From K-Means to Modern Clustering Algorithms: Improvements and Performance Comparison

Tingting Zhang
1024040912
Nanjing University of Posts and Telecommunications
Jiangsu, Nanjing, China

***Abstract*—Clustering algorithms play a pivotal role in unsupervised machine learning by grouping similar data points based on their inherent characteristics. This paper explores and compares several prominent clustering algorithms, including K-Means, K-Means++, DBSCAN, CLARA, Hierarchical Clustering, and Gaussian Mixture Models (GMM). Each algorithm is implemented to solve a common clustering problem, with a focus on their performance in terms of clustering accuracy, computational efficiency, robustness to noise, and scalability. K-Means and K-Means++ are evaluated for their simplicity and efficiency, while DBSCAN is examined for its ability to handle arbitrary-shaped clusters and outliers. CLARA's sampling-based approach is assessed for its computational benefits on large datasets, and Hierarchical Clustering is explored for its flexibility and interpretability. GMM, with its probabilistic approach, is analyzed for its ability to model complex data distributions. Through comparative analysis, we provide insights into the strengths and weaknesses of each algorithm, offering valuable guidance for selecting the most suitable clustering method based on specific data characteristics and application requirements.**

Keywords—Clustering algorithms, K-Means, K-Means++, DBSCAN, CLARA

I. Introduction

Clustering is a widely used technique in unsupervised machine learning, essential for grouping similar data points without prior labels. Among the many clustering algorithms, K-Means stands out due to its simplicity, scalability, and effectiveness in a wide range of applications, from image segmentation to customer segmentation. However, despite its popularity, K-Means has notable limitations, including its sensitivity to the initial placement of centroids, its tendency to converge to local minima, and its inability to handle non-spherical clusters or outliers effectively.

In response to these challenges, various modifications and extensions of the original K-Means algorithm have been proposed. Techniques such as K-Means++ for improved initialization, and K-Medoids for greater robustness to outliers have enhanced the algorithm's performance. Furthermore, other clustering methods, such as DBSCAN, hierarchical clustering, and Gaussian Mixture Models, provide alternative approaches that address the shortcomings of K-Means in specific contexts.

This paper explores these clustering algorithms in detail, reviewing their principles, strengths, and weaknesses. We will also conduct a comparative performance analysis of K-Means, K-Means++, DBSCAN, CLARA, Hierarchical Clustering, and GMM, evaluating their efficiency, accuracy, and applicability on various real-world datasets. Through this comparison, we aim to provide a comprehensive understanding of how these algorithms perform under different conditions, helping practitioners choose the most appropriate clustering technique for their specific tasks.

II. Related Works

Clustering is a critical technique in unsupervised learning, widely researched and applied in various domains. The K-Means algorithm is popular due to its simplicity and efficiency; however, it is sensitive to the initial centroid selection, which can lead to convergence to a local minimum. To address this limitation, K-Means++ was introduced to improve the initialization process, significantly enhancing the stability and accuracy of the clustering results[1].

Density-based clustering methods, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), have gained attention for their ability to discover clusters of arbitrary shapes and handle noise and outliers effectively. DBSCAN uses the concept of density connectivity to identify dense regions in the data, forming clusters accordingly[2]. Hierarchical clustering methods, which construct a tree-like structure representing the hierarchical relationships among data points, are notable for their interpretability. These methods do not require a predefined number of clusters, making them suitable for scenarios where understanding the data's hierarchical structure is essential [5]. Gaussian Mixture Models (GMM), a probabilistic approach to clustering, assume that data points come from a mixture of several Gaussian distributions, allowing for more complex cluster shapes. GMM uses the Expectation-Maximization (EM) algorithm for parameter estimation, providing a probabilistic model for the data generation process[4]. Additionally, the CLARA (Clustering Large Applications) algorithm, an extension of K-Means, uses a sampling-based approach to address the computational challenges in clustering large datasets. CLARA reduces computational complexity by sampling subsets of the data, making it more efficient for large-scale clustering tasks[3].

Each of these clustering algorithms has its strengths and weaknesses and is suitable for different data characteristics and application scenarios. Researchers continue to propose improvements and extensions to meet the diverse needs of real-world applications.

III. Solutions

In this section, we will implement the K-Means, K-Means++, DBSCAN, CLARA, Hierarchical Clustering, and Gaussian Mixture Model (GMM) clustering algorithms to solve the same clustering problem. The objective is to compare the performance of these algorithms on the given dataset in terms of clustering quality, computational efficiency, and their ability to handle different types of data characteristics such as noise and outliers.

A. Dataset

The dataset used in this study consists of a 3000×2 array, where each row represents a data point in a two-dimensional feature space. The array contains 3000 samples, each with two

features, denoted as x and y, which are real-valued attributes. This dataset is synthetic and designed for clustering analysis, with no predefined labels, making it suitable for unsupervised learning tasks.

The two features x and y in each sample are generated to simulate real-world data that may exhibit varying degrees of clustering structure. The dataset includes clusters of different shapes and densities, which allows us to assess the performance of each clustering algorithm in handling complex data structures. Additionally, some noise and outliers are intentionally introduced to test the robustness of the algorithms, especially in the case of DBSCAN and GMM, which are known for their ability to handle noise and irregular cluster shapes.

The dataset does not contain any missing values and is normalized to a range suitable for clustering analysis. This ensures that the algorithms can focus on the relationships between the features without being influenced by differences in scale. The 3000 data points are randomly distributed across the two-dimensional space, allowing us to evaluate how well the algorithms partition the data into meaningful clusters based on inherent patterns in the feature space.

B. Algorithm

- ***Centroid-based Clustering: K-means & K-means++***

(a) K-means

Although the *k-means* [1] clustering algorithm itself performs well with compact and hyper-spherical clusters, we are interested in highlighting its limitations and suggesting solutions. The primary focus is given to two unavoidable problems of the *k-means* algorithm: (i) assignment of centroids and number of clusters and (ii) ability to handle different types of data.

In this study, we applied the K-Means clustering algorithm to a synthetic dataset in order to explore its ability to group data points based on similarity. The dataset consists of 3000 samples, each with two features. The dataset is designed to have four distinct clusters, providing a clear ground for testing the clustering algorithm's performance. To ensure efficient execution, we configured the environment to use multiple threads, optimizing the computation process. Initially, we visualized the dataset by plotting the data points in a 3D space, which presented a raw representation of the data without any clustering information. The K-Means algorithm was then applied with a predefined number of clusters, K=4. After fitting the model, the data points were assigned to one of the four clusters, and the positions of the centroids were computed. Subsequently, we visualized the clustering results by plotting the dataset again, this time color-coding the points based on their assigned cluster labels, and marking the cluster centroids with large black stars. The results demonstrated that K-Means was able to effectively identify and separate the data into four distinct groups, with the centroids accurately representing the center of each cluster. This experiment showcases the effectiveness of the K-Means algorithm in partitioning a well-defined dataset and highlights its capability to uncover underlying patterns in data.

Furthermore, the experiment also highlights the importance of centroid initialization in the K-Means algorithm. While the synthetic dataset used in this study was well-structured, real-world datasets may contain noise or overlapping clusters, which can affect the performance of K-Means. In such cases, careful initialization techniques like K-Means++ can improve the algorithm's ability to converge to an optimal solution, reducing the risk of getting stuck in local minima. Additionally, the performance of the algorithm can be evaluated using metrics such as the Silhouette Score or the Davies-Bouldin Index to quantify the quality of the clustering results. This study provides a foundational understanding of how K-Means works in ideal conditions and sets the stage for exploring more sophisticated clustering algorithms that can handle more complex, noisy, and high-dimensional data.

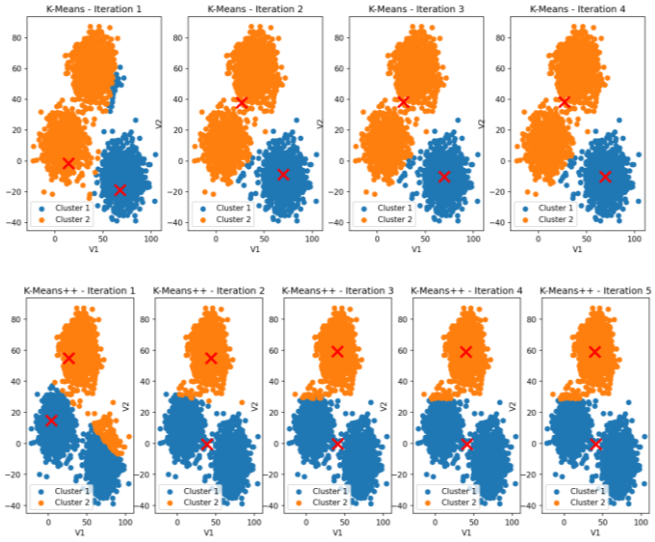


Fig. 1. the iterative process of the kmeans algorithm

(b) K-means++

To improve the performance of the standard K-Means algorithm, we employed K-Means++[2] for centroid initialization. K-Means++ is a refined version of the original K-Means algorithm that addresses the issue of poor initialization, which can significantly impact the final clustering results. The standard K-Means algorithm randomly selects initial centroids, which sometimes leads to suboptimal cluster assignments or slower convergence. K-Means++ overcomes this by selecting initial centroids in a way that maximizes their spread, reducing the likelihood of poor initial cluster assignments and speeding up the convergence process. In the K-Means++ initialization process, the first centroid is chosen randomly from the dataset, and each subsequent centroid is selected with a probability proportional to its distance from the existing centroids. This initialization method results in better initial centroids, leading to more accurate clustering outcomes. The probability formula: $p = \frac{D(x)^2}{\sum_{x \in X} D(x)^2}$. Where X is the dataset, and $D(x)$ is the distance from point x to the nearest selected cluster center.

After applying K-Means++ for centroid initialization, we repeated the clustering process on the same synthetic dataset with 3000 samples and four clusters. As with the standard K-Means, the algorithm successfully assigned data points to one of the four clusters and computed the centroids. The clustering results were again visualized in 2D space, where the data points were color-coded based on their cluster assignments, and the centroids were marked with large black stars. The K-Means++ algorithm demonstrated improved clustering performance, with the centroids being more accurately positioned and the algorithm converging more quickly. This

improvement shows that K-Means++ can effectively handle datasets where proper initialization is critical, and it provides more reliable results, particularly in scenarios involving complex or noisy data.

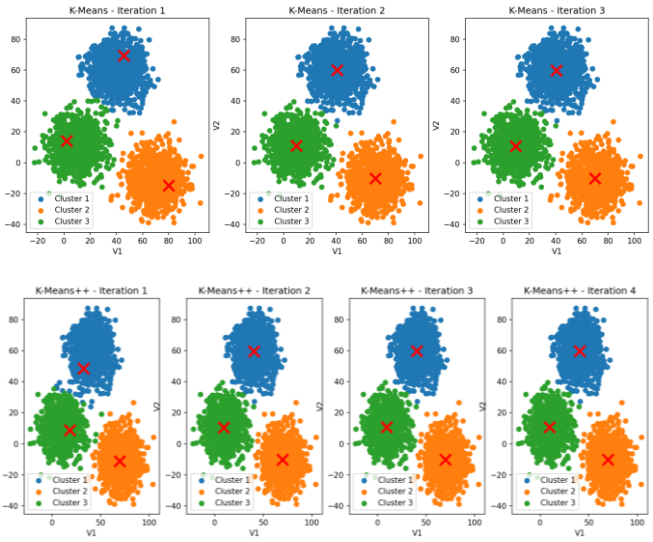


Fig. 2. the iterative process of the kmeans++ algorithm

Algorithm 1: Initial Cluster Center Selection (k-means++)

Input: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters K

Output: Set of K initial cluster centers C= $\{c_1, c_2, \ldots, c_k\}$

1. Randomly select the first cluster center c_1 from X.
2. For $k = 2$ to K, do:
   Compute the distance $D(x)$ from each point $x \in X$ to the nearest selected cluster center:
   $$D(x) = min_{i=1,\ldots,k-1}\|x - c_i\|$$
3. Compute the probability $p(x)$ for each point x to be selected as the next cluster center:
   $$p(x) = \frac{D(x)^2}{\sum_{x' \in X} D(x')^2}$$
4. Select the next cluster center c_k from X based on the probability distribution $p(x)$.
5. Return the set of K selected cluster centers C.

- ***Performance comparison of K-means & K-means++***

In the experimental analysis of clustering with two clusters (K=2), both K-Means and K-Means++ algorithms were observed across multiple iterations. For K-Means, the initial iteration (Iteration 1) demonstrated a random and rough division of data points into two clusters, with initial cluster centers selected arbitrarily, leading to potential misallocations. By the second iteration (Iteration 2), the cluster centers had adjusted to the data distribution, refining the cluster boundaries and leading to a more rational distribution of data points. The third iteration (Iteration 3) saw further optimization of the cluster centers, resulting in more compact intra-cluster data points and enhanced inter-cluster separation. The process reached stability by the fourth iteration (Iteration 4), with no significant changes in data point allocation or cluster center movement. In contrast, K-Means++ showed a more strategic initialization in the first iteration (Iteration 1), with a relatively better initial partitioning of the two clusters. The second iteration (Iteration 2) involved fine-tuning of the cluster centers, leading to even tighter clustering of data points. The third iteration (Iteration 3) brought further optimization, with clearer boundaries between the clusters. Stability was achieved by the fourth and fifth iterations (Iteration 4 and 5), with no notable changes in the allocation of data points or the positions of the cluster centers, indicating that the clustering process had converged.[6]

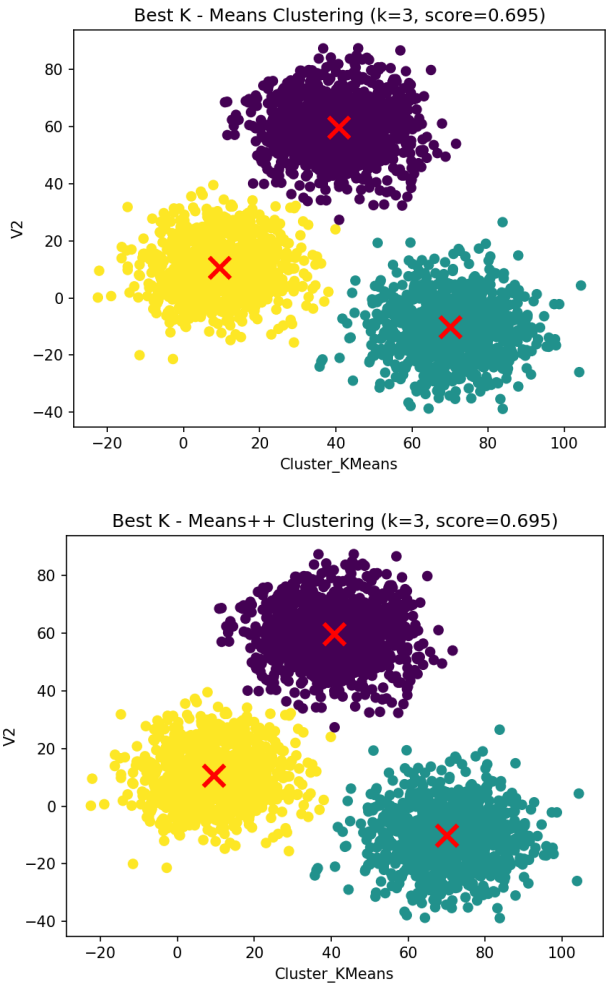


Fig. 3. Clustering results of kmeans&kmeans++

In the overall conclusion of our experimental analysis, we observed that K-Means++ demonstrated a faster convergence rate to stable clustering outcomes due to its more strategic initial selection of cluster centers, as opposed to the random initialization of K-Means, which might require additional iterations to achieve stability. In terms of clustering quality, K-Means++ consistently yielded higher-quality results within the same number of iterations, with data points being more tightly clustered within each cluster and more distinct separation between clusters, a benefit attributed to its initialization method that aims to place initial centers at greater distances from one another, thus avoiding poor initial configurations. Regarding stability, K-Means++ exhibited more consistent clustering results throughout the iterative process, with minimal changes after each iteration, whereas K-Means showed potentially significant fluctuations in the early stages of iteration. These findings underscore the robustness and efficiency of K-Means++ in clustering tasks.

We employed the K-Means and K-Means++ clustering algorithms to evaluate their performance across different values of k, utilizing the Silhouette Score as our primary assessment metric. The Silhouette Score ranges from -1 to 1, with values closer to 1 indicating superior clustering efficiency, where data points are closely packed within their clusters and distinctly separated from other clusters. Our analysis revealed that for k=2, K-Means demonstrated a marginally better Silhouette Score (0.5424) compared to K-Means++ (0.5093), suggesting a slightly more defined cluster distribution and separation. However, this difference was not substantial. When k was increased to 3, both K-Means and K-Means++ achieved an identical Silhouette Score of 0.6946, indicating that the algorithms performed equally well, with data points naturally forming three well-delineated clusters. At k=4, K-Means again showed a higher Silhouette Score (0.5648) than K-Means++ (0.5592), with the random initialization of K-Means++ (0.5390) performing slightly worse. For k values of 5 and above, K-Means maintained a relatively stable Silhouette Score around 0.5, while K-Means++ experienced a gradual decline, particularly noticeable at k=6 (0.3131) and k=7 (0.3157). This trend could be indicative of K-Means++'s increased sensitivity to initialization as the number of clusters grows, possibly leading to less optimal local minima and, in turn, affecting the clustering quality.

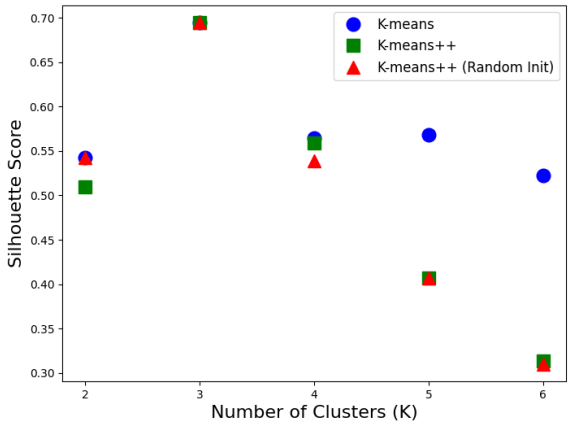


Fig. 4. Silhouette Score for Different K Values and Initialization Methods

In conclusion, the optimal number of clusters, k=3, yielded the highest Silhouette Score for both algorithms, signifying the best balance between intra-cluster compactness and inter-cluster separation. However, the choice of k should also consider business requirements and prior knowledge of data distribution. Despite K-Means++'s potential for instability with random initialization, it did not significantly outperform K-Means in this dataset, highlighting the importance of algorithm selection in the context of specific data characteristics and business logic.

- ***Density-based clustering DBSCAN***

The DBSCAN algorithm[2] has good clustering results in applications and is a typical representative of density algorithms. The algorithm has the feature of discovering arbitrary shape classes in the data set. The DBSCAN algorithm requires the user to set the global constant parameters Neighborhood Distance and Threshold before running. The neighborhood distance sets the radius of the neighborhood range of the sample points. The threshold is set by varying the minimum number of sample points within the radius of the neighborhood range that are marked as core points. It is important to note that both the neighborhood distance and the threshold are constants that are set before the start of the program and do not change after they are set.The DBSCAN algorithm is insensitive to noise points and can be applied to datasets that contain noise points, which it identifies and excludes from the clustering results. For each class in the clustering result, the in-class density is higher than the class edge density, and the density of noise points is lower than the edge density. Based on the data distribution characteristics, the algorithm uses the density differences to identify regions of different densities and label the clustering results.

The steps of the algorithm are as follows:

Algorithm 2: Density-Based Clustering (DBSCAN)

Input: Minimum radius ϵ,
Minimum density threshold min PTs

Output: Clustering results and noise points

1. Initialize Data Structures:
   Initialize an empty list *pointList.*
   Initialize an empty list *resultList.*
2. Sequentially read the data file and store each data point into *pointList.*
3. For each point p in *pointList*:
   Calculate the distance between point p and all other points in *pointList*.
   Initialize an empty temporary list *tmpLst*.
   For each point q in *pointList*:
   If the distance between p and q is less than ϵ
   Add point q to *tmpLst*.
   If the number of points in *tmpLst* is greater than min PTs:
   Mark point p and all points in *tmpLst* as grouped.
   Add the group of points from *tmpLst* as a cluster to *resultList*
4. For each cluster in *resultList*:
   Compare the core points between clusters.
   If any core points are shared between clusters:
   Merge the two clusters into a new cluster
5. Return the final clustering results.

In our analysis of the DBSCAN (Density - Based Spatial Clustering of Applications with Noise) algorithm, we firstly focused on the impact of the *minPts* parameter on clustering outcomes while keeping *eps* (the neighborhood radius) constant at 5. The *minPts* parameter, which dictates the minimum number of points required to form a dense region, significantly influences the clustering results. When *minPts* is low, clusters are more readily formed, but this may also lead to the inclusion of noise points that should otherwise be classified as outliers. Conversely, a higher *minPts* value imposes stricter conditions for cluster formation, potentially misclassifying some points that should belong to a cluster as noise. Specifically, with *minPts* = 5, the data was divided into four clusters (blue, orange, green, yellow) and several noise points (red), as shown in the first figure. These clusters were relatively compact with a high density of points, aligning with DBSCAN's density - based clustering characteristic, and noise points were sparsely distributed at the peripheries of the clusters. Increasing *minPts* to 10, as depicted in the second figure, resulted in three clusters (blue, orange, green) and some noise points (red). This increase in *minPts* led to a

coarser division of data, with some points that were part of distinct clusters at *minPts* = 5 now being merged into larger clusters. For instance, the yellow cluster in the first figure has merged with another cluster at *minPts* = 10, resulting in fewer clusters.

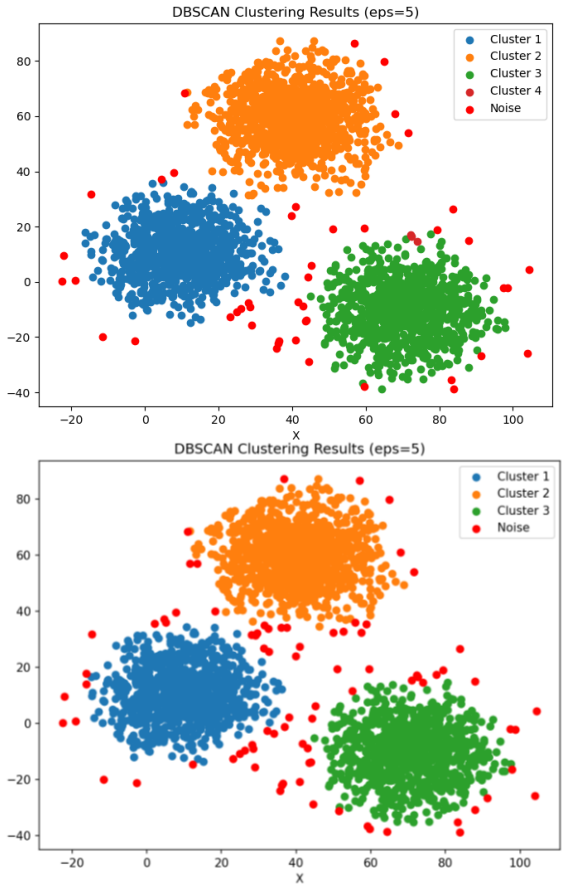


Fig. 5. This picture is the clustering result of DBscan (minpts=5, 10)

Comparatively, raising *minPts* from 5 to 10 could decrease the number of clusters, altering their shapes and sizes. This is because a higher *minPts* value makes cluster formation more stringent, causing areas with relatively lower density not to meet the criteria for cluster formation at *minPts* = 10, hence being merged with other clusters or being classified as noise. Additionally, increasing *minPts* may result in more compact and pure clusters, as only regions with sufficiently high densit

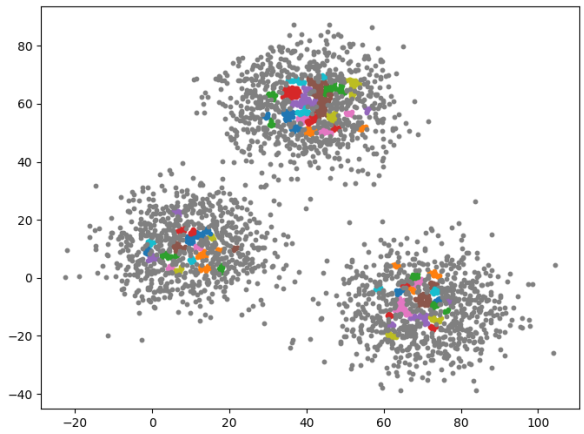
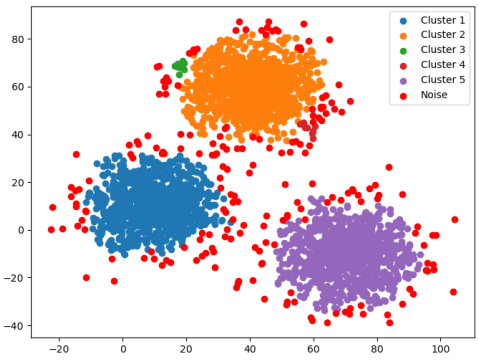


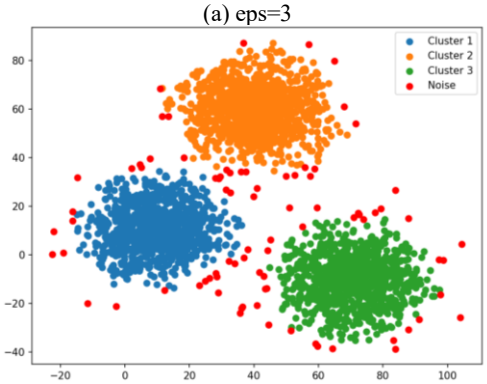
Fig. 6. This figure shows the clustering results of dbscan(eps=1)

The following figures show the DBSCAN clustering results with different values of eps. As eps increases from 1 to 100, significant changes can be observed in the clustering patterns. In the figures where eps = 1, the data points are scattered with numerous small clusters, each represented by different colors. As eps increases to 3, 5, 7, and 10, the clusters start to merge, and the number of distinct clusters decreases. Specifically, at eps = 10, there are mainly three large clusters with a few noise points (represented in red). When eps reaches 100, all the data points are grouped into a single cluster, indicating that a large eps value leads to over - clustering. The quantitative analysis further supports these observations. At eps = 10, the cluster sizes are {1: 2998, - 1: 2}, with a mean standard deviation of 2.9614430147049173, and two noise points. At eps = 100, the cluster size is {1: 3000}, with a mean standard deviation of 2.949838289578235, and no noise points. The community difference is 0.0 in both cases, suggesting that the composition of the clusters remains relatively stable despite the increase in eps.

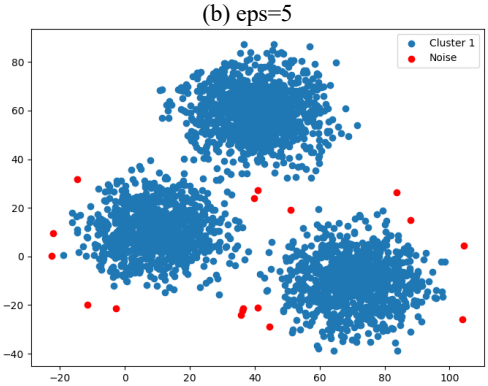
In conclusion, the DBSCAN clustering results are highly sensitive to the eps parameter. A smaller eps value results in more fragmented clusters, while a larger eps value leads to fewer and larger clusters, potentially resulting in over - clustering.



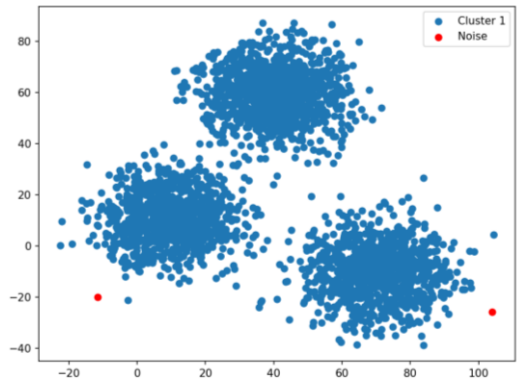
(a) eps=3



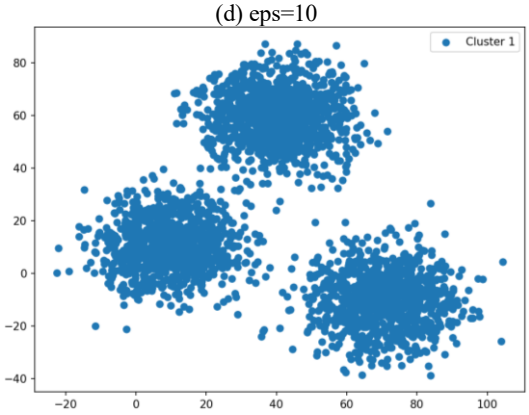
(b) eps=5



(c) eps=7



(d) eps=10



(e) eps=100

Fig. 7. This figure shows the clustering results for different eps values

- ***CLARA***

The CLARA (Clustering Algorithm based on Random sampling) algorithm[3] is a variant of the k-means clustering method, designed to handle large datasets more efficiently by using random sampling. It combines the strengths of the k-means algorithm with sampling techniques to provide better scalability and performance for clustering large datasets.

Algorithm 3: CLARA

Input: Dataset with n objects
Number of clusters k
Number of iterations v (typically v =5)

Output: Best set k of center points and clustering results

1. For each iteration $i = 1$ to v:
2. Randomly draw a sample of $(40 + 2k)$ objects from the entire dataset.
   Use the cost-replacement method to identify k center points from the sample.
3. For each object Q_j in the entire dataset, determine which of the k center points is most similar to Q_j.
4. Calculate the average dissimilarity of the clustering obtained in **step 3**.
5. If this dissimilarity is less than the current minimum value:
   Replace the current minimum dissimilarity with this new value.
   Retain the k center points from **step 2** as the best set of center points found so far.
6. Return the best set of k center points and the corresponding clustering results.

The CLARA does not identify representative objects from the entire dataset but rather discovers them from a sample of the dataset. It then uses a cost-replacement calculation method to select the sample's center points. If the sample is randomly selected, it should be representative of the original dataset. However, it is not guaranteed that the sampling method is truly random, and the quality of clustering is measured by the AAaverage dissimilarity of all objects in the entire dataset, not just the average dissimilarity of the objects in the sample. To better approximate this, CLARA draws multiple samples and takes the best clustering as the output. To improve the efficiency of spatial clustering, the CLARA algorithm randomly selects a number of spatial data points equal to (40+2k) for spatial clustering, where k is the desired number of clusters. For this, v samples can be drawn, with the best clustering result taken as the output.

This approach ensures that the clustering results are not only representative of the sample but also reflective of the overall dataset, enhancing the robustness and accuracy of the clustering process.

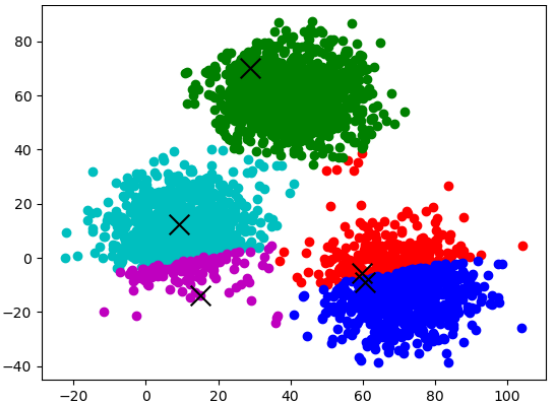


Fig. 8. CLARA Clustering Results

The number of clusters (k) is set to 5. The clustering results under this parameter can fit the distribution of the data well. If the k value is too small, the data may be over-merged; if the k value is too large, the data may be over-segmented. The setting of the sample size (40 + 2k) can fully capture the distribution information of the data while ensuring the efficiency of the algorithm. The number of samplings is 5. Multiple samplings help reduce the errors caused by random sampling and ensure the accuracy of the clustering results. From the CLARA clustering result diagram, we can see that the data is divided into five clusters. These clusters form relatively concentrated areas in two-dimensional space and are marked by cluster centers (black crosses in the figure). This shows that the CLARA algorithm can effectively identify the potential grouping structure in the data set. From the perspective of cluster compactness, the data points in each cluster are relatively tightly clustered, which reflects the effectiveness of the algorithm in determining the structure within the cluster and can cluster similar data points together. In terms of cluster separation, there are relatively obvious intervals between different clusters. Although there may be some ambiguity of data points in the boundary areas of clusters, the clusters are generally distinguished from each other in space, which reflects that the algorithm has a good ability to distinguish different clusters.

- ***Connection-based clustering Hierarchical clustering***

Hierarchical clustering[5] connects objects based on distance to form clusters, roughly describing the groups by the maximum distance required to connect the parts. At different distances, different clusters are formed, which can be presented using a dendrogram (tree-like diagram). This also explains the origin of "hierarchical clustering," where these algorithms do not provide a single partition of the dataset but offer an extensive hierarchical structure of clusters, with these clusters merging at certain distances. In the dendrogram, the y-axis marks the distance at which clusters are merged, while objects are placed along the x-axis to prevent mixing of clusters.

Clustering based on linkage is a group of methods that differ in the way they calculate distance. Popular choices include "single linkage clustering" (minimum object distance), "complete linkage clustering" (maximum object distance), and so on. Furthermore, hierarchical clustering can be agglomerative (starting from individual elements and aggregating them into clusters) or divisive (starting from the full dataset and dividing it into partitions).

The divisive method, in divisive or top-down clustering, involves assigning all observations to a single cluster and then splitting this cluster into at least two similar clusters. This process is then recursively applied to each cluster until each observation is in its own cluster. In some cases, divisive algorithms can produce higher accuracy than agglomerative algorithms but are conceptually more complex.

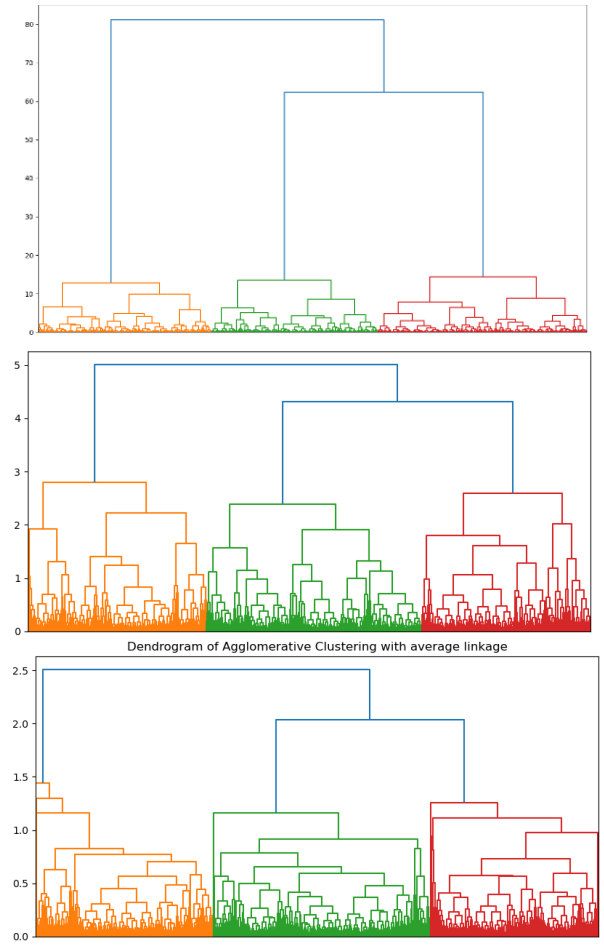


Fig. 9. Dendrogram of Agglomerative Clustering with average linkage

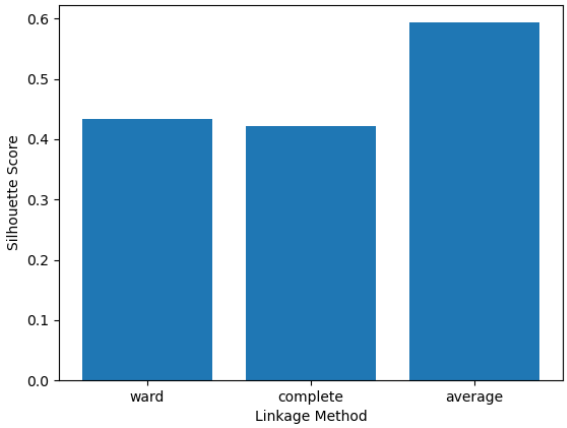


Fig. 10. Effect of Linkage Method on Agglomerative Clustering Performance

The experimental results of agglomerative hierarchical clustering using different linkage methods—Ward, Complete, and Average—are illustrated through dendrograms and compared based on Silhouette Scores. The dendrograms reveal distinct characteristics for each method: the Ward linkage method demonstrates a higher tree height, indicating larger distances at which clusters merge, with a tendency to form clusters of more uniform size. The Complete linkage method shows a lower tree height, suggesting tighter clustering as it merges clusters based on the maximum distance between points, leading to a faster merging process and shorter branches in the dendrogram. The Average linkage method presents a tree height between that of Ward and Complete, considering the average distance between all points in clusters, resulting in moderately shaped clusters. Performance comparison based on the Silhouette Score indicates that the Average linkage method yields the highest score, suggesting optimal intra-cluster compactness and inter-cluster separation for this dataset. The Ward linkage method achieves a moderate Silhouette Score, reflecting a balanced clustering performance. In contrast, the Complete linkage method has a relatively lower Silhouette Score, which may imply less desirable compactness or separation in comparison to the other methods.

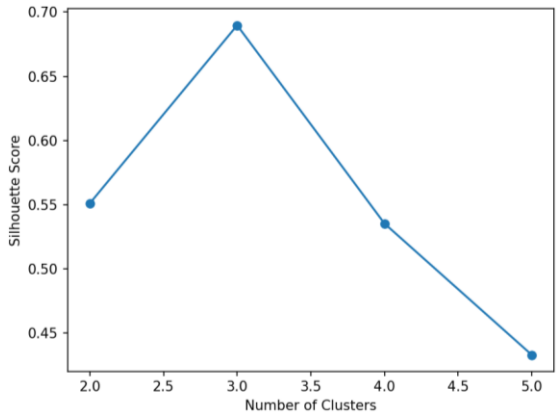


Fig. 11. Effect of Number of Clusters on Divisive Clustering Performance

In conclusion, for this dataset, the Average linkage method outperforms the others, as indicated by the Silhouette Score, offering a better balance between cluster compactness and separation. The Ward linkage method provides a moderate clustering outcome, while the Complete linkage method is somewhat less suitable for this dataset, potentially due to its

inferior performance in cluster compactness or separation. These findings can guide the selection of appropriate clustering methods and parameters to achieve enhanced clustering effectiveness in practical applications.

The chart presents the Silhouette Score, a metric for cluster quality ranging from -1 to 1, on the y-axis. The x-axis represents the number of clusters, ranging from 2.0 to 5.0. Analysis indicates that the peak Silhouette Score, close to 0.70, occurs when the data is divided into three clusters, suggesting optimal divisive clustering performance at this point. As the number of clusters increases from three to five, the Silhouette Score gradually decreases, indicating a decline in clustering quality. At two clusters, the Silhouette Score is moderate, around 0.55.

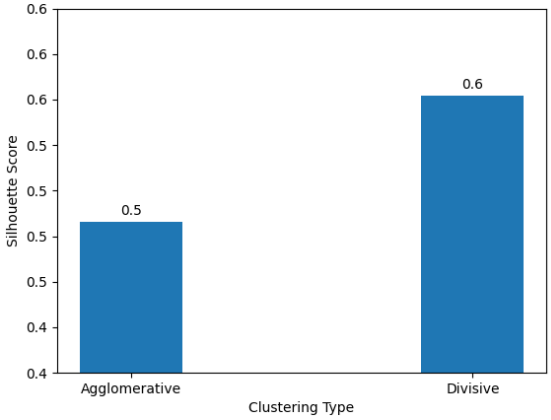


Fig. 12. Performance Comparison between Agglomerative and Divisive Hierarchical Clustering

The chart compares the average Silhouette Score on the y-axis against clustering types—Agglomerative and Divisive—on the x-axis. The analysis reveals that both agglomerative and divisive hierarchical clustering methods yield similar average Silhouette Scores, approximately 0.5, suggesting equivalent overall performance for these methods under the specific dataset and experimental conditions.

In summary, for this dataset, the best clustering outcome is achieved by dividing the data into three clusters using divisive clustering. There is no significant difference in overall performance between agglomerative and divisive hierarchical clustering methods. These results can guide the selection of clustering strategies to optimize clustering effectiveness in practical applications.

- ***Distribution-based clustering GMM***

A Gaussian mixture model (GMM)[4] is a parametric probability density function expressed as a weighted sum of the densities of the Gaussian components. GMM is commonly used as a parametric model for the probability distributions of continuous measurements or features in a biometric system, such as the spectral features associated with vocal tracts in a speaker recognition system. The GMM parameters are estimated from the training data using an iterative Expectation Maximization (EM) algorithm, or from a well-trained The GMM parameters are estimated from training data using the iterative expectation maximization (EM) algorithm or from well-trained a priori models using the maximum a posteriori (MAP).

Algorithm 4: Gaussian Mixture Model (GMM)

Input: The number of components of the GMM and model configuration (e.g., form of covariance matrix, whether parameters are shared, etc.)

Output: Clustering results

1. Determine the number of components of the GMM and the model configuration based on the amount of data and application requirements.
2. Estimate model parameters using Maximum Likelihood (ML) or Maximum A Posteriori (MAP) estimation methods.
3. For each data point:
   calculate the posterior probabilityPr $(i|x,\lambda)$
4. Assign the data point to the corresponding cluster based on the posterior probability, i.e., assign the data point to the cluster corresponding to the component with the largest posterior probability.
5. Return the clustering results.

In the experimental results of the clustering performance, the Silhouette Score, which is a measure of cluster quality ranging from -1 to 1, reaches its peak when the number of clusters (referred to as 'components' in the original text) is set to 3, with a score close to 0.70. This indicates that the clustering effect is optimal at this point. As the number of clusters increases from 3 to 6, the Silhouette Score gradually decreases, signifying a deterioration in clustering quality. When the number of clusters is 2, the Silhouette Score is at a moderate level, approximately 0.55, suggesting a fair balance between intra-cluster cohesion and inter-cluster separation.

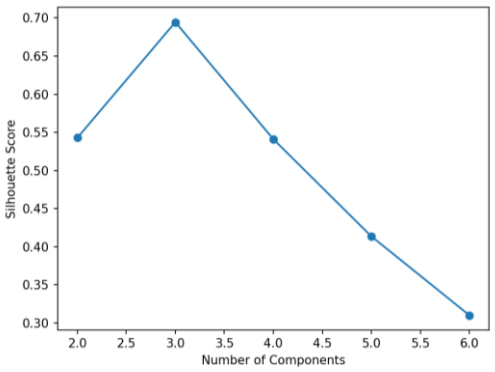


Fig. 13. Effect of Number of Components on Clustering

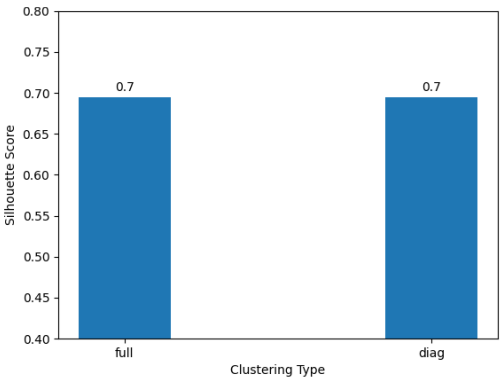


Fig. 14. Effect of Covariance Type on Clustering

In the context of the Gaussian Mixture Model (GMM), "full covariance" refers to a complete and non-diagonal covariance matrix that provides a flexible representation of the data distribution. The covariance matrix, denoted by Σ, captures the variance within each dimension of the data and the correlation between different dimensions.

For a multivariate Gaussian distribution, the probability density function is given by:

$$f(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

where x is the data vector, μ is the mean vector, and Σ is the covariance matrix.

In the case of "full covariance," all elements of the covariance matrix can be non-zero, indicating that it accounts for the variance within each dimension and the correlations between dimensions. For a two-dimensional dataset, the full covariance matrix takes the form: $\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$, where $\sigma_{12} = \sigma_{21}$ represents the covariance between the two dimensions.

Comparison with Other Types of Covariance:

Diagonal Covariance: In a diagonal covariance matrix, off-diagonal elements are zero ($\sigma_{12} = 0 \; for \; i \neq j$). This assumes independence between dimensions, considering only the variance within each dimension.

Spherical Covariance: Under spherical covariance, the covariance matrix is diagonal with equal elements on the diagonal, $\Sigma = \sigma^2 I$, where σ^2 is a scalar and I is the identity matrix. This assumes equal variance across dimensions and independence between them.

In the GMM, selecting different types of covariance can affect the model's ability to fit the data and its complexity. Full covariance can adapt more flexibly to complex data distributions, but with more parameters, it may lead to overfitting.

The experimental outcomes within the Gaussian Mixture Model (GMM) demonstrate that employing both "full" and "diag" covariance types yields Silhouette Scores nearing 0.7, signifying no substantial disparity in clustering efficacy. This consistency in performance across both models is visually substantiated through the dendrograms and Silhouette Score comparisons, which collectively illustrate that each covariance approach effectively delineates clusters and ensures distinct cluster segregation. Consequently, these insights can inform the selection of an appropriate covariance type in GMM, balancing the enhancement of clustering precision against the backdrop of model intricacy.

- ***Overall performance evaluation and solution selection***

The Sum of Squared Errors (SSE) and Adjusted Rand Index (ARI) are two common metrics for evaluating clustering results in order to facilitate the selection of a suitable solution, and we have already used SSE in some of the performance evaluations in the above section. The SSE is easy to compute, and is directly correlated with the distance of the data points from the respective clustering centers. It is a natural metric for distance-based clustering algorithms and effectively measures the compactness of the clusters. However, SSE does not take into account the number of clusters and the distribution structure of the data. Blindly minimizing SSE may lead to overfitting, and in extreme cases, setting the number of clusters to the number of data points leads to zero SSE, which is clearly not a meaningful clustering. Furthermore, for non-spherical data distributions, SSE may not accurately reflect the true quality of the clusters.

In the following, we will choose ARI as an important criterion for selecting a suitable algorithm on the same dataset, ARI is not affected by the number of clusters and the size of the dataset, and is an important tool for comparing the performance of different clustering methods.

Adjusted rand index (ARI): Provides a score of similarity between two different clustering results of the same dataset. For a given set S consisting of a elements and r subsets and two partitions $Y = (Y_1, Y_2, ..., Y_r)$ and $X = (X_1, X_2, ..., X_r)$, the overlap between the two partitions can be summarized as follows:

	Y_1	Y_2	…	Y_r	Sums
X_1	a_{11}	a_{12}	…	a_{1r}	s_1
X_2	a_{21}	a_{22}	…	a_{2r}	s_2
⋮	⋮	⋮		⋮	⋮
X_r	a_{r1}	a_{r2}		a_{rr}	s_r
Sums	s_1	s_2	…	s_r	

The adjusted rand index (ARI) is then calculated by Equation[4]. The ARI score is adjusted to have values between 0 and 1 to represent scores for random and perfect clustering, respectively.

IV. Evaluation

Different clustering algorithms have their own advantages and disadvantages in terms of time and space complexity due to their different principles.

- How to choose the most appropriate algorithm

In our experiment, six clustering algorithms, KMeans, KMeans++, DBSCAN, GMM, Hierarchical and CLARA, are applied to the same dataset, and the consistency of the cluste ring results is measured by Adjusted Rand Index (ARI). The results show that the ARI value of KMeans and KMeans++ i s 1.0, which indicates that the clustering results of the two al gorithms are identical on this dataset, and can be regarded as equivalent choices when largescale datasets and efficient and fast clustering are required.The ARI value of KMeans and G MM is 0.9959, and the clustering results are highly consisten t, although GMM based on the probabilistic model can provi de the probability of attribution of the data points, the clusteri ng structure of the two algorithms is highly consistent on this dataset. The clustering structure is similar between the two o n this dataset.The ARI value of KMeans and Hierarchical is 0.9882, and the clustering results are also very close, and if t he requirement of hierarchical information is not high, KMea ns or KMeans++ can be chosen in preference.The ARI value of DBSCAN and the other methods are all 0.0, because it is a densitybased clustering method, which identifies the noise p oints and excludes them, the The clustering structure is differ ent from other hard clustering methods, but it has an advanta ge in dealing with noisy data.The ARI value of GMM and Hi erarchical is 0.9922, and the consistency of their clustering re

sults is high, GMM can provide more finegrained probability information, and Hierarchical can present the hierarchical str ucture of the clusters, so if it does not pay attention to the hie rarchical information, GMM is a good choice. To summarize , on the noisy standard dataset, KMeans++ or GMM is the pr eferred choice because of its high consistency of clustering re sults and better computational efficiency; when there is noise or outliers, DBSCAN is more suitable; if we want to constru ct the clustering hierarchy to deeply understand the data distr ibution, Hierarchical is a good choice.

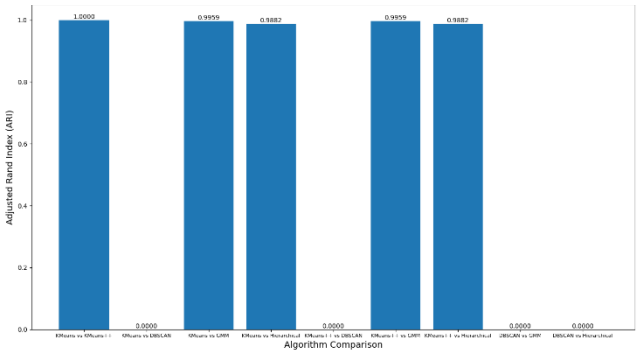


Fig. 15. ARI Comparison among Different Clustering Algorithms

It should be noted that CLARA is not involved in ARI value comparison in this experiment. This is because the clus tering mechanism of CLARA is different from other algorith ms in that it adopts a sampling strategy and clusters based on data subsets. The number of labels in its clustering result is d etermined by the subset size, which is inconsistent with the n umber of labels produced by algorithms based on clustering t he complete dataset, leading to the inability to assess its cons istency with other algorithms directly through the ARI value.

- ***Time complexity analysis and space complexity analysis***

K-Means and K-Means++: The core of these two algorithms is to iteratively assign data points to different clusters to minimize the sum of the distances from the data points to the cluster centers. In terms of time complexity, each iteration requires calculating the distance from all data points (n) to each cluster center (k), with a time complexity of O(nk) per iteration. Typically, the number of iterations t is relatively stable, so the overall time complexity is O(nkt). K-Means++ mainly optimizes the initialization of cluster centers, with a time complexity of O(nk) for the initialization phase, but it does not change the overall order of time complexity. In terms of space complexity, only the data points and cluster centers need to be stored, which is O($n + k$), as the main storage is for the feature values of the n data points and the coordinates of the k cluster centers.

DBSCAN: It divides clusters based on the density of data points and first constructs the neighborhood relationships between data points. If the distance between each pair of data points is calculated directly, the time complexity is as high as O(n^2), where n is the number of data points. In the subsequent process of finding connected components and forming clusters, the time consumption is closely related to the density of the data points. If a spatial index structure is used to optimize neighborhood queries, the time complexity can be reduced to O(nlogn), but in the worst-case scenario, it remains O(n^2). In terms of space complexity, at least O(n) space is needed to store the data points themselves. If a complex spatial index is used, such as a tree-based structure, the space complexity will increase to O(nlogn) because additional information such as index nodes needs to be stored.

Gaussian Mixture Model (GMM): The algorithm assumes that the data is composed of a mixture of multiple Gaussian distributions and uses the Expectation-Maximization (EM) algorithm to solve iteratively. In each iteration, the E-step calculates the posterior probability of n data points belonging to k Gaussian components, involving the calculation of the probability density of the multivariate Gaussian distribution, with a time complexity of O (nkd) , where d is the dimensionality of the data; the M-step updates the parameters of the Gaussian components based on the results of the E-step, with the same complexity of O(nkd). Assuming the number of iterations is t, the overall time complexity is approximately O($nkdt$). In terms of space complexity, it is necessary to store the parameters of each Gaussian component. For k components and d-dimensional data, the covariance matrix requires kd^2 storage, and it is also necessary to store the posterior probabilities of each component corresponding to the n data points, requiring nk space, with a total of O(nk + kd^2).

CLARA: Designed for large-scale data, it uses a sampling strategy. Let the number of samplings be s, and the size of each sample be m. Running a clustering algorithm similar to K-Means on the sampled data (assuming the number of iterations on the sample set is t, with a time complexity of O(mkt)), the overall time complexity of CLARA is approximately O($smkt$). It can be seen that the sampling size and the number of times play a key role in controlling the time cost. In terms of space complexity, in addition to the original data occupying O(n) space, each sampling also requires additional O(m) space to store the sampled data and cluster centers, and other temporary information. After s samplings, the total space complexity is roughly O(n + sm).

Hierarchical Clustering (Agglomerative Clustering): It constructs a hierarchical structure of clusters. The common agglomerative method starts with individual data points as initial clusters and continuously merges close clusters. Initially, constructing a distance matrix between each pair of data points has a time complexity of O(n^2), where n is the number of data points. In subsequent rounds of cluster merging, updating the distance matrix and finding the closest pair of clusters, if not optimized, will still require traversing the matrix, with the worst-case overall time complexity reaching O(n^3); if optimization methods such as priority queues or approximation algorithms are used, the complexity can be reduced to close to O(n^2 logn). In terms of space complexity, the main consumption is in storing the distance matrix, which reaches O(n^2). Although there are some methods of compressed storage or approximate calculation, the storage of the distance matrix for high-dimensional data is still a difficult issue in conventional implementations.

In summary, different clustering algorithms have their own advantages and disadvantages in terms of time and space complexity due to their different principles. The K-Means series is simple and direct, suitable for small to medium-sized and more regular data; DBSCAN is good at handling unevenly dense data but the computational cost is uncontrollable; GMM, based on probability models, is good at handling complex distributions, but parameter estimation is complex for high-dimensional data; CLARA finds a balance in large datasets through sampling; Hierarchical Clustering provides a hierarchy of clusters but has high space requirements. In

practical applications, it is necessary to fully consider factors such as data characteristics and computational resources, choose the appropriate algorithm, and if necessary, combine optimization strategies to achieve efficient clustering.

V. Conclusion

- ***K-Means***

K-Means requires the specification of the number of clusters in advance and forms clusters that are typically spherical or convex in shape. It is not well-suited for handling outlier data, as outliers can significantly affect the cluster centroids. Additionally, K-Means only requires one parameter, the number of clusters, for model training. However, the need to specify the number of clusters can be a limitation, especially when the optimal number is unknown.

- ***K-Means++***

K-Means++ initializes cluster centers differently from the traditional K-Means algorithm by selecting initial centroids in a more strategic way to improve clustering results and reduce the likelihood of poor convergence. However, like K-Means, it is still not suitable for handling outlier data, as outliers can influence the centroids. K-Means++ only requires one parameter for model training, which is the number of clusters, and it does not require the specification of the number of clusters upfront. This makes it more flexible in some cases, but the choice of the optimal number of clusters remains a challenge.

- ***DBSCAN***

DBSCAN can form clusters of any shape, offering significant flexibility compared to algorithms like K-Means. It is also effective at handling data with noise and outliers, as it classifies points that do not meet the density criteria as noise, rather than forcing them into a cluster. The algorithm requires two parameters for model training: *epsilon (ϵ)*, which defines the radius within which points are considered neighbors, and *minPts*, the minimum number of points required to form a dense cluster. Unlike some other clustering algorithms, DBSCAN does not require the specification of the number of clusters, making it more adaptable in situations where the number of clusters is unknown. However, it does rely on choosing appropriate values for ϵ and *minPts* to perform well.

- ***GMM***

Gaussian Mixture Models (GMM) are based on a probabilistic model and assume that the data is generated from a mixture of multiple Gaussian distributions. This approach allows GMM to handle complex data distributions more effectively than simpler clustering algorithms, making it useful for modeling real-world data that may not conform to spherical or convex shapes. However, GMM is sensitive to outliers, as they can distort the parameter estimation. The algorithm typically uses the Expectation-Maximization (EM) algorithm for training, which involves multiple parameters such as the mean, covariance, and mixture weight for each Gaussian component. While GMM is flexible, it requires the specification of the number of clusters (or Gaussian components) beforehand, which can be challenging if the optimal number of components is unknown.

- ***CLARA***

CLARA performs clustering by taking multiple samples from the data, making it capable of handling large-scale datasets efficiently. The algorithm works by clustering each sample individually and then combining the results, which allows it to scale well with large datasets. However, the sampling process can be affected by outliers, as they may influence the quality of the clusters in the sampled subsets. CLARA has parameters related to the sampling process, such as the sample size and the number of samples to draw. One of the key advantages of CLARA is that it does not require prior specification of the number of clusters, as the results naturally form a hierarchical structure, which can be interpreted at various levels of granularity.

- ***Hierarchical Clustering***

Hierarchical clustering includes both agglomerative and divisive methods, allowing it to form clusters with a wide variety of shapes, making it more flexible compared to other clustering algorithms. However, it is relatively sensitive to outliers, with its sensitivity depending on the chosen distance metric and the merge/split strategy. Additionally, hierarchical clustering can have high computational costs, particularly when calculating the distance matrix, which grows quadratically with the number of data points. Despite these challenges, hierarchical clustering remains a powerful tool for discovering clusters at different levels of granularity.

References

[1] Ahmed M, Seraj R, Islam S M S. The k-means algorithm: A comprehensive survey and performance evaluation[J]. Electronics, 2020, 9(8): 1295.

[2] Deng D. DBSCAN clustering algorithm based on density[C]//2020 7th international forum on electrical engineering and automation (IFEEA). IEEE, 2020: 949-953.

[3] Kassambara A. Practical guide to cluster analysis in R: Unsupervised machine learning[M]. Sthda, 2017.

[4] Su T, Dy J G. In search of deterministic methods for initializing K-means and Gaussian mixture clustering[J]. Intelligent Data Analysis, 2007, 11(4): 319-338.

[5] Murtagh F, Contreras P. Algorithms for hierarchical clustering: an overview[J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2012, 2(1): 86-97.

[6] Kapoor A, Singhal A. A comparative study of K-Means, K-Means++ and Fuzzy C-Means clustering algorithms[C]//2017 3rd international conference on computational intelligence & communication technology (CICT). IEEE, 2017: 1-6.