An investigation of dimensionless DBSCAN hyper-parameters

Louis Nass*

Dinuka Malith, Yi Tang

Department of Mathematics

Tulane University

New Orleans, LA 70118

DUE: May 11, 2020

Abstract

DBSCAN is a sorting algorithm that requires hyper-parameters ϵ and *minPts* in order to cluster data. Here we will establish dimensionless constants that are functions of these hyper-parameters. We will do this by sampling pair-wise distances between data points and establishing constant values for ϵ and *minPts*. Then we will set our dimensionless constants to be the ratios of varied values of ϵ and *minPts* and our initialized constants. Our goal is to discover a trend between the values of these dimensionless constants and the 'goodness' of the DBSCAN algorithm.

Key Words: clustering, classification, supervised, hard-clustering, DBSCAN, dimensional analysis, hyper-parameters

*E-mail address: lnass@tulane.edu

Contents

1 Introduction **3**

2 Dimensional analysis **3**

2.1 Scaling data 4

2.2 Dimensionless parameters 4

2.2.1 Dependent and independent variables 4

2.2.2 Dimensionalization 5

2.2.3 Non-dimensionalization 5

3 Classification **6**

3.1 Hard versus soft classification 6

3.2 Supervised versus unsupervised learning 7

4 DBSCAN **7**

4.1 DBSCAN outline 9

4.2 The DBSCAN algorithm 9

4.3 Advantages and disadvantages of DBSCAN 9

5 Experiment **11**

5.1 Dimensionalization of hyper-parameters 11

5.2 Scalar selection 12

5.3 Data 12

6 Results **13**

6.1 K-th nearest points 13

6.2 Fixed $\mathbf{S}\epsilon$, varied $\mathbf{S}minPts$ 14

6.3 Fixed $\mathbf{S}minPts$, varied $\mathbf{S}\epsilon$ 15

6.4 Varying $\mathbf{S}minPts$ and $\mathbf{S}\epsilon$ 15

7 Conclusion **16**

7.1 Continued Study 16

A Buckingham's π Theorem **17**

B Fundamental Dimensions **17**

1 Introduction

Clustering is an effective tool to establish trends in data. We do this by grouping data points that behave similarly. There are a number of techniques that identify clusters in unique ways that depend on the behavior of the data set. The DBSCAN algorithm is a robust technique that clusters data similar to the way we would naturally cluster observational data, by clustering points that are near each other. Generally, data points that lie near each other in space have similarities. When identifying trends, our best guess will be to observe the trends of points that are close together. However, DBSCAN requires hyper-parameter inputs before it is able to sort any data.

Hyper-parameters are values that are pre-determined before running an algorithm. These values are crucial in deciding how the algorithm will sort data points together. Herein lies the problem, how do we choose such hyper-parameters? We like to use clusters to establish trends in data, but we need to know trends in order to properly cluster the data. This circular argument takes away from the effectiveness of DBSCAN. Hence we want to gain an insight of the data set before we begin our clustering

In differential equations, we use dimensional analysis of variables and parameters to gain an insight of our system before we solve anything. It is a technique that allows us to establish scaling relationships for our variables and parameters, and in many cases makes problems easier to solve. Although DBSCAN is not a differential equation that we are trying to reduce, we do want to gain insight of our parameters before we do any work.

Hence, we will propose a method of establishing non-dimensionalized constants from the hyper-parameters of DBSCAN. Our goal is to understand the relationship between these constants and how well the algorithm clusters our data.

2 Dimensional analysis

Before we begin any work on our data, we first want to ensure that it is properly standardized. Pre-processing and scaling the data allows an algorithm and method to be unique for any data set. It is a powerful tool that will reduce the workload of the user. Once the standardization is complete, we can run our algorithm, then reverse the standardization to review our findings.

Additionally, we will define and discuss the advantages of creating dimensionless parameters. In mathematical modeling, we non-dimensionalize parameters to yield insight on scaling relations between predictors or parameters without knowledge of a governing equation [8]. This is where our interests lie.

2.1 Scaling data

We must first standardize our data to ensure our algorithm will be useful for any data set. There are a number of standardization techniques that are useful for different situations. We will discuss normalization, the most common type of scaling used in machine learning algorithms.

Normalization, as its name suggests, means that we will treat the data as normally distributed. We then transform the data to have a 0 mean and a unit standard deviation. Given data samples X_j for $j = 1, \ldots, n$ we have

$$\bar{X} = \frac{1}{n}\sum_{j=1}^{n} X \tag{1}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(X_j - \bar{X})^2} \tag{2}$$

. Then we define our transformation to be

$$X'_j = \frac{X_j - \bar{X}}{s} \tag{3}$$

for $j = 1 \ldots, n$. Hence, we now have data that has mean 0 and unit standard deviation [2]. From this point forward, we will assume that all data has been standardized. Now that our data has been standardized, we next consider dimensionless parameters.

2.2 Dimensionless parameters

As mentioned earlier, non-denationalization of parameters and predictors will allow us to observe relations between desired variables and parameters. In addition, performing dimensional analysis minimizes the amount of necessary parameters and predictors before we do any work on our data.

The backbone of dimensional analysis comes from the Buckingham π Theorem, featured in appendix (A). We perform dimensional analysis in three main steps, as explained by Groesen [4].

2.2.1 Dependent and independent variables

First, we decide the dependent and independent variables of our system. We need to evaluate all of the variables in our system before we can non-dimensionalize our predictors or parameters. In this example, Q_0 is the dependent variable and $Q_1, \ldots, Q_n$ are the independent variables where, for some function f,

$$Q_0 = f(Q_1, \ldots, Q_n) \tag{4}$$

. We must make sure that none of the independent variables $Q_1, \ldots, Q_n$ are affected by arbitrary adjustments of each other.

2.2.2 Dimensionalization

Second, we dimensionalize the independent variables. Here we decide which fundamental dimensions, listed in appendix (B), are associated with each of our independent variables. Each of our independent quantities has a fundamental dimension and is of the form

$$[Q_i] = L^{l_i} M^{m_i} N^{n_i} t^{\tau_i} \tag{5}$$

where l_i, m_i, n_i, and τ_i are dimensionless numbers that follow from the definition of Q_i.

We now select a complete, dimensionally independent subset $Q_1, \ldots, Q_k$ from set our independent variables. The value k is, at maximum, equal to the number of fundamental dimensions included in our system. Considering L, M, N, and t as our fundamental dimensions, then $k \leq 4$. We can express the dimensions of the remaining independent variables as a product of the dimensions from our subset. Namely, we now have, for $i > 4$

$$[Q_i] = [Q_1^{N_{i,1}} Q_2^{N_{i,2}} \ldots Q_k^{N_{i,k}}] \tag{6}$$

where $N_{i,j}$ are dimensionless numbers and

$$l_i = \sum_{j=1}^{k} N_{i,j} l_j \tag{7}$$

$$m_i = \sum_{j=1}^{k} N_{i,j} m_j \tag{8}$$

$$n_i = \sum_{j=1}^{k} N_{i,1} n_j \tag{9}$$

$$\tau_i = \sum_{j=1}^{k} N_{i,j} \tau_j \tag{10}$$

. Generally, the subset $Q_1, \ldots, Q_k$ and the corresponding $N_{i,j}$ values are found by inspection.

2.2.3 Non-dimensionalization

Finally, we non-dimensionalize. Here we define dimensionless form of our independent variables to be Π_i to be, for $i = 1, \ldots, n-k$,

$$\Pi_i = \frac{Q_{k+i}}{Q_1^{N_{(k+i),1}} Q_2^{N_{(k+i),2}} \ldots Q_k^{N_{(k+i),k}}} \tag{11}$$

and a dimensionless form of our dependent variable to be

$$\Pi_0 = \frac{Q_0}{Q_1^{N_{0,1}} Q_2^{N_{0,2}} \dots Q_k^{N_{0,k}}} \tag{12}$$

$$= f(\Pi_1, \dots, \Pi_{n-k}) \tag{13}$$

. We observe that $Q_1, \dots Q_k$ are still dimensionalized. However, because of the construction, and the application of Buckingham's π Theorem, these quantities are absent from the dependent variable Π_0. Hence we have non-dimensionalized the system. Our next stage will be to classify our data.

3 Classification

The goal of classification is to assign data vectors into distinct classes, or clusters, C_i for $i = 1, \dots, k$. Data vectors have the form $X^{(j)} \in \mathbb{R}^p$ for $j = 1, \dots, n$, where p is the number of predictors and n is the number of samples. We classify data in order to make it easy to read, access, and to identify trends within classes. There are different ways to classify data, which will depend on the purpose of the classification.

Classification requires functions called classifiers $\mathcal{C}$. We input data vectors into the classifiers to produce outputs $\hat{Y}^{(j)} \in \mathbb{R}^k$, where

$$\mathcal{C}(X^{(j)}) = \hat{Y}^{(j)} \tag{14}$$

. The output, $\hat{Y}$, is a k dimensional vector where k is the number of classes. We define the values of our output where

$$\hat{Y}_i^{(j)} > 0, X^{(j)} \in C_i \tag{15}$$

$$= 0, X^{(j)} \notin C_i \tag{16}$$

. The definition of $\hat{Y}^{(j)}$ depends on whether the classifier is hard or soft [5].

3.1 Hard versus soft classification

Classification can be considered hard or soft. Hard classification occurs when data is either in or not in a class. A sample will only belong to one class. If a sample does not belong to a class, then it is considered an outlier. For hard classifiers, we adjust equation (15) where $\hat{Y}_i^{(j)} = 1$ for $X^{(j)} \in C_i$. Advantages of hard classifiers are that each data point is assigned to exactly one cluster, and work well with distinct clusters. However, hard classifiers can incorrectly classify outliers and are not as effective with noisy data. An example

of a hard classifier is the K-means algorithm.

Soft classification allows for flexibility of data. A soft classifier will produce a confidence that a sample belongs to a class. Soft classifiers are useful for noisy data where the sample could be in more than one class. When $\hat{Y}_i^{(j)} < \hat{Y}_m^{(j)}$, from equation (15), this indicates that we have confidence that $X^{(j)}$ more likely belongs to C_m than C_i. Soft classifiers are effective with noisy data, but can be over-kill for data that has distinct clustered shapes. Soft classifiers are useful in neural networks and support vector machines [5].

Another important distinction in classification is supervised learning versus unsupervised learning.

3.2 Supervised versus unsupervised learning

Unsupervised learning is a method of classification where the data does not have a given output. The purpose of unsupervised learning is to sort raw data. These methods exploit trends that exist in the structure of the data based on proximity of samples. Unsupervised learning algorithms are normally distance based. They perform by clustering data samples that are physically close in space. Generally, we use unsupervised algorithms to initialize the labels for our data. Then we use the data and labels to teach and and optimize our supervised learning classifiers.

Supervised learning occurs when we optimize classifiers by classifying data that has associated labels. We define the label of a data vector $X^{(j)}$ as $Y^{(j)}$. The label is defined the same as in equations (15) and (16). The goal of supervised learning is to minimize the residual R, with some L-p norm, $||\cdot||_p$, where

$$R = ||\hat{Y}^{(j)} - Y^{(j)}||_p \tag{17}$$

. Supervised learning is normally used in tandem with soft classifiers.

In particular, we will focus on the unsupervised, hard clustering algorithm known as DBSCAN.

4 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) algorithm creates clusters based on dense regions existing in data space, which are separated by regions of lower object density [1]. Essentially, DBSCAN clusters points that are close together, then these clusters are separated by regions without data points. The algorithm will group together points that are close in space, then recursively add points to the cluster based on location. This allows DBSCAN to be effective in identifying arbitrarily shaped clusters

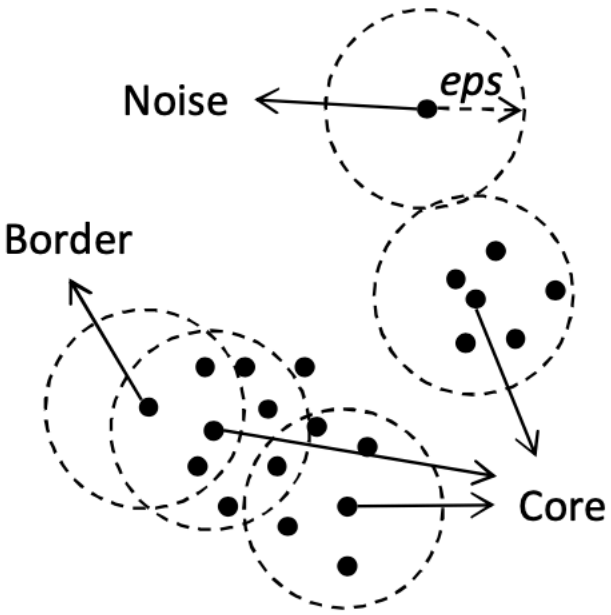

Figure 1: Here is a 2-dimensional visualization of core, boarder, and noise points that were identified by DBSCAN. In this case, the $\epsilon-$neighborhood is defined with the dashed circle and $minPts = 4$. We observe that DBSCAN clusters points that are close together in space and labels points that are 'far away' to be noise. [7].

4.1 DBSCAN outline

The algorithm DBSCAN identifies clusters of data points by inputting two hyper-parameters, a real number epsilon, $\epsilon > 0$, and a natural number, minimum points, $minPts \in \mathbb{Z}$. Here, the hyper-parameters work in tandem to classify data as into one of three categories, core points, boarder points, or noise.

If an $\epsilon-$neighborhood centered at point $X^{(j)}$ contains more than *minPts* other data points, then $X^{(j)}$ is considered a core point for some cluster C_i. A core point is an interior point for a cluster.

If an $\epsilon-$neighborhood centered at $X^{(j)}$ contains less than *minPts* other data points, but at least one core point, then $X^{(j)}$ is considered a boarder point. Clusters, C_i, are defined by their core and boarder points.

If a data point is neither a core point nor a boarder point, then it is considered a noise point. Essentially, noise points are their own clusters and are outliers. Observe figure (1) for a visualization how DBSCAN categorizes data.

First, DBSCAN randomly selects a point from the data set. If the point has *minPts* or more points in its $\epsilon-$neighborhood, the point is assigned as a core point to the cluster. The algorithm then recursively categorizes the other points inside of the $\epsilon-$neighborhood as core points and boarder points and transitively adds them to the cluster. This continues until the algorithm can no longer identify uncategorized points. It will then randomly select another data point that has not been categorized to start a new cluster. This continues until all the points have been categorized into separate clusters or as noisev [7]. A detailed algorithm of DBSCAN is shown below.

4.2 The DBSCAN algorithm

In summary, DBSCAN has three steps:

1. For each point, find the points within its neighborhood ϵ and identify the core points with more than minPts neighbors

2. Find all connected components of core points and assign them to a cluster

3. If a non-core point is within ϵ of a core point, assign it to that core-point's cluster, otherwise label it as noise

We present the pseudo-code for these steps in algorithm (1) [3].

4.3 Advantages and disadvantages of DBSCAN

An advantage of DBSCAN is its ability to cluster arbitrary shaped data sets. Because the algorithm is not centroid based, we do not have to worry about the misclassification of non-spherical data sets. It simply

Algorithm 1 DBSCAN pseudo-code

```
input: X, distFunc, ε, minPts
output: clusters, noise
C = 0
for point p in X do
    label(p) = visited
    neighborPts = RangeQuery(p, ε)
    if neighborPts < minPts then
        label(p) = noise
    else
        C = next cluster
        expandCluster(p, neighborPts, C, ε, minPts)
    end if
end for

function expandCluster(p, neighborPts, C, ε, minPts)
add p to cluster C
for point p′ in neighborPts do
    if p′ is not visited then
        mark p′ as visited
        neighborPts′ = rangeQuery(p′, ε)
        if sizeof(neighborPts′) ≥ minPts then
            neighborPts = join(NeighborPts, neighborPts′)
        end if
    end if
    if p′ is not in a cluster then
        add p′ to cluster C
    end if
end for

function rangeQuery(p, ε)
return all points within p's ε-neighborhood
```

clusters data points with points that are close in space. Additionally, we do not have to pre-decide the number of clusters for our data set before running the algorithm. DBSCAN naturally establishes the proper number of clusters based on densities of data points. Lastly, DBSCAN identifies potential outliers that may cause issues for other sorting algorithms. Potential outliers are clearly identified, allowing us to interpret them properly [6].

A major disadvantage of DBSCAN is its difficulty clustering data of varying densities. Not all data sets have clusters a uniform density. Because ϵ and *minPts* are fixed beforehand, this can cause issues trying to effectively cluster data sets that vary in density. That is why it is important to choose ϵ and *minPts* carefully before starting the algorithm [6]. However, there is not a universal technique used to choose proper ϵ and *minPts* values. Hence, we will consider non-dimensionalizing ϵ and *minPts* prior to running DBSCAN in order to give insight in choosing these values.

5 Experiment

Our goal is to create dimensionless constants $S\epsilon$ and $SminPts$ in order to observe their relationship with how well DBSCAN clusters data. We will use a data set that has a predetermined clustering. We then will vary ϵ and *minPts* to get a spectrum of $S\epsilon$ and $SminPts$ values. Finally, we will plot the error against $S\epsilon$ and $SminPts$ to see if trends occur.

5.1 Dimensionalization of hyper-parameters

We first must consider the definitions of our parameters. We do this to establish units.

The parameter ϵ represents a distance. We will use the Euclidean distance metric to measure length between points. Hence the dimensionalization of ϵ is

$$[\epsilon] = L \tag{18}$$

where L represents length from appendix (B).

The parameter *minPts* is defined as the number of points in an $\epsilon-$neighborhood. In set notation, this is, for some center point $X^{(0)}$, $minPts = card(\{X^{(j)} : ||X^{(j)} - X^{(0)}||_p < \epsilon\})$, where $|| \cdot ||_p$ represents the $p-$dimensional volume, corresponding to the number of predictors for our data. Hence the dimensionalization of *minPts* is

$$[minPts] = \frac{N}{L^p} \tag{19}$$

Figure 2: Generated data that looks like a smiley face. This data depends on 2 predictors X_1 and X_2.

. Hence, we have 2 fundamental dimensions. Since the rest of our predictors will be dimensionless, we non-dimensionalize ϵ and $minPts$ by multiplying scalars with dimensionalization $\frac{1}{L}$ and $\frac{L^p}{N}$ respectively.

5.2 Scalar selection

We now need to choose our scalars ϵ_0 and $minPts_0$. We will do this by calculating the pairwise distances between our data points. This will help us establish the mean and standard deviation of the distance from the $k-$th closest data point. We will let $k = 1, \ldots, n$. Then we will initialize $minP_0$ to be the value k with the smallest standard deviation, and ϵ_0 to be the mean distance from $minP_0-$th point.

Next, we define

$$S\epsilon = \frac{\epsilon}{\epsilon_0} \tag{20}$$

and

$$SminPts = \frac{minPts}{\epsilon_0} \tag{21}$$

. These are our dimensionless parameters. We will now vary $SminPts$ and $S\epsilon$ to see how well they cluster our data.

5.3 Data

For this study, we generate a simple smiley face. It can be seen in figure (2). We observe 4 distinct clusters. We have 2 eyes, 1 mouth, and the face shape. For the data, in this case, we generate the points using the

same deviation, $\sigma = 0.3$.

6 Results

We will now evaluate our data using the techniques mentioned above. First, we will observe the histograms of the $k-$th nearest point.

6.1 K-th nearest points

Sampling the $k-th$ nearest point, we found that the value k with the smallest standard deviation σ_ϵ was the nearest point, $k = 1$. This value had an average distance of $\mu_\epsilon = 0.021468$

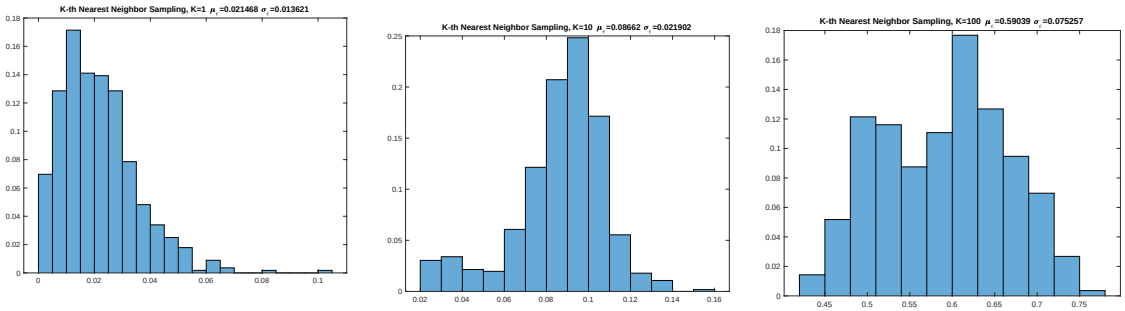

Figure 3: Histograms of the $k-$th nearest point for $k = 1, 10, 100$. We observe that the variance of the distances increases as k increases in figure(4)

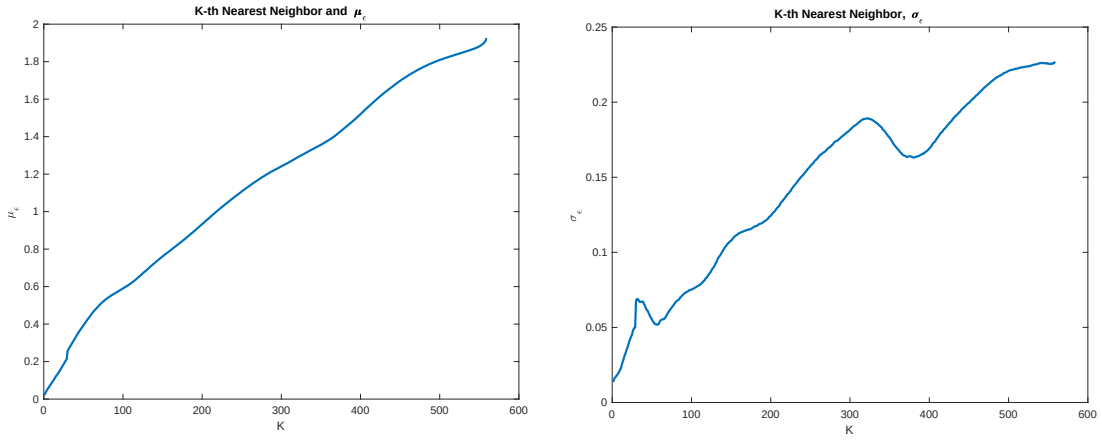

Figure 4: Here we plot $k = 1, \ldots, n-1$, against corresponding means μ_ϵ and deviations σ_ϵ. We observe that in both cases, means and deviations increase almost linearly. We only see some variation in the deviations.

We observe that the nearest point will establish our constants. Hence we will set $\epsilon_0 = 0.021468$ and $minPts_0 = 1$. We will now observe the results of the DBSCAN for $S\epsilon, SminPts = 1$.

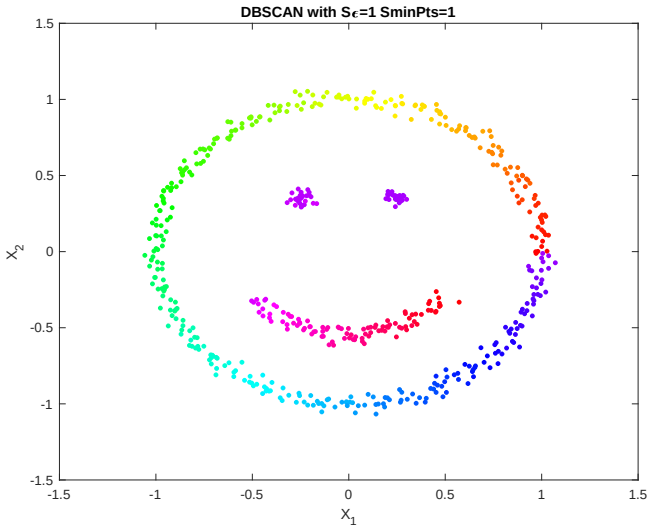

Figure 5: We observe the DBSCAN results for $S\epsilon, SminPts = 1$. We see that every point has its own cluster, making this a bad choice for ϵ and $minPts$.

We see that each point is its own cluster. Hence, this is a bad choice for ϵ and $minPts$. Next, we will fix $S\epsilon$ and vary $SminPts$.

6.2 Fixed Sϵ, varied S*minPts*

Here we fix $S\epsilon = 1$ and vary $SminPts$. The results are featured in figure (6).

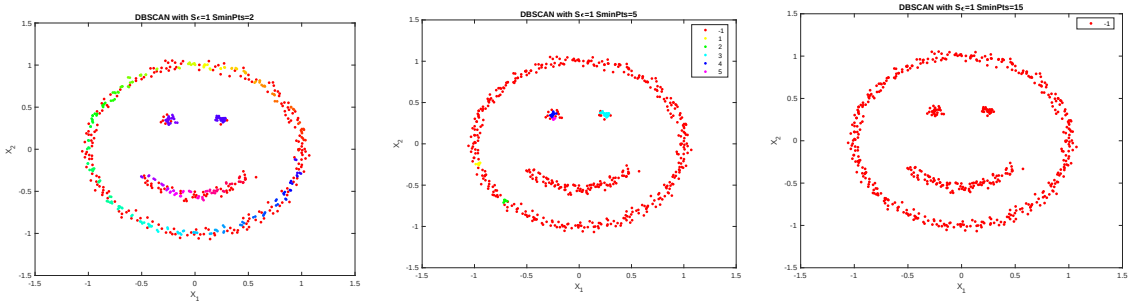

Figure 6: We have three samples where $S\epsilon = 1$ and $SminPts$ is increasing. We observe that the algorithm is labelling most of the data as outliers as $SminPts$ increases.

We observe that the DBSCAN algorithm labels outlier values -1. From our three figures, we see that fixing $S\epsilon$ and increasing the $minPts$ value will undoubtedly lead our data to be misclassified as outliers. This is not something we aim to do.

Next, we will fix $SminPts = 1$ and vary $S\epsilon$.

6.3 Fixed S*minPts*, varied Sϵ

Now we will fix $SminPts = 1$ and vary the $S\epsilon$ values. We observe the results in figure (7).

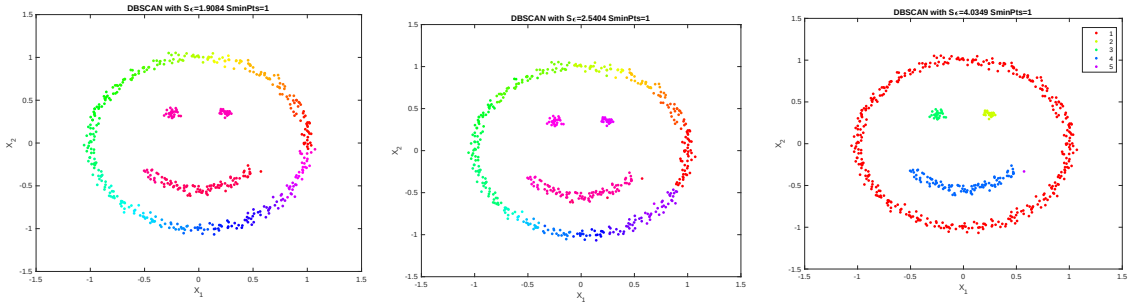

Figure 7: We observe the results where $SminPts = 1$ and $S\epsilon$ increases. We observe that as $S\epsilon$ gets larger, we achieve a reasonable classification of the samples in the smiley face.

We observe that adjusting $S\epsilon$ is more effective in creating a proper cluster. $S\epsilon$ has more flexibility in its selection, and, once we find an optimal $S\epsilon$, we properly cluster the data.

Finally, we will observe examples of variation in both $SminPts$ and $S\epsilon$. The results are shown in figure (8).

6.4 Varying S*minPts* and Sϵ

Here we will experiment and vary both $SminPts$ and $S\epsilon$.

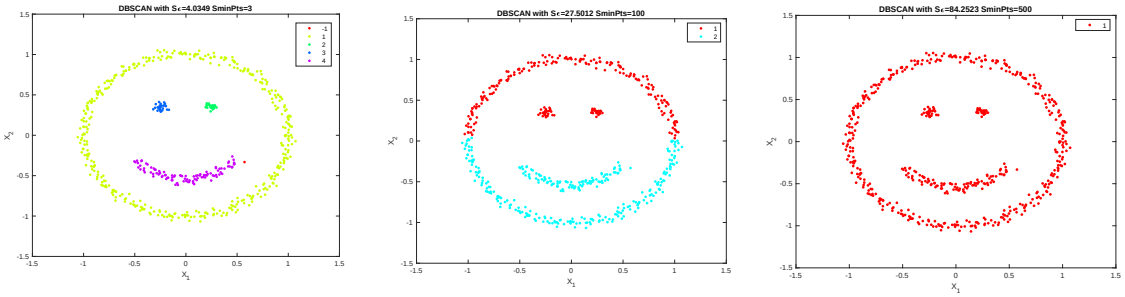

Figure 8: Here we vary both $SminPts$ and $S\epsilon$. We see that in the first figure, adjusting both values will result in decent clustering. However, if we vary these dimensionless values too much we will lose the generality of DBSCAN.

Here we see that subtle adjustments in both $SminPts$ and $S\epsilon$ can be good estimates for the algorithm to run well. However, we observe that major variations in our parameters cause the algorithm to lose its

ability to cluster arbitrary shapes.

7 Conclusion

From the results above, we recognize that varying $S\epsilon$ more liberally than $SminPts$ can increase the quality of the results of the DBSCAN. We observed that variations in $SminPts$ without variations in $S\epsilon$ caused misclassification problems, namely every point was classified as an outlier.

Then, we observed that fixing $SminPts$ and varying $S\epsilon$ did not affect the classification as negatively. In fact, we were able to properly classify the data set, separating the eyes, mouth and face shape, by increasing the value of $S\epsilon$. However, inevitably, if $S\epsilon$ is too large, we will observe all of the data points being clustered into one group.

Finally, we observed that small variations in both $SminPts$ and $S\epsilon$ can yield correct results. However, as stated previously, if the variations are too large, we will lose the ability for DBSCAN to recognize general clusters. Hence we must be careful adjusting these values.

7.1 Continued Study

This experiment was very limited. Moving forward, something to consider would be to evaluate other data sets. To enhance the results, it would be interesting to evaluate the algorithm and the dimensionless parameters on a data set with clusters of different densities. For the smiley face, we used a deviation of $\sigma = 0.3$, hence the density was uniform for all of the clusters. Considering different cluster densities would be interesting.

Lastly, an interesting, potential future study, would lie in the optimization of a cost function for the DBSCAN. We have establishing dimensionless parameters, and these parameters should be effective for any data set. However, we are unable to properly optimize these parameters without a cost function attached. This cost function can not simply work the same as cost function for machine learning algorithms. Because we do not know the number of clusters before we begin, this would take some creativity. I would propose a function that penalizes for having too many clusters and mislabeling noise, but rewards for properly clustering points together.

A Buckingham's π Theorem

Consider a system with predictors $X_1, \ldots, X_k$ and parameters $p_1, \ldots, p_l$ in which m fundamental dimensions are involved. Then, $k + l - m$ dimensionless quantities Q_i can be defined, which are products and quotients of the original predictors and parameters.

B Fundamental Dimensions

Quantity	Label
Length	L
Mass	M
Amount	N
Time	t

References

[1] Michael R Anderberg. *Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks*, volume 19. Academic press, 2014.

[2] Wikipedia contributors. Feature scaling — Wikipedia, the free encyclopedia, 2020. Online; accessed 2020-04-10.

[3] Onur Danaci, Harley Hanes, Elliot Hill, Dinuka Malith, Louis Nass, Noah Rahman, Yi Tang, Zheng Wang, Helen Weierbach, and James M. Hyman. Machine learning algorithms for classifying data. *Tulane Math Archive*, pages 1–49, 2020.

[4] E Van Groesen and Jaap Molenaar. *Chapter 1: Dimensional Analysis and Scaling*. SIAM, 2007.

[5] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[6] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady. Dbscan: Past, present and future. In *The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014)*, pages 232–238, 2014.

[7] Mohamamd Rezaei. Clustering validation. *Publications of the University of Eastern Finland*, pages 1–66, 06 2016.

[8] Ain A. Sonin. *The Physical Basis of Dimensional Analysis*. SIAM, 1992.