Evaluation of DBSCAN algorithm: An exploratory study

***Abstract*—DBSCAN is a well-known algorithm that is commonly used to find associations and structures in large spatial data. Due to its popularity, built-in functions for DBSCAN have been implemented on top of many different programming languages. Researchers and practitioners (data scientists) have been using these built-in functions to cluster and analyze a prolific area of research in data science. Due to the many implementations of DBSCAN and its utilization in many different languages, the output of each built-in DBSCAN function is assumed to be identical. In this paper, we present a systematic approach to evaluate the built-in functions of DBSCAN algorithms and to identify discrepancies in their output. The evidence shown describes such differences and recommends caution in dealing with some built-in functions implemented in some programming languages.**

***Index Terms*—Clustering, DBSCAN, Geo-coordinates, machine learning, spatial**

I. Introduction

Identifying classes in spatial domain and grouping them together is often done in various studies [1]. For example, studies like [2]–[5]. One of the popular clustering algorithm for spatial data is Density-Based Spatial Clustering of Applications with Noise (DBSCAN). The *DBSCAN* clustering algorithm finds neighbors of a given spatial point and cluster the desired and neighbor point if they satisfy number points required to be cluster.

Clustering is the most well-known, commanding and frequently utilized unsupervised learning technique in data mining [6]. This is a method of detecting comparable data objects into a group dependent on some likeness which is often predefined. For example, sorting books in a library, fraud detection in insurance, customer segmentation in marketing are some of the real world applications of clustering. Bigger issues like earthquake analysis or urbanization may require clustering as well. There are seven categories of clustering algorithms, in particular Graph-based algorithm, Hierarchical clustering algorithm, Partitioning clustering algorithm, Density-based clustering algorithm, Grid-based algorithm, Combinational clustering algorithm, and Model-based clustering algorithm. Among these, DBSCAN algorithm is taken into consideration for this study which is one of the most used unsupervised algorithms of machine learning (i.e., in spatial domain).

Though computational similarities are found among the most traditional clustering methods in the literature, such as k-means [7], Single-Linkage clustering (SLINK) [8], and other centroid-based clustering algorithms, they are not strong or adaptable enough to be considered in numerous clustering applications. These are known for their ability to detect clusters with arbitrary shapes. However, they are sensitive to noise and have a serious drawback of identifying only dense and spherical shaped clusters [9]. In this specific situation, DBSCAN shows up as an appealing way out to many clustering drawbacks. It contrasts in quite a few points. It categorizes clusters dependent on the data points' density in the feature space rather than utilizing the position of the centroids as led by k-means. These encourage a more precise proof of identity and segregation of the clusters that are of various sizes and shapes, particularly convex-shaped data clusters. A second preferred position of DBSCAN is its capacity to isolate outliers (noisy data points) when they are excessively different to the rest of the data points. Furthermore, the DBSCAN algorithm utilizes a deterministic approach rather than casually choosing the first locations of the cluster centroids [10].

Contributions of this paper are as follows:

- Provide an evaluation of built-in DBSCAN functions from different programming languages
- A comparisons of result provided by different languages to show the similarities and differences in outcome
- Show the results using visualization to help readers to better understand the obtained evidences
- Provide an explanation to understand the obtained results.

A. Motivation

DBSCAN algorithm is a popular algorithm in clustering spatial data. However, there are different programming languages and most of them provide built-in functions that can facilitate clustering of user data. Often, programmers use these built-in functions with good faith and avoid implementing complex algorithms [11], [12]. However, there is a lack of studies that evaluate built-in functions provided by the available libraries. This study attempts to fill the gap of evaluation-based studies that can show a systematic process of evaluation of built-in functions.

B. Purpose of the study

The purpose of this study is to evaluate built-in DBSCAN clustering algorithms offered by a few popular programming languages. Also, compare built-in DBSCAN functions from different programming languages to see if there are any similarities and differences in the results.

The research question that guided this study is presented below:

- Are all popular programming languages that have a built-in DBSCAN function provide the same result with the same parameters?

- Which programming languages provide the same result and which one provide different results?

The rest of the paper is organized as follows: background of the area is presented in section II, the section III describes the method used to conduct this study, IV presents the experimental results of this study, V provides further evidences, VI discusses the findings, and finally section VII concludes this study.

II. Applications of DBSCAN algorithm

There are various cluster algorithms for class identification in computer science. At the point when we have more than one algorithm, we need to choose the one that provides optimum solution. Performance investigation encourages us to choose the best algorithm from numerous algorithms to take care of a problem. There are as of today a great deal of research on comparative investigation on DBSCAN in the literature. For instance, J. Gan, and Y. Tao (2015) studied the performance of DBSCAN in different systems where they found DBSCAN does not scale well for huge datasets because of its high time complexity like a number of other clustering algorithms [13]. A comparison study of some clustering algorithms are presented in [14].

Many key metrics such as precision, recall value, accuracy, time complexity, and several others are found in the literature that have been used to evaluate the performances of DBSCAN clustering algorithm [15]–[17]. There are different software packages/tools and most of them provide built-in functions that can facilitate clustering of user data. Oyelade et al. (2016) explained some clustering algorithms and software packages/tools corresponding to those algorithms [18]. Often, programmers, data scientists, and researchers use these built-in functions with good faith and avoid implementing complex algorithms. However, there is a lack of studies that evaluate built-in functions provided by the available libraries of different software packages/tools. This study attempts to fill the gap of evaluation-based studies that can show a systematic process of evaluation of built-in functions.

III. Method

Different programming languages were used to understand whether the DBSCAN clustering algorithm provides the same results [19]. To accomplish this, we used seven different geo-coordinate values (See Table I) and four different languages. Such as, Python (i.e., sklearn library), Postgres (i.e., postgis library), R programming language, and JavaScript. Note that the DBSCAN clustering algorithm tries to group together data points if its neighbors are within a given distance (ϵ). The neighborhood of two data points can be checked using the equation below (see Eq. 1).

$$N_\epsilon(p_1) : \{p_2 | d(p_1, p_2) \leq \epsilon\} \tag{1}$$

The function $N_\epsilon(.)$ tries to determine whether p_1 and p_2 are neighbors or not using the spatial distance d, and given distance ϵ. The distance between p_1 and p_2 has to be smaller than ϵ then p_1 and p_2 would be considered as neighbors. A typical DBSCAN algorithm requires at least two parameters, such as epsilon, *MinPts*. In this study, we used the same parameters for all programming languages mentioned above. That is the same epsilon, *MinPts* for all considered programming languages.

In general, distance in a 2D space is calculated using Euclidean distance. The euclidean distance between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ can be computed as,

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)} \tag{2}$$

where d is the distance between p_1 and $p2$

Often haversine distance is used to compute distance between two geo-coordinate points. The Haversine distance considers the circle distance or spherical distance between two geo-coordinates. Because Euclidean distance does not consider the Spherical shape or great-circle, which is the case for spatial data. Therefore, in spatial distance Haversine formula is often used.

$$d = r \sin^{-1}(\sqrt{\sin^2(\frac{y_2 - y_1}{2} + \cos(y_1)\cos(y_2)\sin^2(\frac{x_2 - x_1}{2})} \tag{3}$$

Where d is the distance between two geo-coordinates p_1 and $p2$, r is the radius of earth, which is 6371 km, x_1, x_2 are longitude, and y_1, y_2 are latitude values in spatial coordinates. Please note that Haversine is not necessary when geo-coordinates are not far from each other [20]. Which means when distances in geo-coordinates are not far then the great-circle effect is negligible.

TABLE I
SAMPLE GEO-COORDINATES

Index	lat	long
0	35.85576000	-80.28050000
1	35.85323196	-80.27931830
2	35.85489867	-80.26675832
3	35.85172000	-80.26230000
4	35.85049000	-80.25765300
5	35.85067000	-80.25450000
6	35.86787000	-80.25580000

In this study, in our first experiment. We used data from Table I, and clustered the data using Python, Postgis, R, and JavaScript. For the first experiment we computed the distance matrix between all seven points (i.e., Table I, index 0 to 6) and found that geolocations were between the geolocation index 4 and 5, which was 284 meters or .284 kilometers (see Table 2). So, for the first experiment, we used eps = .00548 and min_sample = 2. Results are presented in the Results section. Furthermore, in the second experiment we used eps = .005 and min_sample = 5. Obtained results were cross checked using the mapview visualization. On the mapview we converted the eps value to a meter distance and used that as a radius of the circle. The circles show which neighboring points fall within its eps reach and should be considered as a neighbor for the cluster (see Eq. 1). Note that eps is the ϵ as mention in Eq. 1.

IV. Results

A. Experiment 1

1) Python: In Python programming language, the "sklearn" library provides a built-in DBSCAN function. The function takes eight parameters. Most of the parameters are set to default values, so those do to require user inputs. However, eps or the minimum distance two points, min_sample or number samples that are required for a cluster, and metric or distance calculation parameters were modified for this study. Using the data from Table 1, with eps = .00548, and min_sample = 2, we found two clusters and an outlier.

The Fig 1 (A), shows the original clustering results obtained from Python's built-in function from sklearn library. However, using the degree distance when circles were drawn then Fig 1(B) indicated some anomalies. Therefore, a distance matrix (see Table III) was computed using the haversine distance Equation 3.

2) Postgis: Using the data from Table I, with eps = .00548, and min_sample = 2, we found two clusters and an outlier.

3) JavaScript: Using the data from Table 1, with eps = .00548, and min_sample = 2, we found two clusters and an outlier.

4) R language: Using the data from Table 1, with eps = .00548, and min_sample = 2, we found two clusters and an outlier.

B. Experiment 2

A second experiment was conducted to further gather evidence. In the second experiment, built-in DBSCAN functions from all four programming languages were tested. The parameters used in the second experiment were eps = .00548, and min_sample = 2. Obtained results are presented in Table 3 (See Table 3). Table 3. Cluster results using the experimental data presented in Table 1.

V. Evaluation

To evaluate the built-in functions further a publicly available data was used. In the publicly available data, locations of all countries name and corresponding latitude and longitude values were present. The data was downloaded from Google data [21]. In the evaluation experiment the parameters used were eps = .9000009 , and min_sample = 4. Obtained results presented in Figure 6. Evidence suggests that all three programming languages (i.e., Postgis, JavaScript, and R) provided the same results. Whereas Python programming language provided a different result (see Fig. 4). In Python, results presented in Figure 4 indicated only one cluster and no outliers. However, the visual map presented in Figure 3 shows that there should have been only four geo-coordinates in a cluster and the rest should have been outliers. To verify the finding of the countries data, a distance matrix between all geo-coordinates were prepared and results indicated that indeed only four geo-coordinates should have been formed a cluster as shown in Figure 3.

VI. Discussion

Many programmers and researchers trust built-in library functions and often use them to solve problems and obtain research results [22]. DBSCAN is one of popular algorithms of clustering spatial data [23]. Applications of built-in DBSCAN algorithms can be found in many research studies. However, to the best of our knowledge, there is no such research that shows the comparison of DBSCAN's implementation on all platforms and evaluating them. Almost in all cases, researchers are using the built-in DBSCAN function with a blind trust [22]. Therefore, a study to evaluate built-in DBSCAN algorithms in multiple platforms or programming languages was a necessity of time. In this study, an evaluation of the DBSCAN algorithm in cross language, such as Python, R, PostGis, JavaScript was explored. To accomplish this the same hyper parameter, such as eps, min_points were used.

Evidence suggests that in three programming languages, such as, R, Postgis, and JavaScript built-in DBSCAN functions provide the same results. However, the built-in function from Python language seems to provide results that are different from the other languages. So, it is the recommendation of this study that researchers should use the built-in DBSCAN function with caution. Perhaps, researchers should try to use other languages for spatial clustering. How the built-in DBSCAN function is implemented in Python was not included in the scope of this study. A future study should focus on comparing the implementation of DBSCAN in different programming languages.

The dissimilarity that was found in Python programming language could be because the unit of the eps value [24]. Evidence suggest that three programming languages, such as, Postgis, JavaScript and R languages are using the eps value in degree distance. Therefore, the eps value in Python is not in degree distance. One of the plausible answer for this is that the eps unit for Python is in radian distance [25]. In a study [25], authors mentioned that the library of Python language uses the eps value in radian distance. However, this should not have been the case. Because, this makes the Python implementation not similar to other programming language and will confuse programmers. Moreover, because of this dissimilarity results produced by Python language without proper eps unit will be inaccurate.

VII. Conclusion

In this paper, we evaluated the DBSCAN built-in function in multiple programming languages such as Python, R, PostGIS and Javascript. The input values and parameters used to make this comparisons were identical for each case. Results of our experiments show that Python scikit-learn's implementation of DBSCAN produces a different output than the other implementations. As shown in the benchmark section, scikit learn's DBSCAN tends to put all the coordinates of publicly available countries csv in a single cluster where in PostGis we can see only four coordinates from all the data points fall under a single cluster. This unexplained behaviour of DBSCAN's official implementation in python validates our hypothesis

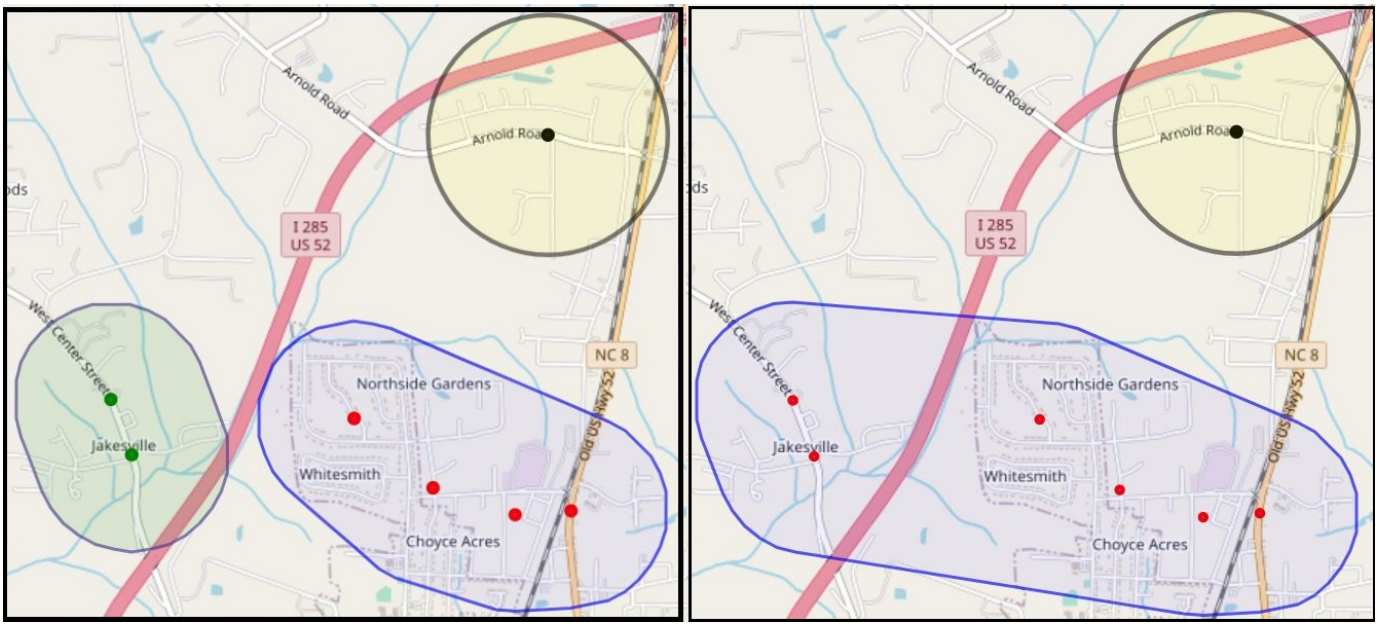

Fig. 1. Clusters from experimental data, (A) shows a cluster with five geo-coordinates and an outlier, (B) shows two clusters, one with two geo-coordinates and another with four geo-coordinates, and an outlier. The radius distance for the experiment was .609 kilometers.

TABLE II
DISTANCE MATRIX USING THE EXPERIMENTAL DATA PRESENTED IN TABLE I

Index	0	1	2	3	4	5	6
0	0	0.3006033356	1.242143822	1.700680877	2.140865675	2.410652495	2.601484789
1	0.3006033356	0	1.147036357	1.542987586	1.976294525	2.254870936	2.672305751
2	1.242143822	1.147036357	0	0.5351435976	0.9559023625	1.200695055	1.74801817
3	1.700680877	1.542987586	0.5351435976	0	0.4405913054	0.7126280264	1.888919156
4	2.140865675	1.976294525	0.9559023625	0.4405913054	0	0.2848798541	1.939769109
5	2.410652495	2.254870936	1.200695055	0.7126280264	0.2848798541	0	1.916137561
6	2.601484789	2.672305751	1.74801817	1.888919156	1.939769109	1.916137561	0

TABLE III
CLUSTER RESULTS USING THE EXPERIMENTAL DATA PRESENTED IN TABLE I

Programming language	Clusters and outliers
Python	Cluster with six points, and one outlier
Postgis	Seven outliers, no cluster
Javascript	Seven outliers, no cluster
R	Seven outliers, no cluster

of "we can't trust built in function blindly", and this study supports this hypothesis. From this research we see high abnormality in scikit-learn's DBSCAN implementation, whereas considering the same parameters we get similar results in all the other platforms. DBSCAN is for clustering spatial data thus the clustering should be identical in all the other platforms. Though all the other platform clusters similarly and accurately scikit-learn's DBSCAN behaves totally differently and acts like an outlier compared to other platforms. Future work will be in determining the factors that are affecting Python's Scikit learn results.

Evidence suggested that the eps (i.e., ϵ) value in DBSCAN algorithms in most languages are in degree; at least the ones were tested in this study. It appears that it does not work for Python's built-in DBSCAN function. Also, in the Python's official website there is no clear information about what is the unit of eps should be [26]. A future study can focus on finding how many research studies used degree as a unit of eps and how many of them used radian as a unit of eps in built-in DBSCAN function.

REFERENCES

[1] J. Aldstadt, "Spatial clustering," in *Handbook of applied spatial analysis*. Springer, 2010, pp. 279–300.
[2] K. M. Kumar and A. R. M. Reddy, "A fast dbscan clustering algorithm by accelerating neighbor searching using groups method," *Pattern Recognition*, vol. 58, pp. 39–48, 2016.
[3] T. M. Thang and J. Kim, "The anomaly detection by using dbscan clustering with multiple parameters," in *2011 International Conference on Information Science and Applications*. IEEE, 2011, pp. 1–5.
[4] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, "Real-time superpixel segmentation by dbscan clustering algorithm," *IEEE transactions on image processing*, vol. 25, no. 12, pp. 5933–5942, 2016.
[5] J. Liu, J. Z. Huang, J. Luo, and L. Xiong, "Privacy preserving distributed dbscan clustering," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, 2012, pp. 177–185.
[6] M. W. Berry, A. Mohamed, and B. W. Yap, *Supervised and unsupervised learning for data science*. Springer, 2019.

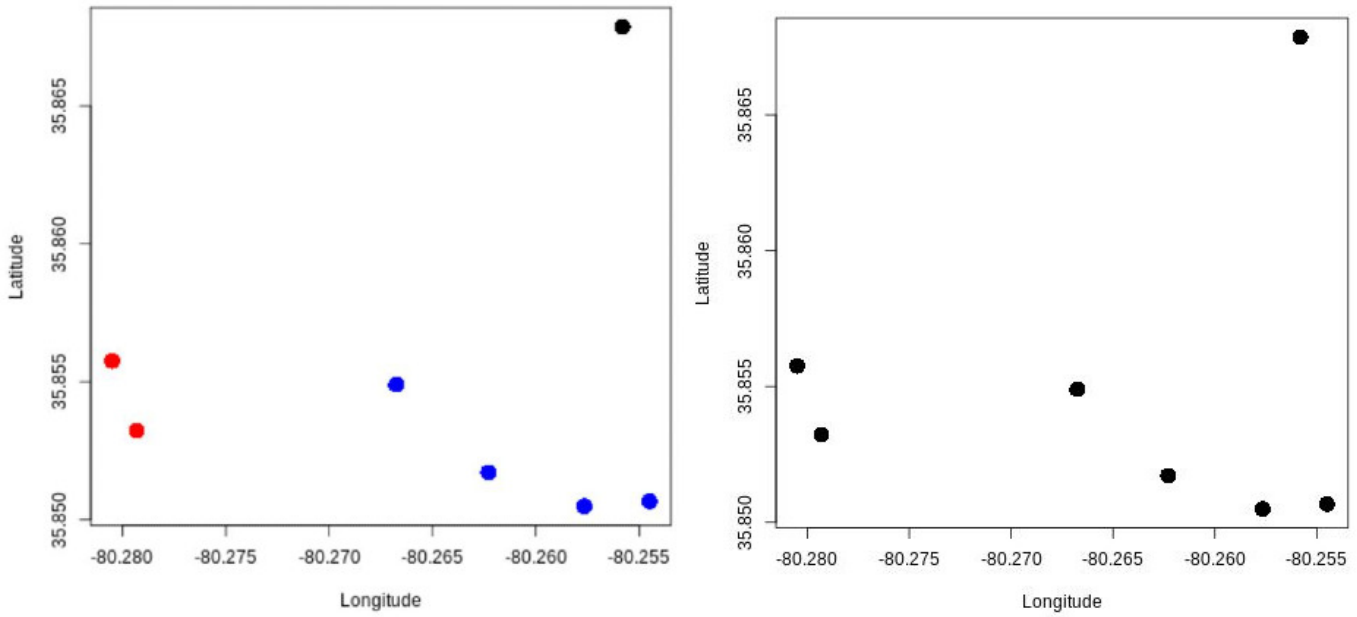

Fig. 2. (A), shows the original clustering results obtained from Python's built-in function from sklearn library. However, using the degree distance when circles were drawn then Fig 1(B) indicated some anomalies. Therefore, a distance matrix (see Table III) was computed using the haversine distance Equation 3

Fig. 3. One cluster: four geo-coordinates (red triangles), and rest are outlier (black circle)

[7] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[8] R. Sibson, "Slink: an optimally efficient algorithm for the single-link cluster method," *The computer journal*, vol. 16, no. 1, pp. 30–34, 1973.

[9] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[10] S.-I. Handra and H. Ciocârlie, "Anomaly detection in data mining. hybrid approach between filtering-and-refinement and dbscan," in *2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE, 2011, pp. 75–83.

[11] L. F. Cranor, "Programming perl: an interview with larry wall," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 1, no. 2, pp. 10–11, 1994.

[12] L. Ramalho, *Fluent python: Clear, concise, and effective programming*. " O'Reilly Media, Inc.", 2015.

[13] J. Gan and Y. Tao, "Dbscan revisited: Mis-claim, un-fixability, and approximation," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 519–530.

[14] M. R. Karim, O. Beyan, A. Zappa, I. G. Costa, D. Rebholz-Schuhmann, M. Cochez, and S. Decker, "Deep learning-based clustering approaches for bioinformatics," *Briefings in Bioinformatics*, 2020.

[15] A. Zhou, S. Zhou, J. Cao, Y. Fan, and Y. Hu, "Approaches for scaling dbscan algorithm to large spatial databases," *Journal of computer science and technology*, vol. 15, no. 6, pp. 509–526, 2000.

[16] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[17] A. Malhotra and K. Bajaj, "A hybrid pattern based text mining approach for malware detection using dbscan," *CSI transactions on ICT*, vol. 4, no. 2-4, pp. 141–149, 2016.

[18] J. Oyelade, I. Isewon, F. Oladipupo, O. Aromolaran, E. Uwoghiren, F. Ameh, M. Achas, and E. Adebiyi, "Clustering algorithms: their application to gene expression data," *Bioinformatics and Biology insights*,

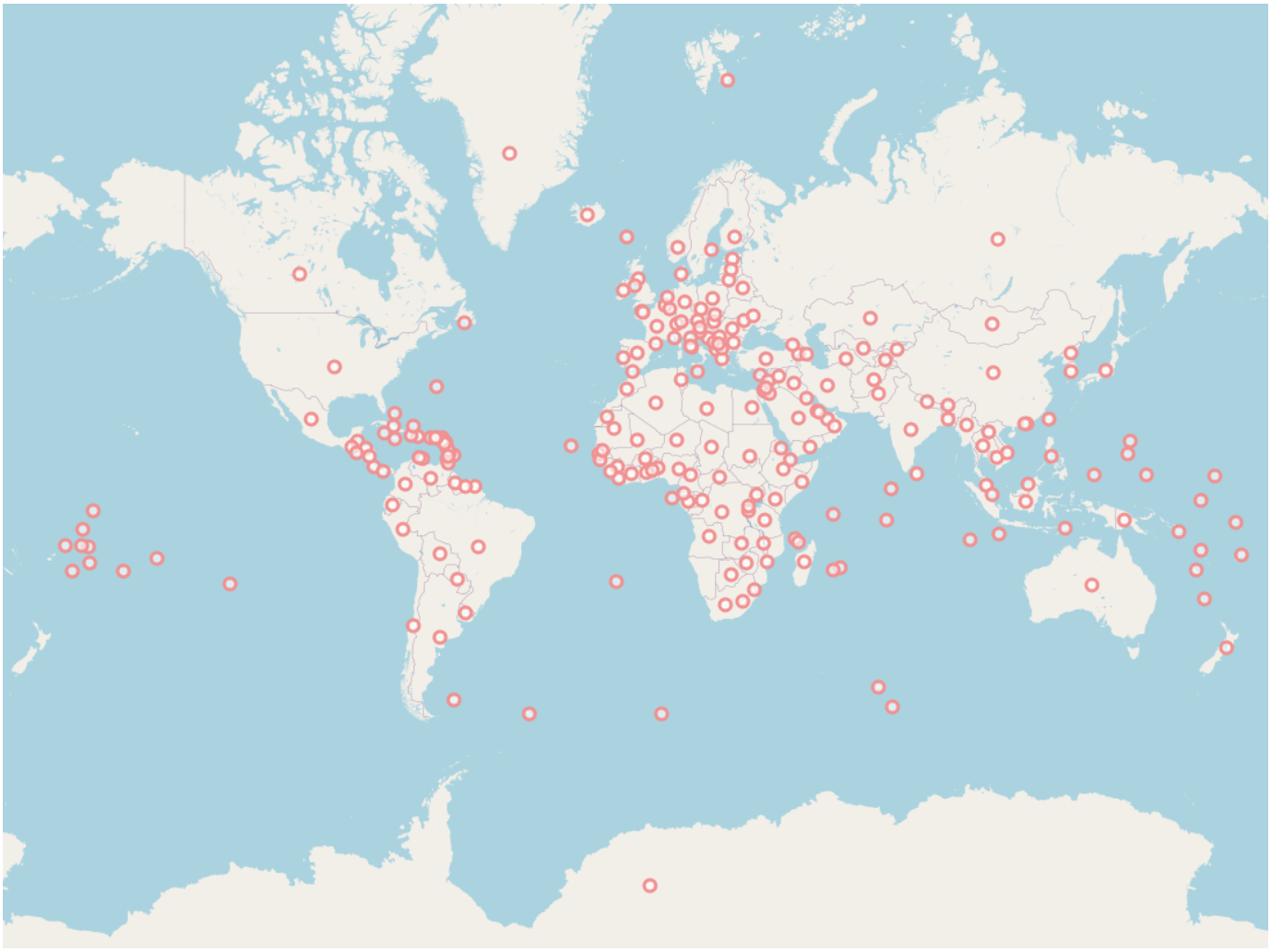

Fig. 4. One cluster all geo-coordinates (red circles) and no outliers

vol. 10, pp. BBI–S38 316, 2016.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[20] D. A. Prasetya, P. T. Nguyen, R. Faizullin, I. Iswanto, and E. F. Armay, "Resolving the shortest path problem using the haversine algorithm," *Journal of Critical Reviews*, vol. 7, no. 1, pp. 62–64, 2020.

[21] "Google: Dataset publishing language," https://developers.google.com/public-data/docs/canonical/countries_csv, accessed: 2021-01-12.

[22] J. Hao and T. K. Ho, "Machine learning made easy: A review of scikit-learn package in python programming language," *Journal of Educational and Behavioral Statistics*, vol. 44, no. 3, pp. 348–361, 2019.

[23] M. Hahsler, M. Piekenbrock, and D. Doran, "dbscan: Fast density-based clustering with r," *Journal of Statistical Software*, vol. 91, no. 1, pp. 1–30, 2019.

[24] A. Starczewski and A. Cader, "Determining the eps parameter of the dbscan algorithm," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2019, pp. 420–430.

[25] G. Boeing, "Clustering to reduce spatial data set size," *arXiv preprint arXiv:1803.08101*, 2018.

[26] "sklearn.cluster.dbscan," https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html, accessed: 2021-01-22.