MCS Portfolio Project Report

Maclay Teefey
CSE 511: Data Processing at Scale
Arizona State University
Tempe, Arizona, United States
mjteefey@asu.edu

Abstract— **Innovations in the storage and processing of large-scale geospatial data have enabled in-depth location-based analyses. This project report describes the data analysis methods used to process NoSQL location data and pickup location. Restaurant information stored in a NoSQL format, use city and coordinate data to find businesses within a specified city or within a specified maximum distance from the user's current location. New York City taxi information spanning years is analyzed using the coordinate data from each pickup to assess the popularity of various pickup locations in the city. The data is split into equivalently sized rectangular zones using coordinate data and equivalently sized cubic cells using the coordinate data combined with the time of the pickup. Each of the 165 zones, with at least one taxi pickup, found the total number of visits in each zone with the maximum discovered to be 2033 taxi pickups. The spatio-temporal cells are assessed for popularity using the Getis-Ord (g-star) statistic, and the highest scoring 50 cells are returned.**

Keywords— Data Processing, Geospatial Data, NoSQL, Apache Spark, Data Science

I. Introduction

Geospatial information has become an essential part of using the internet, including "check[ing] a news website or a weather app or look[ing] up directions to an appointment" [1]. The most common form of geospatial data is organized geographic information systems (GIS) and has become essential for many large-scale research projects and businesses [1]. According to Margaret Pizer, the National Science Foundation (NSF) awarded from 2021-2023 "about 180 grants, totaling more than $83 million, to support research related to GIS". This popularity has been supported by its ability to be used and stored in various formats including comma-separated values files (CSV), JavaScript Object Notation (JSON), Geographic JSON (GeoJSON), and other formats [2]. Because of the various storage formats, many companies have created various formats of NoSQL databases to store large amounts of information, including MongoDB and CosmoDB [3]. This report will cover two projects that process geospatial data in NoSQL and SQL formats. To help users find restaurants in their general area, I created two functions that will process a NoSQL database made up of restaurant information and will provide the restaurants within a specified city or within a distance limit specified by the user. To provide useful popularity data from New York City taxi pickup locations, I analyzed both coordinate data and coordinate data over time to determine which pickup locations were the most popular across the entire period or over small time periods across the dataset.

II. Explanation of Solutions

A. *Processing Resturant Data*

To reflect the use of NoSQL databases to store information without having a strict schema, the restaurant information uses a JSON format. Each restaurant includes the following attributes: 'buisness_id', 'type', 'state', 'latitude', 'name', 'full_address', 'categories', 'stars', 'city', 'neighborhoods', '__id', 'review_count', and 'longitude'.

B. *Finding Resturants Based On Their City*

To filter businesses based off a specified city, I wrote FindBusinessBasedOnCity. The function FindBusinessBasedOnCity is split into two parts: filtering the specified NoSQL collection based off of city names and then writing the filtered cities to a specified file path. To parse the NoSQL document, I used UnQLite filter function. The filter function selects which items should be included from a collection by using a function or lambda to return true if the item should be included and false if the item should not be included. To filter for businesses that should be selected because they are in the specified city, I set the lambda to only select businesses that have a 'city' JSON attribute equal to the specified city. To record the filtered businesses, I opened the file location to write specified by the user and looped through each business to record relevant information. For this function, the user needs to know the name of the business, the full address of the business, the city of the business, and the state the business is located in. For each row of the file, the businesses are stored in the format: "Name$FullAddress$City$State".

C. *Finding Resturants Based On Their Location*

The second task of this software was to report to businesses if they are within a specified maximum distance away and fit one of the specified lists of categories. This task required the creation of two functions: the main method, FindBusinessBasedOnLocation, and a distance calculation helper function, distanceFunction. The method FindBusinessBasedOnLocation is split into the same two parts as FindBusinessBasedOnCity: the filtering step and the writing step. Due to users expecting to find businesses in dynamic locations, the businesses' NoSQL collection does not store the distance from users as an attribute but instead stores their coordinates. These coordinates must be used to calculate the distance from the user's specified coordinates. The formulas used for distance calculation from coordinates cannot fit in one statement, so I wrote the function distanceFunction to return the distance from the business's coordinates to the user's coordinates. When programming distanceFunction, I assumed that the user would not need precise enough data to factor in hills and bumpiness that is not captured using a coordinate projection, so I utilized the haversine formulas to calculate the distances between two pairs of coordinates [4]. The haversine formula code, which was provided by the professor as part of this project, converts coordinate pair to radians and uses those coordinates in the radians with sine, cosine, and atan2 functions to calculate their distance apart. This distance value is checked against a maximum distance value specified by the user, so only businesses that are closer than the maximum distance is included in the filter. Once the distance from the business is calculated using distanceFunction, the business is checked to see if it fits under a list of categories provided by the user. The task of checking if a business list of categories shares a value with the specified list of categories can be accomplished in one line. By converting the business list of categories to a set, I was able to run the intersection function between the two lists, and then only include a business if the shared set between the business categories and the specified categories is not empty. Using the filter method discussed in the previous task, I set up the filter to only include businesses that pass both conditions. Once the list of businesses is filtered down to businesses that pass both conditions, I follow the same format as the previous task. However, the output written to the specified files follows the format of "Name" instead of "Name$FullAddress$City$State".

D. Processing Taxi Data

For the hot spot zone analysis project, I was provided with a sample New York City taxi company with geo-spatial data, and template code provided to load in the specific data used by the taxi company. Due to the ability to optimally load and manipulate data, I utilized Apache Spark and Scala to analyze the taxi company's data. This project was the first time I had utilized these tools, so I started by analyzing the template code in order to focus on the correct areas required to complete this project. The starter code uses Apache Spark's read, and SQL select statements to extract only the required data for each subtask. This is necessary because the data provided by the taxi company includes unnecessary data like the price of each trip. The template code is split into two mutually exclusive sections: HotZoneAnalysis and HotCellAnalysis.

E. Analyzing Hot Zones

For this project, I chose to start with the HotZoneAnalysis. The goal of the HotZoneAnalysis algorithm was to calculate the number of taxi pickups occurring in specific zones and return this information in a csv file. The HotZoneAnalysis code takes two inputs with the spark session information: pointPath and rectanglePath. The pointPath is the file location of the data for each pickup spot point, and the rectanglePath is the file location of the data for each zone, represented in rectangle form through two of the zone diagonal data points. To select all the data points in each rectangle, the HotZoneAnalysis code uses a function in HotZoneUtils called ST_Contains. ST_Contains, when a data point and a rectangle is input, can determine whether a point is contained within the rectangle. To implement this function, I split the code into three parts. Firstly, the point string and rectangle string are split using the split command on each comma in the string. The point string is split into point_x and point_y, and the rectangle string is split into corner_one_x, corner_one_y, corner_two_x, and corner_two_y values. Secondly, I use Scala's Math library to select the rectangles maximum and minimum coordinate values and assign them to variables: max_rect_x, min_rect_x, max_rect_y, and min_rect_y. Finally, I checked for each point coordinate values if it was greater than the maximum value of the coordinate or less than the minimum value of the coordinate. If either of those conditions were met for the point's x and y values, ST_Contains returned false. If none of the conditions were met, ST_Contains returned true. After selecting only, the points in a zone, I made sure that the result csv for the HotZoneAnalysis algorithm fits the required specifications by grouping points by their rectangle string and coalescing all the partitions into one file.

F. Analyzing Hot Cells

The HotCellAnalysis code determines the top 50 cells for popularity by using the G* statistic as a metric of popularity. Unlike the HotZoneAnalysis code, the HotCellAnalysis code tracks the z dimension, which corresponds to the pick-up time.

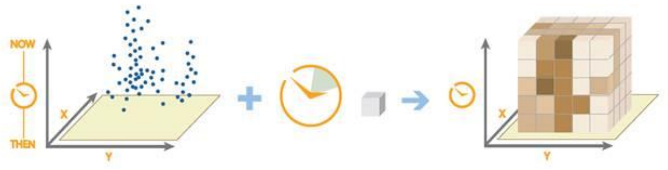

Fig. 1. Visual representation of the addition of time to create three dimensional cells from two dimensional points. [5]

The Gi* statistic formula uses neighboring cell values to calculate the popularity of certain cells, with the highest neighboring cell values resulting in the largest Gi* values. To run this formula, I created a helper function in HotCellUtils called calculateGetisOrd. This function takes in the neighbor sum, the neighbor count, the mean, standard deviation, and n value to calculate the formula (1).

$$Gi^* = (\Sigma(wij * xj) - \bar{X} * \Sigma(wij)) / (S * \mathrm{sqrt}((n*\Sigma(wij^2) - (\Sigma(wij))^2) / (n-1))) \quad (1)$$

To use this function, I needed to calculate all these values. Firstly, I gathered all of the valid cells from the data and calculated the number of pickup points in that cell. Once I got the cell counts, I calculated the global mean, standard deviation, and n value. Secondly, I calculated the sequence of values to connect the neighboring cells. This spanned 27 cell values from (-1, -1, 1) to (1, 1, 1) and were transformed into a dataframe called offsetsDF.

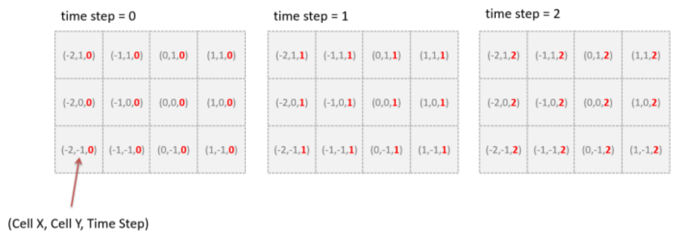

Fig. 2. Grid pattern of the combinations of cell values used in Gi-star calculation [5]

With all of these values, I joined the cell count data on the offsets dataframe by selecting values from each coordinate values added to the neighboring cell values. After gathering all of the cells with their neighboring cells, I calculated the neighborCount and neighborSums and used all of the information to add a column called gistar corresponding to the values found from running the rows in calculateGetisOrd. After each Gi* value was added, I ordered the data using gistar, and selected x, y, and z columns from the top 50 cells.

III. Description of Results

A. Resturant Analysis

The FindBusinessBasedOnCity and FindBusinessBasedOnLocation functions were tested using a sample business NoSQL database provided as part of the project. The test for FindBusinessBasedOnCity searched for businesses in the city of Tempe and wrote to output_city.txt three businesses: VinciTorio's Restaurant, Salt Creek Home, and P.croissants. The test for FindBusinessBasedOnLocation wrote to output_loc.txt only the business VinciTorio's Restaurant after searching for businesses that were categorized with the category "Buffets" and was 10 miles from the coordinates 33.3482589 latitude, -111.9088346 longitude.

B. Hot Spot Analysis

The HotZoneAnalysis and HotCellAnalysis functions were tested using a sample taxi company database provided as part of the project. The test for HotZoneAnalysis found 165 zones with more than one taxi pickup, with the most occurring at the rectangle (-73.981649, 40.764287, -73.949222, 40.800591) containing 2033 taxi pickups. The test for HotCellAnalysis found the top 50 cells in G score, with the cell (-7399,4075,15) having the highest G score.

IV. Description of Contributions

The two projects were completed individually, so this report and the code for the two projects were entirely done by me.

V. Self Reflection

To accomplish the creation of machine learning and clustering models for sensor data, multiple new skills were learned.

A. NoSQL Project

Through the process of programming this software, I learned how to process geolocation data, and furthered my understanding of processing NoSQL data. Through my previous work of processing data from the Nobel Prize API, I have learned how to process NoSQL data, as their API sent out the data using JSON. This process taught me how to navigate the various attributes in a JSON object, but I had utilized different libraries to process that data. UnQLite, a library to better process NoSQL data in Python, made the process of filtering data down to a small list of elements much easier than using loops to cycle through each data point or running the filtering through Pandas Dataframe objects. Through this filtering step, I gained more experience of using the lambda function in Python, because most of the datasets I would have used could have been done lazily through using loops to return the filtered values. This project provided my first instance of creating a distance function to calculate 3D distance. The haversine formula is an elegant form of calculating distances without having to factor in projections of the Earth or Earth's topology.

B. Hot Spot Analysis Project

This was my first time using Scala, and I was impressed by the good library functions for running joins and other large data calculations. It was also my first time using Apache Spark to package scala files into a jar file and then run the jar files using spark-submit. This process was much more difficult than it had to be for me due to the naming of the test file names switching from dash to underscore, and the inability to make spark-submit work without temporarily adding the path to the terminal window. I had previous experiences calculating and using the Getis Ord statistic, but it was useful for creating code to calculate realistic data. The calculation is much more complex when using a z dimension, and I had issues verifying my previous formula was incorrect because I didn't include the z score in the test files to see that I was clearly incorrect in my calculation. My previous formula did not include the cell itself and only included the neighbors, so I needed to script awkward joins to make the neighbor joins work well.

References

[1] M. Pizer, "Mapping science: How GIS transformed our view of the world | NSF - U.S. National Science Foundation." Accessed: Dec. 29, 2025. [Online]. Available: https://www.nsf.gov/science-matters/mapping-science-how-gis-transformed-our-view-world

[2] S. Greene and C. Rinner, "Examining the Value of Geospatial Open Data," in *The Future of Open Data*, DGO-Digital original., P. Robinson and T. Scassa, Eds., University of Ottawa Press, 2022, pp. 159–178. doi: 10.2307/jj.17610837.10.

[3] "What Is A JSON Database? | All You Need To Know," MongoDB. Accessed: Dec. 29, 2025. [Online]. Available: https://www.mongodb.com/resources/basics/databases/json-database

[4] "Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript." Accessed: Dec. 28, 2025. [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html

[5] "ACM SIGSPATIAL GIS Cup 2016." Accessed: Dec. 28, 2025. [Online]. Available: https://sigspatial2016.sigspatial.org/giscup2016/problem