain city (`cityToSearch`), and then saves them to a file at a certain location (`saveLocation1`).

2) `FindBusinessBasedOnLocation`: This function takes five parameters— `categoriesToSearch`, `myLocation`, `maxDistance`, `saveLocation2`, `collection`— and retrieves all the businesses from the data (`collection`) that belong to a specified category (`categoriesToSearch`) and are located within a certain distance (`maxDistance`) from a given location (`myLocation`), and then saves them to a file at a certain location (`saveLocation2`). The Haversine formula is used to calculate distance between two coordinates/locations.

B. Project-2: Hot Spot Analysis

The objective of this project is to perform Hot Spot Analysis on spatial and spatio-temporal data. The data is a collection of NYC Yellow Cab taxi trip records. Only taxi pick-up locations are considered for the analysis. This project has two parts: Hot Zone Analysis and Hot Cell Analysis.

1) Hot Zone Analysis: The goal of this analysis is to identify hotness of a given rectangular region. A region is defined by two coordinates that correspond to two diagonally opposite vertices of a rectangle. Given a few rectangular regions and a set of coordinates as points, the task is to count the number of points that lie within each of those regions. Greater the number of points within a region, higher will be it's hotness.

2) Hot Cell Analysis: The goal of this analysis is to apply spatial statistics to spatio-temporal data to identify statistically significant spatial hot spots. Unlike Hot Zone analysis, this analysis includes temporal dimension. The task is to identify significant clusters in space-time cube based on taxi pick-up times and locations using Getis-Ord Gi* statistic. This task is inspired from the 5^{th} ACM SIGSPATIAL GIS CUP (2016) [1].

II. Solution Approach and Explanation

A. Project 1- NoSQL

The NoSQL database provided for the first project is a collection. A collection is an ordered list of JSON objects (records) [4]. Each record in the data contains various attributes of a unique business/establishment such as name, city, latitude, longitude, stars, etc. Fig. 1. shows the business locations colored with respect to the *city* attribute. The *collection* class available in *UnQLite* library for Python is used for handling the data.

To find businesses based on a given city (i.e., to complete the `FindBusinessBasedOnCity(...)` function) the data needs to be searched for businesses whose *city* attribute matches the city name. The first step to accomplish this is to use the *collection.all()* method to iterate over the

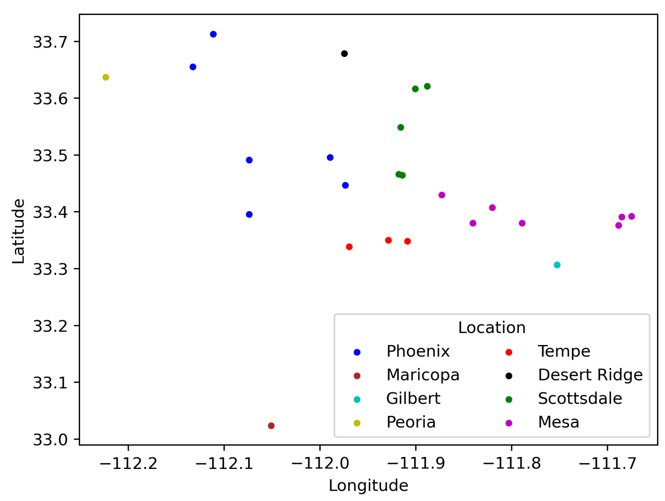

Fig. 1. Coordinates of the businesses from the NoSQL Database

data, which is the argument passed into `collection` parameter. This method fetches each entry from the data as a dictionary, whose keys are the data attributes. For each dictionary fetched, a conditional statement is used to check the value of the *city* key. If the value matches with the argument passed into `cityToSearch` parameter, the corresponding business data is retrieved. The expected output of `FindBusinessBasedOnCity(...)` function is a text file containing business's *name*, *full_address*, *city*, and *state* attributes separated with a '$' symbol. These attributes of retrieved businesses are appended to a text file, which is saved at the location argument passed into `saveLocation1` parameter.

To find businesses belonging to given categories within a certain radius of a given location (i.e., to complete the `FindBusinessBasedOnLocation(...)` function), distance between the businesses and the given location is calculated. This is done using the Haversine formula. Haversine formula calculates the great-circle distance between two points– i.e., the shortest distance over the earth's surface [2]. It is defined by equations (1), (2), and (3):

$$a = sin^2(\Delta\varphi/2) + cos\ \varphi_1 \cdot cos\ \varphi_2 \cdot sin^2(\Delta\lambda/2) \quad (1)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \quad (2)$$

$$d = R \cdot c \quad (3)$$

where φ is latitude (in Radians), φ is longitude (in Radians), and R is earth's radius (mean radius = 3959 miles) [2]. The value of d gives the required distance in miles.

Similar to the previous task, *collection.all()* method is used to iterate over the data. For each business dictionary, the value (a list) of the *category* key is compared against the list argument passed into `categoriesToSearch` parameter. The elements in category list describe the type of business (such as restaurant, spa, Shopping, etc.). If there is at least one element common in these lists, the corresponding business is retrieved for calculating distance from the given coordinates, which are passed through `myLocation` parameter. If the calculated distance is less than or equal to the given distance, viz., `maxDistance`, the business's name is appended to a file, which is saved at the location given by `saveLocation2` parameter. Unlike the previous task, the expected output of the `FindBusinessBasedOnLocation(...)` function is a text file containing business names alone.

B. Project 2- Hot Spot Analysis

The second project undertakes a more extensive application of spatial analysis. The data used for this project contained attributes from NYC taxi trips from January of 2009. Only pick-up points, and the date and time of pick-ups are used for this project. For Hot Zone Analysis, a sample of 10,000 entries from the data is used. For Hot Cell Analysis, all 100,000 data entries are used. Due to the huge amount of data, Apache Spark is used to process the computation. The programs executed on top of Spark framework are written in Scala, as suggested in the ACM SIGSPATIAL competition [1].

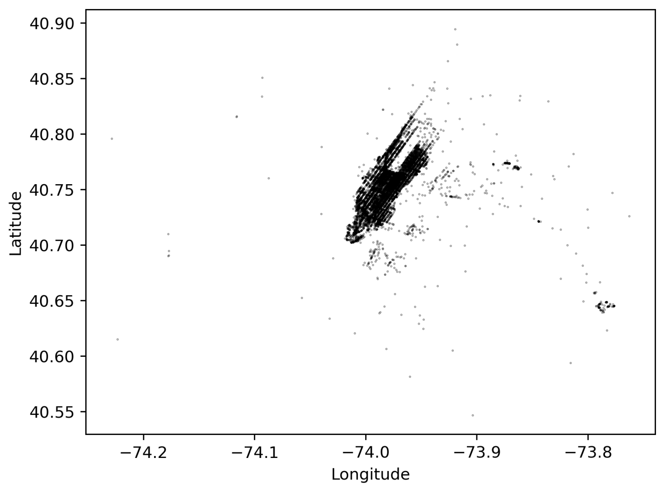

Fig. 2. A sample of 10,000 coordinates of NYC taxi pick-ups in Jan 2009

1) Hot Zone Analysis: The idea of this analysis is to perform range join operation on a rectangle data set (zones) and a point data set (pick-up locations). A zone data set is provided that contains the coordinates of the (rectangular) zone's diagonal. The point data is extracted using the *pick-up* feature from the NYC taxi data. Fig. 2. visualizes the plot points data.

The first step in this analysis is to complete a Scala object's method (named `ST_Contains(...)`) that checks whether a given point lies within a given rectangular region. Since it is provided that the region's edges are aligned with the X and Y axes, comparing the X and Y coordinates (longitude and latitude) of the region's vertices with those of the point is sufficient to determine whether the point lies within the region. If the values of X and Y coordinates of the point are in between the values of X and Y coordinates of the given region, then the point will lie inside the region. Implementing this logic completed the `ST_Contains(...)` method.

The next step in this analysis is to complete another Scala object's method (named `runHotZoneAnalysis(...)`) that performs range join operation to determine the hotness of given regions. This method reads the rectangle and point data, runs an SQL query to join them, and then runs another SQL query to count the points in each rectangle/zone. The method finally returns a csv file containing rectangles and their corresponding points count (hotness), arranged in ascending order of the rectangles. The conception of this analysis can be visualized in Fig. 5.

2) Hot Cell Analysis: This analysis focuses on identifying hot spot cells in space-time cube of the taxi pick-ups. The coordinates (latitude and longitude) along with a temporal dimension creates a space-time cube of the pick-up data. The spatial neighborhood for each cell in the cube is determined by neighboring cells within a grid formed by uniformly subdividing latitude and longitude [1]. For this project, each cell size is 0.01*0.01 degrees in terms of latitude and longitude. A spatial relationship is established between the time periods that precede, coincide with, and succeed the current time period [1]. A time-step size of 1 day is used for this analysis. The cell containing higher number of pick-up points will be hotter. Fig. 3. depicts how a space-time cube is created and used for hot spot analysis.

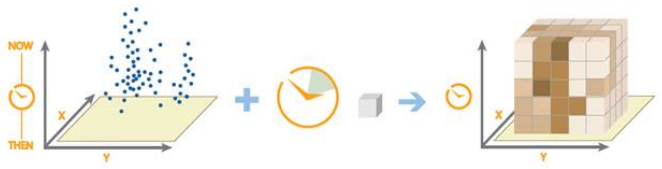

Fig. 3. Creation of a space-time cube (credit: ACM SIGSPATIAL [1])

The first step in conducting this analysis is to complete a Scala object by defining various methods for that object. Two methods were defined: `GetNeighbourCount(...)` and `GetisOrd(...)`. The former method counts the number of neighbors of a given cell based on its position in the space-time cube. Since the data used for this analysis only corresponds to January 2009, the cube is bounded between Jan 1 and Jan 31 in temporal dimension. The number of neighbors of the cells vary based on its position in the cube. The latter function computes the Getis-Ord Gi* statistic of a cell. It is defined by the equations (4), (5), and (6):

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{n \sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j}\right)^2}{n-1}}} \tag{4}$$

where x_j is the attribute value for cell j, $w_{i,j}$ is the spatial weight between cell i and j, n is the total number of cells, and

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n} \text{ , is the mean of attributes} \tag{5}$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - \bar{X}^2} \text{ , is the std. dev. of attributes} \tag{6}$$

For simplicity, all the spatial weights are set to be 1. The Gi* statistic is a z-score. For statistically significant positive z-scores, the larger the z-score is, the more intense the clustering of high values (hot spot) [3].

The next step in this analysis is to complete another method (named `runHotcellAnalysis(...)`) that applies the Gi* statistic to retrieve hot cells from the data. A cell's boundary is used to retrieve all the points (pick-ups) that lie within the cell. SQL queries are used to retrieve and count the points in each cell. Then, the neighbors of each cell are determined, and the valid neighbors with at least one pick-up point are counted. These steps are also executed as join operations using SQL queries. Proceeding further, the Gi* statistic is calculated for each cell. Finally, an SQL query is used to order the cells in descending order of their Gi* score. The method then creates a csv file containing the coordinates of the 50 hottest cells.

III. Results

A. Project 1- NoSQL

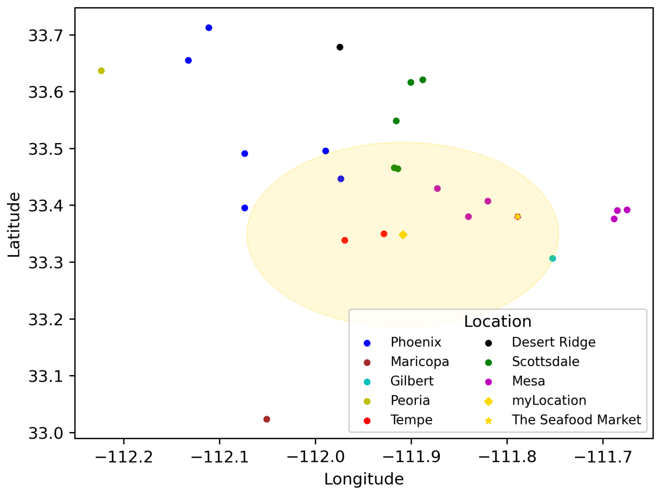

Fig. 4. Conception of `FindBusinessBasedOnLocation`

As explained in section II-A, the results for this project are two text files created by `FindBusinessBasedOnCity` and `FindBusinessBasedOnLocation`. Fig. 1. visualizes the conception of `FindBusinessBasedOnCity(...)` function. The function returns the desired attributes of businesses located in a given city. Fig. 4. shows a test case of the `FindBusinessBasedOnLocation(...)` function. Given a location (gold diamond in the plot), the function was asked to return the names of all businesses within 10 mile radius that fall under "Specialty Food" category. The gold area in the plot represents the region to be considered. The result was only one business- The Seafood Market (marked as gold star in the plot). A text file is returned contained the business's name.

B. Project 2- Hot Spot Analysis

As explained in section II-B, the results for this project are two csv files, one from Hot Zone Analysis and one from Hot Cell Analysis. The output file of Hot Zone Analysis contains coordinates sorted in ascending order. The output file of Hot Cell Analysis contains coordinates of cells, sorted in descending order of their hotness (Gi* statistic).

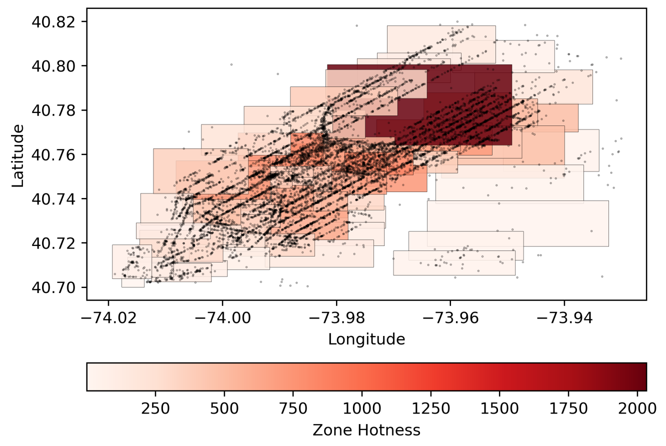

Fig. 5. Visualization of a subset of points and zones, showing the hotness

TABLE I
FIRST FIVE ROWS FROM HOT ZONE ANALYSIS OUTPUT

-73.789411,40.666459,-73.756364,40.680494	1
-73.793638,40.710719,-73.752336,40.730202	1
-73.795658,40.743334,-73.753772,40.779114	1
-73.796512,40.722355,-73.756699,40.745784	1
-73.797297,40.738291,-73.775740,40.770411	1

1) Hot Zone Analysis: Fig. 5. visualizes a subset of the rectangle zones. This visualization provides an overview of the Hot Zone Analysis. The black points in the plot represent pick up locations. The face color of the zones is mapped to their hotness. The red zone has the highest number of pick up locations, making it the hottest zone. In the plot, the hottest zone had 2033 pick up locations.

Table I shows a sample from the Hot Zone Analysis output file. The first column contains the coordinates of the rectangles (zones) as strings, and the second column contains the number of points (pick ups) in that zone. The rows are sorted by the first column.

TABLE II
FIRST FIVE ROWS FROM HOT CELL ANALYSIS OUTPUT

-7399	4075	15
-7399	4075	29
-7399	4075	22
-7399	4075	28
-7399	4075	14

2) Hot Cell Analysis: Table II shows a sample from the Hot Cell Analysis output file. The first and second columns contain the spatial coordinates (longitude and latitude) of the cells, while the third column contains the temporal coordinate (day of the month). The rows are sorted in descending order of Gi* statistic, which is not included in the table.

IV. DISCUSSION AND TAKEAWAYS

The two projects were completed individually without any collaboration. I started the first project by getting familiar with the *UnQLite* library in Python (see [4]). Realizing that the provided data is a JSON document collection, I learned about the *collection* class available in *UnQLite* library. I also learned about the various methods available for a *collection* object. By completing the first project, I gained hands-on experience in using the *UnQLite* library. I learned how to work with NoSQL databases— specifically, JSON document collections— and process queries using Python. I also got familiar with various methods available for a *collection* object that can be used to perform database operations like retrieving, updating, deleting, etc.

I started the second project with familiarizing myself with the configuration of data processing tools like Apache Spark. I looked into the fundamentals of Scala programming language and Apache Spark for hot spot analysis. I learned that Scala is preferred to Python for Spark because of its faster performance and better concurrency handling capabilities [5]. By completing the project on hot spot analysis, I learned how to use Apache Spark to analyze spatial data. I learned how to utilize the processing power of Spark and run SQL queries on it. I learned about various functionalities of Spark that allows running SQL queries seamlessly. This helped me achieve the desired outputs easily. I also gained understanding on various terminologies used by GIS engineers, and gained hands-on experience on how they analyze geo-spatial and geo-temporal data. I also learned that Hot Spot Analysis on taxi trips can provide valuable insights into spatial patterns and trends of the trips. It can be used for identifying popular pick-up (and drop-off) locations that can help taxi companies optimize service coverage. It can also be used for analysing taxi demand, improving traffic management, improving urban infrastructure, etc.

REFERENCES

[1] ACM SIGSPATIAL. (2016). ACM SIGSPATIAL GIS Cup Problem Definition [Online]. Available: https://sigspatial2016.sigspatial.org/giscup2016/problem
[2] C. Veness. (2019). Calculate distance, bearing and more between Latitude/Longitude points [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html
[3] Esri. How Hot Spot Analysis (Getis-Ord Gi*) works [Online]. Available: https://pro.arcgis.com/en/pro-app/3.1/tool-reference/spatial-statistics/h-how-hot-spot-analysis-getis-ord-gi-spatial-stati.htm
[4] Anon. (2014). unqlite-python Documentation [Online]. Available: https://unqlite-python.readthedocs.io/en/latest/quickstart.html#collections
[5] Simplilearn (2023, Mar. 30). Scala vs Python for Apache Spark: An In-depth Comparison [Online]. Available: https://www.simplilearn.com/scala-vs-python-article