The Use of Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

1st Lt Kevin Choe

I. Introduction

WHILE practicing precise science, we often struggle with analyzing collected data that is complex and not conceptually easy to organize. This problem becomes further complicated and intensified in certain research topics which require many data management tools. Computing tools are becoming increasingly important in assisting engineers, scientists, and data analysts for their projects. In the nuclear engineering world, we are often required to use computing tools more than any other field of study due to the substantial number of data points that we deal with. DBSCAN is a data clustering algorithm that has been used by scientists and engineers around the world who deal with data heavily during their research. In this paper, we will introduce how this algorithm works and explain how to code it explicitly using Python, a well-known software. Hopefully, this paper provides some useful guild lines for those scientists and engineers who need a computing tool for their data analysis projects, especially for those who are beginners in the programming world.

A. Problem Description

1) Nuclear Forensics: In the nuclear forensics career field, samples of new data are often collected from areas where nuclear activities occur, such as locations with nuclear reactors and/or places building and testing nuclear weapons. With the data received from these sites, investigations (data analysis) are conducted to define whether those activities were done legally or illegally. For instance, the US has allowed countries like North Korea to run nuclear reactors for the economic growth and prosperity of their people. However, they later used those facilities to enrich materials solely to build nuclear weapons which currently threaten world peace. In order to prevent a catastrophic nuclear situation, we need a mechanism to find potential nuclear schemes before these problems become world issues. One of the many ways to achieve this goal is by collecting materials at the location of the investigation and comparing its data with sets of preexisting data. This strategy, which will trace nuclear activity, is possible with the enormous amount of data that our country has gathered through previous experiments and tests.

Now that the problem is presented, engineers and data analysts are left with the task to investigate by developing several mechanisms to distinguish characteristics of data and understanding what is occurring with the sample materials collected. As introduced before, DBSCAN is a data clustering algorithm that groups together data points that may have the same material properties. This concept is one of many useful tools used in nuclear forensics. If we have a set of preexisting data whose characteristics and properties have been defined already and separately recoded by groups, we could simply add a new set of data (subjected to investigation) into that preexisting data to find similarity with pre-characterized clusters. By defining their data properties, perhaps, as an outcome, we will detect crime in a nuclear forensics case.

2) Clustering Data: Before we introduce the DBSCAN algorithm, we need to know what clustering means in data analysis and recognize how powerful a data clustering computing tool can be. Clustering is the process of grouping similar objects together; it can be thought of as a search for similarity. The topic of clustering deals with how rational region would understand similarity and how this concept could be used to categorize items. One approach to this problem is to relate similarity with distance. Under this approach, two items in a search space are similar to each other if the distance between them is small. DBSCAN is the algorithm which takes this approach. As figure 1 shows, data points within certain distances from each other are considered to be in the same group.

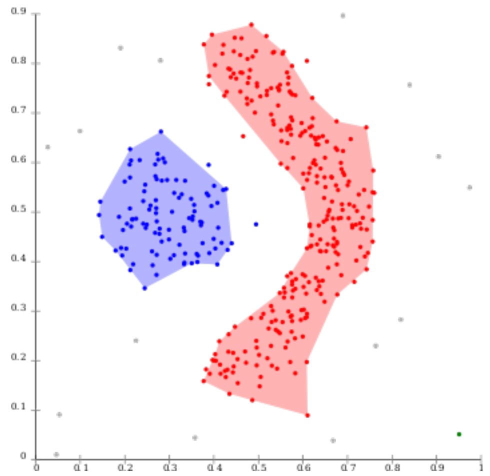

Fig. 1. Clustering data points

3) DBSCAN Algorithm: DBSCAN algorithm takes three parameters as input:

1) A set of data points is to be subjected in this method.

2) Neighborhood values (epsilon) is used to set the cutoff distance for similarity in the data set.

3) Minimum points parameter defines a minimum number of neighborhood points required to form a group (cluster).

For the output, DBSCAN will return the collection of clusters with each cluster as a high density region in the search base.

In DBSCAN, a single object is represented as a numerical point in space. The neighborhood of a point is the set of all points that are within a distance set by the neighborhood parameter. In this algorithm, each point is labeled as either core, border, or noise point (see figure 2):

1) A core point is any point that has at least a defined minimum number of points in its neighborhood.

2) A border point is a non-core point that has at least one core point in its neighborhood.

3) A noise point is neither a core point nor a border point. It represents outliers in a data set that dont belong to any particular clusters.

An algorithm picks an unassigned core point in the dataset and then performs a search starting at that point. For the search, let X be the current point that is being explored. At all the points in Xs neighborhood, the program recursively applies the search to each unexplored core point in that neighborhood. The resulting tree structure represents a single cluster of points. Then, the DBSCAN algorithm repeats the search until all core points have been assigned to a cluster.

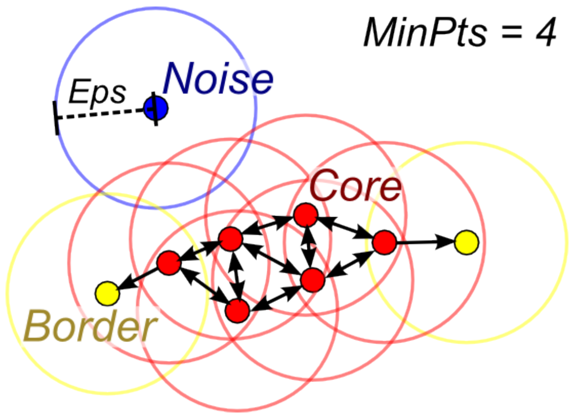

Fig. 2. Core points, border points, noise points defined by given Eps and MinPts

B. Description of Work

Achieving this mechanism in programming can be challenging but feasible. In this paper, we will talk about how the DBSCAN algorithm can be coded in Python, a free software that is being used by many engineers and scientists who seek for cost-effective ways to acquire useful research tools. First, a software called "pandas" can be used to import data from ".csv" or Microsoft Excel type programs to Python.

Once that data gets loaded in Python, then we define a couple of key variables that are essential to the DBSCAN algorithm:

1) Epsilon (ϵ) is known as neighborhood values (mentioned above) and is the radius of points which are used to define its neighborhood points within its parameter.

2) Minimum sample points (μ) are a numerical value that defines the number of neighborhood points required to form a cluster.

For example, if the number of points do not satisfy the given μ value within its radius (ϵ), then those multiple points would not be considered neighborhood points; this would result in no cluster being formed. Therefore, with the high numerical value of ϵ, we expect to see clustering behavior between widely-spread-out points in space; with smaller numerical value of μ, we expect to see high number of clusters being formed in space.

When those key variables are defined, we incorporate "if" and "for" loops along with conditions which enforce the rules of ϵ and μ over all data points being analyzed. As a result, the program calculates the number of clusters detected and visually represents cluster groups in a plot. Figure 3 is the plot we created using Python with a data set inherited from a previous graduate student. For further analysis of the clustering pattern, we reduced point sizes to prevent some data points from being hidden behind other points (Figure 4).

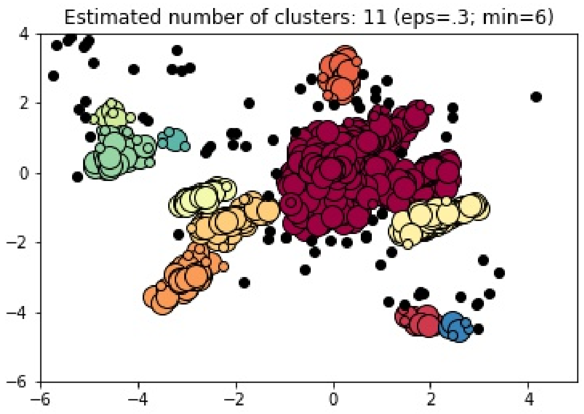

Fig. 3. Result plot @ ϵ =0.3; μ = 5

As the product evaluating processed, error bars were applied to all points in the plot. Error bars were set (projected in both x and y directions from all points on the plot) to have the same length as ϵ. From Figure 3, applying error bars, we get Figure 5. As we zoom in to a certain space of interest (Figure 6), those error bars would help us recognize how ϵ and μ are effecting clustering patterns as we change the magnitude of those variables per trial. The analysis of the ϵ and μ effects are captured in the result section of this paper.

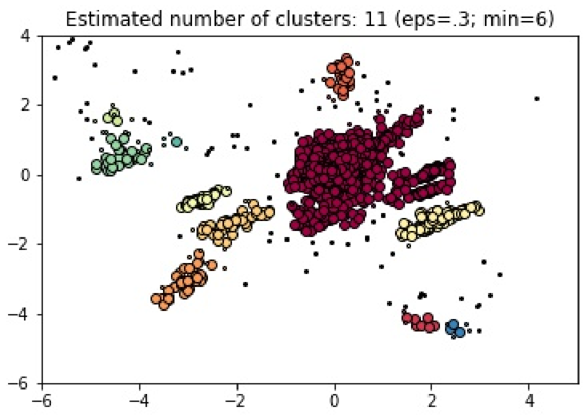

Fig. 4. Same set up as Figure 3, reduced point size

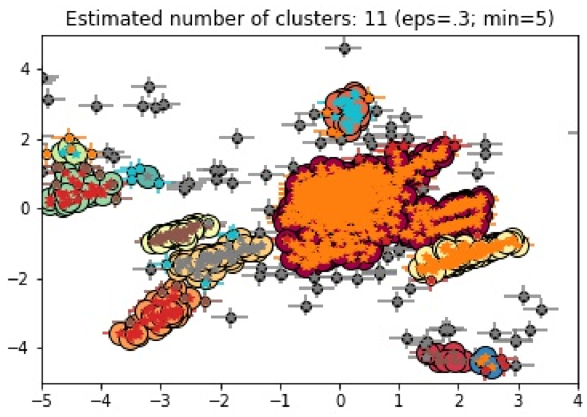

Fig. 5. Length of error bars = ϵ = 0.3 in both x,y directions

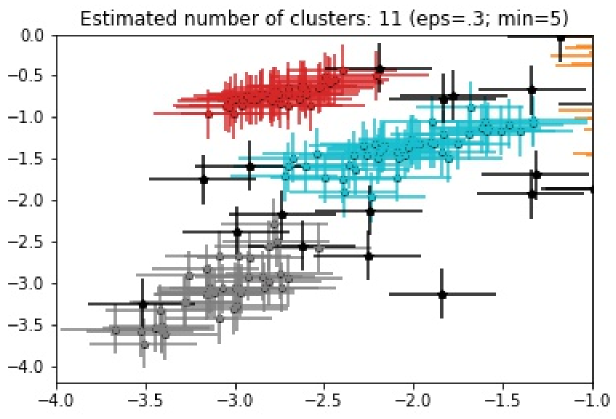

Fig. 6. Zoomed in to a space of interest for analysis

The product outcome is a plot that visually displaces clusters of data. However, in reality, this visual representation does not help much to investigate data property unless there is a data summary sheet which provides information such as what a data point is defined as (either core, border, or noise) and which cluster does it belong to. Therefore, we added more features to this program so it does not only print out a plot, but also creates a ".csv" (Microsoft Excel) type data summary table, which contains information about the classification of all data points. For data points that were recognized to be in a cluster, this summary table also classifies which cluster group they belong to. This feature would help anyone interested in knowing the properties of a specific data point. We will go over how this feature works in the result section as well.

C. Results

As expected, changes in ϵ and μ effect clustering patterns. From Figure 6, changing $\mu \rightarrow 10$, we get Figure 7. Consider that the "Estimated number of clusters" found in Figure 7 is not well represented in this "zoomed in" snapshot. However, if we just visually inspect this specific space, we can see that each cluster gets less core points. With only a certain amount of points located next to each other, fewer points are found as core while points the μ value gets larger. When compare to Figure 6, Figure 7 gets more outliers because fewer points are found as core points. The less core points are found in a cluster, the more points are found as outliers.

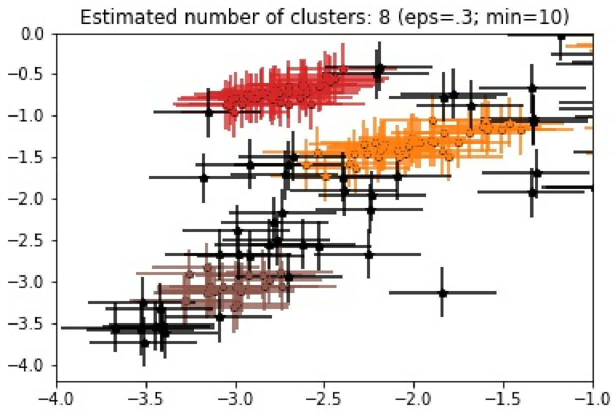

Fig. 7. Same set up as Figure 6, but μ = 5 $\rightarrow$ 10

From Figure 6, changing $\epsilon \rightarrow$ 0.7, we get Figure 8. As we increase the parameter (ϵ), each point will have more neighborhood points around them, causing fewer large clusters to form. Therefore, we see less outliers with the bigger value of ϵ.

When we change both $\epsilon \rightarrow$ 0.5 and $\mu \rightarrow$ 8, we get Figure 9. It is hard to predict many more or less clusters would be found as we change both the ϵ and μ concurrently. As we see in Figure 7, the larger value of μ can cause more or less clusters depending on how densely points are located by each other.

With the human eye, it is hard to tell the difference between border points and core points in a result plot. Border points are hard to detect because they reside between core points

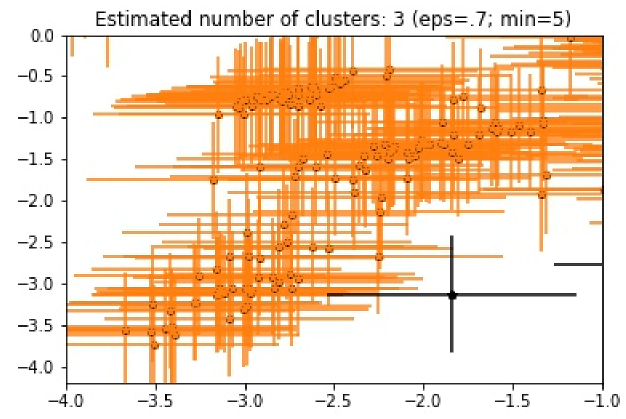

Fig. 8. Same set up as Figure 6, but ϵ = 0.3 $\rightarrow$ 0.7

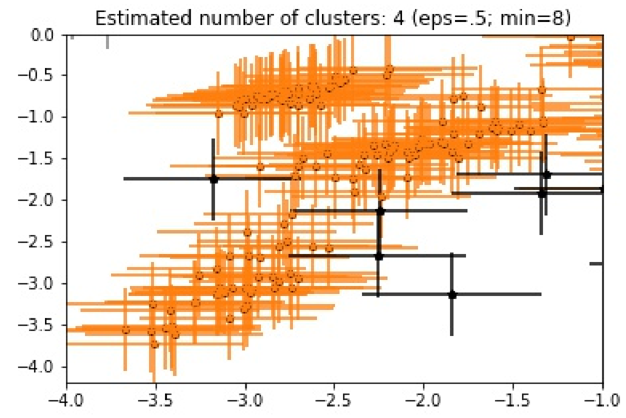

Fig. 9. ϵ = 0.3 $\rightarrow$ 0.5 and μ = 5 $\rightarrow$ 8

and noise points; border points are especially difficult to differentiate because they are colored the as same as core points and are located right next to them. We need a data table to display this type of information more explicitly, otherwise, we will not know the numerical representation of each point.

During the last step, the program is updated so it creates a data summary. Table 1 shows a sample data summary table that Python print out in ".csv" format. The first column is an index of each datum. The second and third columns display the original X and Y input values. The forth, fifth, and sixths columns display core, border, and noise information; respectively, "1" means "Yes" and "0" means "No". The seventh column displays information about which cluster each datum belongs to if it is not defined as a "noise" point; cluster numbers are indicated from 0th to nth; "-1" means the datum is a "noise".

Suppose we wanted to analyze a set of random data. That data set would simply need to be put into a preexisting data set and compare their relation to each other. We can add a new data set to a preexisting data set in "`DBSCAN_data.py`". Then, the run file "`DBSCAN_analyzed_data_table` " could be used since it is already programmed to import data from the "`DBSCAN_data.py`" file. This run file will display information in the "Variable Explorer" box (located above the console box in Spyder) in array form and also generates summary data in ".csv" format.

Table 1. Result Table "`Data_Summary.csv`"

Index	X Data	Y Data	Core	Border	Noise	Cluster
0	-0.454	0.74143	1	0	0	0
1	-0.0127	0.149	1	0	0	0
2	-0.0801	0.0513	1	0	0	0
...	...	...	...	...	...	...
692	-2.73	-2.17	0	0	1	-1
693	-2.77	-2.28	0	1	0	3
694	-1.77	-0.738	0	1	0	5
...	...	...	...	...	...	...

D. Conclusion

We have gone over the fundamental knowledge of DBSCAN, what needs to be done with coding, and how we can further improve the program. This project was successful because a robust computing tool was built to:

1) Define which cluster each datum belongs to by looking at the "labels" array or the "`Data_Summary.csv`" table;

2) Define how accurately new points are related to preexisting points by determining core, border, or noise points;

3) Regulate variables (ϵ and μ) to analyze data in a various ways as each situation and desired outcome may vary based on the intention of use.

The DBSCAN algorithm is a powerful tool because it can:

1) Discover any number of clusters;

2) Find clusters that vary in size and shape;

3) Detect and ignore outliers (noise points) in data.

On the other hand, this method is not the absolute solution to all problems:

1) The algorithm is sensitive to the choice of the neighborhood parameter.

2) If the number of neighborhood points are too small, then that small cluster would be labeled as an outlier. We could miss out a potentially valuable cluster by this defect.

3) If the neighborhood is too large then dense clusters would be merged together. In other words, if ϵ is too large than we would detect a cluster that may be better indicated in separate groups.

Due to the drawbacks listed above, it would be wise to have a good understanding of those two variables (ϵ and μ) before use.

To further the knowledge of the DBSCAN algorithem, additional coding may be implemented in the future so the program can generate percentage displays about how close each points are located to one another. To future DBSCAN users, I hope that this paper guides you to success.

References

[1] DBSCAN. (2017, December 01). Retrieved December 01, 2017, from https://en.wikipedia.org/wiki/DBSCAN

[2] Sklearn.cluster.DBSCAN. (n.d.). Retrieved December 02, 2017, from http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html