DBSCAN: Density-Based Spatial Clustering of Applications with Noise

Peter Wu

December 7, 2022

1 Introduction

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised learning algorithm for clustering proposed by Ester in 1996 [1]. It was the first algorithm to address some of the major shortcomings with previous well-known clustering algorithms: (1) minimal domain knowledge to determine input parameters (2) discovery of clusters with arbitrary shapes (3) good efficiency on large databases. DBSCAN has a vast array of applications in biology, statistics, medicine, mathematics, and computer science. Because it is an algorithm based on densities, it is particularly effective for abnormal data, especially in the clustering of spatial data [2, 4, 6].

In clustering, we are given a dataset $\{x_i\}_{i=1}^{i=n}$ and our task is to group the $x_i \in \mathbb{R}^d$ that are similar based on the closeness of the features. This is an unsupervised learning problem because we are not given the labels y_i associated with each input vector x_i. Intuitively, we want to minimize the intra-cluster distance within any particular cluster and maximize the inter-cluster distance between clusters. There are many metrics that capture these specifications, like the Dunn index [5]. More formally, DBSCAN solves the problem of assigning each point in d-dimensional space to a particular cluster or group. For C clusters named $\{1, 2, \ldots, C\}$, we assign each point $x_i \in \mathbb{R}^d$ to exactly one of the C clusters, or a label as a noise or outlier point (a characteristic unique to the DBSCAN setup).

DBSCAN has several advantages over popular clustering algorithms. In a traditional K-Means clustering setup, we would have to input K (or the number of clusters, C in our instance) [3]. Typically in a DBSCAN setup, we are not given this value of C, and one of the algorithm's most salient characteristics is to be able to generate an output that determines the number of clusters without being explicitly told. Furthermore, DBSCAN is able to cluster well when the data exhibits arbitrarily shaped clusters where methods like Gaussian Mixture Models (GMM) would not perform well because this particular method, for example, make assumptions about data being centered at a point with some variation around it for each cluster. DBSCAN does not make assumptions about the data being drawn from a Gaussian like in GMMs, which makes it generalize well on a variety of different cluster shapes, even the ones where GMM performs well on.

DBSCAN solves the clustering problem by defining what are referred to as core points, border points, and noise points. It first finds points that are surrounded by many points (specifically, at least `minPts` within an ϵ-radius neighborhood, or an ϵ-radius hypersphere) which are called core points. Next, for points that are not core points but contain a core point in an ϵ-radius neighborhood are then labelled as border points. The remaining points are noise points or outliers and are not considered in the next step. Finally, DBSCAN starts creating clusters with the core points by essentially grouping the core points that are in the same neighborhoods.

In 2, we will describe the DBSCAN algorithm in detail, defining along the way some important key terms and concepts needed in understanding the algorithm. In 2.1, we review the problem setup and crystallize the linear algebra notation we will use throughout the section. In 2.2, we discuss the hyperparameters of the algorithm, `MinPts` and `Eps`, which define what constitutes being in the **neighborhood** of a point. In 2.3,

we use those hyperparameters to label points as core, border, or noise points, which will help us intuitively cluster the points. In 2.4, we will use the concepts in 2.3 to define what a **density-edge** is and also what it means for two points p and q to be **density-connected**. Finally, in 2.5, we will use all the concepts and terms we use previously in the section to define exactly what the DBSCAN algorithm does to cluster points.

In 3.1 and 3.2, we will discuss the strengths and weaknesses of DBSCAN, explaining and understanding them intuitively from a geometric point of view. To close out, in 4 we will utilize DBSCAN on different simulated datasets to better understand where DBSCAN may be more effective than others and also where it will perform worse, reinforcing our geometric intuitions from 3.1 and 3.2.

2 Method

2.1 Notation and Problem Setup

We are given an $N \times d$ data matrix, where we have N data points x_i, note $i \in \{1, 2, \ldots, n-1, n\}$, with each $x_i \in \mathbb{R}^d$. In a traditional clustering setup, our goal is to assign each point x_i to exactly one of C clusters: $\{1, 2, \ldots, C-1, C\}$. DBSCAN performs this task, but adds a twist in that it can also assign points as a **noise** point, thus not belonging to any one of the C clusters.

We view $\vec{x_i}$ as a d-dimensional vector representing the d observations taken from the i^{th} data point More formally, we can write $\vec{x_i} = (x_{i1}, x_{i2}, \ldots, x_{id})^T$.

We define the **distance** between two points x_i and x_j as $d(x_i, x_j)$, and write:

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{id} - x_{jd})^2}.$$

Note that for the purposes of this term paper, we will only consider the **Euclidean distance** between two points x_i and x_j. But one can use any distance metric they see fit, and the DBSCAN algorithm will still work. For example, one can define $d'(x_i, x_j)$ using the Manhattan distance and use the following formula:

$$d'(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{id} - x_{jd}|.$$

As a consequence of using distances that are not Euclidean distance, note that our geometric intuition understanding of the algorithm will weaken. Our original ϵ-hypersphere neighborhood around our data points to define a *dense* region will no longer hold, and the new interpretation will depend on the new distance metric being used.

2.2 Hyperparameters (MinPts and Eps)

The two hyperparameters we must input into DBSCAN are `MinPts` and `Eps`. Loosely, these two hyperparameters roughly define what it means for a point to be in a *dense* region. For a given point, we are specifically interested in how many points (including itself) are within an ϵ-radius around it, or an ϵ-hypersphere in d dimensions for our example. The number of points that are within an ϵ-hypersphere around that point, including itself, is associated with our hyperparameter `MinPts`. We will see exactly how this definition is applied in the next section.

2.3 Core, Border, and Noise Points

Using 2.2, we define a point to be a **core point** if there are at least `MinPts` around it (including itself) within an ϵ-hypersphere in d dimensions. Given some point x_i, we compute the distances between that point and every point in the dataset: $\{d(x_i, x_1), d(x_i, x_2), \ldots, d(x_i, x_N)\}$. Note that $d(x_i, x_i) = 0$, and

the given point itself would be contained in the ϵ-hypersphere. The total number of values in the set $\{d(x_i, x_1), d(x_i, x_2), \ldots, d(x_i, x_N)\}$ that are less than or equal to ϵ is our value of interest. Going back to our definition, if this value is greater than or equal to `MinPts`, then that given point is a core point.

Any point that is not a core point but is within an ϵ-hypersphere neighborhood of another core point is called a **border point**. Formally, if x_i is not a core point but x_j is a core point and $d(x_i, x_j) \leq \epsilon$, then x_i is a border point.

A point that is neither a core point or border point according to the previous two definitions is deemed a **noise point**.

The following image encapsulates these three definitions well [7].

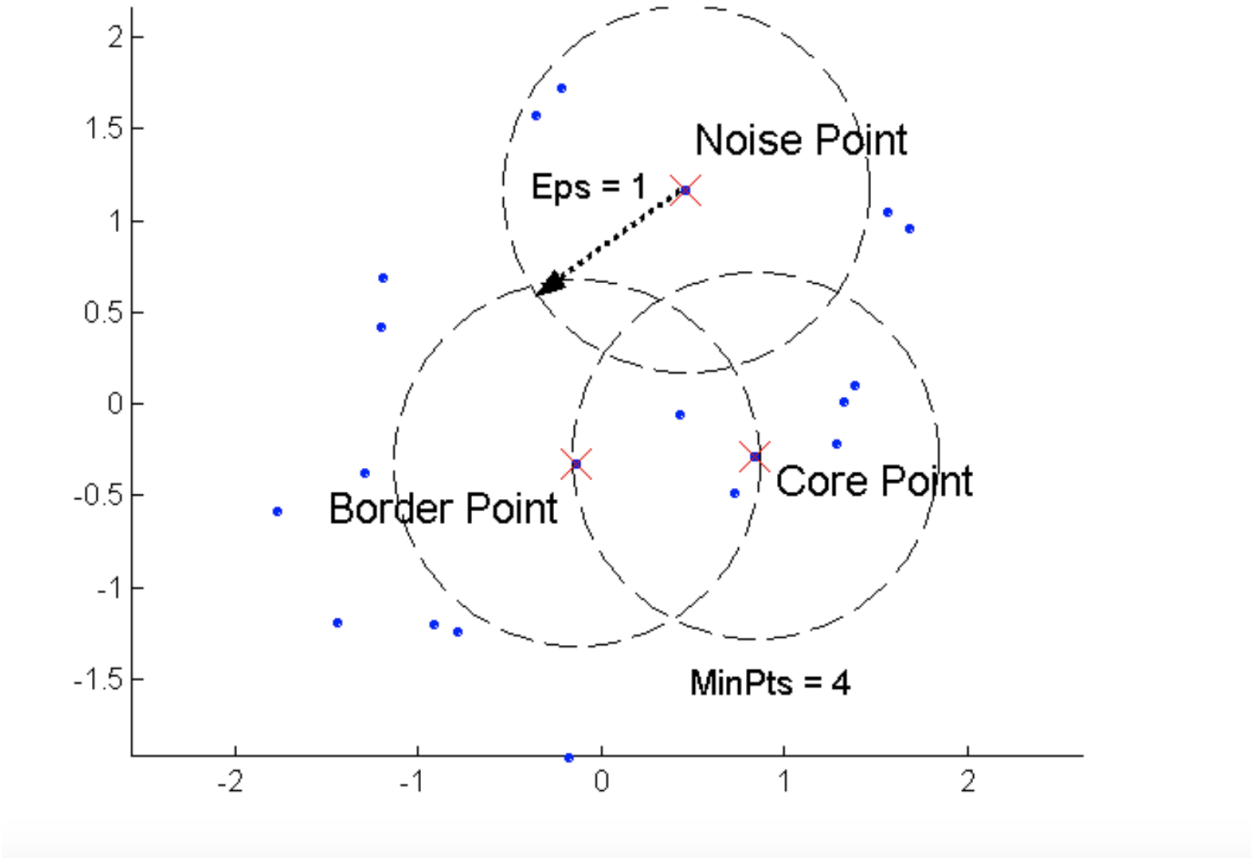

Figure 1: Core, Border, and Noise Points Geometric Intuition

2.4 Density-edge and Density-connected

Between two core points points x_i and x_j, we construct a **density-edge** between them if the distance between them is less than ϵ, or if $d(x_i, x_j) \leq \epsilon$.

Between two points x_i and x_j, we say x_i is **density-connected** to x_j if there is a path of **density-edges** that one can take to go from x_i to x_j.

The following image encapsulates these two definitions well [7].

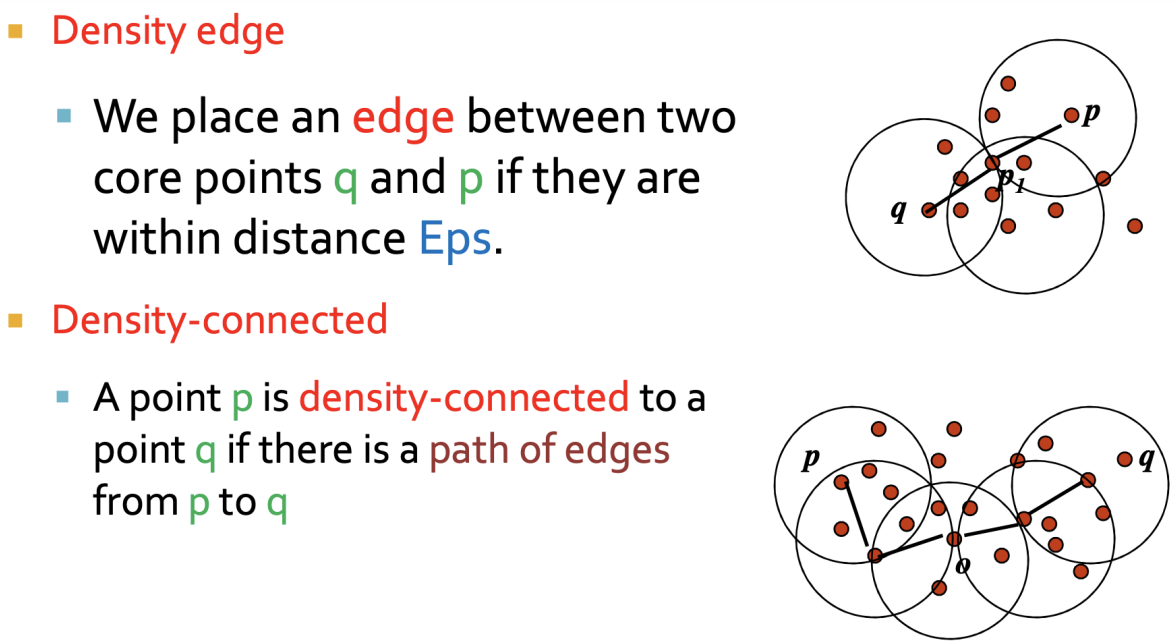

Figure 2: Density-edge and Density-connected Geometric Intuition

2.5 Formal DBSCAN Algorithm

We now describe the exact DBSCAN Algorithm in full detail. The inputs and outputs of the DBSCAN algorithm are as follows:

- Input: Dataset: $\{x_i\}_{i=1}^{i=n}$ with $x_i \in \mathbb{R}^d$, Hyperparameters: `MinPts` and `Eps`
- Output: For each x_i, one label from $\{1, 2, \ldots, C-1, C\}$ or a label as **Noise Point**

The algorithm is listed step-by-step below. We reference above sections where the terms or definitions we define previously are used in that step.

1. Label each $\{x_i\}_{i=1}^{i=n}$ as either a **core**, **border**, or **noise** point. [2.3]
2. Ignore the **noise** points for the rest of the algorithm (they can be outputted). [2.3]
3. For each core point x_i that has not been assigned to a cluster:

   3.1 Create a new cluster with x_i and add all the points that are **density-connected** to x_i to that cluster. [2.4]
4. Assign **border** points to the cluster of the nearest **core** point.
5. Output the assignments of each point as one label from $\{1, 2, \ldots, C-1, C\}$ or a label as **Noise Point**.

3 Discussion

3.1 Strengths

The strengths of DBSCAN are:

- **Ability to cluster arbitrary shapes and sizes**: Whereas modeling-based clustering techniques like GMM make assumptions about the points coming from a Gaussian distribution, DBSCAN does not and is able to handle clusters of various shapes and sizes. Intuitively, from 2.5, we can see that it is able to do so because of Step 3.1 when each **core points** adds all points that are **density-connected** to it. DBSCAN relies on the notion of neighborhoods to discover these clusters, and furthermore, as we discussed in 2.1, we are not limited to Euclidean distance. We can use whichever distance metric we believe captures the notion of neighborhoods best for the problem we are tackling.
- **Does not require user to input number of clusters**: In a traditional clustering algorithm like K-Means, the user has to input explicitly the number of clusters. In practice, though, the user may not know the number of clusters before seeing the data. DBSCAN solves this issue by being able to cluster points without being inputted the number of clusters. Intuitively, from 2.5, DBSCAN roughly will find clusters based on neighborhoods of core points, or regions that are dense.
- **Robust to outliers and noise due to nature of algorithm**: As illustrated in 2.3, one step in the DBSCAN algorithm is determining which points are points of interest to be assigned a cluster (core or border points), and which are noise points or outliers. Given a dataset with noise, DBSCAN is not required to assign **every** point to a cluster like in a traditional setup in K-Means. Thus, it is able to detect this noise and output that to the user. Intuitively, the noise points are regions that are less dense between the regions with core points that delineate the clusters the algorithm outputs.
- **Only requires two hyperparameters to run, `MinPts` and `Eps`**: As illustrated in 2.2, DBSCAN only requires two parameters to run. They define what it means for a region to be *dense*, and they are quite intuitive. A domain expert who understands their datasets very well would be able to tune these parameters very intuitively. If they know approximately how dense the regions should be with their datasets, DBSCAN would give them excellent results.

3.2 Weaknesses

The weaknesses of DBSCAN are:

- **`MinPts` and `Eps` are very sensitive to even the slightest changes**: Even just a small tweak in the values of `MinPts` and `Eps`, and the DBSCAN output can be vastly different. We will see a specific example of this in 4. Intuitively, tweaking `Eps` just slightly enough can make the difference between a point being a core point or a border point, or between a noise point and a border point. Since core points are what entirely drives how the clusters are formed in the DBSCAN algorithm, it is very inuitive geometrically to see why just small changes in these hyperparameters can lead to vastly different outputs.
- **Does not cluster well when true clusters have large difference in densities**: When there are clusters with varying densities, DBSCAN does not perform as well. Intuitively, when we pass in a `MinPts` and `Eps` in one run of the algorithm, those two hyperparameters define what a *dense* region is in that one run. Thus when there are varying densities, DBSCAN will not be able to capture those varying densities because those hyperparameters we passed in will be used as the threshold for a *dense* region. DBSCAN cannot capture both densities. We will see a specific example of this in 4.
- **Need to choose the correct distance measure between points**: Geometrically, Euclidean distance is the most intuitive for us to understand the data. But it is not always the case that Euclidean distance may be the best metric to capture the notion of neighborhoods in the data. It may be the case that the Minkowski distance of order 3 may in fact be the best metric for the data at hand. The main takeaway is that one has to choose the right distance metric to use for their dataset, and if they do not know their dataset that well, this may be particularly challenging.
- **Curse of dimensionality makes notion of neighborhoods break down in higher dimensions**: It is well known that in higher dimensions, the notion of neighborhoods breaks down and vanishes, a

phenomenon called the **curse of dimensionality**. Thus, finding the appropriate value for ϵ will be difficult when using a distance measure like Euclidean distance. Because neighborhoods vanish, the geometric interpretation for ϵ starts to become not as accurate or intuitive.

3.3 Practical Considerations when Choosing Hyperparameters

There are a couple practical considerations to make when choosing and tuning the hyperparameters `MinEps` and `Eps`.

- The larger a dataset, the larger `MinEps` should be. This makes intuitive sense because we would expect the *dense* regions to contain more points on average.
- For a dataset that is noisier, use a larger `MinEps`. This makes intuitive sense because having more points in order to deem a region *dense* would make the algorithm less susceptible to possible noise or outliers.
- For 2-d data, we can use DBSCAN's default value of MinEps = 4 [1].
- When the data has more than two dimensions, as a rule of thumb, it is recommended that we use MinEps $= 2 \times d$, where d is the dimensionality of our data [9]. This in turn satisfies the other rule of thumb that MinEps $\geq d$. Intuitively, if we think about possible `MinEps`, a MinEps $= 1$ would mean every point would be a core point, which is not what we want. When MinEps ≤ 2, we get the same result as agglomerative hierarchical clustering with single linkage (using the MIN distance) with the dendogram cut at a cluster distance of ϵ. Thus, we should generally have MinEps ≥ 3.
- Suppose we have chosen MinEps $= 4$. To determine ϵ, we could take the distance of each point to its fourth nearest neighbor, sort these distance in ascending order, and then plot them. Intuitively, for core points, the distance to the fourth nearest neighbor will be very similar, while the distance to the fourth nearest neighbor for noise points will be significantly higher. In the plot, we choose ϵ to be at the *elbow* of the plot, or where the plot makes a significant jump. We would expect that point to be sort of the demarcation between the core points and noise points. Note that this is just a heuristic and in practice, due to the randomness of data, this may not always be clear-cut. We could do the same with a different MinPts $= k$, and instead plot the k^{th} nearest neighbor for all the points.

3.4 Assessing Performance of Clustering and DBSCAN Assumptions

DBSCAN works on the assumption that the dataset exhibits clusters that are dense regions separated by sparser regions of lower density. The core points are what define these *dense* regions, the border points are on the edges between the *dense* and *sparse* regions, and the noise points are the points in these sparser regions. These sets of assumptions are very much in accord with what *intuitively* appear to be a good clustering geometrically. We can quantify these mathematically.

We can assess the performance of our DBSCAN clustering output through some of the different metrics we had briefly eluded to in 1. Note that for DBSCAN, we would remove the noise points in order to calculate these metrics. Three popular metrics for clustering are the **Dunn Index**, **Silhouette Index**, and **Davies-Bouldin Index** [12]. As we have previously mentioned, intuitively, a good clustering is one where the intracluster distances are relatively small and the intercluster distances are relatively large.

The **Dunn Index** is defined as:

$$\frac{\min_{1 \leq i \leq j \leq C} \delta(C_i, C_j)}{\max_{1 \leq k \leq C} \Delta_k}.$$

The numerator captures that between any pair of distinct clusters, we want the *minimum* intercluster distance. The denominator captures that within any given cluster, we want the *maximum* intracluster

distance. This fraction emphasizes the notion that we want the intracluster distances to be relatively small and the intercluster distances to be relatively large. Larger values of the Dunn index are better because they would enforce the good clustering where the intracluster distances are relatively small (denominator) and the intercluster distances are relatively large (numerator).

The **Silhouette Index** is defined as:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}}.$$

For a given point x_i, $a(x_i)$ is the average distance of x_i to every other point in its cluster. $b(x_i)$ is found by calculating the average distance of x_i to every point in cluster 1, the average distance of x_i to every point in cluster 2, ..., the average distance of x_i to every point in cluster C, and then taking the smallest value, excluding the average distance to its own cluster.

If $s(x_i)$ is close to 1, then $b(x_i)$ is much greater than $a(x_i)$ and x_i can be interpreted geometrically to be assigned to its *correct* cluster. If $s(x_i)$ is close to -1, then $a(x_i)$ is much greater than $b(x_i)$ and x_i would have been better being assigned to the other cluster associated with $b(x_i)$. If $s(x_i)$ is close to 0, then x_i is on the fence between the two clusters for its *correct* cluster.

We can combine $s(x_i)$ compactly by taking the average for every point in the dataset:

$$SC = \sum_{i=1}^{N} s(x_i).$$

The **Davies-Bouldin Index** is defined as:

$$DB = \frac{1}{C} \sum_{i=1}^{C} \max_{j \neq i} \frac{\sigma_i + \sigma_j}{d(C_i, C_j)}.$$

C_i denotes the centroid of cluster i, for $i \in \{1, 2, \ldots, C-1, C\}$. σ_i is the average distance of all points to the centroid C_i for cluster i. We take the average of that particular expression on the right for every cluster. The numerator captures that we want dense clusters so intracluster distances being small and the denominator captures that we want clusters to be well-spaced and so large intercluster distances. We take the maximum such value for every cluster, when compared with every other cluster. Smaller values of DB index are better because they capture both of intuitions revolving around well-spaced clusters and small, dense clusters.

3.5 Theoretical Guarantees

In the original DBSCAN paper, Ester et al. had stated that the average run time complexity of DBSCAN is $\mathcal{O}(n \log n)$ with index structures. It turns out that while range queries can be supported efficiently using R^*-trees in a wide range of applications and datasets, no formal analysis of the average runtime complexity has yet been conducted in the literature [10]. In a research paper where they revisited DBSCAN, they stated that they meant to derive informally (making a wrong assumption about the runtime complexity of range queries in R^*-trees) an average runtime complexity. They made the assumption that a range query would take $\mathcal{O}(\log n)$ time, and if were true, the average run time complexity of DBSCAN would actually be $\mathcal{O}(n \log n)$ [10].

The correct theoretical guarantee (albeit not strong) is that the worst-case runtime of DBSCAN is quadratic $\mathcal{O}(n^2)$. Even though the original paper had this slight confusion, it was noted two years later by Sanders et al. that *The runtime of GDBSCAN obviously is $\mathcal{O}(n \times$ runtime of a neighborhood query$)$. n objects are visited and exactly one neighborhood query is performed for each of them. [...] Without any index support, the runtime of GDBSCAN is $\mathcal{O}(n^2)$.* In a master's thesis 17 years later, Gunawan noted: *The running time of DBSCAN depends on how many times the function RangeQuery(p, ϵ) is called. We can see that DBSCAN*

calls this function exactly once for each point. Therefore, the total running time complexity for DBSCAN is $\mathcal{O}(n^2)$ *using the most naive way to do a range query.*

Ester et al. states: *Most secondary sources are clearly aware of the worst-case complexity but, unfortunately, often also re-state the complexity of DBSCAN when using index structures as* $\mathcal{O}(n \log n)$ *– sometimes even without qualifying it further. This runtime complexity is formally incorrect since there is no theoretical guarantee for the assumed runtime complexity of range queries, and it should not be repeated in this form. This mistake in the stated runtime complexity, however, does not mean that there is an unfixable flaw in DBSCAN, nor does it mean that DBSCAN cannot be efficient with index structures in many clustering applications.*

All in all, we should be very careful when discussing the runtime complexity of DBSCAN especially if we are talking about it in the context of range queries. The runtime complexity will essentially boil down to have fast the runtime of one particular neighborhood query will take. The controversy in the past 20 years was related to the context of this range query when supported by computer science optimizations like indexes and R^* trees.

We should reiterate what we *know* to be theoretically true. In the worst case, the neighborhood query will take $\mathcal{O}(n)$ time because we will traverse all the other elements, and if n objects are visited, the naive way of running DBSCAN will amount to $\mathcal{O}(n^2)$ runtime.

4 Example(s)

We illustrate the DBSCAN on several examples that help us reinforce our understanding of the strengths and weaknesses of the algorithm we discussed intuitively from 3.1 and 3.2.

4.1 Example 1

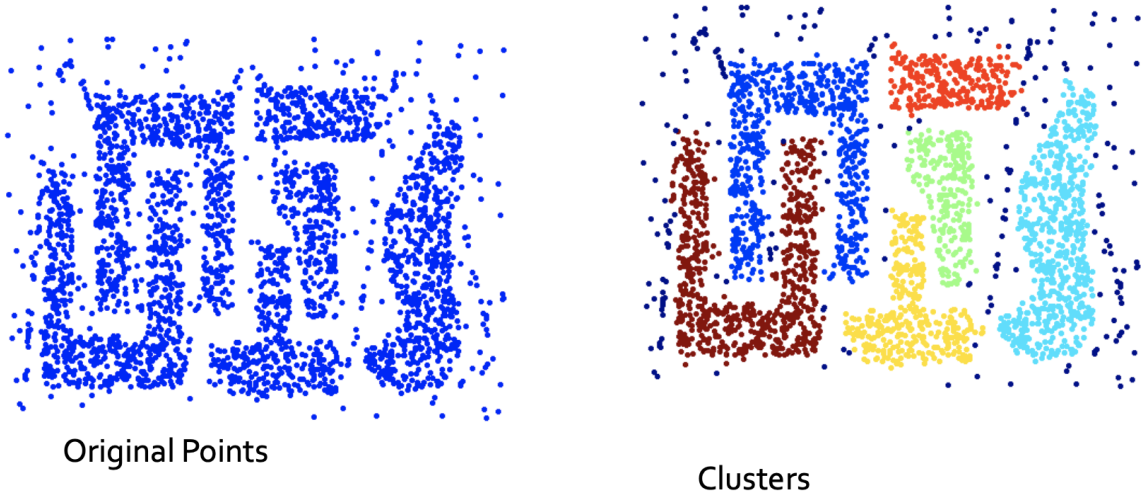

Figure 3: Example 1 Geometric Intuition

The first example is taken from [7]. It looks like a maze with these very dense regions acting like boundaries and noise points surrounding these maze-like pieces. The pieces do vary in size and shape, but they are all of similar densities.

This example demonstrates many of DBSCAN's strengths. First, it is **able to cluster arbitrary shapes and sizes**. This clustering is able to capture these maze-like Tetris shapes of different sizes. For instance,

the blue and brown U-shaped pieces are much bigger than the red and green pieces to the right. Second, the algorithm does **not require the user to input number of clusters**. Looking at the image, we can clearly see there are six clusters. If we used an algorithm like K-Means for example, we would have had to input $K = 6$ at the outset of using the algorithm. With DBSCAN, it clusters together points in the same neighborhood, from Step 3.1 in 2.5, and so inputting the number of clusters is not required. Third, it is **robust to outliers and noise due to nature of algorithm**. The darker blue points around the maze-like Tetris pieces are the noise points outputted by DBSCAN. They are in sparser regions around these denser colored regions, and the model is able to identify them and output them as noise, instead of hardly assigning them to a cluster like in K-Means.

4.2 Example 2

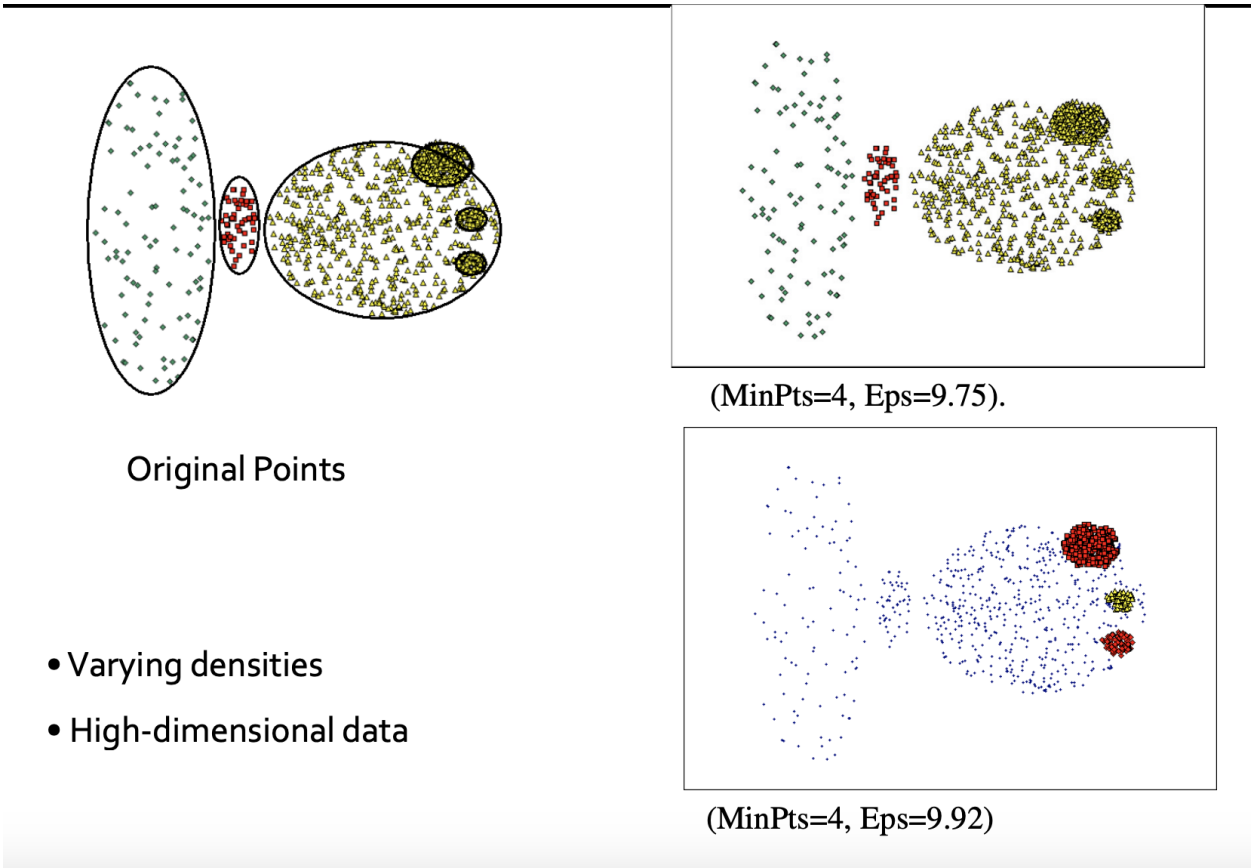

Figure 4: Example 2 Geometric Intuition

The second example is taken from [7]. It is composed of three clusters of varying densities and subclusters within a given cluster of different densities. The left shape is a vertical, long ellipse. The middle shape is a vertical, smaller elliptical shape. The right shape is a horizontal, medium-sized ellipse. On the left, we see green points very sparsely scattered. The red points in the middle are more dense. The right points are decently dense overall, but they have these three regions on the right that are noticeably more dense than the whole region overall.

This example highlights many of DBSCAN's weaknesses. First, **`MinPts` and `Eps` are very sensitive to even the slightest changes**. Even just increasing ϵ slightly from 9.75 to 9.92 caused DBSCAN to output a completely different clustering. It is likely that ϵ was increased just enough so that the points in the middle cluster were able to be **density-connected** to the cluster on the left and part of the cluster on the right.

Second, the algorithm **does not cluster well when true clusters have large difference in densities**. As we can see from the example, having different densities within the cluster can cause a completely different model output. The right cluster has the three sub-clusters that are more dense, which can output a clustering that is not good if ϵ is not set well. Third, one can understand geometrically from this example why one must **choose the correct distance measure between points** because the **curse of dimensionality makes the notion of neighborhoods break down in higher dimensions**. Although we cannot visualize higher dimensions $d > 3$, if everything in higher dimensions is all very far apart, then DBSCAN in general will not be able to pick up on dense cluster shapes as in 4.1 and 4.2 discussed so far.

4.3 Example 3

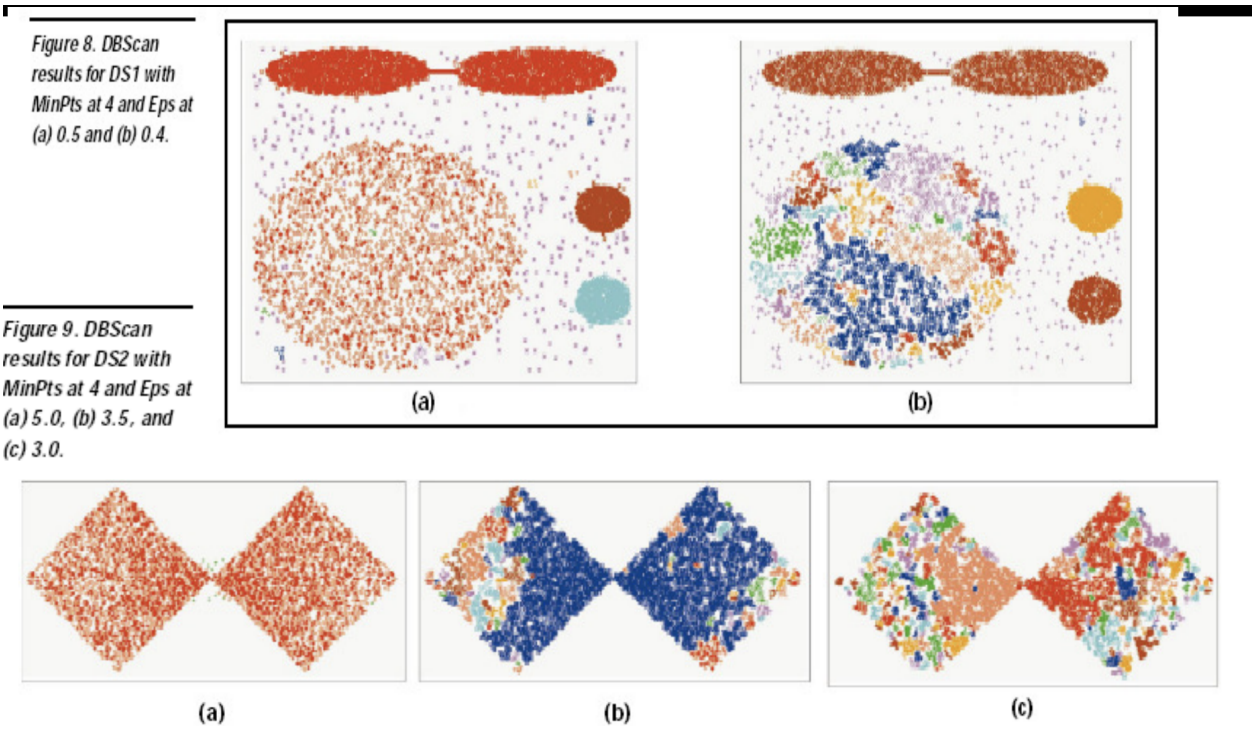

Figure 5: Example 3 Geometric Intuition

The third example is taken from [7]. In the first row, we see on the top two elliptical shapes about the same density. Then, there is a large elliptical shape to the left underneath them, and then two smaller elliptical shapes to the right. The large elliptical shape to the left underneath them is less dense than the other shapes, while the other elliptical shapes are very dense. On the second row, we see two sideways squares with the areas where they are touching more dense than the areas that are further apart from each other.

This example highlights many of DBSCAN's strengths and weaknesses. First, an advantage of DBSCAN is that it **only requires two hyperparameters to run, `MinPts` and `Eps`**. Both hyperparameters define in concert was defines a *dense* region very intuitively. Needing to only tune two parameters is a very manageable task. A disadvantage of DBSCAN, however, is that **`MinPts` and `Eps` are very sensitive to even the slightest changes**, as we had discussed previously. Even just slightly decreasing ϵ from the first row caused the globular cluster on the bottom left to completely break apart. On the bottom row, decreasing ϵ also caused the points in these two squares to completely break apart.

4.4 Example 4

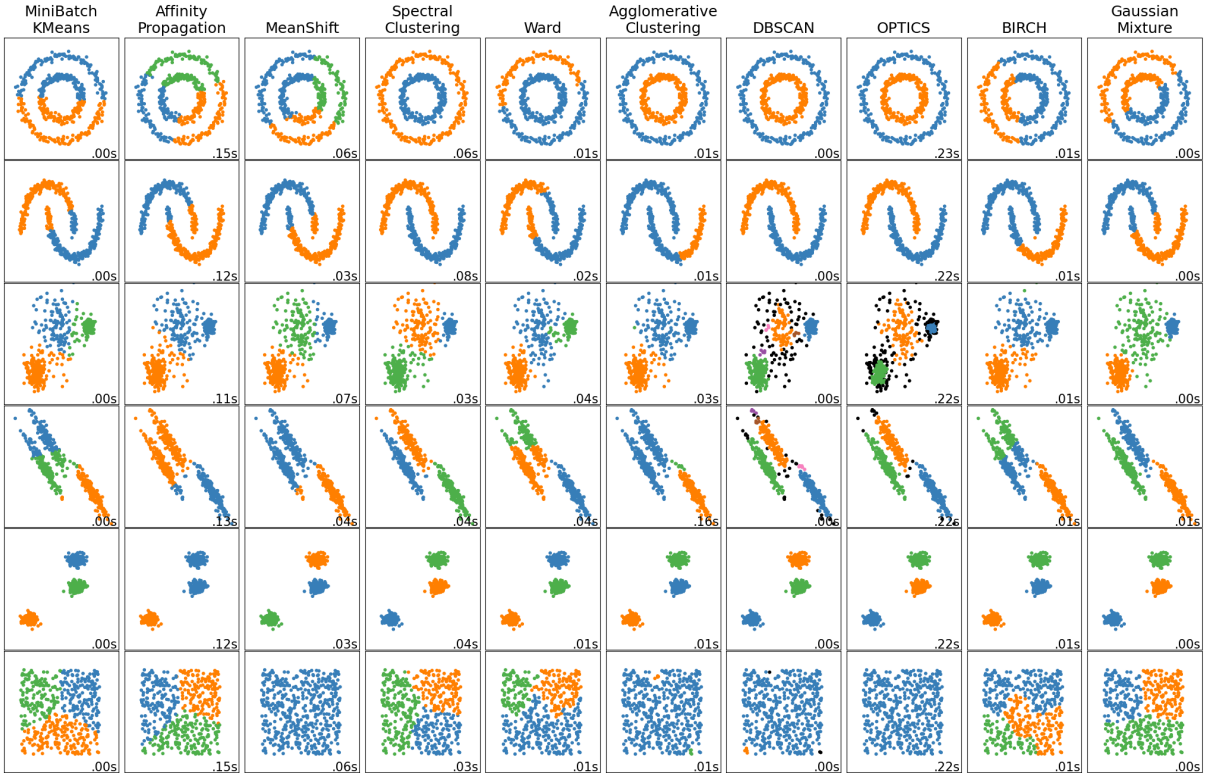

Figure 6: Example 4 Geometric Intuition

The fourth and final example is taken from [8], `Scikit-learn`'s official documentation page for clustering on toy datasets. We compare how DBSCAN performs against the popular clustering algorithms like K-Means, Agglomerative Clustering, and Gaussian Mixture Models (GMMs). Some of these examples are similar to some of the strengths and weaknesses discussed in 4.1, 4.2, 4.3. We will talk about the datasets that accentuate the major characteristics of DBSCAN.

On the first and second row, we observe that GMMs and K-Means have a hard time clustering these concentric circles and spiral shapes. They both break up the circles and spirals in some way. DBSCAN is however able to cluster both of these very well, not breaking them up. This accentuates its strength and **ability to cluster arbitrary shapes and sizes**. Furthermore, on the third row, it is able to identify these noise points in black surrounding the middle cluster. For this particular cluster, agglomerative clustering has trouble with clustering the middle and right cluster, likely due to the noise points. DBSCAN is **robust to outliers and noise due to nature of algorithm**, and this trait makes it particularly effective for slightly noisier datasets where methods like K-Means and Agglomerative Clustering will fail like on row 3 and row 4. It does not have the hard assumption that each point *has* to belong to exactly one cluster, making the choice not intuitive when there is a lot of noise. It relies on the notion of neighborhoods to cluster points, and if `MinPts` and `Eps` are chosen very well to define a *dense* region, DBSCAN can perform excellently.

5 Bibliography

[1] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." kdd. Vol. 96. No. 34. 1996.

[2] Yang, Yang, et al. "An efficient DBSCAN optimized by arithmetic optimization algorithm with opposition-based learning." The Journal of Supercomputing (2022): 1-39.

[3] Gong, Wenrong, et al. "A social-aware K means clustering algorithm for D2D multicast communication under SDN architecture." AEU-International Journal of Electronics and Communications 132 (2021): 153610.

[4] Fouedjio, Francky. "Clustering of multivariate geostatistical data." Wiley Interdisciplinary Reviews: Computational Statistics 12.5 (2020): e1510.

[5] Misuraca, Michelangelo, Maria Spano, and Simona Balbi. "BMS: An improved Dunn index for Document Clustering validation." Communications in statistics-theory and methods 48.20 (2019): 5036-5049.

[6] Güting, R.H. An introduction to spatial database systems. VLDB Journal 3, 357–399 (1994). https://doi.org/10.1007/BF01231602

[7] Al-Fuqaha, Ala. Clustering Analysis - Computer Science | Western Michigan University.

[8] "Comparing Different Clustering Algorithms on Toy Datasets." Scikit.

[9] Sander, Jörg, et al. "Density-based clustering in spatial databases: The algorithm gdbscan and its applications." Data mining and knowledge discovery 2.2 (1998): 169-194.

[10] Schubert, Erich, et al. "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN." ACM Transactions on Database Systems (TODS) 42.3 (2017): 1-21.

[11] Ade Gunawan. 2013. A faster algorithm for DBSCAN. Master's thesis. Technical University of Eindhoven. http://repository.tue.nl/760643

[12] Liu, Yanchi, et al. "Understanding of internal clustering validation measures." 2010 IEEE international conference on data mining. IEEE, 2010.