HW4

Experiment settings

1. 目标：对iris数据集进行dbscan聚类分析。
2. 数据集：iris数据集，来自uci公开数据集，大小5kb，特征数量共4个，分别是sepal length，sepal width，petal length，petal width。
3. 参数设置：距离度量采用欧式距离。eps设置为0.1，0.2，0.4，0.8；minPts设置为2，5，10,分别观察聚类的结果。
4. 聚类效果评价指标：采用轮廓系数评价。
5. 实验环境：Python3.10
6. 需要安装的包：sklearn
7. 实验过程：先进行数据预处理，然后进行dbscan聚类，画出图像进行可视化分析。

选择Iris数据集的原因

1. **噪音点处理**：Iris数据集可能包含一些异常值或噪音点。DBSCAN能够识别并处理这些噪音点。
2. **不需要预设簇的数量**：与K-Means算法不同，DBSCAN不需要事先指定簇的数量。这对于Iris数据集非常有用，因为如果我们没有关于应有簇数量的先验信息，DBSCAN可以自动确定簇的数量。
3. **簇的形状**：Iris数据集中的簇可能不是球形的，而DBSCAN不像K-Means那样假设簇是超球形的。DBSCAN可以识别任何形状的簇。
4. **参数设定**：DBSCAN需要两个参数：半径（eps）和最小点数（min_samples）。这些参数可以通过观察数据的空间分布来合理设置。Iris数据集的特征空间表现出较为清晰的分布，使得这些参数的设置相对较为直观。
5. **可解释性**：DBSCAN生成的簇边界有着基于密度的直观解释，这增加了聚类结果的可解释性。

数据预处理

Code:

```
# 加载数据集
def load_iris_dataset(filename):
    with open(filename, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        next(csvreader)  # Skip header if present
        dataset = [list(map(float, row[:-1])) for row in csvreader if row]  #
Exclude the label
    return dataset
```

下载uci iris数据集，读取csv内容。

实现dbscan算法

Code:

```
def rangeQuery(X, P, eps):
    distances = euclidean_distances(X[P].reshape(1, -1), X)
    neighbors = set(np.where(distances <= eps)[1].tolist())  # 使用集合存储
neighbors
    return neighbors
```

```
def dbscan(X, eps, minPts):
    labels = np.full(X.shape[0], -1)
    C = 0
    for P in range(X.shape[0]):
        if labels[P] != -1:
            continue
        neighbors = rangeQuery(X, P, eps)
        if len(neighbors) < minPts:
            labels[P] = -2  # Mark as noise
            continue
        C += 1
        labels[P] = C
        neighbors.remove(P)  # 删除自身，避免后面重复检查
        seeds = neighbors.copy()
        while seeds:
            Pn = seeds.pop()
            if labels[Pn] == -2:
                labels[Pn] = C
            if labels[Pn] == -1:
                labels[Pn] = C
                Pn_neighbors = rangeQuery(X, Pn, eps)
                if len(Pn_neighbors) >= minPts:
                    seeds |= Pn_neighbors  # 使用集合的并集操作添加新的元素
    return labels
```

`rangeQuery` 函数

- 输入参数
  - `X`是数据点的集合，通常是一个二维数组，其中每一行表示一个数据点。
  - `P`是查询点的索引。
  - `eps`是邻域的半径，表示距离阈值。
- **功能**：该函数用于寻找给定数据点`P`在半径`eps`内的所有邻居。
- 实现细节
  - 首先，计算点`P`到数据集`X`中所有点的欧氏距离。
  - 然后，找出距离小于等于`eps`的点，这些点被认为是`P`的邻居。
  - 返回这些邻居点的索引集合。

`dbscan` 函数

-
  - `X`是数据集。
  - `eps`表示两个数据点被认为是邻居的最大距离。
  - `minPts`是形成稠密区域所需的最小邻居数目。
- **功能**：执行DBSCAN算法，对数据集`X`进行聚类。
- 实现细节
  1. 初始化所有点的聚类标签为-1（尚未分类）。
  2. 遍历数据集中的每个点：
     - 如果点已被分类，跳过。
     - 否则，查询其邻居。
     - 如果邻居数量少于`minPts`，则将该点标记为噪声(-2)。
     - 如果邻居数量足够，则创建一个新的簇，将该点及其邻居加入簇中。
  3. 对于每个新找到的邻居点：

- 如果之前被标记为噪声，现在更改为当前簇的标签。
- 如果未被分类，也加入当前簇。
- 如果该邻居点的邻居数量也达到`minPts`，则将其邻居加入到当前考虑的邻居集合中，以此方式扩展簇。

4. 最终返回所有点的聚类标签。

算法时间复杂度与空间复杂度分析

- **时间复杂度**：对于每一个数据点，我们都执行了一次`rangeQuery`。因此，时间复杂度将是O(n^2)。这是最简单实现版本的时间复杂度，在实际使用中，这个时间复杂度可以通过空间索引数据结构（如KD树，球树等）来优化。
- **空间复杂度**：`dbscan`函数主要存储了输入数据集和每个点的标签，因此其空间复杂度为O(n)。

指标分析

Code:

```
labels = dbscan(iris, eps=0.2, minPts=5)
end_time=time.time()
print(f'consume time:{end_time-start_time}')
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)

if n_clusters > 1 and n_noise != len(labels):
    silhouette_avg = silhouette_score(iris, labels)
    print(f"轮廓系数: {silhouette_avg}")
else:
    print("轮廓系数无法计算，需要至少两个簇且不能全部为噪声。")
```

1. `n_clusters = len(set(labels)) - (1 if -1 in labels else 0)`：计算簇的数量。在`dbscan`执行的结果中，-1代表噪声，其他每个不同的正值代表一个簇。通过生成标签的集合并计算长度，可以得到簇的数量。
2. `n_noise = list(labels).count(-1)`：计算噪声的数量，也就是标签值为-1的数据点的数量。
3. 接下来，如果生成的簇的数量大于1且噪声的数量不等于数据的总数量，则计算轮廓系数，否则输出提示信息。
    - 轮廓系数是聚类效果的一个度量，其值范围在-1到1之间。越接近1，表示聚类效果越好；越接近-1，表示聚类效果越差。如果所有数据都是噪声或只有一个簇，轮廓系数无法计算。
4. `silhouette_avg = silhouette_score(iris, labels)`：调用`silhouette_score`函数计算轮廓系数。其中，`iris`是数据集，`labels`是聚类结果。

可视化分析

Code:

```
# Function to plot pairplot
def plot_pairplot(data, labels):
    # Convert the data to a DataFrame for seaborn compatibility
    df = pd.DataFrame(data, columns=[f'feature{i}' for i in
range(data.shape[1])])
    df['label'] = labels
    sns.pairplot(df, hue='label', palette='bright')
    plt.show()
```

```
def plot_cluster(labels):
    # 假设 'labels' 是DBSCAN后得到的簇标签数组
    # 'iris' 是原始的数据集数组

    # 获得簇的数量（去除噪声点）
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

    # 设置不同簇的颜色
    # 注意：如果簇的数量很多，则需要调整此处的颜色列表
    colors = ['royalblue', 'maroon', 'forestgreen', 'mediumorchid', 'tan',
'deeppink', 'olive', 'goldenrod', 'lightcyan', 'navy']
    vectorizer = np.vectorize(lambda x: colors[x % len(colors)])

    # 为每个簇的样本分配颜色
    colored_labels = vectorizer(labels)

    # 创建散点图
    plt.figure(figsize=(12, 9))
    for class_label, plot_color in zip(range(n_clusters_), colors):
        # 基于簇标签过滤数据点
        class_member_mask = (labels == class_label)
        xy = iris[class_member_mask]

        # 绘制每个簇的数据点
        plt.scatter(xy[:, 0], xy[:, 1], s=50, c=plot_color, label=f'Cluster
{class_label}')

    # 绘制噪声点，如果有的话
    if -1 in labels:
        # 获取噪声点
        noise_mask = (labels == -1)
        xy_noise = iris[noise_mask]
        plt.scatter(xy_noise[:, 0], xy_noise[:, 1], s=50, color='k',
label='Noise')

    # 添加图例
    plt.legend()

    # 添加各种标签
    plt.title('DBSCAN Clustering')
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')

    # 显示图像
    plt.show()
```

画出成对关系图与聚类散点图。

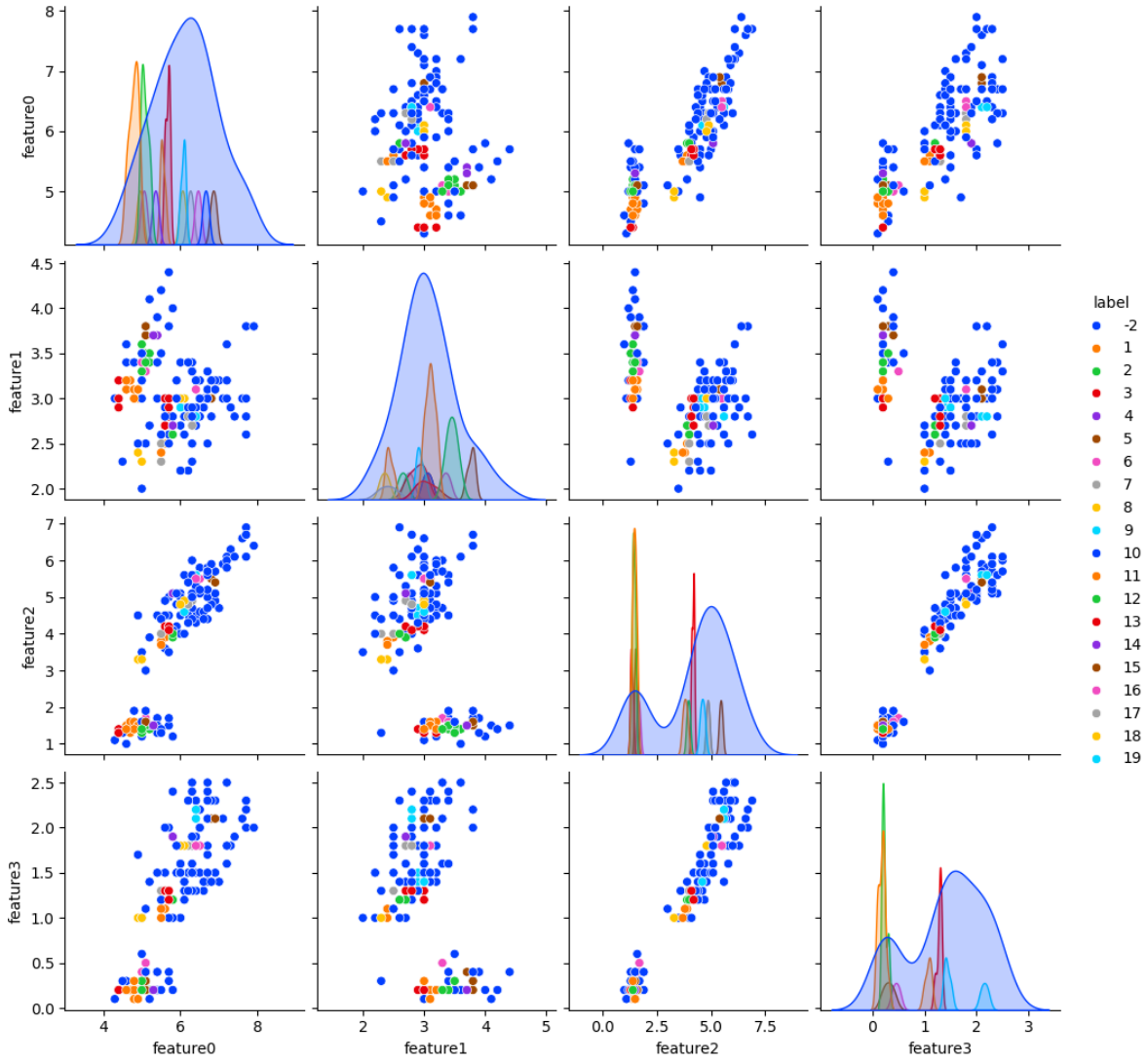
Result

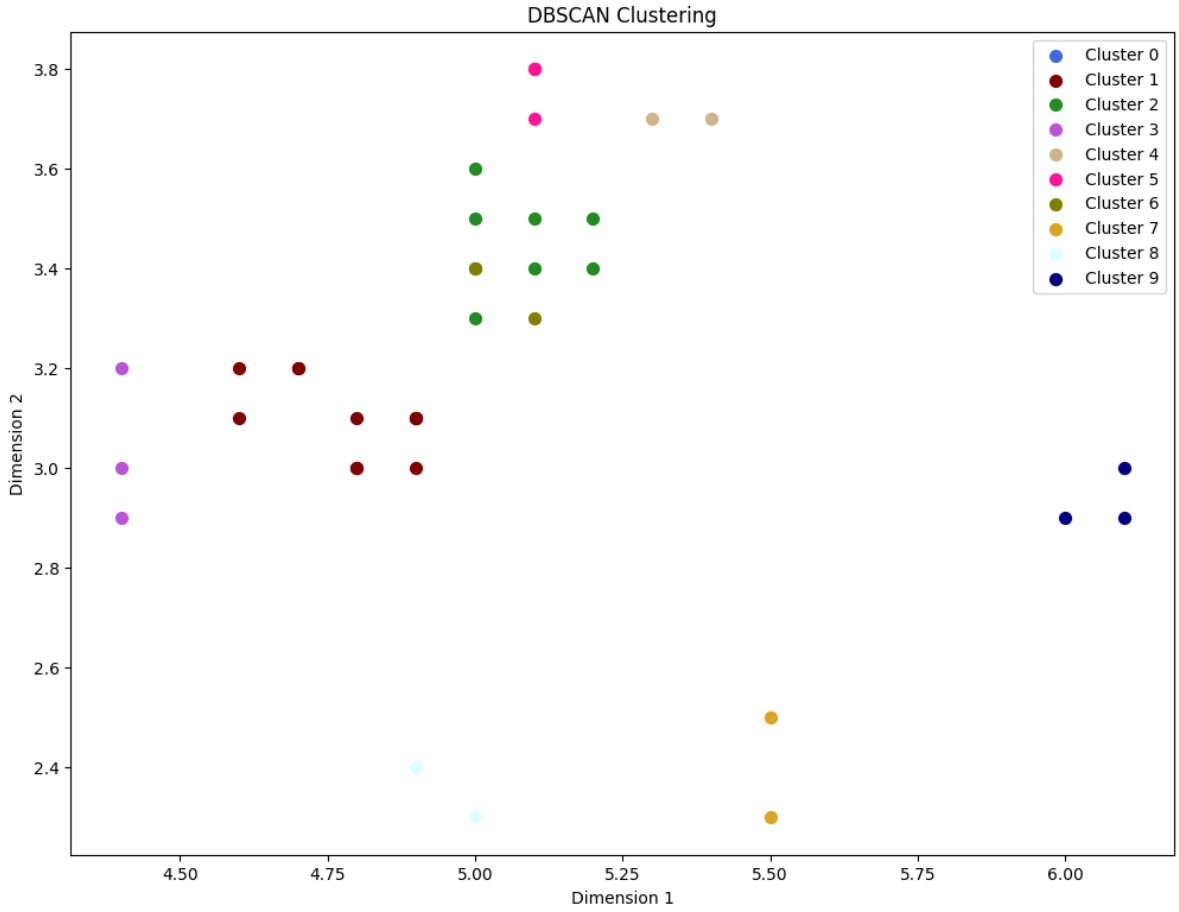
运行时间、运行结果与可视化比较分析

eps=0.2 minPts=2

经过观察，轮廓系数<0说明聚类效果很差，且成对关系图发现不同类的聚类有很大部分重合，证明此参数选择不合理。

```
consume time:0.025002717971801758
轮廓系数: -0.25717882502797673
```

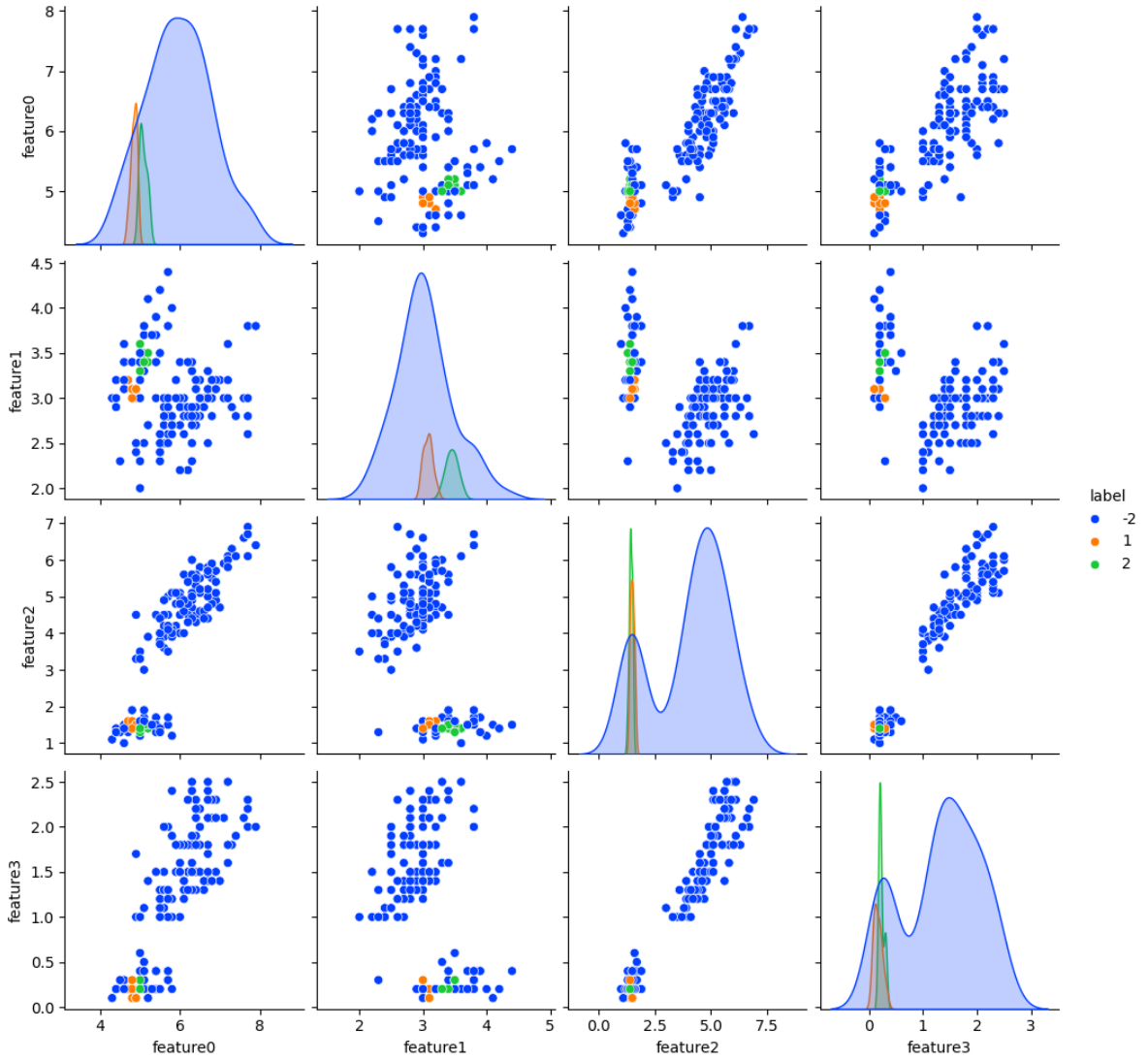


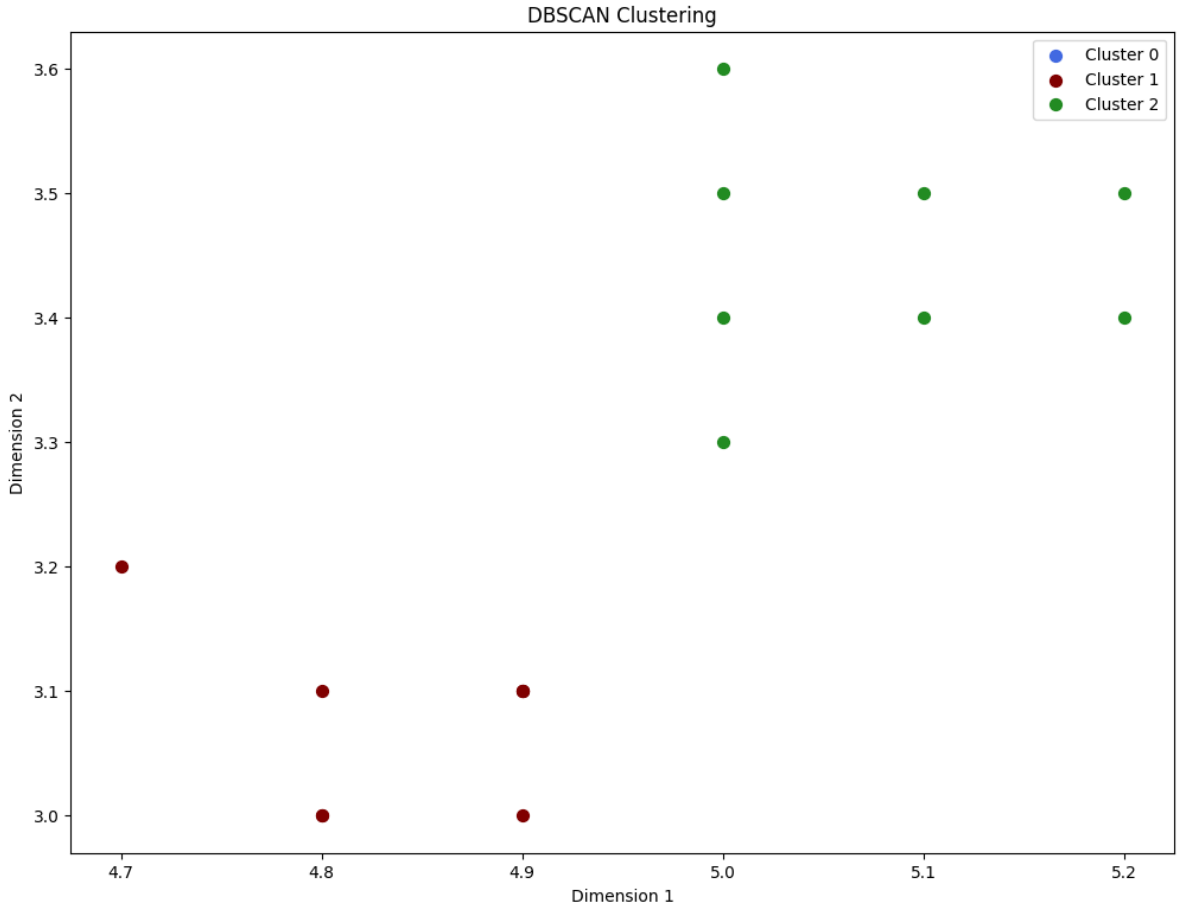


eps=0.2 minPts=5

同上，聚类效果也很差。

```
consume time:0.035149812698364 26
轮廓系数: 0.18025315745322368
```


feature0
feature1
feature2
feature3
label
-2
1
2


DBSCAN Clustering
Cluster 0
Cluster 1
Cluster 2
Dimension 1
Dimension 2

eps=0.2 minPts=8

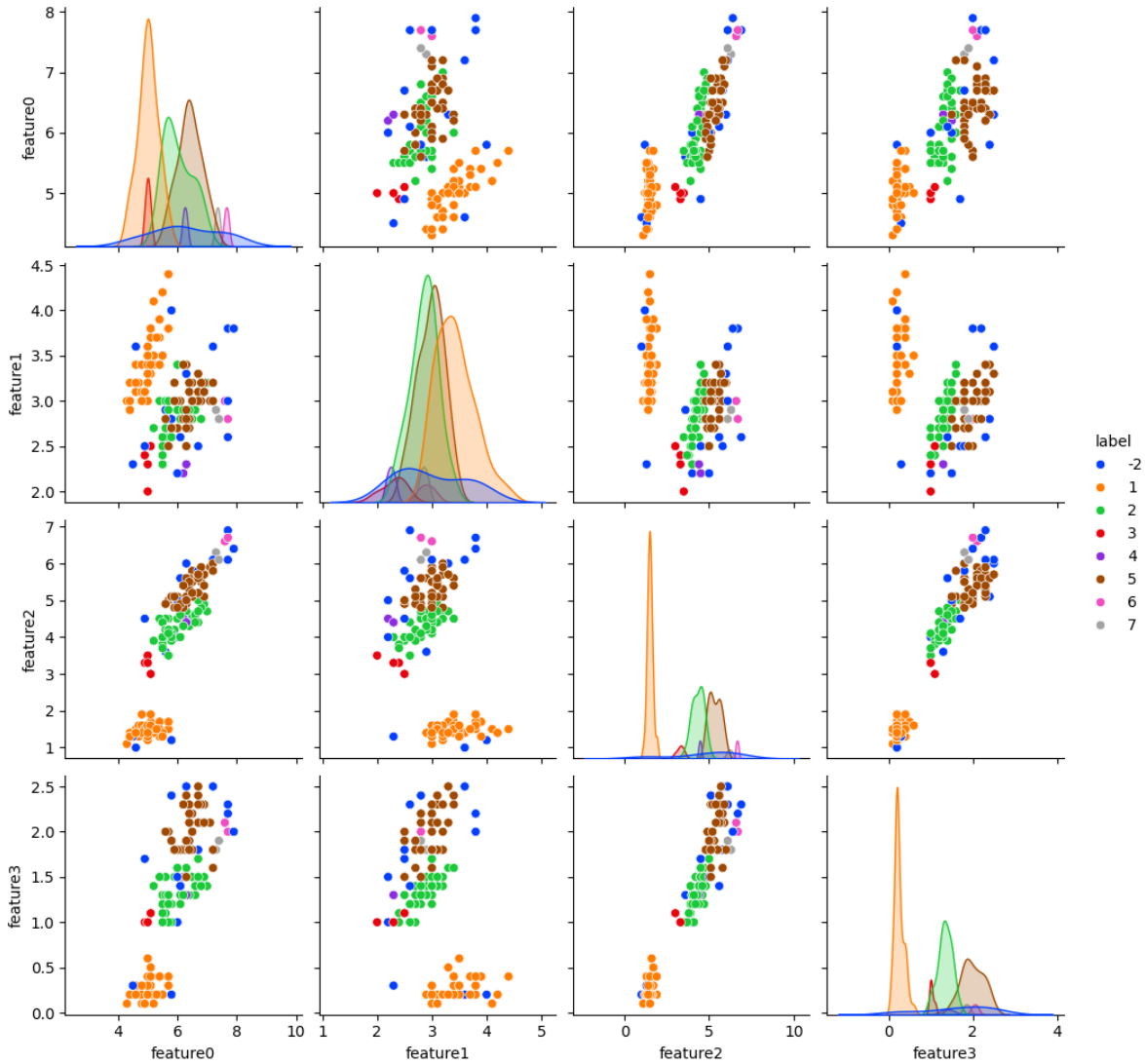
全部聚成了一个簇，聚类失败。

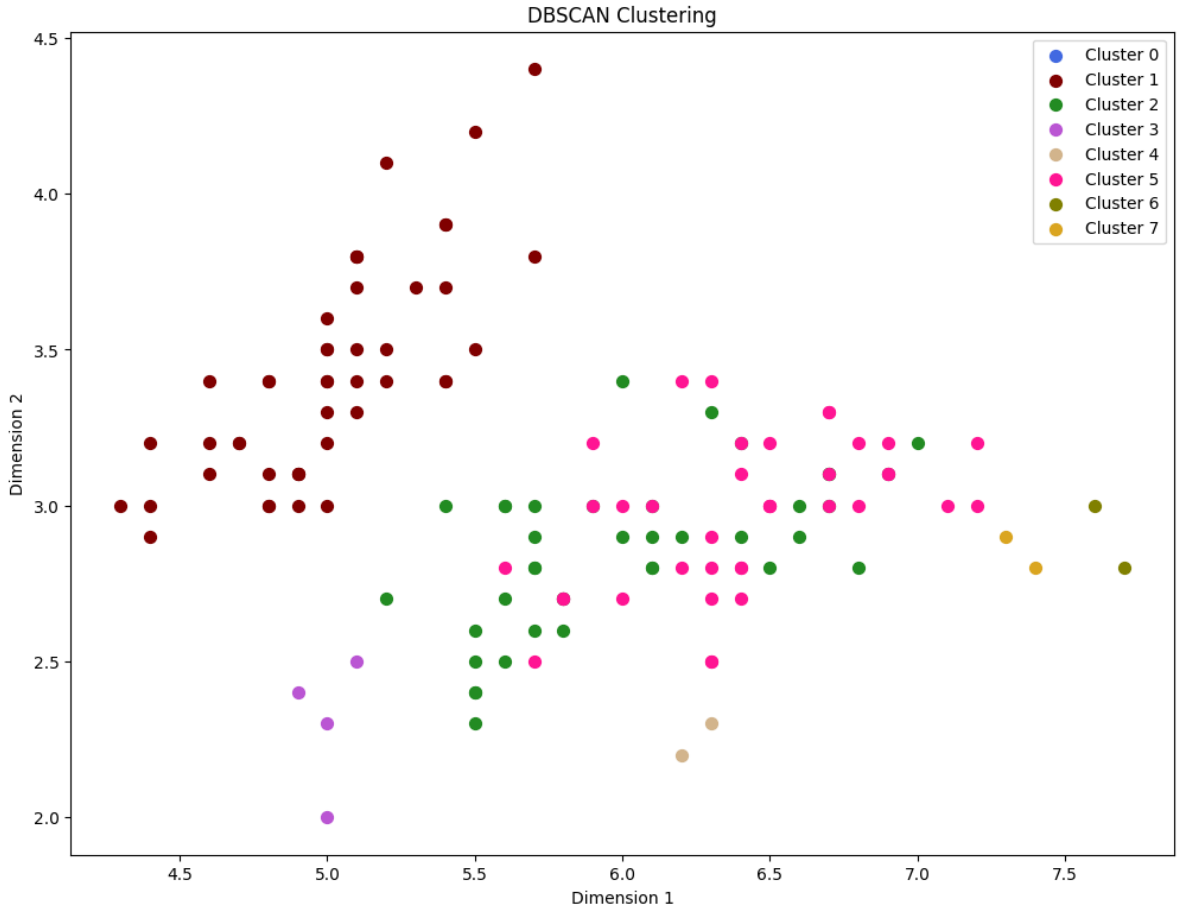
```
consume time:0.3100204467773437
轮廓系数无法计算，需要至少两个簇且不能全部为噪声。
```

eps=0.4 minPts=2

聚类数量多，轮廓系数小且成对图显示重合较多，聚类较差。

```
consume time:0.05054783821105957
轮廓系数: 0.23774364294415778
```

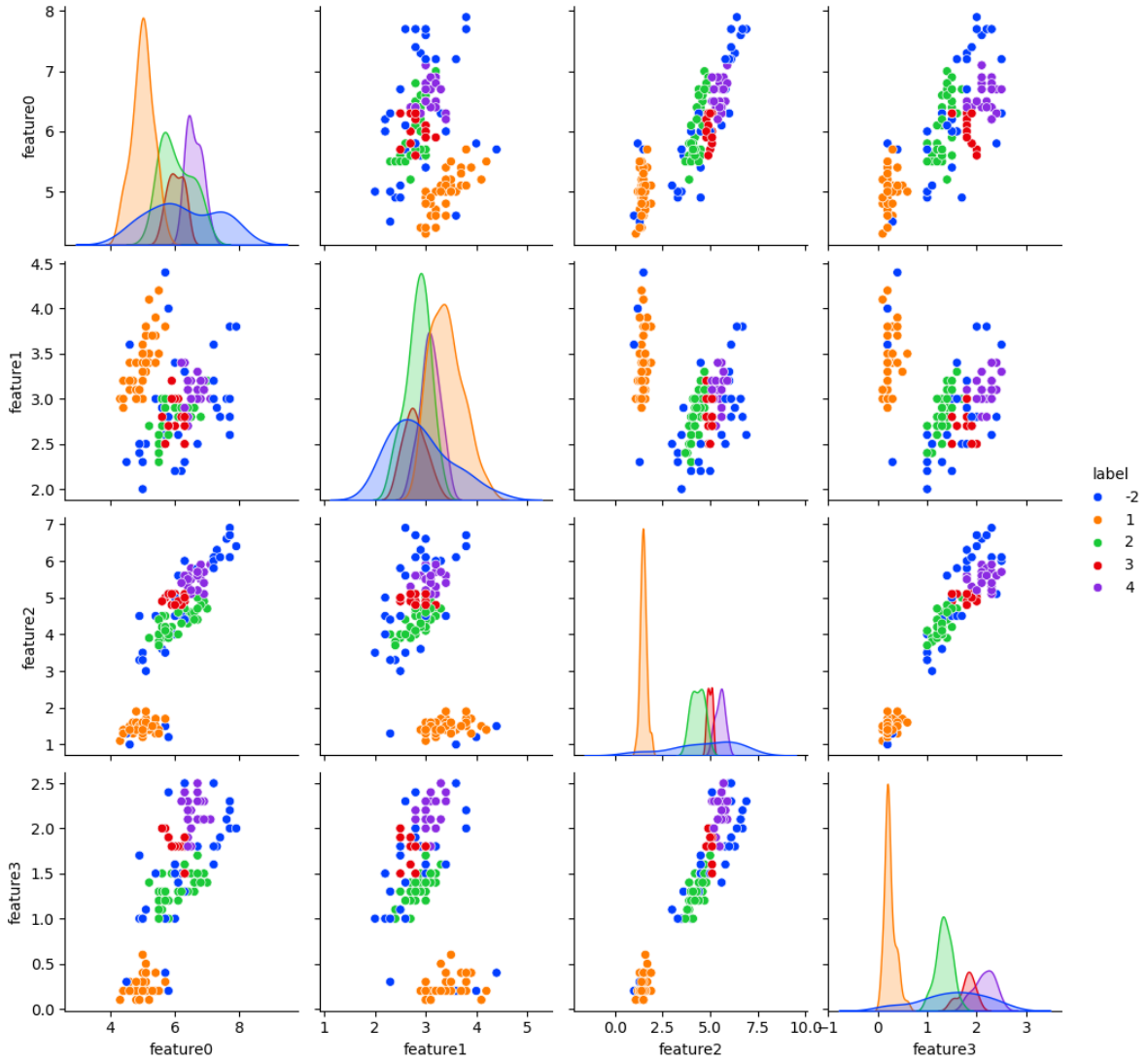


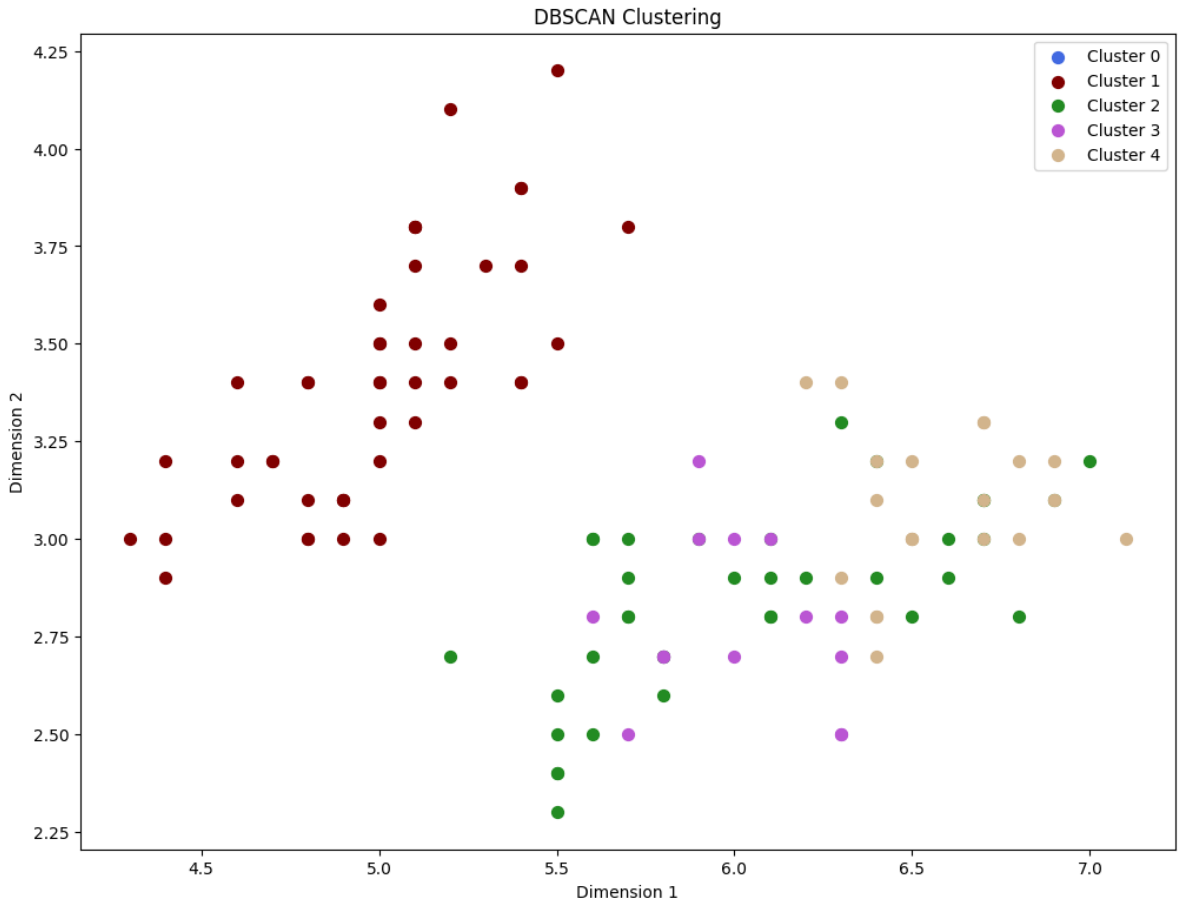


eps=0.4 minPts=5

同上，聚类较差。

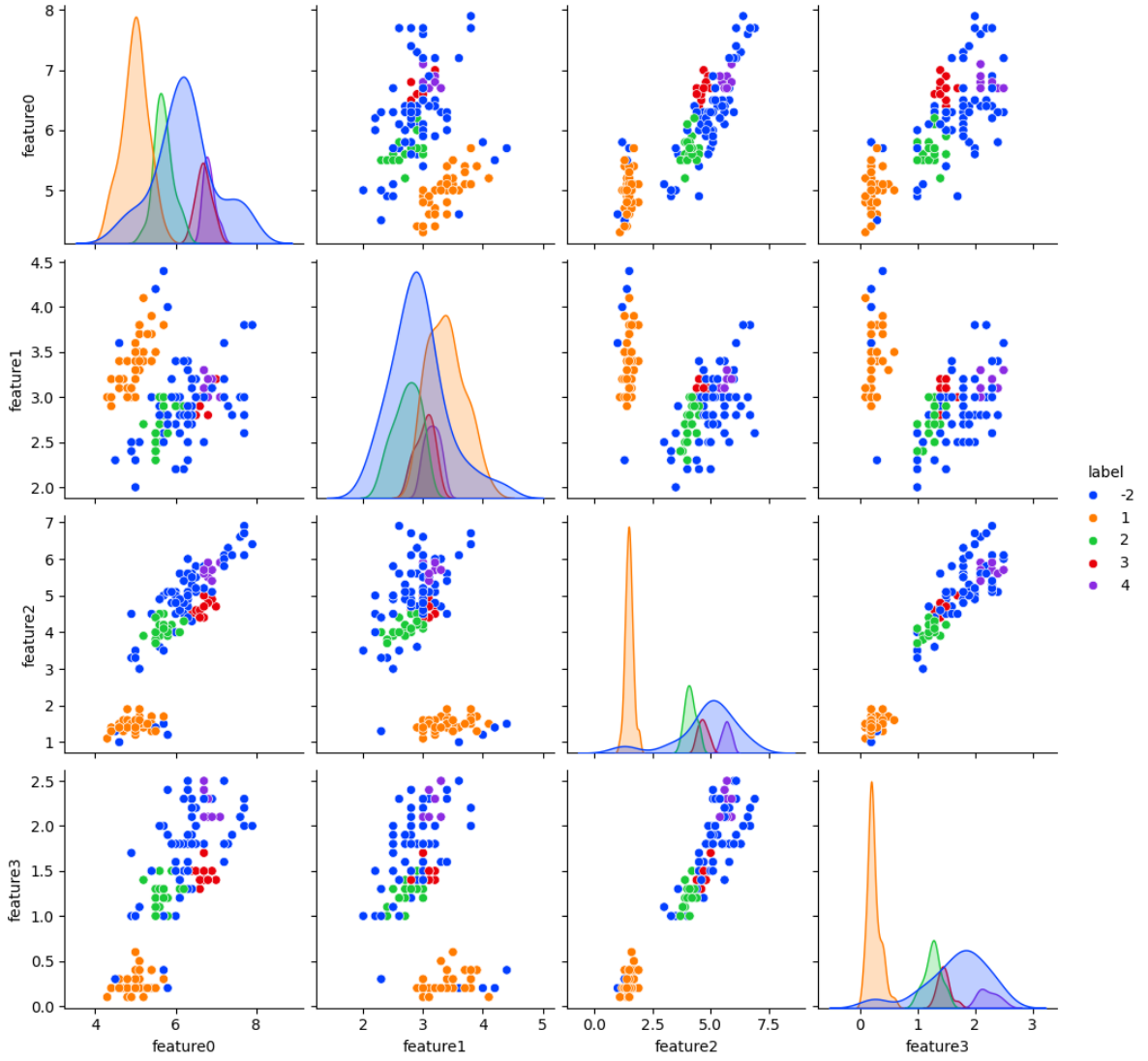
```
consume time:0.029967784881591797
轮廓系数: 0.2736076815092494
```

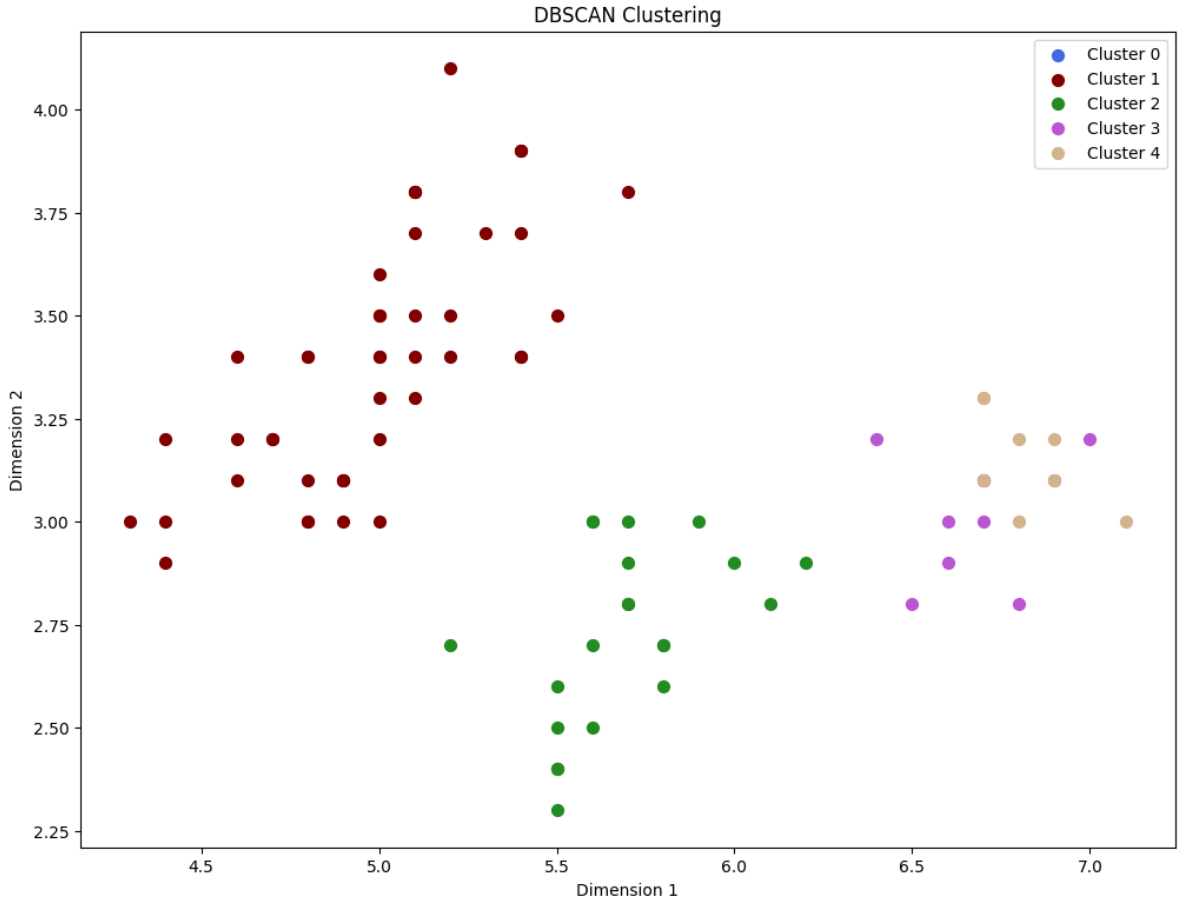

feature0
feature1
feature2
feature3
label
-2
1
2
3
4


DBSCAN Clustering
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Dimension 1
Dimension 2

eps=0.4 minPts=8

```
consume time:0.025051355361938477
轮廓系数: 0.1651597415590381
```

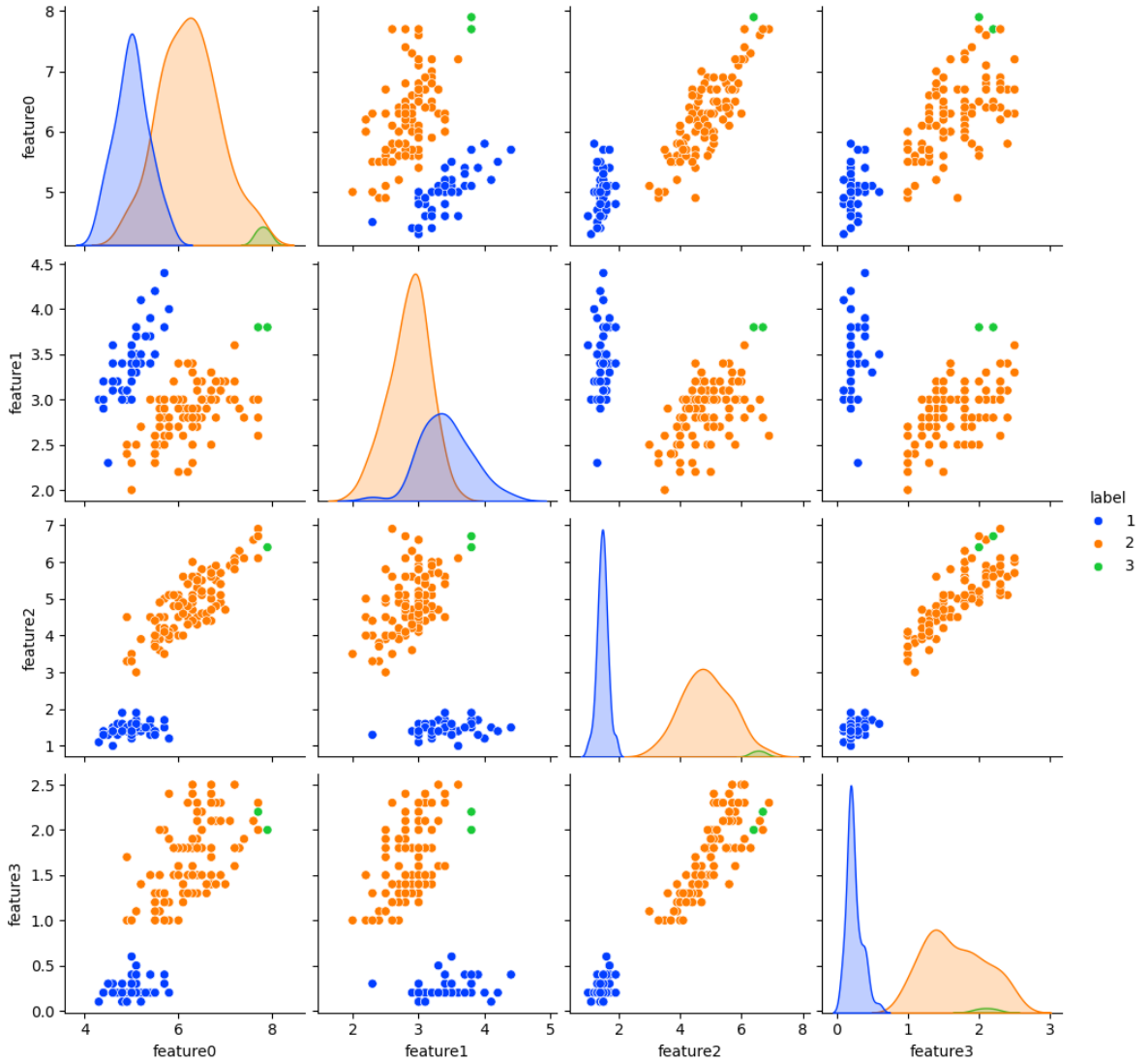


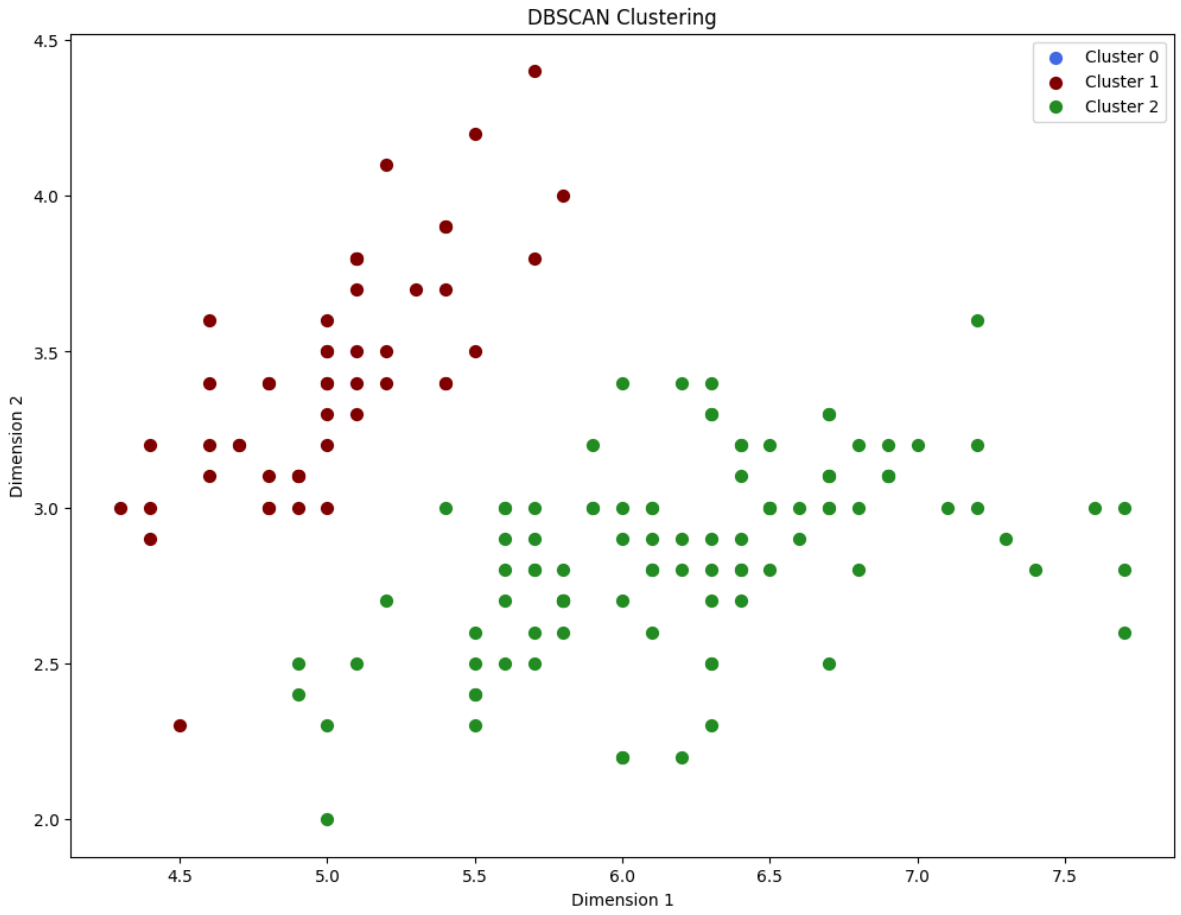


eps=0.8 minPts=2

聚成了2类，聚类效果不错。

```
consume time:0.03447747230529785
轮廓系数: 0.5090689670612821
```


feature0
feature1
feature2
feature3
label
1
2
3

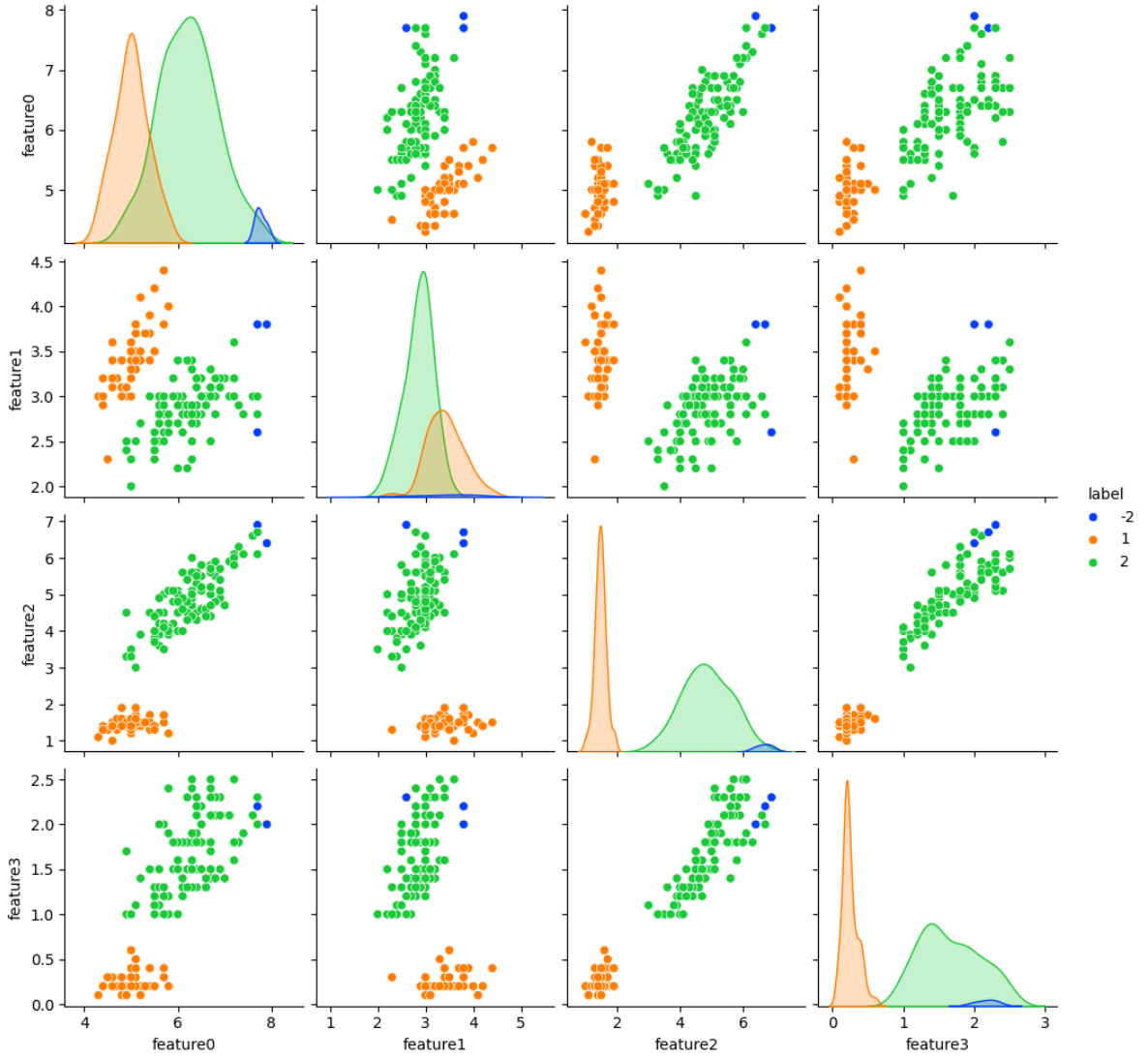

DBSCAN Clustering
Cluster 0
Cluster 1
Cluster 2
Dimension 1
Dimension 2

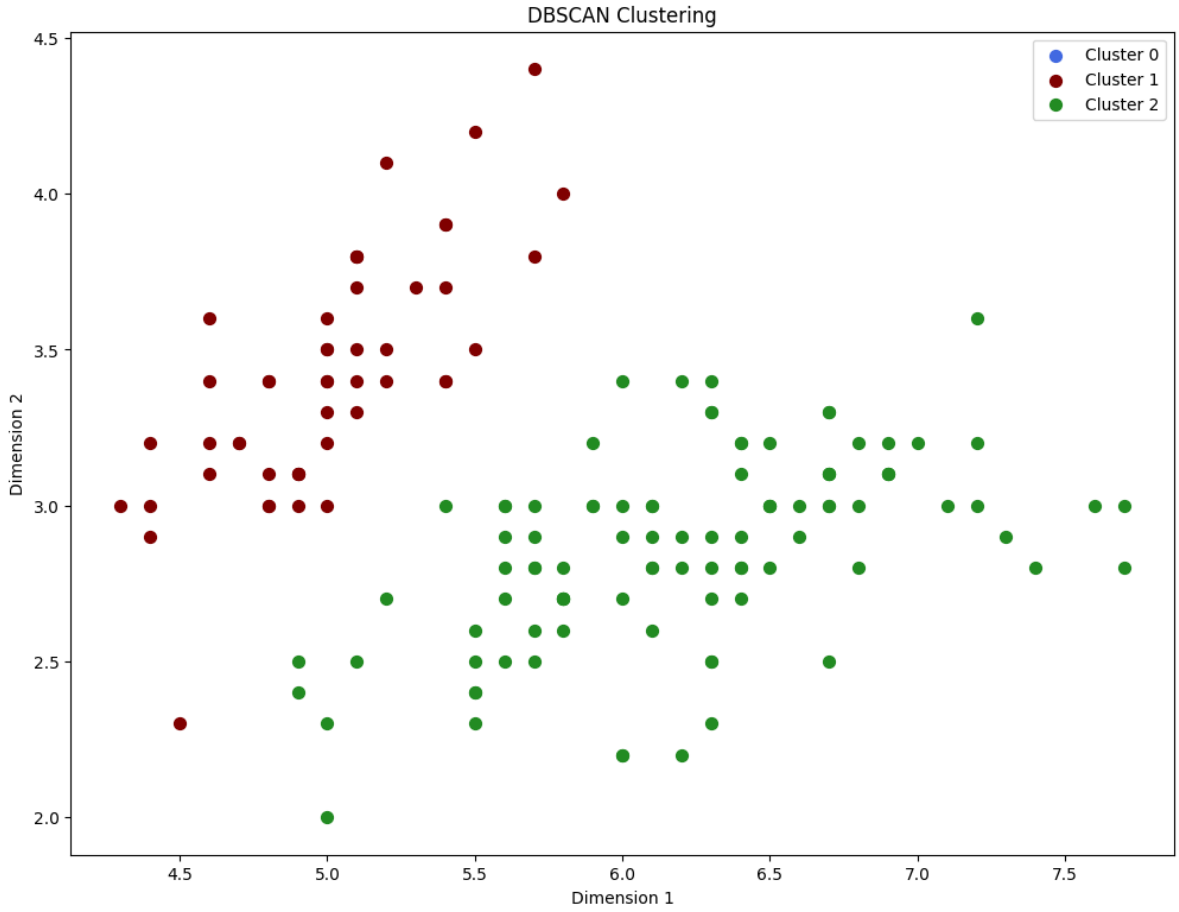
eps=0.8 minPts=5

与上面eps=0.8 minPts=2结果相同。

eps=0.8 minPts=8

```
consume time:0.027003288269042 97
轮廓系数: 0.5193363720017944
```





综上，dbscan对于eps与minpts的选择很有说法，对最终聚类结果影响非常大。比如对iris数据集，太小的eps与minpts会使聚类结果非常差。而合适的eps与minpts（如最后几个）能使其得出较为合理的聚类结果。