MIJST
MIST International Journal of Science and Technology

Logical analysis of built-in DBSCAN Functions in Popular Data Science Programming Languages

Md Amiruzzaman[1*], Rashik Rahman[2], Md. Rajibul Islam[3], and Rizal Mohd Nor[4]

[1]West Chester University, West Chester, PA, USA
[2,3]University of Asia Pacific, Dhaka, Bangladesh
[4]International Islamic University Malaysia, Kuala Lumpur, Malaysia

emails: [1]*m.amiruzzaman@gmail.com;* [2]*17201012@uap-bd.edu;* [3]*md.rajibul.islam@uap-bd.edu; and* [4]*rizalmohdnor@iium.edu.my*

ARTICLE INFO

Article History:

Received: 21st March 2022
Revised: 21st May 2022
Accepted: 23rd May 2022
Published online: 26th June 2022

Keywords:

Clustering
DBSCAN
Geo-coordinates
Machine learning
Spatial

ABSTRACT

DBSCAN algorithm is a location-based clustering approach; it is used to find relationships and patterns in geographical data. Because of its widespread application, several data science-based programming languages include the DBSCAN method as a built-in function. Researchers and data scientists have been clustering and analyzing their study data using the built-in DBSCAN functions. All implementations of the DBSCAN functions require user input for radius distance (i.e., eps) and a minimum number of samples for a cluster (i.e., min_sample). As a result, the result of all built-in DBSCAN functions is believed to be the same. However, the DBSCAN Python built-in function yields different results than the other programming languages those are analyzed in this study. We propose a scientific way to assess the results of DBSCAN built-in function, as well as output inconsistencies. This study reveals various differences and advises caution when working with built-in functionality.

1. INTRODUCTION

Identifying and classifying classes in the spatial domain is a common practice in many investigations. For example, research such as Rizvee et al., (2021) for locating accident-prone locations and Islam et al., (2021) for locating densely populated areas. The Density-Based Spatial Clustering of Applications with Noise (i.e., DBSCAN) is a clustering approach for location-based data. The DBSCAN clustering method locates the neighboring points of a given spatial point and groups the neighbors if they meet multiple clustering requirements within the given adjacent distance (Amiruzzaman, et al., 2021).

Clustering in the spatial realm can be useful in a variety of applications. DBSCAN is commonly used for clustering in planar space. It can produce reasonable results when used to map the impact of natural catastrophes or to plot the position of weather stations in a city. It can also be utilized when the data is made up of non-discrete points and has outliers. DBSCAN is used by many systems nowadays that provide recommendation services, such as a Recommender engine, to propose goods or things to its clients. It is also utilized to identify typical events, such as finding areas where frequent road accidents occurred, in other applications (Rizvee et al., 2021; Amiruzzaman et al., 2018).

Clustering techniques is one of the most renowned, powerful, and widely used unsupervised learning approaches in data mining (Berry et al., 2019). This is a way of classifying comparable or similar data members into a set or group based on some preset resemblance (Fischer et al., 2003). Some real-world uses of clustering include book sorting in a library, consumer segmentation in marketing, and fraud detection in insurance (Mahmoudi et al., 2020). Larger challenges, such as seismic analysis or perhaps urbanization analysis, may benefit from clustering as well. According to (Limwattanapibool et al., 2017), clustering algorithms are classified into seven types: (i) hierarchical clustering algorithms, (ii) graph-based algorithms, (iii) density-based clustering algorithms, (iv) partitioning clustering algorithms, (v) model-based clustering algorithms, (vi) combinational clustering algorithms, and (vii) grid-based algorithms. Among them, the DBSCAN method is widely used as an unsupervised machine learning techniques, that is taken into account for this study (i.e., in the spatial domain).

Although, the most clustering approaches in the literature, for example, k-means (Macqueen et al., 1967; Islam et al., 2021), Single-Linkage clustering (SLINK) (Sibson et al., 1973), and other centroid-based clustering approaches, share computational similarities, they are not powerful or

adaptable enough to be considered in a wide range of clustering use. These are recognized for their capacity to recognize clusters of any form (Wu et al. 2014). They are, however, susceptible to noise and have the major limitation of recognizing groups based on only density and data points spherical-shaped clusters (Jain et al., 2010).

In this case, DBSCAN appears to be a promising solution to several clustering issues. It contrasts in a number of ways (Amiruzzaman, et al., 2021). It classifies clusters based on the density of data members in their feature space as a substitute of the position of the computed centroids like k-means does (Dudik et al., 2015). These provide more exact evidence of identification and segregation of clusters of varying sizes and forms, particularly when they form a shape of a convex clusters of data. DBSCAN's ability to extract noisy data members or outliers make them stand out from other algorithms (Luchi et al., 2019). Importantly, rather than arbitrarily selecting the first positions of the cluster centroids, the DBSCAN method employs a deterministic approach (Handra et al., 2011). This paper's contributions are as follows:

- Present a methodology as to how a built-in function can be compared among different programming languages.
- Provide comparable results obtained from the experiment.
- Provide visual output as evidence for easier exploration.
- Provide implications and detailed discussion on the evidence.

A. Motivation

The DBSCAN method is a well-known technique for grouping geographical data. Nevertheless, there are several programming languages, and virtually almost all of the languages provide built-in functions that can help with user data clustering. Often, programmers utilize these built-in functions in good faith in order to avoid inventing difficult procedures or algorithms on their own (Cranor et al., 1994; Ramalho et al., 2015).

Commonly Java implementation of DBSCAN can be found in Apache Common Math and Environment for DeveLoping KDD-Applications Supported by Index-Structures (ELKI). C++ implementation can be found in mlpack and pyclustering. In python, DBSCAN is included in the scikit-learn or sklearn library and R contains a package for DBSCAN. There are applications like Weka and SPMF that provide their implementation of DBSCAN.

There are few studies that demonstrate a systematic review of such built-in functions to offer a technique to identify similarities and variations in the outcomes produced by the accessible libraries (Amiruzzaman, et al., 2021). This study aims to address a necessity in evaluation-based studies by demonstrating a methodical approach of evaluating built-in functionalities.

B. Objective

The goal of this study was to explore how the built-in DBSCAN algorithm provides output, and if the obtained outputs from different programming languages are comparable. We wanted to see where similarities and differences lies in the output obtained from different implementations of DBSCAN. We focused on following research questions for this study:

- Are there any differences in clustering results produced by DBSCAN built-in clustering among data science based programming languages, when the same parameter values are used?
- If there are any differences in results and similarities, then which ones are providing pairing results?

2. RELATED WORK AND USE OF DBSCAN IN EXISTING STUDIES

In computer science, there are different cluster methods for class identification (Amiruzzaman, et al., 2021). Since we have more than one algorithm, hence we must select the one that fits the dataset and offers most optimal result. Performance analysis allows us to select the best algorithm from a set of algorithms to solve a problem. There is indeed a large number of study in the literature on the comparative investigation of DBSCAN. In one study, authors examined the performance of DBSCAN and concluded that the DBSCAN clustering technique does not scale well for big datasets (Gan & Tao, 2015). While studying deep learning clustering approaches for bioinformatics, Karim et al. (2020) utilized DBSCAN to compare several clustering methods in a different study. Particularly, Karim compared machine learning approaches in clustering algorithms such as K-means, DBSCAN, OPTICS which is an extension of DBSCAN, GMM, AC and the Partial Mix Model (PMM). For each algorithm, Karim describes the parameters being used, scalability of sample sizes, geometry used for calculating distances, its use cases in bioinformatics and its limitations. Among its notable limitations for DBSCAN is the difficulty of separating nearby clusters and the quadratic computational complexity. However, he also noted that it is a good algorithm for uneven cluster size with non-flat geometry and hence used in many bioinformatics analysis.

Many essential criteria, for example, the accuracy of the result, precision, recall values, the complexity of the algorithm, and others, can be found in the existing works that have been used to appraise the DBSCAN clustering algorithm's performance (Zhou et al., 2000). Schubert et al. (2017) presented a strong argument about why we should still use DBSCAN as a rebuttal to a previous paper misrepresenting DBSCAN of having poor performance. In Schuberts argument, he explains that choosing the right parameters is important to achieve both meaningful results and good performance. Schubert conducted experiments to show that other suggested methods do not appear to offer practical benefits if the DBSCAN parameters are well chosen and thus they are primarily of theoretical interest. Schubert concluded that the original DBSCAN algorithm with effective indexes and reasonably chosen parameter values performs competitively compared to the method proposed by others claiming to achieve better results. Clearly, DBSCAN remains relevant in many scientific

research work and requires constant revision on how we use DBSCAN to achieve meaningful results.

There are several software libraries or tools available, and most of those provide built-in functions that aid with consumer data clustering. Authors of an existing study described several clustering techniques as well as the software packages/tools associated with such approaches (Oyelade et al., 2016).

Often, data technologists, programmers, and academics rely on these built-in clustering functions and avoid constructing difficult algorithms (Amiruzzaman, et al., 2021). Nevertheless, there is a scarcity of research that compares the built-in functionalities offered by various software packages/tools' libraries. This research sought to address a gap in evaluation-based studies by demonstrating a systematic approach to evaluating built-in functionalities. Figure 1 depicts how many articles in the last 7 years employed the DBSCAN technique in their study. The data shown in Figure 1 was collected in May of 2022.

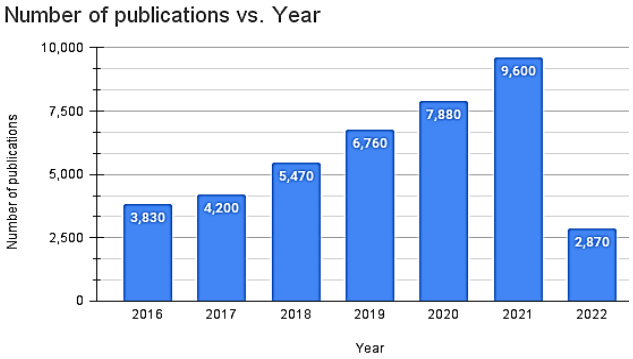

Figure 1. The DBSCAN algorithm used in studies over the last 7 years

For example, 3830 papers published in 2016 used DBSCAN as an analysis or as a clustering tool. Subsequently, in 2017, 4,200 papers were published that somehow used DBSCAN in their work. Furthermore, there were 5,470 papers in 2018, 6,760 papers in 2019, 7,880 papers in 2020, and 9,600 papers in 2021. Most interestingly, in 2022 (up to mid-May 2022), 2,870 papers used the DBSCAN algorithm. These results are based on a search in Google scholar using the keywords "BDSCAN as an analysis tool". Perhaps, in reality, these numbers may change slightly, however, these numbers give us an idea about how popular the DBSCAN algorithm is in data-science-related studies or clustering in general. From Figure 1, it is evident that the popularity of the DBSCAN algorithm is experiencing an upward trend. As shown in Figure 1, by the end of the year 2022, the search quantity may exceed that of the previous year. Research related to DBSCAN is showing an increasing trend, so it is very important to justify various approaches to the DBSCAN algorithm.

3. METHOD

A. Opening discussion

DBSCAN clustering algorithm helps in computing distances in between points of data given and outputs high density cluster areas. The focus for DBSCAN clustering is highly dependent on neighborhood computation. Generally, the computations as follows: a point p_1 is said to be the neighbor of another point p_2, *iff* p_1 and p_2 are located less than or equal to the radius distance of *eps* distance. This computational process continues discovering other points, p_1 and p_2 which are neighbors and these points form a cluster if they meet the goal of min_sample or minPts (see Figure 2).

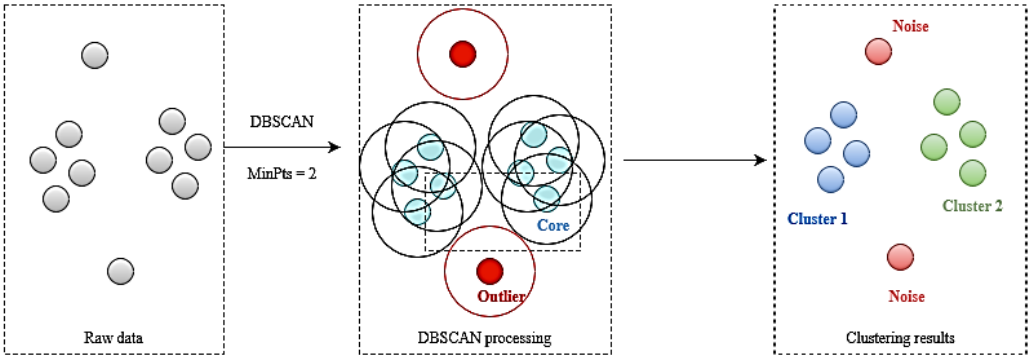

Figure 2. Working procedure of DBSCAN algorithm

Based on the above discussion the DBSCAN algorithm depends on two basic parameters:

- **minPts (min_sample):** The minimum number of points to form a cluster.
- **eps (ε):** The minimum distance to form a neighboring relationship among points.

Throughout this study of DBSCAN clustering algorithms, numerous programming languages were utilized to compare the accuracy, consistency and reliability of the DBSCAN clustering algorithm implemented. Particularly, we were interested in producing the same results despite the programming language given, hence a similar approach to Ester et al, 1996 was taken to discover clusters in large spatial databases with noise. In doing so, we also took the approach of Davies-Bouldin (Davies et al., 1979) and Silhouette Coefficient (Rousseeuw et al., 1987) by first calculating the appropriate number of clusters. Consequently, we employed seven distinct geo-coordinate location coordinates from Lexington, North Carolina, United States (refer to Table 1) with the use of four different computer languages, which includes the Python programming language. Also, we considered other languages, such as the, R programming language, Javascript, and Postgres with the postgis library.

As for the selection of geo-coordinates, points were selected using a pseudo-random generator as suggested by most researchers in literature to enhance the validity of the research (Niemierko *et al.*, 1990). As a recall to what was mentioned, the DBSCAN clustering algorithm attempts to group data points together provided that they are neighbors and within a certain distance (ε). It can best be described mathematically in the equation below (see Equation (1)) that determines if two data points are within the same neighborhood.

$$N_\epsilon(p1): \{p2 | d(p1, p2) \leq \epsilon\} \quad (1)$$

Using the spatial distance denoted as *d*, and the provided distance *ε*, the function $N_\varepsilon(.)$ attempts to identify whether coordinate point p_1 and coordinate point p_2 are considered to be neighbors. If coordinate point p_1 and coordinate point p_2 is less than ε, then p_1 and p_2 are considered as neighbors. It is common that the DBSCAN method requires at least 2 parameters, such as *epsilon* and *MinPts*. To be consistent with our study, we employed identical parameters for all of the above-mentioned programming languages for our experiments. Therefore, for brevity, it is enough to say that all epsilon and MinPts, for all programming languages studied, are identical.

To determine the 2D space distance, it is pretty common practice to use Euclidean distance as a method for measurement. The Euclidean distance between two points can be calculated as such, consider point (x_1, y_1) denoted as p_1 and point (x_2, y_2) denoted as p_2, then the Euclidean distance may be calculated as follows:

$$d = \sqrt{((x2 - x1)^2 + (y2 - y1)^2)} \quad (2)$$

where *d* denotes the distance between two points, which is p_1 and p_2.

In some other occasions, Haversine distance is largely preferred. This is usually the case when it is necessary to take into consideration the distance between two geo-coordinates that are intersecting with the circumference line of the circle or the circumference of the sphere. While Haversine is particularly more accurate for distances between two points on earth, it is not always used since planar distances may not have any curvatures, or it is too small of a distance to provide significant differences. It is for this reason that the Haversine formula is frequently utilized in spatial distance calculations as opposed to Euclidean distances. The Haversine distance may be calculated as follows:

$$d = r\, sin^{-1}\left(\sqrt{\sin^2\left(\frac{y2-y1}{2}\right) + \cos(y1)\cos(y2)\sin^2\left(\frac{x2-x1}{2}\right)}\right) \quad (3)$$

where, the notation *d* represents the distance between two geo-locations, such as point p_1 and point p_2, here note that *r* represents the radius of the earth, which is approximately 6371 km, x_1, x_2 are longitude, and y_1, y_2 are latitude values in spatial coordinates. It is important to note that the Haversine distance calculation approach is inconsequential when geo-locations are not spacious enough with each other (Prasetya *et al.*, 2020). This means that when distances between the data points in terms of their geo-coordinates are not spacious enough from each other, then the consequence of the great-circle effect can be forgiven.

Table 1
Data used in experiment, detail of the data presented in section 4(A)

Position	latitude	longitude
0	44.9438	-76.1083
1	44.9414	-76.1073
2	44.9427	-76.0940
3	44.9396	-76.0897
4	44.9383	-76.0850
5	44.9384	-76.0817
6	44.9549	-76.0814

In this study, we conducted our first experimental observation with grouped data from Table 1. The programming languages Python, Postgis, R, and JavaScript were used as tools for our experiment. Firstly, we calculated the matrix of distances between all seven coordinates (see Table 1) and revealed that the geolocations were in the middle for index 4 and index 5, which was exactly 284 meters or 0.284 kilometers as in Table 2. As a result, we selected eps = 0.00548 and min sample = 2 as configuration 1 and eps = 0.00548, min samples = 5 is selected as configuration 2. Figure 3(A) and Figure 3(B) shows a map visualization of clusters with configuration 1 and 2 respectively. The mapview visualization was used to cross-check the obtained results. The corresponding mapview was then transformed to the eps value to match the eps with meter space as the circle's radius. The circles indicate which surrounding points are within its eps range and should be treated as a cluster neighbor (see Equation (1)). It should be noted that eps is the ε as specified in Equation (1).

Table 2
Obtained distance matrix based on the data presented in Table 1

Index	0	1	2	3	4	5	6
0	0.000000	0.282715	1.128214	1.537755	1.935583	2.180580	2.450144
1	0.282715	0.000000	1.053958	1.399959	1.790722	2.044759	2.534211
2	1.128214	1.053958	0.000000	0.488360	0.866289	1.085500	1.682116
3	1.537755	1.399959	0.488360	0.000000	0.398247	0.645561	1.825003
4	1.935583	1.790722	0.866289	0.398247	0.000000	0.260446	1.866630
5	2.180580	2.044759	1.085500	0.645561	0.260446	0.000000	1.833376
6	2.450144	2.534211	1.682116	1.825003	1.866630	1.833376	0.000000

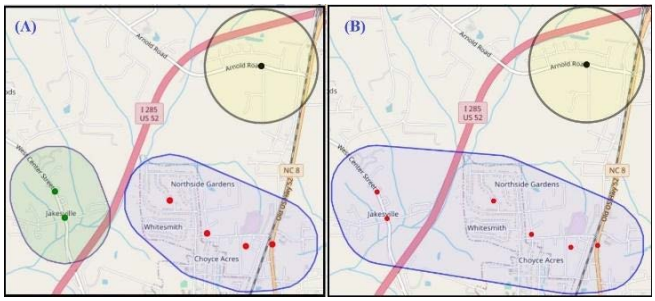

Figure 3. Two different results: (a) results obtained by configuration 1 with other languages, (b) results obtained by configuration 1 and 2 with python's DBSCAN.

4. RESULTS

A. *Investigation 1*

i. *Analysis using Python*

The "sklearn" package of the Python programming language has a built-in DBSCAN function. In general, there are several parameters used in this function. However, often the majority of the settings are not set by the users, which means those are set to the default system values and so no user input is required. However, the parameters eps (the distance between two points), minimum number of samples (i.e., number of samples required to be classified as a cluster), and the standard metric scale (the metric scale used for distance calculation) were changed for this study. Data used in this experiment is presented in Table 1. We noticed one cluster and an outlier using both configurations. However, according to the other programming languages, they consider all the data points as outliers for configuration 2 and only make two clusters and an outlier for configuration 1 as shown in Figures 5(a) and 5(b). Only the built-in function of Python demonstrates the same outcome for both configurations.

ii. *Analysis using Postgis*

The experimental data used in this study is presented in Table 1, where we used eps = .00548, and then the min_sample = 2. The clustering results we discovered were two clusters and an irregular that is an outlier data point.

iii. *Analysis using JavaScript*

Data used in this experiment presented in Table 1, we used eps = 0.00548, and as for min_sample = 2.0. The obtained result indicated that there are two clusters and an outlier (see Figure 4).

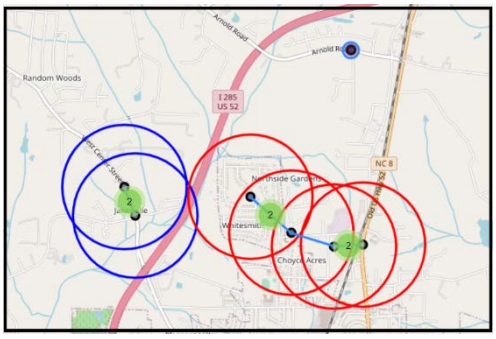

Figure 4. Clustering results obtained from JavaScript's DBSCAN function

iv. *Analysis using R*

Using the given data from Table 1, with the same configuration of eps = 0.00548, and parameter min_sample = 2.0, we discovered two clusters and an irregular outlier. Later on, we repeated our configuration with eps = 0.00548 and the parameter min sample = 5 and discovered that if the cluster needs at least 5 minimum points to form, then it decides to label all the data points as outliers.

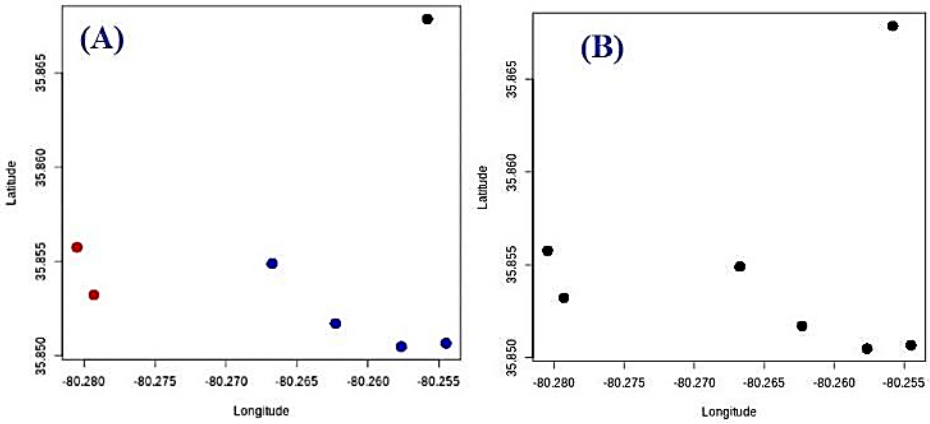

Figure 5. Results were obtained from all the other programming languages except python considering (a) configuration 1 and (b) configuration 2

B. *Investigation 2*

We conducted a second experimental observation to acquire further evidence. In contrast to previous experiments, in the second experiment, we used all four programming languages to evaluate the results of the DBSCAN clustering function. In the second trial, the settings were eps = 0.00548 and min sample = 2.0. Table 3 displays the obtained results. Table 3 shows the cluster findings based on the experimental data shown in Table 1.

C. *Investigation 3*

In the third experimental observation, the built-in function of sklearn was reevaluated, and the intention was to find the reason why the function was not providing the same results as other languages. For spatial data, often degree and radian concepts are used, so, for the sake of analysis, the longitude and latitude values were converted to radians and then passed in the built-in function of sklearn.

Table 3
Experimental results from different programming languages (data used in this experiment is presented in Table 1)

Language name	Results with configuration 1	Results with configuration 2
Python	Two clusters with an outlier	Two clusters with an outlier
Postgis	Two clusters with an outlier	All data points are outliers.
Javascript	Two clusters with an outlier	All data points are outliers.
R	Two clusters with an outlier	All data points are outliers

This time, the built-in function provided the same result as other languages. So, the evidence from this study suggests that sklearn's DBSCAN function requires radians as input, unlike other languages. So, despite the similar parameter settings, sklearn's built-in function requires different types of input than the rest of the languages (see Figure 6).

Figure 6. Clustering results from Sklearn's DBSCAN function. (a) Shows results before converting to radians, and (b) shows the results after converting to radians

5. EVALUATION

Further, publicly accessible data was utilized to evaluate the built-in functions. The locations of all countries' names, as well as the associated latitude and longitude numbers, were contained in the publicly available data. The data was obtained from Google data (Google: Dataset publishing language). The specifications utilized in the assessment study were eps = 0.9000009 and min_sample = 4 which is considered configuration 3. Figure 7 depicts the obtained results. According to the evidence, JavaScript, R, and Postgis produced identical outcomes. The programming language Python, on the other hand, produced a different outcome (see Figure 8). In Python, the findings shown in Figure 4 exhibited just a cluster but no outliers. However, the aesthetic map in Figure 3 demonstrates that there seemed to be just four geospatial information in a cluster, with the remainder being outliers. To validate the countries' records, we created a distance matrix among all geospatial data, and the findings showed that only four geospatial data might as well have produced a cluster, as seen in Figure 7.

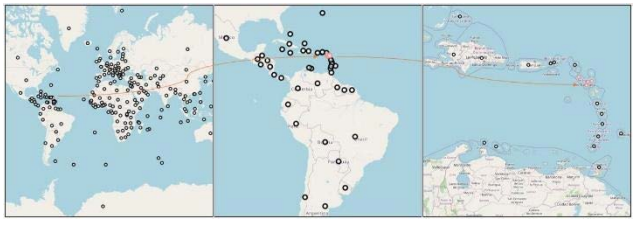

Figure 7. Clustering result: A single cluster formed using four geo-locations (i.e., red markings) and the rest geolocations did not form any cluster (i.e., those are considered as outliers)

Figure 8. Clustering results on the world map, center of each countries and territories are used in the experiment to cluster them

Figure 7 and Figure 8 uses configuration 3 to form clusters where the clusters illustrated in Figure 7 is the output of PostGis, R, and JavaScript implementations of DBSCAN, and Figure 8 is the output of Python's implementation of DBSCAN. This variation may be caused due to Python's built-in function requiring a different type of input, namely radian input as the eps than the rest of the languages (see Figure 6).

6. DISCUSSION

Numerous software engineers and specialists have faith in using the functions of the integrated library and regularly use them to acquire research results and take care of issues (Hao *et al*., 2019). Applications of such built-in library functions (e.g. DBSCAN algorithm) can be found in several exploration studies. DBSCAN is one of the well-known spatial data algorithms (Hahsler *et al*., 2019). However, as far as we know, no such research exists that compares and evaluates DBSCAN's implementation across all platforms. Usually, researchers rely solely on the built-in DBSCAN function (Hao *et al*., 2019). Hence, research to assess DBSCAN algorithms integrated into numerous programming languages or platforms was long overdue. This analysis looks at how the DBSCAN algorithm performed in several languages, for example, R, Python, R, JavaScript, and PostGis. And the identical hyper-parameters, for example, eps, and min_points were used to accomplish this.

Built-in DBSCAN functions in three computer languages, including R, Postgis, JavaScript, and R, appear to provide the same findings. Conversely, the Python language's built-in function appears to produce results that differ from those of other languages. As a result, the study advises researchers to exercise caution while using the built-in DBSCAN tool. Perhaps researchers should experiment with different languages for spatial clustering. The implementation of Python's built-in DBSCAN function was not covered by this study. DBSCAN implementations in other programming languages should be compared in a future study.

The Python programming language's dissimilarity might be attributed to the unit of the eps value (Starczewski et al., 2019). Evidence reveals that the eps value is used in degree distance by three programming languages: JavaScript,

Postgis, and R. As a result, in Python, the eps value is not in degrees. One possible cause is that Python's eps unit is measured in radians (Boeing et al., 2018). According to the authors of research (Boeing et al., 2018), the Python language library use the eps value in radian distance. This, on the other hand, should not have happened. As a result, the Python implementation will be distinct from other implementations.

7. CONCLUSIONS

In this paper, we assessed the built-in DBSCAN function in a variety of computer languages, including R, Python, Javascript, and PostGIS. The input numbers and parameters used to perform these comparisons were identical in each case. Our results reveal that the Python scikit-learn DBSCAN implementation generates conflicting outcomes than the other implementations. As demonstrated in the benchmark section, DBSCAN of scikit-learn prefers to group all of the geospatial data from publicly accessible countries csv into one cluster, whereas built-in DBSCAN of PostGis only groups four geospatial data from all data points into one cluster.

This inexplicable response of DBSCAN's Python implementation verifies our assumption that "we should not simply entrust the results of built-in functions undoubtedly," and this investigation backs this up. According to our study, scikit-learn's DBSCAN implementation has a high level of abnormality, whereas all other platforms produce similar results when the same parameters are used. DBSCAN is used to cluster geospatial data, so the clustering should be the same across all platforms. Despite the fact that all of the other platforms clusters identically and properly, scikit-learn's DBSCAN performs completely differently as well as turns as an outlier when equated to the other programming languages. The next step will be to identify the elements that influence Python's Scikit-learn outcomes.

According to the findings from this investigation, the eps (i.e., ε) value in DBSCAN functions across many programming languages is in degree; particularly the ones investigated in this study. Apparently, it appears to be ineffective for Python's installed DBSCAN algorithm. Furthermore, there is no clear information on the Python official website regarding the unit of eps [32]. In the integrated DBSCAN function, a deeper analysis may look at how many past researches utilized degree as an eps unit against how many utilized radian as an eps unit.

ACKNOWLEDGEMENTS

The part of this research was supported by the Institute of Energy, Environment, Research, and Development (IEERD, UAP), the University of Asia Pacific. Md Amiruzzaman was supported by West Chester University tenure-track faculty start-up grand.

REFERENCES

Amiruzzaman, M. (2018, November). Prediction of traffic-violation using data mining techniques. In *Proceedings of the Future Technologies Conference* (pp. 283-297). Springer, Cham.

Amiruzzaman, M., Rahman, R., Islam, M. R., & Nor, R. M. (2021, November). Evaluation of DBSCAN algorithm on different programming languages: An exploratory study. In *2021 5th International Conference on Electrical Engineering and Information & Communication Technology (ICEEICT)* (pp. 1-6). IEEE.

Berry, M. W., Mohamed, A., & Yap, B. W. (Eds.). (2019). *Supervised and unsupervised learning for data science*. Springer Nature.

Boeing, G. (2018). Clustering to reduce spatial data set size. *arXiv preprint arXiv:1803.08101*.

Cranor, L. F. (1994). Programming perl: an interview with larry wall. *XRDS: Crossroads, The ACM Magazine for Students*, *1*(2), 10-11.

Clustering algorithms: their application to gene expression data. *Bioinformatics and Biology insights*, *10*, BBI-S38316.

Dudik, J. M., Kurosu, A., Coyle, J. L., & Sejdić, E. (2015). A comparative analysis of DBSCAN, K-means, and quadratic variation algorithms for automatic identification of swallows from swallowing accelerometry signals. *Computers in biology and medicine*, *59*, 10-18.

Davies, D., & Bouldin, D. (1979). A cluster separation measure, IEEE transactions on patter analysis and machine intelligence. vol.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* (Vol. 96, No. 34, pp. 226-231).

Fischer, B., & Buhmann, J. M. (2003). Path-based clustering for grouping of smooth curves and texture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(4), 513-518.

Gan, J., & Tao, Y. (2015, May). DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (pp. 519-530).

Google: Dataset publishing language. https://developers.google.com/public-data/docs/canonical/countriescsv, accessed: 2021-01-12.

Handra, S. I., & Ciocârlie, H. (2011, May). 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)* (pp. 75-83). IEEE.

Hao, J., & Ho, T. K. (2019). Machine learning made easy: a review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics*, *44*(3), 348-361.

Hahsler, M., Piekenbrock, M., & Doran, D. (2019). dbscan: Fast density-based clustering with R. *Journal of Statistical Software*, *91*(1), 1-30.

Islam, M. R., Jenny, I. J., Nayon, M., Islam, M. R., Amiruzzaman, M., & Abdullah-Al-Wadud, M. (2021, August). Clustering Algorithms to Analyze the Road Traffic Crashes. In *2021 International Conference on Science & Contemporary Technologies (ICSCT)* (pp. 1-6). IEEE.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern recognition letters*, *31*(8), 651-666.

Karim, M. R., Beyan, O., Zappa, A., Costa, I. G., Rebholz-Schuhmann, D., Cochez, M., & Decker, S. (2021). Deep learning-based clustering approaches for bioinformatics. *Briefings in Bioinformatics*, *22*(1), 393-415.

Limwattanapibool, O., & Arch-int, S. (2017). Determination of the appropriate parameters for K-means clustering using selection of region clusters based on density DBSCAN (SRCD-DBSCAN). *Expert Systems*, *34*(3), e12204.

Luchi, D., Rodrigues, A. L., & Varejão, F. M. (2019). Sampling approaches for applying DBSCAN to large datasets. *Pattern Recognition Letters*, *117*, 90-96.

Mahmoudi, M. R., Baleanu, D., Mansor, Z., Tuan, B. A., & Pho, K. H. (2020). Fuzzy clustering method to compare the spread rate of Covid-19 in the high risks countries. *Chaos, Solitons & Fractals*, *140*, 110230.

MacQueen, J. (1967, June). 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, pp. 281-297).

Niemierko, A., & Goitein, M. (1990). Random sampling for evaluating treatment plans. *Medical physics*, *17*(5), 753-762.

Oyelade, J., Isewon, I., Oladipupo, F., Aromolaran, O., Uwoghiren, E., Ameh, F., ... & Adebiyi, E. (2016).

Prasetya, D. A., Nguyen, P. T., Faizullin, R., Iswanto, I., & Armay, E. F. (2020). Resolving the shortest path problem using the haversine algorithm. *J. Crit. Rev*, *7*(1), 62-64.

Rizvee, M. M., Amiruzzaman, M., & Islam, M. R. (2021). Data Mining and Visualization to Understand Accident-Prone Areas. In *Proceedings of International Joint Conference on Advances in Computational Intelligence* (pp. 143-154). Springer, Singapore.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53-65.

Ramalho, L. (2015). *Fluent Python: Clear, concise, and effective programming*. " O'Reilly Media, Inc.".

Sibson, R. (1973). SLINK: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, *16*(1), 30-34.

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)*, *42*(3), 1-21.

Starczewski, A., & Cader, A. (2019, June). Determining the EPS parameter of the DBSCAN algorithm. In *International Conference on Artificial Intelligence and Soft Computing* (pp. 420-430). Springer, Cham.

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

Wu, C. H., Ouyang, C. S., Chen, L. W., & Lu, L. W. (2014). A new fuzzy clustering validity index with a median factor for centroid-based clustering. *IEEE Transactions on Fuzzy Systems*, *23*(3), 701-718.

Zhou, A., Zhou, S., Cao, J., Fan, Y., & Hu, Y. (2000). Approaches for scaling DBSCAN algorithm to large spatial databases. *Journal of computer science and technology*, *15*(6), 509-526.