Data Clustering and Representation: Displaying High Volumes of Geodata Efficiently and Meaningfully

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Austin Tran

Spring, 2025

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Briana Morrison, Department of Computer Science

Data Clustering and Representation: Displaying High Volumes of Geodata Efficiently and Meaningfully

CS4991 Capstone Report, 2025

Austin Tran
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
juj8pk@virginia.edu

ABSTRACT

A software solutions company developing an application to manage customer geodata and related workflows found issues displaying high volumes of data on the map of a web application in a clear, efficient, and meaningful manner. By clustering data points hierarchically, I developed software that displayed point clusters reducing the load on a user's browser and allowing more data to be represented. Each point cluster could be broken down into smaller clusters via zoom to provide a clearer representation of data within the map's context. To build this, I used the programming language Typescript, the web framework Angular, several open-source JavaScript libraries, and Git version control. Throughout development, I relied on an Agile methodology with regular code reviews from full-time team members to ensure my code was concise and functional. The major result of this project was the increased capacity of the web application from struggling to display 400 data points to loading 75,000 points easily. Although unit tests were created, future work lies in end-to-end testing and functional testing of the software on different browsers with varying system resources.

1. INTRODUCTION

Arming leaders with meaningful data to aid in decision-making can often be the difference between sinking and swimming. Modern-day businesses and governments collect lots of data throughout their operations. Having ways to analyze high volumes of data and visually represent important trends to nontechnical leaders is essential to the survival of any organization. Furthermore, enabling leadership to easily generate what they need, when they need it, can remove the slack in decision-making processes and streamline operations.

However, as data grows more complex and irregular it becomes harder to aggregate and visualize quickly. For example, counting red cards in sorted piles is easy, but clustering geodata with varying attributes is more difficult. Every standard playing card has the same set of attributes (color, suit, value) while geodata points may have different features. Ensuring that users can view complex data accurately and intuitively presents challenges. The concept of clustering has its obstacles as well. When analyzing large datasets, having a few large clusters may obfuscate important details while having an excessive number of small clusters could overwhelm a potential viewer and prevent them from identifying larger trends. Additionally, the computing power of an end user's browser limits how intensive a web application attempting to cluster data can be. An adequate solution must take all these complications into account to work both efficiently and effectively.

2. RELATED WORKS

Although there are many ways to cluster points, Du, et al. (2024) found that using KD-trees (k-dimensional) to divide datasets and computing K-nearest neighbors (KNN) could

prove "efficient" and "accurate," especially when considering clusters with "uneven density and complex shapes." Du, et al. defines the existence of "adhesive points" that connect clusters to allow the merging of adjacent clusters to automatically identify the "optimal number of clusters." These benefits support the use of KD-trees to cluster geodata and allow the utilization of its hierarchical structure to enable the decomposition of clusters into smaller clusters. However, a drawback of this proposed method is that there is no optimized K value selection strategy for KNN. This issue was mitigated by implementing the KD-tree clustering with a set minimum and maximum cluster size (radius-wise).

An additional task of generating a polygon encompassing the points within a cluster for transparency to the end user was also implemented. This was done using an adaptation of a library, concaveman, that generates a 2D concave hull from a point set (GitHub, 2021). This library performs this using an algorithm based on the ideas of Park (2012). Park proposed an "easy to understand and implement" concave hull algorithm that first creates a convex hull using a known algorithm, then "digs" on the concave hull until it becomes a complete concave hull. Although its proposed time complexity of $O(n\log n + rn)$ where n is the number of points and r is the number of points in the convex hull after the first step, the algorithm has the ability to be easily adjusted using a threshold value. Thus, the algorithm proved extremely adaptable and easy to implement for quick polygon generation around clusters in the two-dimensional geodata in the project.

3. PROJECT DESIGN

The project consisted of making upgrades to a pre-existing web application made to handle and display geodata to nontechnical users.

3.1 System Architecture

The web application used a complex technical stack to handle the insertion, manipulation and retrieval of data. The frontend or user-facing portion of the project utilized the Angular web framework to display content from and interact with the backend, the server-side of the project which used a myriad of technologies. On the backend, a Java Spring Boot framework provided APIs for creating, deleting, and manipulating data that persisted in a PostgreSQL database. Additionally, all data in PostgreSQL would be ingested into an OpenSearch NoSQL database using a message broker. To handle data retrieval requests from the frontend, the backend used a microservice built in Golang that queried the OpenSearch database before returning search results to the frontend.

3.2 Development Environment

To assist developers in working on the complex application, many different strategies, both technical and organizational, were used. Docker, a platform used to package microservices into individual containers, was used in combination with Kubernetes, an open-source container deployment platform, to allow developers to run local, individual instances of the application during development. Additionally, a hot reload feature was implemented to allow code changes to immediately be reflected in the deployed containers without having to rebuild the entire application. Furthermore, Git version control and an Agile methodology allowed multiple developers to work on individual features and microservices simultaneously without collaboration issues.

3.3 Requirements

The main objective of the project was to clearly and quickly represent large volumes of geodata on the frontend of the web application using the Angular framework.

3.3.1 Clustering

The goal of the data clustering project was to represent 75,000 individual data points

retrieved from the backend quickly and accurately on the user's browser (frontend). Moreover, data clusters were expected to break apart into smaller clusters dynamically when users zoomed in or clicked on them. Visual requirements like styling clusters to have tooltips (information popups on hover) and dynamic labels and colors generated based on the contents of each cluster were also requested by the client. A further requirement involved rendering individual geodata points upon extreme zoom and representing cases where geodata share the same latitude and longitude (a "stack" of points).

3.3.2 Polygon Generation

To increase visibility on what each cluster contains, another consideration was a requirement that each cluster have a colored polygon that encompassed all its geodata points. This would help enable users to visually identify what cluster certain regions of data belong to.

3.4 Solutions

To cluster queried geodata on the user's browser, the MapLibre mapping library's clustering layers were used. Using these map layers, clusters could be configured to have a maximum size, minimum size and aggregate information from their data points. Furthermore, the underlying data structure of the library, a KD-tree, enables large clusters to break down into smaller clusters upon zoom. Features like tooltips on cluster hovers and zoom-in on cluster clicks were implemented by adding event listeners which allowed the binding of browser events to specific actions. For example, the triggering event of hovering over a cluster would create a tooltip popup that contained aggregated information about that cluster. An issue presented by using event listeners was browsers repeatedly triggering an event too quickly (several times per second), thus triggering an action many times. This could result in the function of rendering clusters (the action) upon zoom or moving of the map (the event) to be called faster than the function could respond, leading to lag on the user's browser. To solve this problem, the RxJS debounce operator which helps to rate-limit event notifications was used (GitHub, 2025). This operator delays the call of a designated action from events until a certain delay has passed without an event being triggered. Thus, the re-rendering of clusters, including the breaking down of clusters, only occurred once the user finished zooming in or out after they finished moving the map.

To visually represent "stacks" of geodata points, they first had to be identified as clusters that do not break apart under the max level of zoom. This allowed the original cluster click event handler to have a special case for clusters at max zoom (stacks) in which a popup with a scrollable list of each item in the stack would be displayed. Finally, using the information each cluster aggregated, a colored ring around each cluster indicating the composition of the cluster, like a pie or donut chart, would be displayed.

The concaveman library was used to create polygons based on the points in each cluster. In addition to tooltips, the cluster hover event listener was used to trigger the generation and display polygons. The debounce operator was again used to rate-limit the display and remove polygons from clusters. However, the generation of polygons for each cluster upon hover proved to be slow, especially for clusters containing over 10,000 points. To solve this problem, clusters containing over 10,000 points would be exempted from polygon generation and a polygon-caching strategy would be implemented, making the user's browser only generate a cluster's associated polygon once (on the first hover) and simply redisplay the polygon on later cluster hovers.

4. RESULTS

The implemented data clustering allowed end users to view up to 75,000 points of data without lag or noticeable load on the user's browser or computer. This was a vast

improvement from the previous maximum capacity of 400 points of data (over 180 times more data). Additionally, the aggregated information in each cluster provided insight into each region of the map, whereas the prior display only rendered individual data points separately without extra categorization of styling to help make the data more readable. Thus, clustering helped enable users to utilize data in decision-making more effectively. Finally, the polygon generation added transparency to the underlying clustering process. Users could clearly identify which regions were covered by which clusters. Data clustering geodata had an overall significant impact in improving end users' ability to view and understand data to make key decisions while also providing transparency on which clusters covered which areas of the map.

5. CONCLUSION

The project played a key role in assisting business leaders in analyzing and viewing high volumes of geodata with speed and accuracy. It increased the working capacity of the pre-existing web application from struggling to display 400 data points to quickly generating clusters representing 75,000 data points. Furthermore, it added visual clarity on clustering through the display of polygons encompassing all the geodata points within a given cluster. The solution utilized a variety of open-source libraries to help implement clustering, enable polygon generation, and boost overall performance. Through the features developed in this project, nontechnical business leaders were introduced to new insights and trends in their data through gaining the ability to view high volumes of data.

6. FUTURE WORK

As part of the development process, unit tests for the clustering and polygon generation features were created. However, more comprehensive end-to-end tests still need to be implemented to ensure the consistency and accuracy of the clustering. End-to-end tests on the web application using Cypress, a Javascript testing framework, were already written for other features, thus any further end-to-end testing on the clustering and polygon generation features should be done using Cypress. Additionally, the clustering feature should be tested for higher volumes of data. Although the customer requested 75,000 points of data be displayable at one time, understanding the limitations of the current clustering implementation could be useful if the volume of data required by users at one time increases.

REFERENCES

Du, H., Hu, Z., Lu, D., Liu, J. (2024). Density clustering algorithm based on kd-tree and voting rules. *Computers, Materials & Continua*, *79(2)*, 3239–3259. https://doi.org/10.32604/cmc.2024.046314

GitHub. (2021, August 9). *GitHub—mapbox/concaveman: A very fast 2D concave hull algorithm in JavaScript*. GitHub. https://github.com/mapbox/concaveman

GitHub. (2025, February 21). *GitHub—ReactiveX/rxjs: Reactive extensions for JavaScript.* GitHub. https://github.com/ReactiveX/rxjs

Park, J., Oh, S. (2012). A new concave hull algorithm and concaveness measure for n-dimensional datasets. *Journal of Information Science and Engineering* 28(3), 587-600. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=14df973438a9ba4634bb41740072b9e4704ba47b