CSE304 Introduction to Data Mining
Group #2
Contact point tmdvy830@unist.ac.kr

Assignment #2

1 K-means++

1.1 Introduction

Among various clustering techniques, the K-means algorithm is widely used due to its simplicity and effectiveness. The algorithm divides data into K clusters by repeatedly calculating the center of each cluster and assigning data points to the nearest cluster center. However, its performance can significantly vary depending on the initial selection of cluster centers, as randomly chosen centers may not guarantee optimal results. Additionally, K-means is an NP-hard problem,. [4, 2] meaning that finding an optimal solution in polynomial time is difficult, and the computational cost can increase sharply with data size and complexity.

To address this, the K-means++ algorithm [1] was proposed. K-means++ improves upon K-means by selecting initial cluster centers more intelligently, considering the distance information between data points. This enhancement reduces the likelihood of poor clustering results and improves overall efficiency.

Determining the value of K, the number of clusters, is another crucial issue. This report employs the silhouette method, a metric for evaluating clustering quality. A higher silhouette value indicates better assignment of data points to clusters. By using this method, the optimal value of K can be determined, ensuring more effective clustering.

This report demonstrates that combining the K-means++ algorithm with the silhouette method improves the implementation of the K-means algorithm, addressing the issues of initial center selection and optimal K determination, and achieving better clustering outcomes.

1.2 Problem Statement

Given a set X of n points in a 2-dimensional space and an integer k, initialize k cluster centroids using the k-means++ algorithm. Group the points into k clusters $C = C_1, C_2, \ldots, C_k$ such that $Cost(C) = \sum_{i=1}^{k} \sum_{x \in C_i} (x - c_i)^2$ is minimized, where c_i is the centroid of the points in cluster C_i.

Additionally, let $a(x)$ be the average distance from x to all other points in C_i, and let $b(x)$ be the minimum average distance from x to points in any other cluster C_j, $j \neq i$. Make the silhouette coefficient

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

close to 1.

1.3 Algorithm Description and Analysis

Algorithm 1 K-means

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

The K-means algorithm is the clustering algorithm that divides a dataset into K clusters. This algorithm iteratively updates the center of each cluster to maximize the similarity within clusters and minimize the similarity between clusters. The main steps of the K-means algorithm are as follows. First, K cluster

centers are randomly selected. Then, each data point is assigned to the nearest cluster center. Next, the center of each cluster is updated to the mean of the data points currently assigned to that cluster. The steps of cluster assignment and center updating are repeated until the cluster centers no longer change. The time complexity of the K-means algorithm is O(n * k * t), where n is the number of data points, k is the number of clusters, and t is the number of iterations.

Algorithm 2 K-means++

1: Choose an initial center c_1 uniformly at random from X, The set of all data points.
2: **for** $i = 2$ to K **do**
3: Choose the next center c_i, selecting $c_i = x' \in X$ with probability $\frac{D(x')^2}{\sum_{x \in X} D(x)^2}$
4: **end for**

The K-means algorithm is sensitive to the initial selection of cluster centers. Since the initial centers are chosen randomly, the final clustering results can vary depending on the initial choice, even for the same dataset. To overcome this limitation, we see the K-means++ algorithm. K-means++ aims to improve the performance of the K-means algorithm by selecting the initial cluster centers more carefully. The main steps of the K-means++ algorithm are as follows.

The first cluster center is initialized by randomly selecting one data point. The remaining cluster centers are chosen with a probability proportional to the square of the closest distance from the selected cluster centers. This process is repeated until k centers are selected.

Since selecting each center involves calculating the distance for n data points, this step takes O(n) time. This process needs to be repeated k-1 times, so the total time complexity is O(n) * (k-1) = O(n * k).

If k isn't given, the algorithm should estimate the optimal number of cluster k for given dataset. There are several metric to evaluate the result of clustering.

First, WCSS(Within Cluster Sum of Squares) is the sum of the squared distance between each point and the centroid of cluster. So, It can be represented as follow formula.

$$WCSS = \sum_{C_k}^{C_n} (\sum_{d_i in C_i}^{d_m} distance(d_i, C_k)^2)$$

Where d_i indicate each point in Cluster i.

To find the optimal number of cluster using WCSS, we can use elbow method. When we plot WCSS against k, the resulting graph resembles an elbow as follow figure. Then, the point of inflection will be the optimal number of cluster we find. But the decreasing pattern doesn't not appeared in every pattern, so it is difficult to define the elbow point mathematically, and also it only reflect the density inside each cluster, but not completely reflect the effect with between clusters. The good clustering result have to show perfectly separated cluster.

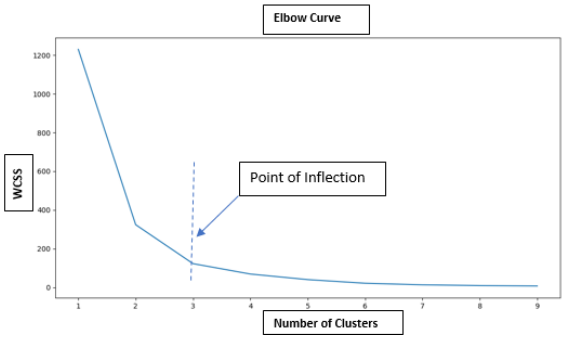


Figure 1: Elbow-method for WCSS

Second metric we have considered is Silhouette Coefficient. Silhouette Coefficient is expressed with a and b where a is the average distance between each point o in dataset and all other points in the cluster to which o belongs and b is the minimum average distance from o to all clusters to which o does not belong. Therefore, a indicate how the point is close together in same cluster, and b indicate how the cluster is separated from other clusters It can be also represented as follow formula.

$$a(o) = \frac{\sum_{o' \in C_i, o \neq o'} dist(o, o')}{|C_i - 1|}$$

and

$$b(o) = \min_{C_j : 1 \leq j \leq k, j \neq i} \frac{\sum_{o' \in C_j} dist(o, o')}{|C_j|}$$

where, C is the cluster in result.

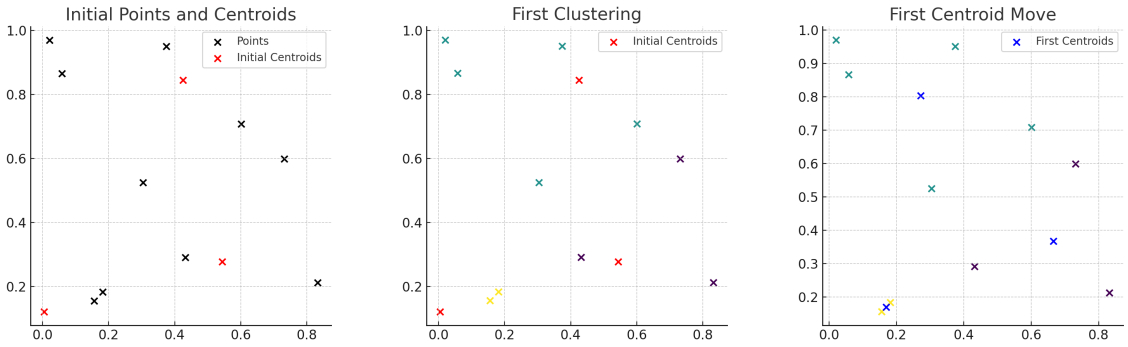
Then, the silhouette coefficient is as follow:

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

By the definition of the coefficient, it should be in $-1 \leq s(o) \leq 1$, and the larger value means the greater result, well-clustered.

So, in here, we decide to use Silhouette Coefficient to evaluate the result of K-means++ algorithm, and choose the highest Silhouette Coefficient among the several k.

1.4 Example



(a) Initial state (b) First clustering (c) First move of centroids

Figure 2: K-means Example

Example 1. *The first plot shows the initial distribution of data points (black x-marks) and the initial positions of the centroids (red x-marks). Second plot, the data points are colored according to their assigned clusters after the first iteration of K-means. Each point is colored differently based on the nearest initial centroid. The red x-marks still represent the initial centroids. Third plot shows the updated positions of the centroids (blue x-marks) after the first iteration. The centroids have moved to new positions that are the mean of the points assigned to each cluster.*

1.5 Advantages and Disadvantages

K-means++ initialization improves convergence speed and consistency by setting initial cluster centroids in well-dispersed positions, reducing improper clustering from poor initialization. The silhouette method helps determine the optimal number of clusters by calculating the silhouette coefficient for each data point and selecting the number of clusters that maximize the average silhouette coefficient. This method considers both intra-cluster cohesion and inter-cluster separation, providing a reliable criterion for cluster determination.

K-means scales well with the number of data points and features, making it suitable for large-scale clustering tasks. With K-means++ initialization, the algorithm typically converges quickly to a local optimum, yielding reliable clustering results.

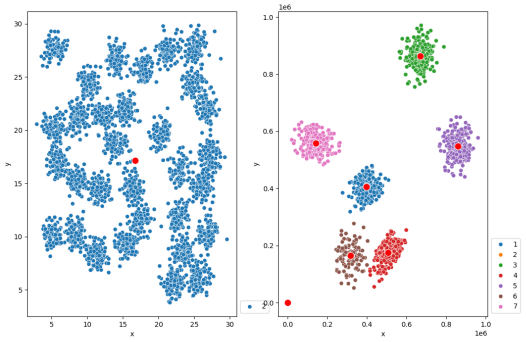
However, K-means++ can increase computational cost during initialization due to repeated distance calculations for all data points when selecting initial centroids. Similarly, using the silhouette coefficient to determine the optimal number of clusters incurs significant computational cost as clustering must be performed multiple times for various k values.

K-means++ is effective for arbitrarily distributed datasets but struggles with datasets of irregular density and is sensitive to data scaling. Incorrect scaling can negatively impact results. The silhouette method is useful for data with separated cluster structures but may struggle when cluster boundaries are ambiguous. Additionally, K-means partitions clusters linearly, which may not be suitable for data with complex cluster structures.

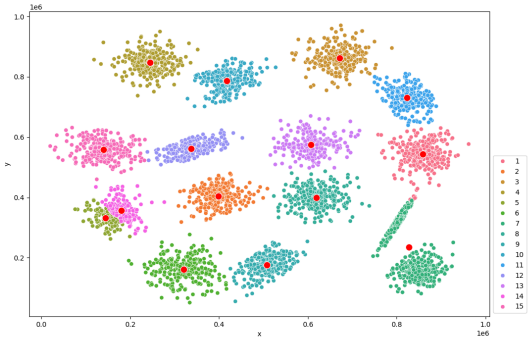
1.6 Experiment

We implemented the algorithm using the basic functions of Java and verified its performance through six datasets. The implementation code and dataset were all put on this GitHub link[1].

We executed clustering algorithms in this experiment using six different datasets and analyzed the results. Each dataset has diverse characteristics and features, allowing for a comprehensive review of various outcomes. The artd-31 dataset contains 5,000 data points, some clustered within a relatively small value range, while others are spread across a wider range. The artset1 dataset also contains 5,000 data points, but they are widely dispersed and primarily consist of large values. Also, we use some dataset in the external site [2] which has a various form of dataset usually used in the clustering test. In this site, we use s3, a2, t4.8k, s2, each corresponds to data1, data2, data3, data4, and data1 and data4 are distributed arbitrarily, data3 and data4 doesn't. For this, we expect that K-means++ shows good performance in data1 and data4, DBScan algorithm shows it in data2 and data3.



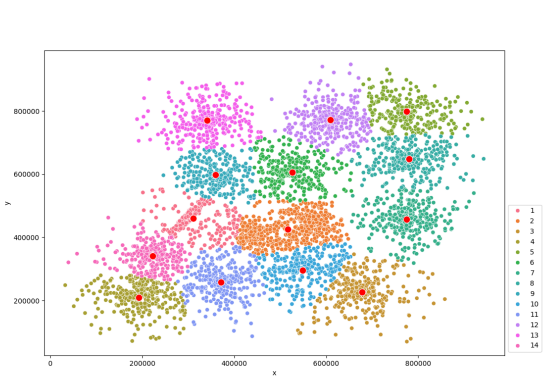
(a) artd-31(K: 7)



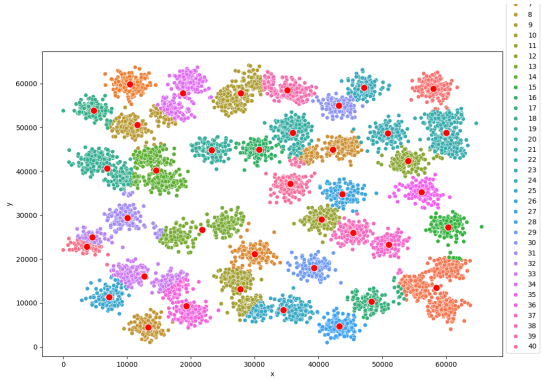
(b) artset1(K: 15)

The left graph of artd-31 shows a cluster pattern with points densely placed in a small range and widely dispersed clusters. The right graph displays well-separated clusters over a wide range, with distinct clusters represented in different colors and centroids marked with red dots. However, points near the origin in the left graph are poorly clustered because when we allocate each centroid in distance-proportional random method, the points near the origin can be allocated only one point due to its large scale.

The artset1 dataset demonstrates clear, well-separated clusters, with each cluster densely placed around its centroid, indicating meaningful groupings distinct from other clusters.



(a) data1(K: 14)

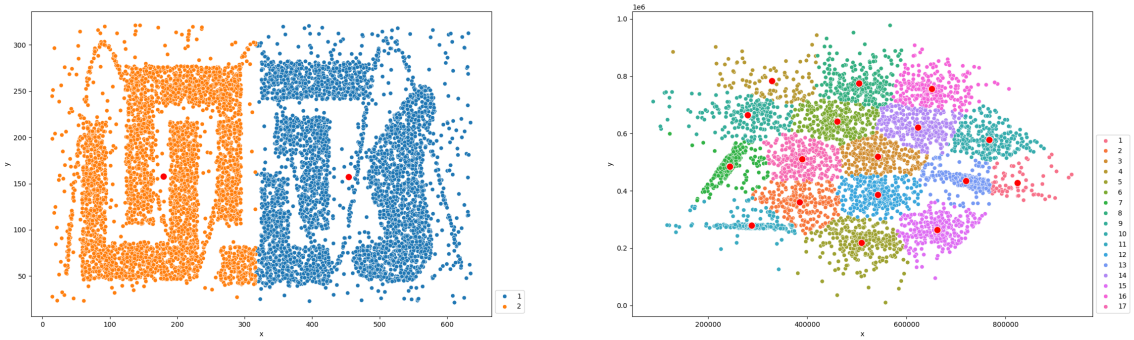


(b) data2(K: 40)

[1] https://github.com/eqcoder/A2_G2.git
[2] https://cs.joensuu.fi/sipu/datasets/

The graph for data1 shows clearly defined clusters with well-separated data points, high cohesion around centroids, and central positioning of centroids within each cluster even though the dataset has ambiguous boundary at a glance.

Similarly, the graph for data2 indicates that centroids are centrally located within clusters, with points gathered around each cluster's centroid. There are some clusters which should be separated each other but doesn't, or which should be one cluster but separated. It is also because data2 has some densely block. So data2 represent well that K-means++ doesn't proper this condensed dataset.



(a) data3(K: 2) (b) data4(K: 17)

Overall, the result of the data3 image shows some separation, but it is not perfect because it has some blocks with higher density. So data3 isn't proper for K-means++ algorithm. Specifically, the blocks at the bottom, which are not grouped into the same cluster despite being adjacent, indicate that the clustering did not perform well.

For data4, the K-means clustering result shows 17 clusters. Unlike data3, it has fewer, larger clusters spread over a wider range. Although the cluster boundaries are less distinct than in data3, the clusters are still well separated.

1.7 Conclusion and Future Directions

We applied K-means clustering to various datasets using the silhouette method and K-means++. The silhouette method evaluates how well each data point fits within its cluster, helping assess clustering quality. K-means++ improves clustering efficiency by setting appropriate initial centroids.

For uniformly distributed datasets, K-means formed clear and distinct clusters. However, for densely placed datasets within a small range, it often grouped data points into a single cluster, highlighting its limitations in reflecting cluster size and density. Additionally, K-means struggled with noisy datasets, often assigning noise to separate clusters, indicating its sensitivity to noise and inability to handle it effectively.

To address these issues, using a density-based clustering method like DBSCAN [3] is beneficial. DBSCAN forms clusters based on data point density, producing better results for dense or noisy data by effectively ignoring noise and focusing on dense regions.

A hybrid approach combining K-means with other clustering methods can also be effective. For instance, using a density-based method to remove noise before applying K-means to further subdivide clusters can enhance overall clustering performance.

In conclusion, recognizing the limitations of K-means and using density-based methods like DBSCAN or hybrid approaches can improve clustering accuracy and efficiency. We will explore these methods in the next steps to analyze clustering performance on various datasets.

2 DBSCAN

2.1 Introduction

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that identifies clusters based on the density of data points, making it suitable for datasets with

complex and irregular distributions. Unlike k-means, DBSCAN can discover clusters of various shapes and effectively handle noise.

DBSCAN requires two parameters: MinPts and Eps. MinPts is the minimum number of neighbor points required to form a dense region, distinguishing noise from clusters. Eps is the maximum distance at which a point is considered a neighbor, defining the neighborhood's radius. Selecting appropriate values for these parameters is crucial for successful clustering.

MinPts can be determined using the silhouette coefficient, which measures the quality of clustering. By analyzing silhouette scores for various MinPts values, we can identify the best value for clustering quality. Eps can be determined using the k-dist graph, which plots the distance to the k-th nearest neighbor for each data point. The knee point in this graph represents a threshold where point density significantly decreases, and the Eps value at this point is selected for DBSCAN.

By optimizing MinPts and Eps, DBSCAN can generate high-quality clustering results tailored to specific datasets.

2.2 Problem Statement

Given a set X of n points in a 2-dimensional space, a minimum number of points $MinPts$, and a distance threshold Eps, group the points into clusters using the DBSCAN algorithm.3 Identify $C = C_1, C_2, \ldots, C_k$ consists of points that are density-connected within the cluster, while points from different clusters are not density-connected to each other. Identify points that do not belong to any cluster as noise.

2.3 Algorithm Description and Analysis

Algorithm 3 DBSCAN

```
1:  procedure DBSCAN(SetOfPoints, Eps, MinPts)
2:      // SetOfPoints is UNCLASSIFIED
3:      ClusterId ← nextId(NOISE)
4:      for i ← 1 to SetOfPoints.size do
5:          Point ← SetOfPoints.get(i)
6:          if Point.ClId = UNCLASSIFIED then
7:              if ExpandCluster(SetOfPoints, Point, ClusterId, Eps, MinPts) then
8:                  ClusterId ← nextId(ClusterId)
9:              end if
10:         end if
11:     end for
12: end procedure
```

The algorithm iterates over each point in the dataset to check if a new cluster can be formed from that point. It uses the Eps and MinPts parameters to determine the density of points and identify clusters and noise. This process involves marking points as part of a cluster or as noise, and incrementally assigning cluster IDs whenever a new cluster is formed.

The algorithm retrieves each point in the dataset one by one through the for loop on line 4. In this loop, the statement on line 5 assigns the current point. For each point, the if statement on line 6 attempts to expand the cluster only if the point has not already been assigned to a cluster. This prevents reprocessing of already classified points. The if statement on line 7 calls the ExpandCluster function. This function checks if there are at least MinPts points within the radius Eps to determine if the current point can be a core point of a cluster. This assesses the point's density. If the ExpandCluster function successfully expands the cluster, it returns true; otherwise, it returns false. As the ExpandCluster function expands the cluster, it marks points with the current cluster ID. If the cluster expansion fails, the point is subsequently marked as noise. On line 8, the cluster ID is incremented when a new cluster is formed, assigning a unique ID to each new cluster.

ExpandCluster4 represents a method to expand a cluster starting from a given point. Here's an algorithm and a detailed explanation of each part of the algorithm.

In line 2, store all neighboring points within the radius Eps into seeds. In line 3, check if the number of neighboring points is less than MinPts. If the number of neighboring points is less than MinPts, designate the current point as noise and return False. If the number of neighboring points is greater than or equal to MinPts, start expanding the cluster. Then, change the cluster ID in seeds to ClId. Since the current

Algorithm 4 ExpandCluster

```
1:  procedure ExpandCluster(SetOfPoints, Point, ClId, Eps, MinPts)
2:      seeds ← SetOfPoints.regionQuery(Point, Eps)
3:      if seeds.size < MinPts then
4:          SetOfPoint.changeClId(Point, NOISE)
5:          return False
6:      else
7:          // all points in seeds are density-reachable from Point
8:          SetOfPoints.changeClIds(seeds, ClId)
9:          seeds.delete(Point)
10:         while seeds ≠ Empty do
11:             currentP ← seeds.first()
12:             result ← SetOfPoints.regionQuery(currentP, Eps)
13:             if result.size ≥ MinPts then
14:                 for i ← 1 to result.size do
15:                     resultP ← result.get(i)
16:                     if resultP.ClId IN {UNCLASSIFIED, NOISE} then
17:                         if resultP.ClId = UNCLASSIFIED then
18:                             seeds.append(resultP)
19:                         end if
20:                         SetOfPoints.changeClId(resultP, ClId)
21:                     end if
22:                 end for
23:             end if
24:             seeds.delete(currentP)
25:         end while
26:         return True
27:     end if
28: end procedure
```

point has been used, remove the current point from seeds. Repeat while seeds is not empty. Select the first point from seeds as currentP. Store all neighboring points within the radius Eps of currentP into result. Check if the number of points in result is greater than or equal to MinPts. If so, repeat for all points in result. Each point is resultP. Check if the cluster ID of resultP is UNCLASSIFIED or NOISE. Change the cluster ID of resultP to ClId. And add resultP to seeds only if it is UNCLASSIFIED. Then remove currentP from seeds. Repeat this process, and once the cluster expansion is complete, return True.

If MinPts or Eps aren't given, the algorithm should estimate the optimal value. For the estimation of MinPts, we apply the Silhouette Coefficient with given Eps, and find the best MinPts has the highest Silhouette score. For Eps, we compute the k-th nearest distance for every point, and sort them with increasing order. Then, the graph form a albow curve we saw above. To determine the elbow point, we connect the first point and last point straightly, and find the point farthest from that straight line. Follow figure is it.

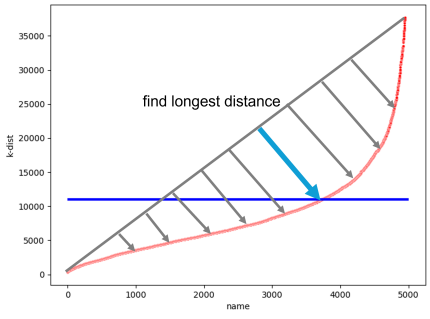
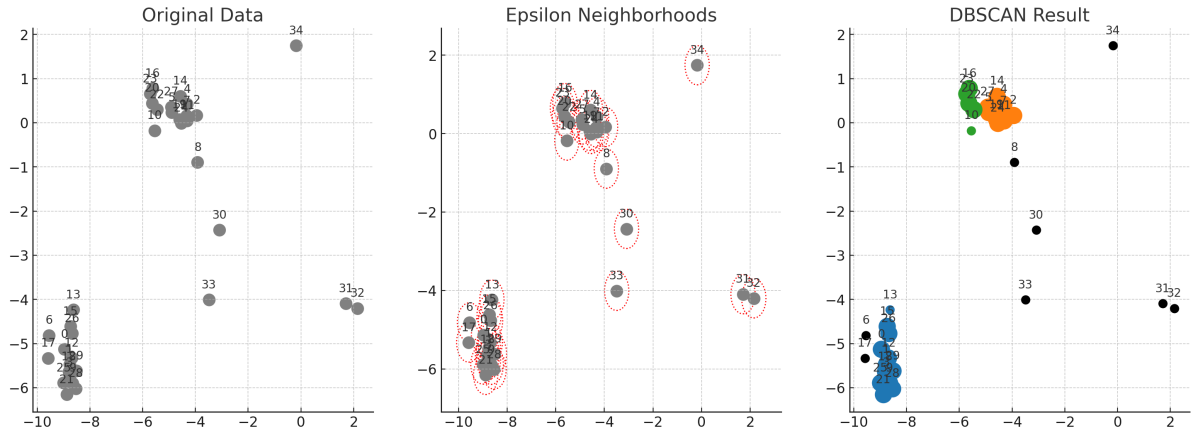


Figure 6: find elbow point in k-dist plot

2.4 Example



(a) DBSCAN Example

Figure 7: DBSCAN Example

Example 2. *The above graphs provide a detailed explanation of the steps in the DBSCAN algorithm. The first graph shows the initial data. In the second graph, circles with a radius of Eps are drawn around each data point to visualize the neighbors. Here, the Eps value is set to 0.5. The final graph shows the result of applying the DBSCAN algorithm. Different colors represent different clusters, and black dots represent noise (data points that do not belong to any cluster).*

2.5 Advantages and Disadvantages

DBSCAN treats low-density data points as noise and excludes them from the clustering process, which helps form more accurate clusters. This algorithm also has the ability to effectively detect non-spherical clusters. It is capable of identifying clusters of various shapes. This provides a significant advantage over other algorithms like K-means clustering.

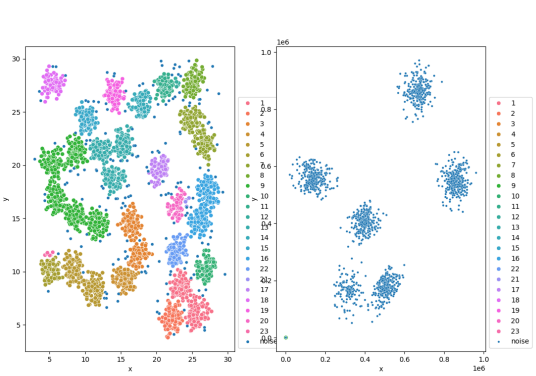
Additionally, another innovative advantage of DBSCAN is that it does not require the number of clusters to be specified in advance. Unlike K-means clustering, where the number of clusters must be determined beforehand, DBSCAN automatically determines the appropriate number of clusters based on the density of the data. Users do not need to perform preliminary work to decide the number of clusters, as DBSCAN helps reveal the structure of the data naturally.

However, there are also some disadvantages. One of the disadvantages of the DBSCAN algorithm is the variability in cluster sizes. If there is a large difference in the densities of the clusters, the same Eps and MinPts values may not effectively detect clusters of various sizes and densities. This can result in some clusters not forming properly, or conversely, being overly divided. This issue negatively affects the quality of the clustering results.

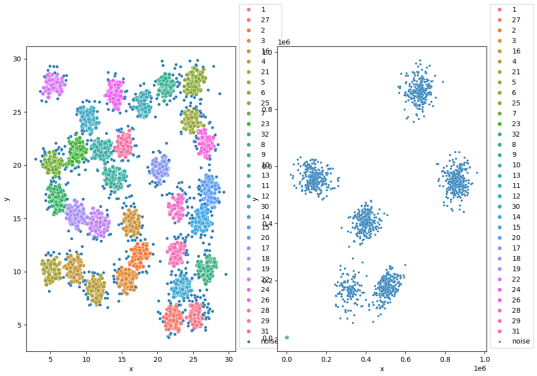
Moreover, setting appropriate values for Eps and MinPts can be challenging. Optimizing these two values to fit the characteristics of the data is necessary. However, finding the optimal values can be difficult and may require several attempts and adjustments through experience, depending on the distribution of the data.

2.6 Experiment

Using the previously employed datasets, we examined the results with MinPts set to 4 and optimal. We started with MinPts set to 4 and found the optimal Eps. Using this optimal Eps, we then searched for the optimal MinPts. This process was repeated until we found good results.



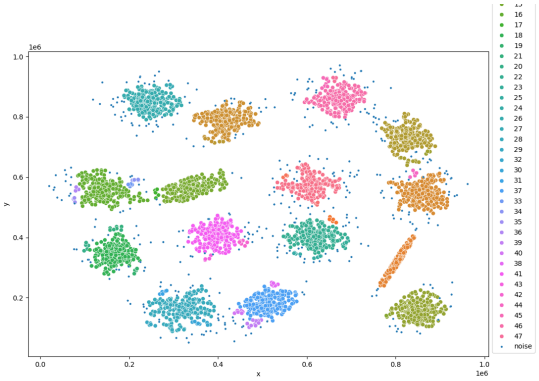
(a) artd-31(MinPts: 4, Eps: 0.5)



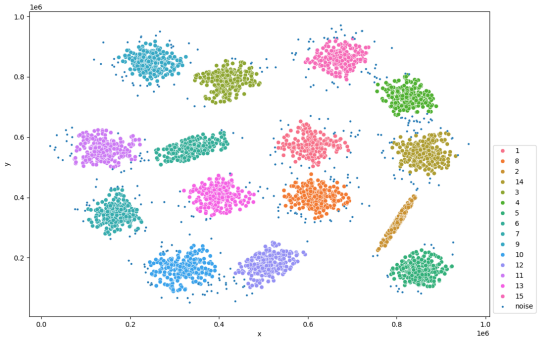
(b) artd-31(MinPts: 9, Eps: 0.7)

In artd-31(4), the left graph shows a cluster pattern that is densely placed in a small range and widely distributed, while the right graph shows well-separated clusters over a wide range. In the left graph, clear clusters represented by different colors are formed. However, it can be seen that data points far from the origin are not well clustered.

In artd-31(9), the dataset is separated with more number of cluster. It seems more natural and obvious clustering. So the optimal ninPts in artd-31 is 9, and it doesn't work in the large domain too.



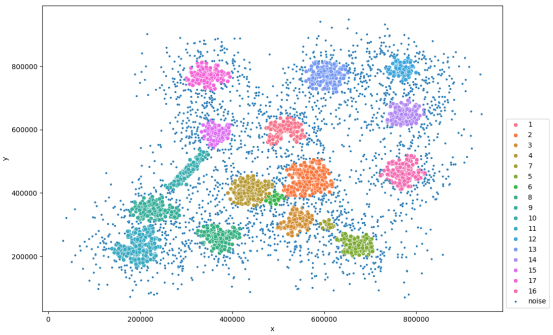
(a) artset1(MinPts: 4, Eps: 13060.2)



(b) artset1(MinPts: 12, Eps: 20362.3)

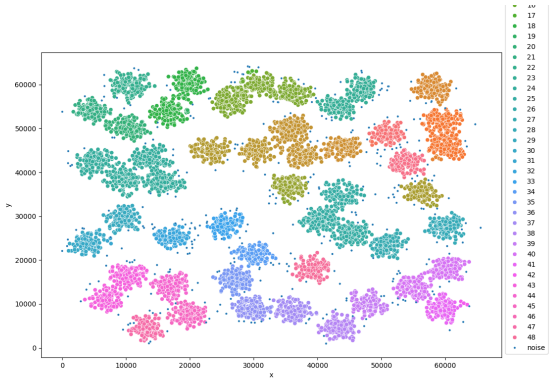
In both artset(4) and artset(12), it can be seen that the main clusters are well classified. However, many of the small local clusters that occurred in artset(4) have been largely integrated in artset(12).

(a) data1(MinPts: 4, Eps: 16852.1)

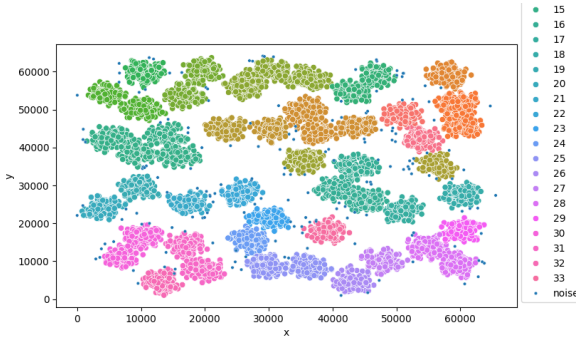


(b) data1(MinPts: 18, Eps: 16852.1)

When the MinPts value was 4 in data1, the clustering failed because most areas were recognized as a few clusters due to the uniform density. When the MinPts value was 18, it was possible to identify several clusters in the denser areas, but a significant amount of noise also appeared.

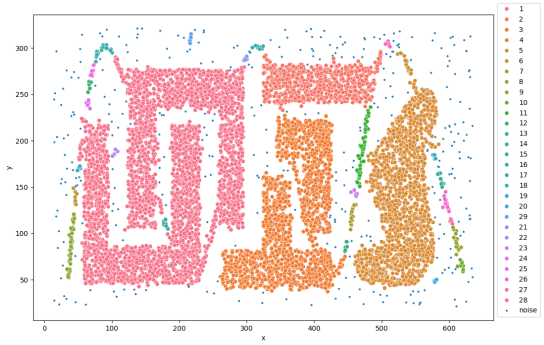


(a) data2(MinPts: 4, Eps: 837.1)

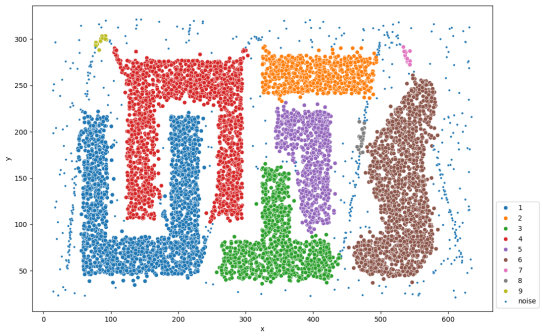


(b) data2(MinPts: 10, Eps: 1254.6)

In the case of data2, it can be seen that all major clusters are well classified.



(a) data3(MinPts: 4, Eps: 6.1)



(b) data3(MinPts: 15, Eps: 10.0)

In both cases, successful clustering of the main clusters was achieved. The difference is that with a MinPts value of 4 in DBSCAN, many small clusters were formed. With a MinPts value of 15 in DBSCAN, more noise points were identified and fewer but larger and denser clusters were formed. There are some small cluster we didn't expect as cluster because we estimate the optimal minPts with Silhouette Coefficient. It doesn't proper this dataset because it has some high density of block, but doesn't condensed with one center.

(a) data4(MinPts: 4, Eps: 16311.8)

(b) data4(MinPts: 22, Eps: 16311.8)

Similar to data1, data4 also has many uniformly spread-out parts, making it difficult to perform clustering overall. In minPts=4, one large cluster can be easily found. However, when we find the optimal minPts=22, it is better clustering has several separated cluster. But even it is the best clustering, the data4 doesn't proper DBScan due to its ditribution.

2.7 Conclusion and Future Directions

By using DBSCAN for clustering, we observed that it works effectively when data points are clearly distinguishable by density. It has shown robustness in identifying clusters even in complex, non-linear distributions, making it useful for handling intricate data structures. Particularly, DBSCAN excels in outlier detection, making it valuable in the data cleansing process.

However, the limitations of DBSCAN also became apparent. When dealing with data that has varying scales, DBSCAN struggled to form accurate clusters. It was also challenging to distinguish clusters in uniformly dense data, resulting in optimal clustering outcomes.

Based on these conclusions, we will next introduce a method to overcome these limitations by integrating DBSCAN with K-means. By combining the strengths of both methods, we aim to develop a hybrid algorithm that can improve clustering performance in specific data scenarios.

3 Hybrid algorithm of K-means++ and DBSCAN

3.1 introduction

K-means++ is efficient and works well for clusters that are spherical and uniformly dense, but its performance drops when dealing with clusters of different densities and shapes. It is also sensitive to noise and outliers, which can degrade clustering performance. Also, DBSCAN excels at identifying clusters of arbitrary shapes and is robust to noise. By defining clusters based on the density of data points, DBSCAN can effectively handle outliers. However, DBSCAN struggles when clusters have significant differences in density and scale, selecting appropriate parameters can be challenging.

To overcome these limitations, a hybrid approach that combines the strengths of K-means++ and DBSCAN can provide a more robust clustering solution. The proposed method involves first applying DBSCAN within each initial cluster to identify dense regions and filter out noise. Then, K-means++ is used to partition the data into final clusters. This hybrid approach compensates for the weaknesses of both algorithms. DBSCAN accurately captures complex shapes and densities of clusters, refining initial clusters, while K-means++ partitions the data within these clusters. By combining both methods, we can achieve more accurate and meaningful clustering results in datasets with varying densities and noise levels.

This hybrid approach has yielded significant clustering results in practical experiments, particularly in areas where K-means++ or DBSCAN alone previously produced ambiguous results.

3.2 Problem Statement

Given a set X of n points in a 2-dimensional space, apply the DBSCAN algorithm first and then use the k-means clustering algorithm. DBSCAN is used initially to identify the core points based on density, and k-means is used to identify meaningful clusters that were not previously detected.

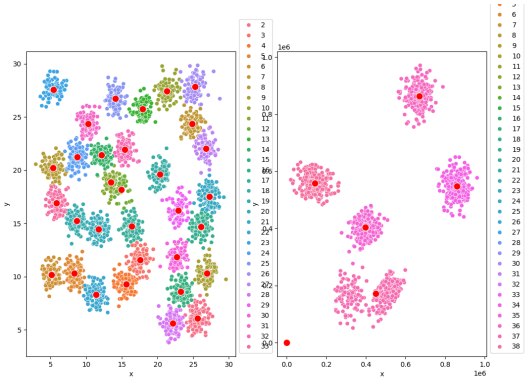
3.3 Algorithm Description and Analysis

Algorithm 5 Hybrid Algorithm

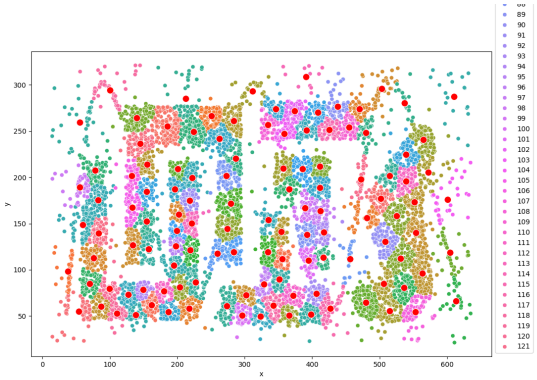
1: $C_{DBSCAN} \leftarrow \text{DBSCAN}(X)$
2: $initial\_centroids \leftarrow \text{ComputeInitialCentroids}(C_{DBSCAN})$
3: $C \leftarrow \text{KMeans}(X, initial\_centroids)$
4: **return** C

Apply the DBSCAN algorithm to the dataset X. The output, C_DBSCAN, consists of clusters formed by DBSCAN. For each cluster identified by DBSCAN, compute the centroid. Apply the K-means algorithm to the dataset X, using the centroids computed in the previous step as the initial centroids. Return final clusters.

3.4 Experiment



(a) artd-31(K: 38, MinPts: 9, Eps: 0.5)



(b) data3(K: 121, MinPts: 15, Eps: 6.1)

In the artd-31 dataset, where significant scale differences previously hindered effective clustering, the hybrid approach yielded relatively clear clustering results. However, for other datasets, existing clustering methods that each had their own strengths showed more good results. The new hybrid method tended to generate a somewhat larger number of clusters.

3.5 Conclusion and Future Directions

In this report, we proposed a hybrid approach that combines the K-means++ and DBSCAN algorithms and evaluates its performance through experiments. The results indicate that the hybrid approach compensates for the drawbacks of the individual algorithms on certain datasets, resulting in improved clustering performance. Specifically, on datasets with non-uniform densities and non-uniform scale structures, the hybrid method outperformed both K-means++ and DBSCAN. The artd-31 dataset, effectively clustered data with significant scale differences.

However, on other datasets, the hybrid approach tended to produce an excessive number of clusters. This issue arises because the small clusters generated during the DBSCAN phase are further subdivided by K-means++. To address this, more refined parameter tuning for DBSCAN or adding a post-processing step to the hybrid approach to merge unnecessary clusters could be considered. Anyway, combination of two algorithm also isn't good approach. To suggest better approach, applying the algorithm individually, and comparing the result of two algorithm also be a good approach since these two algorithms have opposite pros and cons.

Optimizing the parameters of DBSCAN and K-means++ or incorporating a cluster merging algorithm could reduce the number of excessive clusters and yield more consistent clustering results. For instance, merging small clusters within a specific distance threshold might be an effective solution.

Additionally, to ensure efficient performance on large-scale datasets, we can explore ways to reduce the computational complexity of the algorithm. Techniques such as sampling methods or optimized data structures could be utilized to improve the algorithm's speed.

In conclusion, the K-means++, DBSCAN, and hybrid approaches demonstrated decent performance on various datasets, and there is considerable potential for further improvement.

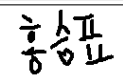
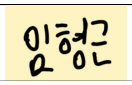
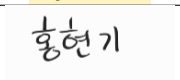
References

[1] D. Arthur, S. Vassilvitskii, et al. k-means++: The advantages of careful seeding. In *Soda*, volume 7, pages 1027–1035, 2007.

[2] S. Dasgupta. The hardness of k-means clustering. 2008.

[3] 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.

[4] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 442:13–21, 2012.

Name	Sign	Individual Contribution	Percentage
홍승표		Writing - Original Draft, Formal analysis, Methodology	50%
임형근			0%
Hong Hyeongi		Validation, Conceptualization, Software	50%