Cluster Analysis by *K-means* and *DBSCAN*

Siyuan Sun*

Abstract. K-means algorithm and DBSCAN algorithm are two important cluster methods in big data and data mining. In this report, I will use these two methods to study the data set released in class and complete the clustering operation.

Keywords. big data and data mining; cluster; K-means; DBSCAN

1. Introduction.

In todays information age, the amount of data being generated is growing at an unprecedented rate. This data is being produced by a variety of sources, including social media, mobile devices, and the Internet of Things (IoT). The sheer volume of data being generated has led to the development of new techniques for analyzing and understanding it. One such technique is clustering, which is used to group similar data points together.

Clustering is a popular approach in data mining and has been widely used in big data analysis. The goal of clustering is to divide data points into homogeneous groups such that the data points in the same group are as similar as possible and data points in different groups are as dissimilar as possible. Clustering is important in pattern recognition, machine learning, image analysis, information retrieval, and many other fields.

However, clustering big data presents a number of challenges. One of the biggest challenges is the sheer size of the data. Traditional clustering algorithms are not designed to handle datasets that are too large to fit into memory. In addition, the high dimensionality of big data makes it difficult to find meaningful clusters. Finally, the presence of noise and outliers in big data can make it difficult to identify true clusters.

To address these challenges, researchers have developed new clustering algorithms that are specifically designed for big data. These algorithms are designed to be scalable and efficient, and they can handle datasets that are too large to fit into memory. Some of the most popular clustering algorithms for big data include k-means, DBSCAN, and hierarchical clustering.

In conclusion, clustering is an important technique for analyzing big data. It is used to group similar data points together and can be used in a variety of applications, such as image segmentation, anomaly detection, and customer segmentation. Researchers have developed new clustering algorithms that are specifically designed for big data, such as k-means, DBSCAN, and hierarchical clustering. These algorithms are designed to be scalable and efficient, and they can handle datasets that are too large to fit into memory.

2. Related works.

2.1. K-means.

K-means is a popular clustering algorithm used in machine learning to group similar data points together. It is a partitional clustering algorithm that divides a set of n observations into k clusters based on similarity scores. The objective is to minimize the total variance of the k clusters.

The algorithm starts by randomly assigning each data point to an initial group and calculating the centroid for each one. A centroid is the center of the group. Then, the algorithm evaluates

* Siyuan Sun (E-mail:2468191255@qq.com) is with the Department of Cyberspace Security, Nanjing University of Posts and Telecommunication, China

each observation, assigning it to the closest cluster. The definition of closest is that the Euclidean distance between a data point and a groups centroid is shorter than the distances to the other centroids. When a cluster gains or loses a data point, the K-means algorithm recalculates its centroid. The algorithm repeats until it can no longer assign data points to a closer set.

K-means clustering is an unsupervised method because it starts without labels and then forms and labels groups itself. It is not a supervised learning method because it does not attempt to predict existing or known group labels. K-means clustering usage has grown recently thanks to machine learning taking off.

If you have a large dataset of observations, but there is no grouping information or labels for the data points, you can use K-means clustering to generate groups comprised of observations with similar characteristics. For example, if you have customer data, you might want to create sets of similar customers and then target each group with different types of marketing.

The flow diagram of K-means can be described as Algorithm 1:

Algorithm 1 Process of K-means Cluster

```
Input: Sample Set $D$ = $\{x_1, x_2, ..., x_N\}$,Number of Clusters $K$,Maximum iterations $m$
Output: Division of Clusters $C$ = $\{C_1, C_2, ..., C_k\}$
 1: for $a = 1 \to k$ do
 2:     $u_a \Leftarrow D$
 3: end for
 4: for $i = 1 \to k$ do
 5:     $C_i \Leftarrow \emptyset$
 6:     $\lambda = \|x_1 - u_i\|^2$
 7:     for $j = 1 \to m$ do
 8:         $d_{ij} \Leftarrow \|x_j - u_i\|^2$
 9:         if $d_{ij} \leq \lambda$ then
10:             $\lambda = d_{ij}$
11:         end if
12:     end for
13:     $C_{\lambda j} = C_{\lambda j} + u_j$
14: end for
15: for $i = 1 \to k$ do
16:     $\tilde{u_i} \Leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$
17:     if $(\tilde{u_i} \neq u_i)$ then
18:         $u_i \Leftarrow \tilde{u_i}$
19:     end if
20: end for
```

2.2. DBSCAN.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups together data points that are close to each other based on a specified distance and density threshold. It is particularly useful for identifying clusters of arbitrary shapes and for detecting noise in the data.

The algorithm starts by selecting a random point and finding all the points within a specified radius (eps) of that point. If the number of points within the radius is greater than or equal to a specified minimum number of points (min-samples), then a new cluster is formed. The algorithm then repeats this process for all the points in the cluster until no more points can be added. DBSCAN is an unsupervised learning method, which means it does not require any prior knowledge about the data. It is also robust to outliers and can handle noise in the data. One of the advantages of DBSCAN is that it can identify clusters of arbitrary shapes, unlike K-means clustering, which assumes that the clusters are spherical. Another advantage is that it does not require the number of clusters to be specified beforehand, unlike hierarchical clustering.

However, DBSCAN has some limitations. It is sensitive to the choice of parameters, such as eps and min-samples, and the results can vary depending on the choice of parameters. It can also be computationally expensive for large datasets.

In summary, DBSCAN is a powerful clustering algorithm that can identify clusters of arbitrary shapes and handle noise in the data. It is a useful tool for exploratory data analysis and can be used in a variety of applications, such as image segmentation, anomaly detection, and customer segmentation.

The flow diagram of DBSCAN can be described as Algorithm 2:

Algorithm 2 Process of DBSCAN Cluster

```
Input: Sample Set $D$ = $\{x_1, x_2, ..., x_N\}$,Neighborhood parameter $(e, MinPts)$,Maximum iterations $m$
Output: Division of Clusters $C$ = $\{C_1, C_2, ..., C_k\}$
 1: for $a = 1 \to k$ do
 2:     $\omega_a \Leftarrow \emptyset$
 3: end for
 4: for $i = 1 \to m$ do
 5:     $N_e(x_i) \Leftarrow x_i$
 6:     if $d_{ij} \geq MinPts$ then
 7:         $\omega = \omega + x_i$
 8:     end if
 9: end for
10: $k = 0$
11: $R = D$
12: for $\omega \neq \emptyset$ do
13:     $R_{OLD} \Leftarrow R$
14:     $Q \Leftarrow \omega, R = R - Q$
15:     $k = k + 1, C_k = R_{OLD} - R$
16:     $\Omega = \Omega - C_k$
17: end for
```

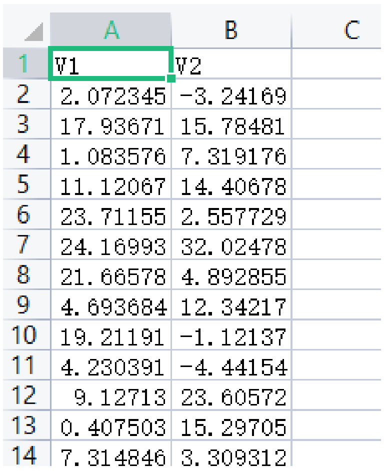
2.3. Others.

Hierarchical clustering is a method of clustering that involves creating a hierarchy of clusters. The algorithm starts by treating each data point as a separate cluster and then iteratively merges the two closest clusters until all the data points belong to a single cluster. Hierarchical clustering can be either agglomerative or divisive. Agglomerative clustering starts with the individual data points and merges them into larger and larger clusters. Divisive clustering starts with all the data points in a single cluster and then recursively splits them into smaller and smaller clusters.

K-DBSCAN is an improved version of DBSCAN that aims to accelerate the execution speed of the algorithm for big datasets. It does this by applying an initial grouping to the data using the K-means++ algorithm, and then performing clustering in each group separately using DBSCAN. The computational burden of DBSCAN execution is reduced, and the clustering execution speed is increased significantly. While Deep learning-based clustering approaches use deep learning techniques to cluster large datasets in bioinformatics. They employ dimensionality reduction methods such as principal component analysis (PCA), non-linear transformation, and spectral methods to reduce the complexity of the data before clustering. Density-based clustering algorithms are similar to DBSCAN in that they use density-based approaches to cluster data points. However, they differ in their approach to clustering. For example, OPTICS is a density-based algorithm that can identify clusters of varying densities and sizes. Fuzzy clustering is a clustering technique that allows data points to belong to multiple clusters with varying degrees of membership. It is particularly useful when dealing with datasets that have overlapping clusters or when there is uncertainty in the data. Subspace clustering is a clustering technique that identifies clusters in subspaces of the data. It is particularly useful when dealing with high-dimensional data, where traditional clustering algorithms may not be effective.

3. Problem and Solution.

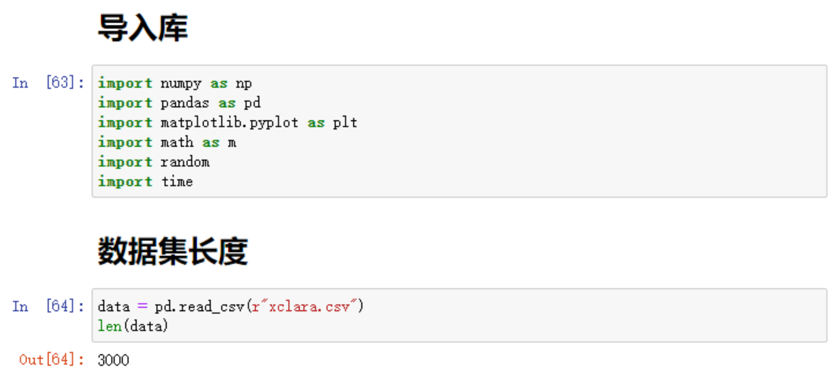
In Fig 1, it can be seen that the data set is divided into two columns with parameters V1 and V2 respectively. To process the dataset, we use the following python libraries, including NumPy, Pandas, Matlotlib, Math, Random, and Time as Fig 2 and get the length of the data set.



	A	B	C
1	V1	V2	
2	2.072345	-3.24169	
3	17.93671	15.78481	
4	1.083576	7.319176	
5	11.12067	14.40678	
6	23.71155	2.557729	
7	24.16993	32.02478	
8	21.66578	4.892855	
9	4.693684	12.34217	
10	19.21191	-1.12137	
11	4.230391	-4.44154	
12	9.12713	23.60572	
13	0.407503	15.29705	
14	7.314846	3.309312	

Fig. 1. Data set features

NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool. It provides data structures for efficiently storing and manipulating large datasets, and tools for working with structured data. While Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.



```
In [63]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import math as m
         import random
         import time
```

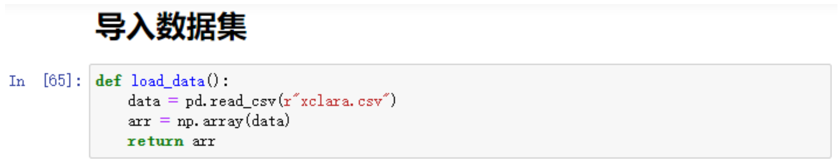
```
In [64]: data = pd.read_csv(r"xclara.csv")
         len(data)

Out[64]: 3000
```

Fig. 2. Code of preprocessing

It provides a variety of customizable plots, charts, and graphs for data visualization. Math is a built-in Python library that provides access to the mathematical functions defined by the C standard. Random is a built-in Python library that provides access to functions for generating random numbers. Finally, Time is a built-in Python library that provides various time-related functions.

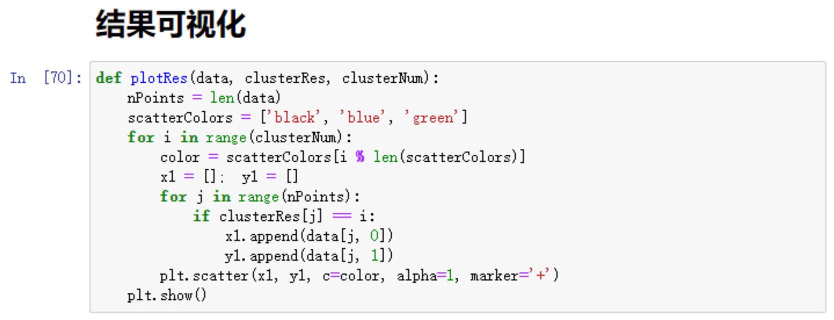
Define function load-data() to import and deal the element in data set by the function in Pandas library and NumPy library as Fig 3.



```
In [65]: def load_data():
             data = pd.read_csv(r"xclara.csv")
             arr = np.array(data)
             return arr
```

Fig. 3. Code of importing data set

Also, we need to define a function plotRes() for visualizing the results as Fig 4 as below:



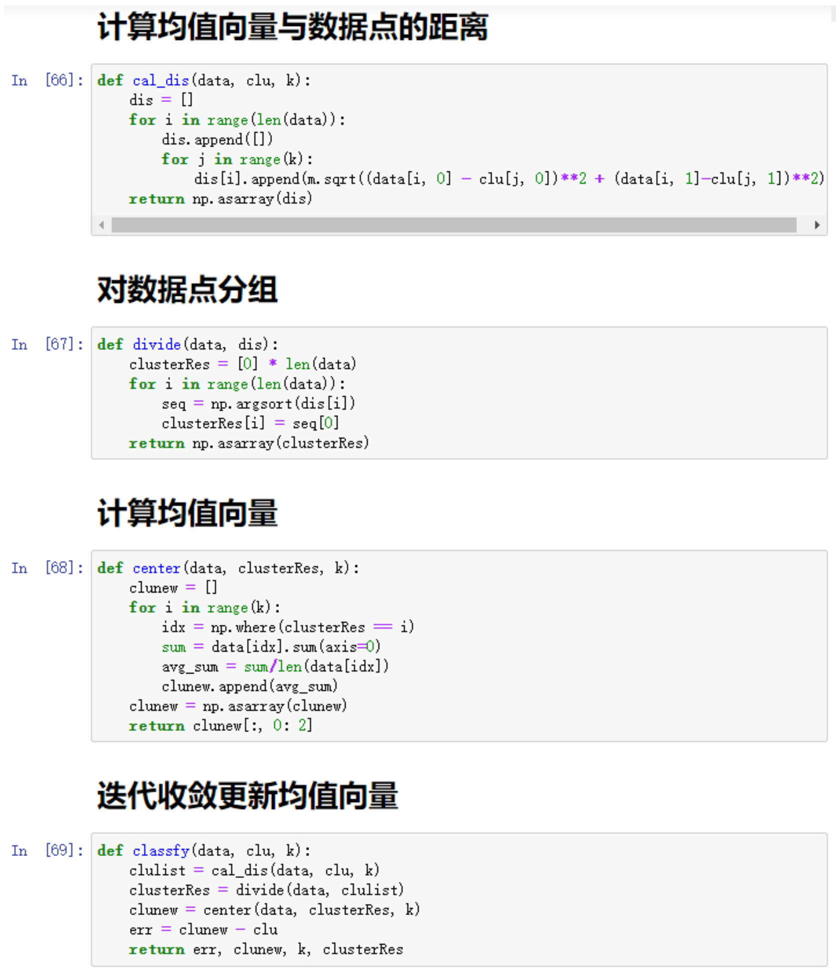
```
In [70]: def plotRes(data, clusterRes, clusterNum):
             nPoints = len(data)
             scatterColors = ['black', 'blue', 'green']
             for i in range(clusterNum):
                 color = scatterColors[i % len(scatterColors)]
                 x1 = [];  y1 = []
                 for j in range(nPoints):
                     if clusterRes[j] == i:
                         x1.append(data[j, 0])
                         y1.append(data[j, 1])
                 plt.scatter(x1, y1, c=color, alpha=1, marker='+')
             plt.show()
```

Fig. 4. Core code of visualizing the results

3.1. K-MEANS.

In order to implement the algorithm, we need to calculate the distance between each sample point and the initial mean vector and updated for several iterations after selecting the initial mean vector, and the final output is cluster partition which could be processed further, some of the code can be seen as Fig 5.

To avoid too long running time, a maximum number of running rounds or a minimum adjustment amplitude threshold is usually set. If the maximum number of rounds is reached or the adjustment range is less than the threshold, the operation is stopped.



计算均值向量与数据点的距离

```
In [66]: def cal_dis(data, clu, k):
             dis = []
             for i in range(len(data)):
                 dis.append([])
                 for j in range(k):
                     dis[i].append(m.sqrt((data[i, 0] - clu[j, 0])**2 + (data[i, 1]-clu[j, 1])**2)
             return np.asarray(dis)
```

对数据点分组

```
In [67]: def divide(data, dis):
             clusterRes = [0] * len(data)
             for i in range(len(data)):
                 seq = np.argsort(dis[i])
                 clusterRes[i] = seq[0]
             return np.asarray(clusterRes)
```

计算均值向量

```
In [68]: def center(data, clusterRes, k):
             clunew = []
             for i in range(k):
                 idx = np.where(clusterRes == i)
                 sum = data[idx].sum(axis=0)
                 avg_sum = sum/len(data[idx])
                 clunew.append(avg_sum)
             clunew = np.asarray(clunew)
             return clunew[:, 0: 2]
```

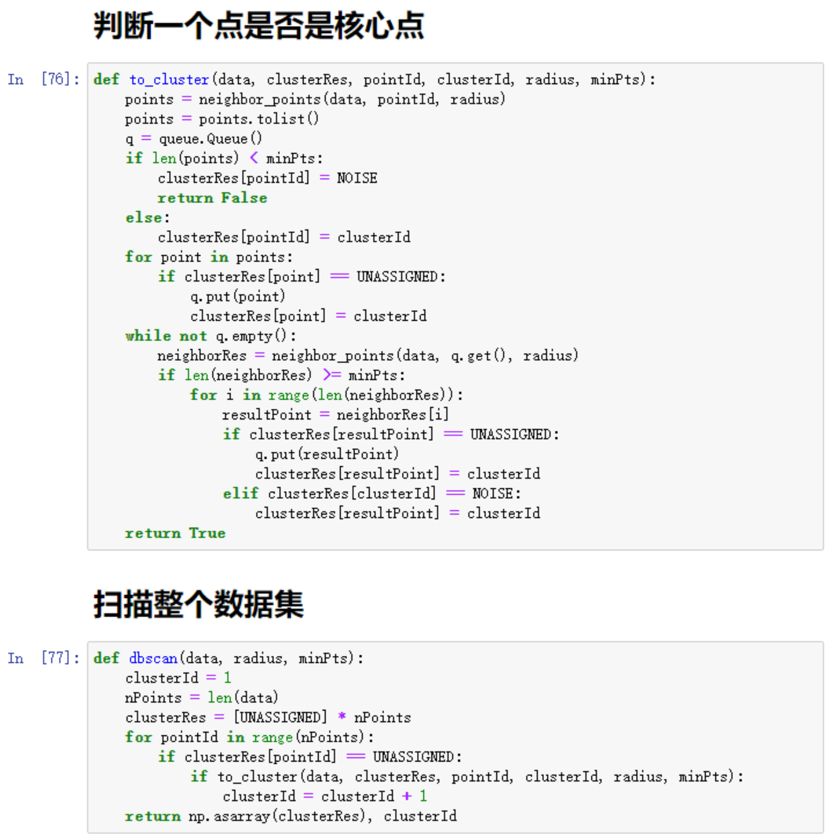
迭代收敛更新均值向量

```
In [69]: def classfy(data, clu, k):
             clulist = cal_dis(data, clu, k)
             clusterRes = divide(data, clulist)
             clunew = center(data, clusterRes, k)
             err = clunew - clu
             return err, clunew, k, clusterRes
```

Fig. 5. Core code of implementing K-means

3.2. DBSCAN.

DBSCAN clustering algorithm is a density-based clustering algorithm, which divides the data into several clusters, so that the data points in each cluster are relatively dense, and the density difference between each cluster and other clusters is large. To implement this algorithm, it is necessary to randomly select an unvisited data point and find all data points within a circle centered on the data point with radius eps. If the number of data points in this circle is greater than or equal to min-samples, all data points in this circle are assigned to the same cluster, and all data points in this cluster are accessed recursively. If the number of data points in the circle is less than min-samples, the data point is marked as a noise point, and the other data points that have not been visited are continued, some of the code can be seen as Fig 6.



判断一个点是否是核心点

```
In [76]: def to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
             points = neighbor_points(data, pointId, radius)
             points = points.tolist()
             q = queue.Queue()
             if len(points) < minPts:
                 clusterRes[pointId] = NOISE
                 return False
             else:
                 clusterRes[pointId] = clusterId
             for point in points:
                 if clusterRes[point] == UNASSIGNED:
                     q.put(point)
                     clusterRes[point] = clusterId
             while not q.empty():
                 neighborRes = neighbor_points(data, q.get(), radius)
                 if len(neighborRes) >= minPts:
                     for i in range(len(neighborRes)):
                         resultPoint = neighborRes[i]
                         if clusterRes[resultPoint] == UNASSIGNED:
                             q.put(resultPoint)
                             clusterRes[resultPoint] = clusterId
                         elif clusterRes[clusterId] == NOISE:
                             clusterRes[resultPoint] = clusterId
             return True
```

扫描整个数据集

```
In [77]: def dbscan(data, radius, minPts):
             clusterId = 1
             nPoints = len(data)
             clusterRes = [UNASSIGNED] * nPoints
             for pointId in range(nPoints):
                 if clusterRes[pointId] == UNASSIGNED:
                     if to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
                         clusterId = clusterId + 1
             return np.asarray(clusterRes), clusterId
```

Fig. 6. Core code of implementing DBSCAN

In this algorithm, samples that do not belong to any cluster are considered noise or anomaly samples.

4. Evalution.

In this data set, it is most appropriate to set the K value as 3. If the appropriate K value is not selected, the experimental results will be greatly affected, as shown in the Fig 7.

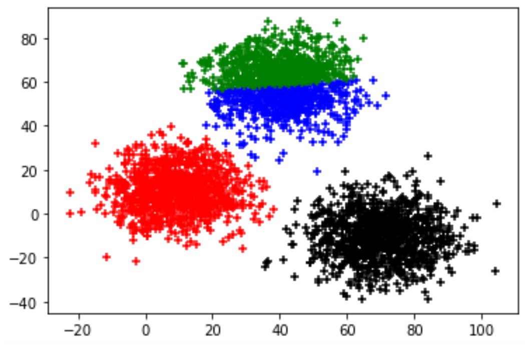
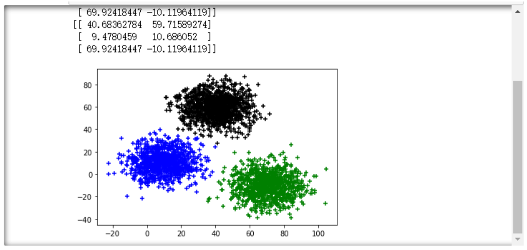


Fig. 7. Using the wrong k = 4

After reading the data set, K-means and DBSCAN were used to process the data set respectively, and the results are displayed by calling the matlablib library. It can be seen in Fig 8 and

Fig 9 that the two methods have good clustering effect.



```
 [ 69.92418447 -10.11964119]]
[[ 40.68362784  59.71589274]
 [  9.4780459   10.686052  ]
 [ 69.92418447 -10.11964119]]
```

Fig. 8. Result of K-means

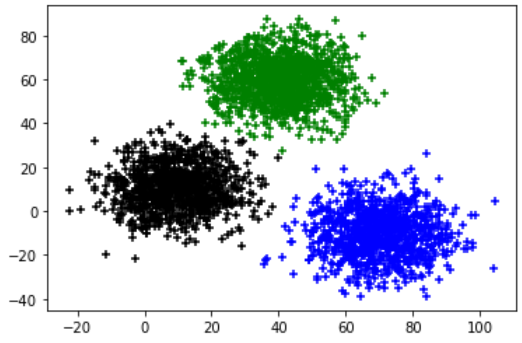


Fig. 9. Result of DBSCAN

Using K-means as an example to process the data several times, it can be seen that the results remain stable in Fig 10, which proves that the algorithm maintains good properties in this experiment.

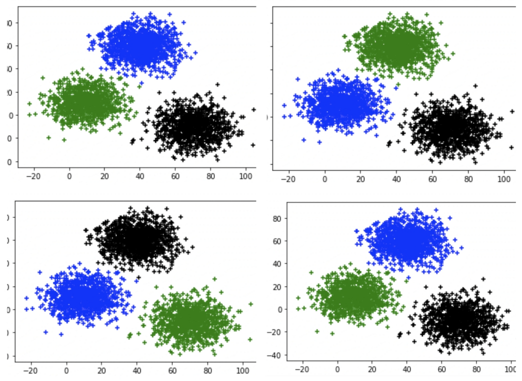


Fig. 10. Results after multiple tests

Use Time library, we can get the whole processing time of the algorithm.

5. Conclusion.

K-means and DBSCAN are two popular clustering algorithms used in unsupervised machine learning. K-means is a centroid-based or partition-based clustering algorithm, while DBSCAN is a density-based spatial clustering algorithm with noise. Both algorithms are used to group data points based on their inherent similarities, but they differ in their approach to clustering.

K-means works by partitioning a dataset into distinct, non-overlapping clusters. The algorithm iteratively performs two primary steps: (1) assigning each data point to the nearest cluster centroid, and (2) updating the centroid of each cluster based on the mean of the data points assigned to it. This iterative process continues until the centroids no longer change significantly or a set number of iterations is reached. K-means is particularly effective when theres a preliminary understanding or estimation of how many clusters the dataset should be segmented into. However, it has several limitations, such as difficulty in handling non-spherical or differently sized clusters, sensitivity to initial conditions, and the need to specify the number of clusters.

DBSCAN, on the other hand, is a density-based algorithm that seeks to build a hierarchy of clusters either through a bottom-up or top-down approach. It segments data into spherical clusters, making it well-suited for such distributions. DBSCAN operates on a density-based approach, where a cluster is defined as a region of high density separated by regions of low density. The algorithm identifies core points, which are data points that have at least a minimum number of other data points within a specified radius. It then expands the cluster by adding density-reachable points, which are data points that are within the specified radius of a core point. DBSCAN is particularly effective when dealing with datasets that have different sizes or shapes of clusters, and it can identify outliers and noise points. However, it has several limitations, such as the need to specify the radius and minimum number of points, sensitivity to the choice of parameters, and the inability to handle clusters with varying densities.

In summary, K-means and DBSCAN are two popular clustering algorithms that differ in their approach to clustering. K-means is a centroid-based or partition-based algorithm that is effective when theres a preliminary understanding or estimation of how many clusters the dataset should be segmented into. DBSCAN is a density-based algorithm that is effective when dealing with datasets that have different sizes or shapes of clusters, and it can identify outliers and noise points. Both algorithms have their strengths and limitations, and the choice between them often depends on the characteristics of the dataset at hand and the desired outcomes from the clustering process.

6. REFERENCES.

[1] K. B M ,Nandan S M ,R. R B , et al.An efficient framework for obtaining the initial cluster centers[J].Scientific Reports,2023,13(1):20821-20821.
[2] Zicai C ,Bocheng Z ,Chen B .A new Chinese text clustering algorithm based on WRD and improved K-means[J].Intelligent Data Analysis,2023,27(4):1205-1220.
[3] Renato A D C ,Vladimir M .On k-means iterations and Gaussian clusters[J].Neurocomputing,2023,553
[4] Jin K ,Zhang L ,SUN D .Research on Pre-classification Method of Industrial Control Data Based on Adaptive BLOCK-DBSCAN[C].School of Artifificial Intelligence and Data Science,Hebei University of Technology;,2023:6.DOI:10.26914/c.cnkihy.2023.034872.
[5] Fengxia Z ,Han Y ,Yu S , et al.PhageTailFinder: A tool for phage tail module detection and annotation[J].Frontiers in Genetics,2023,14947466-947466.
[6] Bing M ,Can Y ,Aihua L , et al.A Faster DBSCAN Algorithm Based on Self-Adaptive Determination of Parameters[J].Procedia Computer Science,2023,221113-120.