

Assignment #2

1 K-means++

1.1 Introduction

In data mining, clustering is the common technique for analyzing characteristics of the data by grouping data points with similar attributes. By identifying a representative point for each cluster, one can understand the characteristics of the data in each cluster without the need of examine all the data points. One of the most commonly used algorithms for clustering is the k-means algorithm. k-means divides clusters based on the distance from centroids and reassigns the centroids to the mean of the data points in each cluster. This process is repeated until the centroids no longer change and converge.

In the original k-means algorithm, initial centroids are chosen randomly, which can lead to significantly different results depending on their initial positions. To address this problem, k-means++ algorithm[3] is proposed. The k-means++ algorithm sets the first of the k initial centroids randomly and then uses a biased probability distribution for the remaining centroids, where the points that are farther from existing centroids are more likely to be chosen, thereby improving the overall performance and convergence speed.

Another issue with the original k-means algorithm is that the user must specify the value of k. Many studies has been conducted to find optimal parameter configuration. One method for estimating the appropriate number of clusters (k) is the silhouette method[6]. The silhouette method calculates the silhouette coefficient for each point, which is based on the average distance to other points within the same cluster ($a(i)$) and the average distance to points in the nearest cluster ($b(i)$). The silhouette coefficient, which ranges from -1 to 1, indicating better clustering with values closer to 1. By calculating the silhouette coefficient for varying numbers of clusters, the most suitable value of k can be estimated.

1.2 Problem Statement

In this report, we use a "centroid" to represent the cluster and cluster the data points based on the distance from the centroids. After k-means++ algorithm execute a clustering for the given data set, we use a "silhouette" method to evaluate the result.

Definition 1. *Cluster: group of the data objects.*

Above definition is the most common definition. Actually, the detailed definition of the cluster is different according to the clustering methods. In k-means and k-means++ algorithm, cluster is defined as the group of the data points that has the same nearest centroid.

Definition 2. *Centroid: The point representing mean of all data points in the cluster, minimizing the total squared euclidean distance to those points. Centroid of the cluster C can be calculated as follow:*

$$Centroid_C = \frac{1}{|C|} \times \sum_{p \in C} p$$

Definition 3. *Silhouette: Silhouette of point i in the dataset is defined as a measure that compares the average distance between point i and other points within the same cluster to the average distance between point i and points within the nearest cluster. This method evaluates the quality of the clustering by determining how well each point of the clusters is separated. Calculation of the silhouette will be describe in more detail in the section 1.4.1*

Definition 4. *Probability of point being chosen as the initial centorids: Let the centorid c_p is the nearest centorid from the point p and D is the whole data set. Then, the probabiltly of point p being chosen as the initial centorids is as follows:*

$$probability(p) = \frac{distance(p, c_p)^2}{\sum_{i \in D} distance(i, c_i)^2}$$

1.3 Algorithm

In this section, we present an algorithm designed to perform clustering on a given dataset. The steps of k-means++ algorithm are detailed in Algorithm1. And the method of setting initial centroids for k-means++ is detailed in Algorithm 2. The main idea of the k-means algorithm is to classify data points into clusters based on their distance and update the position of the centroids until they no longer change and converges. k-means++ algorithm enhances the performance compared to k-means algorithm by selecting the centroids with probability proportional to square of the distance between a point and its nearest centroid.

Algorithm 1 k-means++ algorithm

```

1: procedure K-MEANS++
2:   Initial-Centroid(k) ▷ Choose k of initial centroids
3:   while centroid is changed do
4:     Initialize all points in clusters
5:     for each point p do
6:        $C \leftarrow$  Cluster that has minimum distance from p to centroid ▷ Nearest cluster
7:       Put p into the cluster C
8:     for each cluster C do
9:        $cent \leftarrow$  Centroid of the C
10:       $cent \leftarrow (0, 0)$ 
11:      for each point  $p \in C$  do
12:         $cent \leftarrow cent + (p.x, p.y) \div (\text{number of the points})$  ▷ Update the centroid
13:   return

```

The K-MEANS++ algorithm starts from setting the initial centroids(Line 1). It will be described in detail on Algorithm2. This algorithm iterates until the location of the centroid is not changed(Line 3-12). In each iteration, we initialize all points in the cluster(Line 4) and determine their cluster again and put them into their cluster(Line 5-7). Then, calculate the centroids of the each cluster again(Line 8-12). We iterate this algorithm and select the result that the silhouette is a biggest.

Algorithm 2 Initial-Centroid algorithm

```

1: procedure INITIAL-CENTROID(k)
2:    $C \leftarrow$  Choose one data point randomly
3:    $num\_centroid = 1$ 
4:   while  $num\_centroid \neq k$  do
5:      $TotalDistance = 0$ 
6:     for each Point i do
7:        $C_I \leftarrow$  Nearest Centroid from i
8:        $TotalDistance = TotalDistance + distance(i, C_I)^2$ 
9:      $CurrentDistance = 0$ 
10:     $StopValue = (\text{random value from the interval } [0,1]) \times totalDistance$ 
11:    for each Point j do
12:       $C_J \leftarrow$  Nearest Centroid from i
13:       $CurrentDistance = CurrentDistance + distance(j, C_J)^2$ 
14:      if  $currentDistance > StopValue$  then
15:        add j to the centroids
16:        break
17:     $num\_centroid ++$ 
18:   return

```

Procedure INITIAL-CENTROID(k) is about initializing k centroids. It starts with selecting the first centroid by choosing one data point randomly(Line 2). Part of choosing remaining centroids is in a loop(Line 4-17), which iterates while we choose k centroids. First, Set $TotalDistance$ to 0 (Line 5). For each point i in dataset, find the nearest centroid C_I from i and calculate the distance between two points, and add the square of the distance to $TotalDistance$ (Line 6-8). Then, Set $CurrentDistance$ to 0 and $StopValue$ with random value between 0 and $TotalDistance$ (Line 9-10). For each point j in

dataset, calculate the distance again and add the square of the distance to *CurrentDistance*. if the *CurrentDistance* exceeds *StopValue*, then select *j* to the next centroid (Line 11-16). Lastly, add 1 to *num_centroid*. This procedure chooses the next centroid with each point has probability proportional to square of the distance between point and its nearest centroid, ensuring that point close to existing clusters is not chosen often.

1.4 Our approach

1.4.1 Optimal k value

In this report, we use silhouette method to find the optimal k value for given dataset. The silhouette value can be calculated by using $a(i)$ and $b(i)$ where $a(i)$ is an average distance between point *i* and the points in the same cluster and $b(i)$ is an average distance between point *i* and the points in the nearest cluster. The silhouette value ranges from -1 to 1. If the silhouette value closer to 1, it indicates better clustering. Because, a value close to 1 means that $b(i)$ is much larger than $a(i)$, which means that the distance to points in the same cluster is much smaller than the distance to points in the nearest neighboring cluster. Following formula is to calculate $a(i)$ and $b(i)$ (C_J is the nearest cluster which does not include point *i*):

1. $silhouette(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$
2. $a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} dist(i, j)$
3. $b(i) = \frac{1}{|C_J|} \sum_{j \in C_J} dist(i, j)$

By using above equations, We made a algorithm to calculate the silhouette.

Algorithm 3 Silhouette

```

1: procedure SILHOUETTE(point p)
2:    $C_I \leftarrow$  Cluster that includes point p
3:    $C_J \leftarrow$  Nearest cluster to the point p  $\triangleright C_I \neq C_J$ 
4:    $a(p), b(p) = 0$   $\triangleright$  initialize a(p), b(p)
5:   for each point i in cluster  $C_I$  do
6:      $a(p) = a(p) + Distance(i, p) \div (|C_I| - 1)$ 
7:   for each point j in cluster  $C_J$  do
8:      $b(p) = b(p) + Distance(j, p) \div (|C_J|)$ 
9:    $Silhouette = (b(p) - a(p)) \div \max(b(p), a(p))$ 
10:  return Silhouette

```

When estimating optimal value of k , we execute k-means++ clustering algorithm for $k = 2, 4, 8, 16, 32, 64$ and calculate silhouettes for each execution. To ensure better results, we perform multiple executions for each value of k and compute the silhouette value for each cluster configuration. We then select the value of k that has configuration of the highest silhouette value. And if we select $k = 2^n$, we also execute the algorithm for $k = 2^{n-1} + 1$ to $2^{n+1} - 1$ and find the optimal value of k which has the configuration of the highest silhouette value.

1.5 Advantages and Disadvantages

1.5.1 Advantages

- **Fast execution time:** The time complexity of the k-means algorithm is $O(n \times k \times t)$ where *n* is the number of data points, *k* is the number of clusters, *t* is the number of iterations. Selecting *k* initial centroids by applying a probability proportional to the distance has a time complexity of $O(n \times t \times (k - 1)) = O(n \times t \times k)$ too. This shows a k-means algorithm has fast execution time. Moreover, by choosing the initial centroids to be far each other, it make k-means algorithm can find solution in $O(\log k)$ time[3].
- **Does not choose all initial centroids randomly:** The k-means++ algorithm selects only the first initial centroid randomly, and the other centroids are chosen with a probability proportional to the distance. Consequently, the farther a point is from the existing centroids, the higher the

probability it will be chosen as the next centroid. This approach prevents points that are close to the existing centroids from being chosen often, leading to better initialization of centroids.

1.5.2 Disadvantages and Limitations

- **Clusters of different sizes and densities are not found:** The k-means++ algorithm determines the cluster of a point based on the distance. Therefore, if there are clusters that have a large size and low density compared to other clusters, the k-means++ algorithm may not detect these clusters.
- **Only circular clusters can be found:** The k-means algorithm can only find circular clusters as this algorithm determines a cluster by the distance between the centroid and the data point.
- **Outliers can greatly affect the results:** The centroid is defined as the center of the points included in the cluster. Therefore, if outliers are included, the location of the centroid may differ significantly from the optimal location.
- **Silhouette method has too much calculation:** The silhouette method calculate the silhouette coefficient for all points for each k. Since the calculation of the silhouette coefficient is $O(n)$, the time complexity of the silhouette method is $O(n^2)$.

1.6 Experiment

In this section, we provide a detailed description of our experimental approaches. Our experiment were conducted using Oracle OpenJDK 11 via visual code, and the results were visualized on Kaggle. The implementation code and dataset are available at https://github.com/thinkin9/DataMining_G6

1.6.1 Dataset

To evaluate our k-means++ algorithm, we used five 2-dimensional data sets. Three datasets are selected from clustering datasets available on Kaggle, and two datasets are from provided resources. Data1 has 6 big clusters and 25 small clusters, Data2 has 15 clusters, Data3 has about 30 clusters and we select this dataset to test our algorithm can do clustering well if the k value is large. Data4 designed like a two bananas, and Data5 designed as a set of the five lines. We select Data4 and Data5 if to test whether our algorithm can find non-circular clusters. Table 1 shows the number of data points in the data sets. And the Figure shows how our datasets are look like.

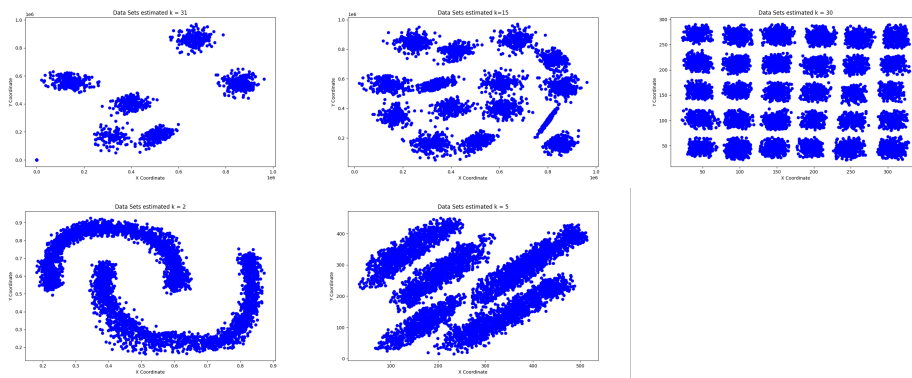


Figure 1: Datasets

1.6.2 Experiment Results

Figure3 shows the result of our algorithm with estimated k value and Table1 shows the accuracy of our results. To evaluate our accuracy, we used Jaccard similarity which can measure the similarity between to sets. Jaccard similarity defined as follow: $Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$

We compared our result set with the ground truth set for every data point in the dataset. The results showed good accuracy for Data2 and Data3, but poor accuracy for Data1, Data4, and Data5.

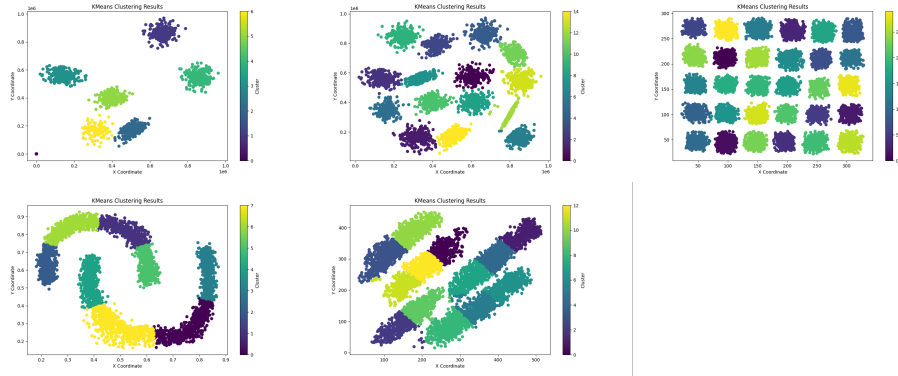


Figure 2: k-means++ Result Visualizaion

Datasets	Number of elements	accuracy
Data1(artd-31)	5000	0.3980
Data2(artset1)	5000	0.9978
Data3(boxes)	8901	0.9997
Data4(banana)	4811	0.303
Data5(lines2)	6195	0.4253

Table 1: Metadata of dataset and Accuracy of the result

Data2 and Data3's circular (or small linear) shaped clusters are easy for the k-means++ algorithm to identify accurately, leading to good clustering performance. However, Data1's results show the limitations of distance-based clustering methods like k-means++, also our silhouette evaluation method. Data1 contains a mix of large and small clusters, which causes the k-means++ algorithm to struggle separating small clusters. In our result, k-means algorithm joined all small clusters (which is shown in bottom left) into one cluster, leading to poor accuracy result. We set the k value to 10 for further investigation, and the k-means++ algorithm failed to separate small clusters, instead it separated big clusters into two or three.

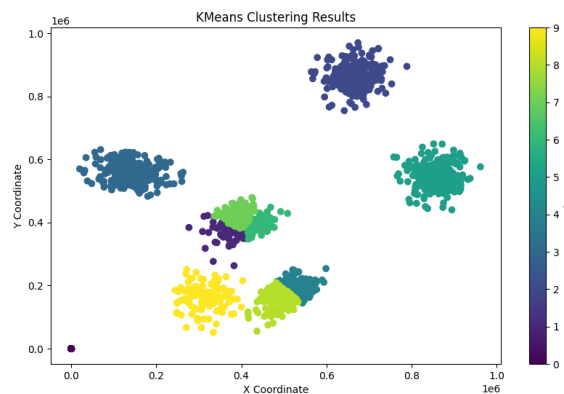


Figure 3: Data1, k=10

For Data4 and Data5, the k-means++ algorithm also performed poorly. Data4 contains curved-shaped clusters, and the k-means++ algorithm, which assumes circular cluster shapes, fails to capture this structure. Instead, k-means algorithm separated curved-shaped cluster into four small clusters.

Data5 consists of five linear shapes that are not very far apart. The k-means++ algorithm tends to divide these linear shapes into two or three clusters each, rather than identifying them as distinct linear clusters. This shows that k-means++ is less effective for identifying these non-circular clusters.

1.7 Conclusion and Future Directions

In conclusion, k-means++ algorithm performs clustering by repeating classify the data points and update the centroids. When we choose initial centroid, k-means++ algorithm select the farther point from other centroids by using a biased probability. It is fast and have simple method, but it has limitations. It is based on the distance, so it can only find circular cluster. And centroid is defined as a mean of the points, so outlier can greatly affect the clustering result.

Future Directions should focus on addressing these limitations by exploring alternative clustering methods that can also detect non-circular cluster shapes and more resistant to outliers, such as Density-Based clustering (DBSCAN), Gaussian Mixture Models (GMM). Research into adaptive algorithms that dynamically adjust to different cluster shapes and distributions can also be useful.

2 DBSCAN

2.1 Introduction

Clustering is about grouping data objects based on how similar they are to each other. There are two broad categories of clustering approaches: partitional and hierarchical methods. Each of these methods is well-developed and widely used in various applications. However, similarity calculations by using radius-based distance measurements, such as k-medoids and k-center, have limitations in identifying clusters of arbitrary shapes.

To address this problem, the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [4] was developed. DBSCAN has three significant strengths: the ability to identify clusters of arbitrary shapes, robustness to noise, and the elimination of the need to specify the number of clusters in advance. As shown in the survey paper [2], various density-based clustering techniques have been proposed in recent two decades, focusing on refining the definition of density, optimizing parameters, and enhancing execution modes in parallel and distributed computing.

In this project, we implement the DBSCAN algorithm and explore methods for determining the optimal values for its two key parameters: epsilon(eps) and minimum points(minPts). Finally, we evaluate clustering performance using two metrics: Adjusted Rand Index(ARI) and Adjusted Mutual Information(AMI).

2.2 Problem Statement

In this project, we consider a set of points, each defined by x and y coordinates. Let $X = \{x_1, \dots, x_n\}$ be a data set of n points, and let $d(x_p, x_q)$, $x_p, x_q \in X$ be a pairwise distances for a metric distance. Density-based clustering is defined as follows:

Definition 5. ε -neighborhood of a point: The ε -neighborhood of a point x_p , denoted by $N_\varepsilon(x_p)$, is defined by $N_\varepsilon(x_p) = \{x \in X \mid d(x, x_p) \leq \varepsilon\}$.

Definition 6. Directly density-reachable: A point x_p is directly density-reachable from a point x_q w.r.t. ε and $minPts$ if $x_p \in N_\varepsilon(x_q)$, and $|N_\varepsilon(x_q)| \geq minPts$ (core point condition).

Definition 7. Density-reachable: A point x_p is density-reachable from a point x_q with respect to w.r.t. ε and $minPts$ if there is a chain of points x_{p1}, \dots, x_{pn} , $x_{p1} = x_q$, $x_{pn} = x_p$ such that $x_{p(i+1)}$ is directly density-reachable from x_{pi} .

Definition 8. Density-connected: A point x_p is density-connected to a point x_q with respect to w.r.t. ε and $minPts$ if there is a point x_o such that both x_p and x_q are density-reachable from x_o w.r.t. ε and $minPts$.

Definition 9. Cluster: Let D be a database of points. A cluster C w.r.t. ε and $minPts$ is a non-empty subset of D satisfying the following conditions: $\forall x_p, x_q$: if $x_p \in C$ and x_q is density-reachable from x_p w.r.t. ε and $minPts$, then $x_q \in C$. (Maximality), and $\forall x_p, x_q \in C$: x_p is density-connected to x_q w.r.t. ε and $minPts$. (Connectivity)

Definition 10. Noise Point: A point x is called a noise point of the clusters w.r.t. ε and $minPts$ if the point does not belong to any cluster C_i for $i = 1, \dots, k$.

Based on these definitions, DBSCAN classifies each point as either a core point, a border point, or a noise point. Two core points belong to the same cluster if they are directly density-reachable. The cluster expands by merging both density-reachable points and density-connected points. The procedure will be detailed in the following section.

2.3 Algorithm Description

Algorithm 4 SetOfPoints.regionQuery Procedure

```

1: procedure SETOFPOINTS.REGIONQUERY(refPoint, eps) : List<Point>
2:   neighbor := empty list of Point
3:   for each point trgPoint in SetOfPoints do
4:     if distance(refPoint, trgPoint) ≤ eps then
5:       append trgPoint to neighbor
6:   return neighbor

```

Algorithm 4 is about querying all neighbor points within a specified radius eps from a reference point. It iterates over each point in the dataset, and calculates the distance between the reference and the target point. If this distance is less than or equal to eps, the target point is appended to the neighbor list.

Algorithm 5 DBSCAN Algorithm

```

1: procedure SCAN(SetOfPoints, Eps, MinPts)
2:   ClusterID := 1 // ClusterID is starting from 1
3:   for each unvisited point Point in SetOfPoints do
4:     if Point.classified = False then
5:       if ExpandCluster(SetOfPoints, Point, ClusterID, Eps, MinPts) then
6:         ClusterID := ClusterID + 1 // Update ClusterID

```

Algorithm 5 is about overview of the DBSCAN algorithm. For each unvisited and unclassified points in the dataset, it attempts to expand a new cluster using the ExpandCluster procedure. If a cluster is successfully expanded, ClusterID is incremented. This procedure continues until all points in the dataset are visited and classified. As a results, all points are classified as core points, border points, or noise points.

Algorithm 6 ExpandCluster Procedure

```

1: procedure EXPANDCLUSTER(SetOfPoints, Point, ClusterID, Eps, MinPts) : Boolean
2:   seeds := SetOfPoints.regionQuery(Point, Eps)
3:   if seeds.size < MinPts then
4:     SetOfPoints.setLabel(Point, -1) // -1 for noise
5:     return False
6:   else
7:     SetOfPoints.setLabel(seeds, ClusterID)
8:     seeds.delete(Point)
9:     while seeds ≠ Empty do
10:      nowPoint := seeds.first()
11:      result := SetOfPoints.regionQuery(nowPoint, Eps)
12:      if result.size ≥ MinPts then
13:        for each point NextPoint in result do
14:          if NextPoint.classified = False then // unclassified
15:            seeds.append(NextPoint)
16:            SetOfPoints.setLabel(NextPoint, clusterID)
17:          else if NextPoint.clusterID = -1 then // noise
18:            SetOfPoints.setLabel(NextPoint, clusterID)
19:      seeds.delete(nowPoint)
20:   return True

```

Algorithm 6 is about ExpandCluster Procedure. It takes five arguments: the dataset(SetOfPoint), the starting point(Point), the cluster identified(ClusterID), the radius(Eps), and the minimum points(MinPts). It finds neighbor points w.r.t. radius of eps. If the number of neighborhood is less than minPts, the point is labeled as noise, and the procedure returns False. Otherwise, the points is labeled as a core point. By iteratively finding neighborhoods and checking whether the number of its neighborhood exceeds minPts or not, unclassified points are added to core points and classified as ClusterID, and noise points are classified as ClusterID. This procedure continues until no more points can be added, and the procedure returns True.

2.4 Optimal MinPts

2.4.1 Our approach

In **Definition 1**, the core point condition, $|N_\varepsilon(x_p)| \geq \text{minPts}$ indicates that the parameter *minPts* is a critical constraint in determining directly density-reachability. Modifying the parameter *minPts* affects the classification of a point as either a core point or not. Thus, proper classification between core points and others is important, as core points can expand a cluster by identifying other reachable points, whereas other noise and border points cannot.

Given ε , determining the optimal *minPts* determines how strictly we classify a point as a core point. A fixed ε allows us to measure each point's density by calculating how many points are within a circle of radius ε . However, exploiting this information to find the optimal value of *minPts* does not fully capture the true meaning of "density" because it treats all neighboring points equally, regardless of their distance from the center. To address this, our approach focuses more on closer points than on distant points, securing more close and dense points to be classified as core points.

Our approach is detailed as follows:

- **Define $|N_{0.5\varepsilon}(x_p)|$, the size of 0.5- ε neighborhood.** To focus more on close points than on distant ones, we define the 0.5- ε neighborhood of a point x_p , denoted by $N_{0.5\varepsilon}(x_p)$, as $N_{0.5\varepsilon}(x_p) = \{x \in X \mid d(x, x_p) \leq \varepsilon/2\}$. Considering the concept of density, we attempt to reflect more close points over distant points in finding optimal *minPts*. The pseudo-code for 0.5- ε neighborhood is a simple modification of Algorithm 4 from ε to 0.5- ε .
- **Exclude a point having only a single 0.5- ε neighborhood.** This point has only one point which is itself within a circle of radius 0.5- ε .
- **Get median¹ from the list of $|N_{0.5\varepsilon}(x_p)|$ where $x_p \in X$ and $|N_{0.5\varepsilon}(x_p)| > 1$.** The median of a set of numbers is the value separating the higher half from the lower half of a data samples. Thus, the median is not skewed by a small proportion of extremely large or small values, providing a better representation of the center. Moreover, we can guarantee at least 50% of top $|N_{0.5\varepsilon}(x_p)|$ points are classified as core points. This percentage can be increased by using median approach which allows other points having the same $|N_{0.5\varepsilon}(x_p)|$ as median to be classified as core points. This approach does not limit Lower $|N_{0.5\varepsilon}(x_p)|$ points to be classified as either noise points or border points. These points can also be classified as core points depending on the number of neighborhood within $0.5\varepsilon \sim \varepsilon$ area.
- **Set median - 1 as *minPts*.** Decrementing the median by 1 is because counting neighborhood of a point includes the point itself.

2.4.2 Advantages

- Reflection of the true meaning of density. The density simply defined by using ε treats all neighboring points equally, regardless of their distance from the center. With more restricted definition of density w.r.t. ε , we can focus more on closer points than on distant points and secure more close and dense points to be classified as core points, thereby properly reflect the notion of density in determining optimal minPts.
- Median-based approach. This approach allows at least 50% of top $|N_{0.5\varepsilon}(x_p)|$ points to be classified as core points. This percentage can be increased as other points having the same $|N_{0.5\varepsilon}(x_p)|$ as median are also classified as core points. Additionally, this approach helps to avoid the increase in uncertainty occurring from treating floating-point of estimated minPts, such as the mean.

¹To avoid floating-point number when calculating median as the arithmetic mean in case of even number of datasets, we take the smaller number as the median.

2.4.3 Disadvantages

- Our constant 0.5 is intuitively defined. We presumed that the area of a circle of 0.5ε radius covers only a center, quarter of the area of a circle of ε radius, and this ratio is deemed sufficient for a restricted definition of density. In case of too high or low ε , the constant 0.5 become meaningless and should be adjusted to appropriately represent the meaning of density. Put another way, our approach is parameter-sensitive to ε .
- $O(n^2)$ time complexity. Each point requires a total n of distance calculations, where n is the number of points in the dataset. Consequently, the time complexity of our approach is $O(n^2)$ which is quite expensive for large datasets.
- Dataset dependency. In case of datasets with well-distributed or too sparse, our approach may fail to find a proper number of points within 0.5ε radius circle, resulting in poor estimation of minPts and clustering results.

2.5 Optimal Eps

2.5.1 Our approach

The parameter ϵ is crucial as it defines the radius for neighborhood points, directly impacting the density calculation. Proper value of ϵ influences the classification of points as either core or not, which is essential since core points can expand clusters by identifying other reachable points, whereas other noise and border points cannot. Distinct clusters may merge into a single cluster if the ϵ is too large, whereas too many clusters are generated if ϵ is too small. Thus, determining the optimal ϵ is critical for the overall performance of DBSCAN.

Given minPts , one of the most renowned methods for determining ϵ is the k-distance method. Let $\text{MinPts} = k$. As shown in Algorithm 7, for each data point, the distance to its k -th nearest neighbor is calculated, then k-distance values are sorted. Therefore, the knee point, which indicates a significant change in distance, is identified. Another approach involves using the average of the smallest k distances instead of the k -th distance.[7]

Algorithm 7 K-Distance Method

```

1: procedure K-DISTANCE(DataPoints  $P$ )
2:   for each point  $p$  in  $P$  do
3:      $k - \text{distance} = d(p, k^{\text{th}} \text{ neighbor})$  ▷ distance of two points
4:      $KD.\text{add}(k - \text{distance})$ 
5:   sort  $KD$ 
6:    $\epsilon = KD.\text{get}(\text{knee})$  ▷ set  $\epsilon$  to the value of knee point
7:   return  $\epsilon$ 

```

However, this method can encounter issues when clustering dense datasets. If many points are closely packed, points that are slightly farther apart may be classified into separate clusters, resulting in an excessive number of clusters. To address this, we consider both the k -nearest neighbors of the target point and the distances among these neighbors. This is demonstrated in Algorithm 8. This approach helps to distinguish noise points and prevents the creation of too many clusters.

Algorithm 8 Changed K-Distance Method

```

1: procedure K-DISTANCE(DataPoints  $P$ )
2:   for each point  $p$  in  $P$  do
3:      $p2 = p$ 's  $k^{\text{th}} \text{ neighbor}$ 
4:      $p3 = p2$ 's  $k^{\text{th}} \text{ neighbor}$ 
5:      $k - \text{distance} = \max(d(p, p2), d(p2, p3))$ 
6:      $KD.\text{add}(k - \text{distance})$ 
7:   ....
8:   return  $\epsilon$ 

```

The traditional method of finding the knee point tends to select an excessively high ϵ value, especially on datasets where a significant portion of the data is densely clustered and the remaining data forms

a low-density distribution. To address this, we adopt an alternative approach for determining the knee point.

First, we plot the k-distance graph with sorted points on the x-axis and their corresponding k-distance values on the y-axis. Then, we draw a straight line connecting the first and the last points of the graph. For each point on the graph, we calculate the perpendicular distance to this line. The point with the highest perpendicular distance is identified as the knee point.

Additionally, to further address the issue, we compare:

1. The difference between the median k-distance and the first k-distance value.
2. The difference between the median k-distance and the k-distance value at the knee point.

If the second difference is ten times larger than the first difference, we consider the original knee point to inadequately represent the k-distances of the majority of the data. In such cases, we update the knee point value to the median k-distance value.

2.5.2 Advantages

- We incorporate more neighboring points than the given number of MinPts, thus ensuring more robust results. The k-distance method suffers from limited information gathering as it scans points only once in the order they are stored. Our devised approach maintains the essence of the k-distance method while incorporating information about neighbors. Consequently, it prevents excessive cluster creation by avoiding excessively small ϵ , particularly in regions of high data density.
- Adopting a density-based approach is rational. By determining core distances for each point based on the given MinPts value, we effectively capture the density of the data. Hence, even users with limited prior understanding of the data can obtain meaningful results through this method.
- The method for determining the knee point is more intuitive compared to other methods and better reflects the data's characteristics. By incorporating a process that compares the differences with the median and updates the knee accordingly, this approach identifies a more optimal knee point than traditional methods. Consequently, it enables more meaningful clustering.

2.5.3 Disadvantages

- Applying it to clusters with diverse densities poses challenges. Since we don't store eps values for individual clusters and instead rely on sorting the k-distance values of each point, there's a lack of cluster-specific information.
- It has a high time complexity. Scanning all points twice results in an $O(n^2)$ time complexity. Compared to the traditional k-distance method, it involves investigating neighbors' neighbors once more, and during the process of determining the knee, sorting all values, making it inefficient for handling large datasets.

2.6 Experiment

2.6.1 Metrics

In density-based clustering, Silhouette Coefficient does not adequately capture the varying density and arbitrary shape of clusters due to the distance metrics measured by points in and between clusters. Thus, evaluation metrics are detailed as follows:

- **Adjusted Rand Index(ARI)** is a metric of the similarity between two data clusterings, adjusted for the chance grouping of elements. It ranges from -1 (indicating poor clustering) to 1 (indicating a perfect match), with a score of 0 representing random clustering.
- **Adjusted Mutual Information(AMI)** is an entropy-based measure of the agreement between two clusterings, adjusted for the chance clustering. It accounts for the shared information between cluster assignments, corrected for the expected mutual information of random clusterings. The AMI returns a value of 1 when the two partitions are same.

2.6.2 Dataset

Total five 2-dimensional datasets were selected from clustering datasets available on GitHub and were visualized on Google Colaboratory. The datasets are denoted as Data1, Data2, Data3, Data4, and Data5. Each dataset comprises multiple clusters, featuring various shapes to evaluate performance of density-based clustering.

Datasets Data1, Data2, and Data3 were designed with shapes such as weak spirals, donuts, and bananas, respectively, to assess clustering performance based on density. Datasets Data 4 and Data 5 are actually from the same dataset "artd-31", however, we extract low x, y points to evaluate high-density dataset. On the other hands, we use raw dataset "artd-31" as Data 5 to evaluate varying-density dataset with extremely high x, y points.

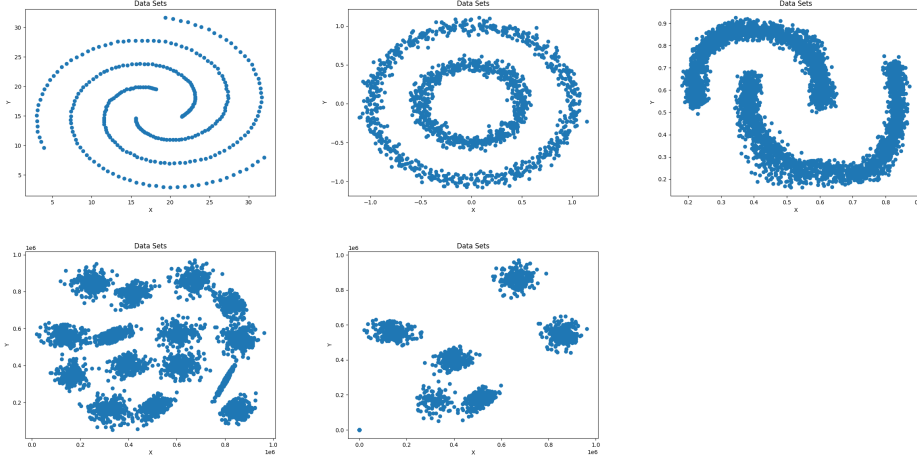


Figure 4: Datasets

2.6.3 Experiment Results

As we mentioned before, both ARI (Adjusted Rand Index) and AMI (Adjusted Mutual Information) were used to compare against ground truth values and evaluate how two parameters were correctly predicted. The experimental process can be divided into three main steps.

First, given ϵ value, we evaluated the method for the optimal $minPts$. For consistency, the optimal epsilon value for each dataset was used as the fixed eps value. We examined scores both estimated $minPts$ and estimated $minPts \pm 2$. As shown in Figure 5, our method generally yields higher evaluation scores compared to estimated $minPts \pm 2$.

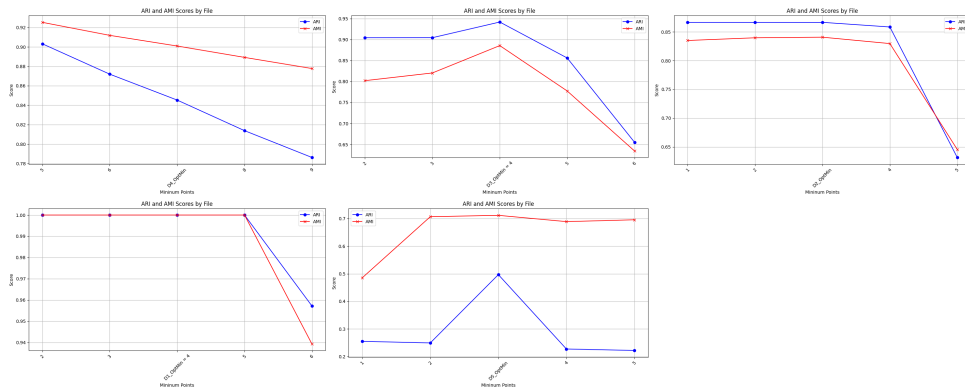


Figure 5: Minimum Points

Second, given $minPts$, we evaluated the method for the optimal ϵ . The optimal $minPts$ values obtained from the previous step were used for this evaluation. We compared the traditional k-distance method with our algorithm. Let $MinPts = k$. We compared the following values:

- EpsMean: Uses the average distance to the k nearest points from each point to calculate the k-distance.
- EpsTh: Uses the distance to the k-th nearest point from each point to calculate the k-distance.
- OptEps: Our proposed algorithm.

The result shows that our algorithm generally achieved the highest evaluation scores. Notably, for Data4, where many points exist at high density, our consideration of the median was proven to be worthy.

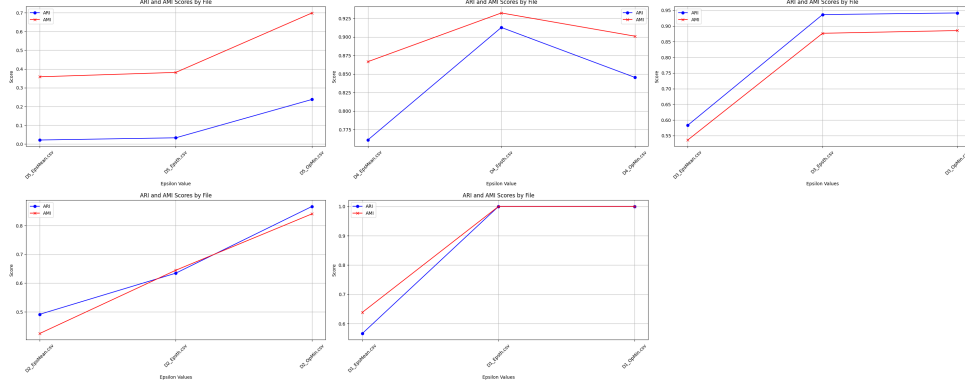


Figure 6: Epsilon

Finally, we attempted to find both optimal eps and minPts by using only default $minPts = 4$ for estimating the optimal eps. The parameters eps and MinPts were updated sequentially, yielding the following results.

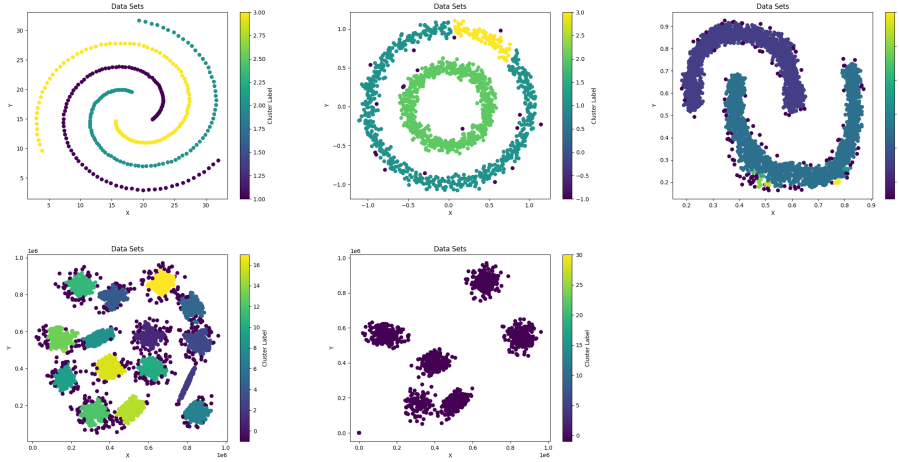


Figure 7: Result Visualizaion

2.7 Conclusion and Future Directions

In this project, we explore methods for determining the optimal values for its two key parameters: epsilon(eps) and minimum points(minPts). Our approach to find the optimal $minPts$ with the size of $0.5\text{-}\epsilon$ neighborhood and median achieved higher scores compared to estimated $minPts \pm 2$. Also, Our approach to find the optimal eps with novel K-dist method and knee point detection proved to be effective in clustering with densely packed in relatively small regions compared to the traditional K-dist method.

Our future works can be addressed with two directions: (1) DBSCAN struggles with clusters of varying densities because it scans data points sequentially and doesn't store neighborhood information separately. By changing the traversal strategy to prioritize neighboring nodes, we can gather more neighbor data without sorting distances. This could improve DBSCAN's performance on clusters with different density. (2) we can evaluate the optimal eps with well-known method, such as OPTICS[1] and HDBSCAN[5], then compare these value with our optimal eps.

References

- [1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.
- [2] P. Bhattacharjee and P. Mitra. A survey of density based clustering algorithms. *Frontiers of Computer Science*, 15:1–27, 2021.
- [3] S. V. David, Arthur. k-means++: The advantages of careful seeding. *Soda*, pages 1027–1035, 2007.
- [4] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.
- [5] L. McInnes, J. Healy, and S. Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2, 03 2017. doi: 10.21105/joss.00205.
- [6] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, pages 53–65, 1987.
- [7] A. Starczewski, P. Goetzen, and M. J. Er. A new method for automatic determining of the dbscan parameters. *Journal of Artificial Intelligence and Soft Computing Research*, 10(3):209–221, 2020.



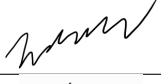
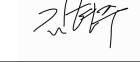
Name	Sign	Individual Contribution	Percentage
SUNGWON BAE		k-means++ draft, dataset, optimal k method	25%
JUNSEO KIM		k-means++ implementation and visualization	25%
HYEONJIN JO		DBSCAN Implementation and Optimal minPts	25%
HYUNJU KIM		Optimal eps, Dataset, and Experiments	25%

Table 2: Percentage of the contribution must be 100% in total. For writing individual contribution, refer to <https://www.elsevier.com/researcher/author/policies-and-guidelines/credit-author-statement>