DBSCAN实现

2051498 储岱泽

算法实现详细描述

DBSCAN算法主要思路

DBSCAN是一种基于密度的聚类算法，其核心思想是通过确定数据点的邻域密度来识别簇，并将低密度区域标记为噪声点。DBSCAN 算法主要依赖两个参数：**eps（邻域半径）和minPts（最小点数）**。DBSCAN是基于一组邻域来描述样本集的紧密程度的，参数(`eps，minPts`)用来描述邻域的样本分布紧密程度。其中，`eps` 描述了某一样本的邻域距离阈值，`minPts` 描述了某一样本的距离为 `eps` 的邻域中样本个数的阈值。算法主要包含以下步骤：

1. **初始化：** 将所有数据点标记为未访问状态，初始化聚类标签列表和簇编号。
2. **核心点确定：** 对于每个数据点，计算其邻域内的点数。如果邻域内的点数大于等于 minPts，则将该点标记为核心点。
3. **簇扩展：** 对于每个核心点，将其邻域内的所有点添加到同一个簇中。如果邻域内的点也是核心点，则继续递归扩展簇。
4. **噪声点标记：** 将未被任何簇包含的点标记为噪声点。
5. **聚类结果：** 返回每个数据点的聚类标签。

和传统的K-Means算法相比，DBSCAN最大的不同就是不需要输入类别数k，最大的优势是可以发现任意形状的聚类簇，而不是像K-Means，一般仅仅使用于凸的样本集聚类。同时它在聚类的同时还可以找出异常点。

程序设计

在我的项目中，我使用了Python来进行DBSCAN算法的程序设计，首先定义一个名为 `DBSCAN` 的 Python 类，其中包含以下主要方法：

- `init(self, eps, min_pts)`：初始化方法，用于设置算法的参数，包括邻域半径 `eps` 和最小点数 `min_pts`。

```
# 初始化函数，设置eps邻域半径和minPts最小点数
def __init__(self, eps, min_pts):
    self.eps = eps  # 邻域半径，两个点成为邻居的最大距离
    self.min_pts = min_pts  # 一个点成为“核心点”所需的最小邻居数目
```

- `fit_predict(self, X)`：拟合数据并预测聚类标签的方法，接受数据集 `X` 作为输入，返回每个数据点的聚类标签。

```
 # 主函数，用于拟合数据并预测每个点的聚类标签
def fit_predict(self, X):
    labels = [0] * len(X)  # 初始化所有点的标签为0
    cluster_id = 0  # 初始化聚类ID

    # 对每个点进行迭代
    for i in range(len(X)):
        if labels[i] != 0:  # 如果点已经被标记，则跳过
            continue

        # 获取点i的邻域点
        neighbors = self.region_query(X, i)
        if len(neighbors) < self.min_pts:  # 如果邻域点数少于minPts，则为噪声点
            labels[i] = -1  # 标记为-1
        else:  # 否则，将该点作为新聚类的核心点
            cluster_id += 1  # 聚类ID自增
            self.expand_cluster(X, labels, i, neighbors, cluster_id)  # 扩展该核心点的聚类
    return labels  # 返回所有点的聚类标签
```

- `expand_cluster(self, X, labels, core_idx, neighbors, cluster_id)`：扩展簇的方法，用于从核心点开始递归地将所有可达的点加入到簇中。

```
    # 递归函数，用于扩展以核心点为核心的聚类
def expand_cluster(self, X, labels, core_idx, neighbors, cluster_id):
    labels[core_idx] = cluster_id  # 将核心点标记为当前聚类ID
    i = 0  # 初始化索引
    # 当存在待处理的邻居点时
    while i < len(neighbors):
        idx = neighbors[i]  # 取出当前邻居点的索引
        if labels[idx] == -1:  # 如果邻居是噪声点，则将其标记为当前聚类ID
            labels[idx] = cluster_id
        elif labels[idx] == 0:  # 如果邻居尚未分配到任何聚类
            labels[idx] = cluster_id  # 将其标记为当前聚类ID
            new_neighbors = self.region_query(X, idx)  # 获取邻居点的邻域点
            if len(new_neighbors) >= self.min_pts:  # 如果邻居点的邻域点数足够
                neighbors.extend(new_neighbors)  # 将这些邻域点添加到待处理列表
        i += 1  # 移动到下一个邻居点
```

- `region_query(self, X, idx)`：邻域查询方法，用于找到与给定点在指定邻域半径内的所有邻居点。

```
# 寻找给定点的邻域点
def region_query(self, X, idx):
    neighbors = []  # 初始化邻域点列表
    for i in range(len(X)):  # 遍历所有点
        if np.linalg.norm(X[idx] - X[i]) < self.eps:  # 如果两点之间的欧氏距离小于eps
            neighbors.append(i)  # 将点i添加到邻域点列表
    return neighbors  # 返回邻域点列表
```

数据集选择

在本项目中，我选择了经典的鸢尾花（Iris）数据集作为研究对象。鸢尾花数据集是一个经典的机器学习和数据挖掘基准数据集，由英国统计学家 Ronald Fisher 在 1936 年首次引入。该数据集包含了来自三个不同品种的鸢尾花（Setosa、Versicolor 和 Virginica）的样本，每个品种收集了 50 个样本，共计 150 个样本。对于每个样本，记录了四个特征：花萼长度（sepal length）、花萼宽度（sepal width）、花瓣长度（petal length）和花瓣宽度（petal width）。

我选择鸢尾花数据集作为研究对象的主要原因包括以下几点：

1. **数据集的广泛应用：** 鸢尾花数据集是一个被广泛应用于机器学习和数据挖掘算法验证的经典数据集，其特征具有明显的差异，适合用于聚类算法的研究和测试。
2. **数据集的简单性：** 鸢尾花数据集相对较小且简单，包含了少量的特征和样本，便于理解和处理。这使得我们可以快速进行实验和验证，加深对聚类算法的理解。

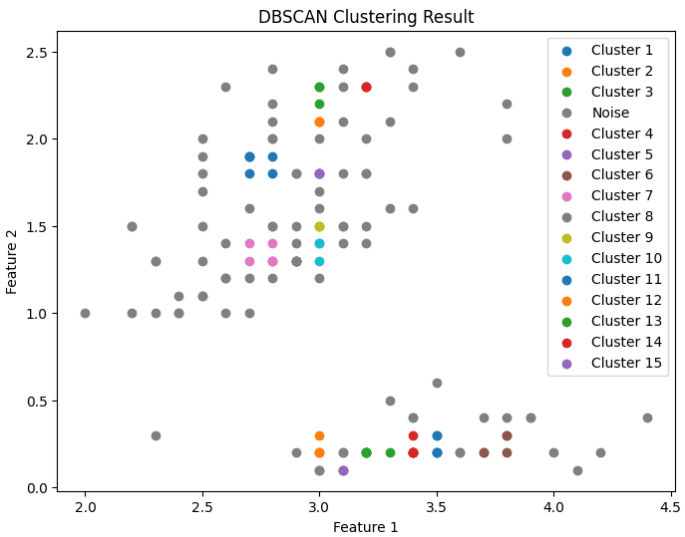
3. **数据集的标签信息：** 鸢尾花数据集包含了每个样本所属的鸢尾花品种标签，这使得我们可以在进行聚类算法研究时，对比聚类结果和真实标签，评估算法的性能和准确性。

```python
from ucimlrepo import fetch_ucirepo
# fetch dataset
iris = fetch_ucirepo(id=53)
# data (as pandas dataframes)
X = iris.data.features
y = iris.data.targets
# metadata
print(iris.metadata)
# variable information
print(iris.variables)
```

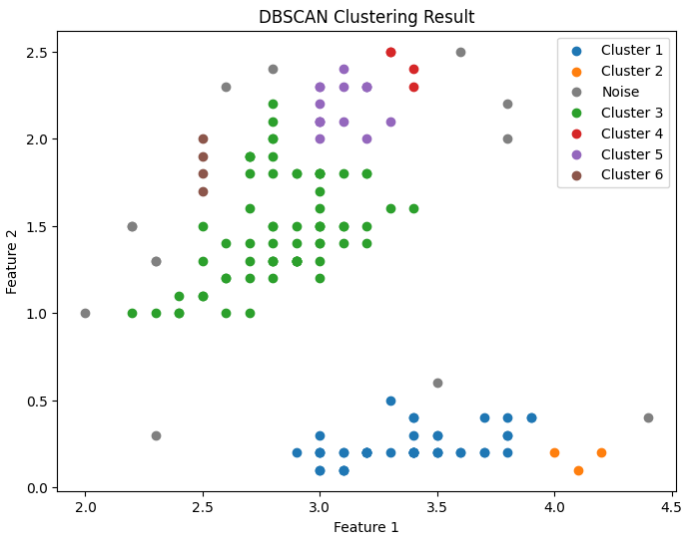
参数调整

在程序设计过程中，我对 DBSCAN 算法中的两个关键参数 `eps` 和 `min_pts` 进行了调整。通过调整这两个参数，我分析了不同设置对聚类结果的影响，并评估了算法的稳健性和鲁棒性。

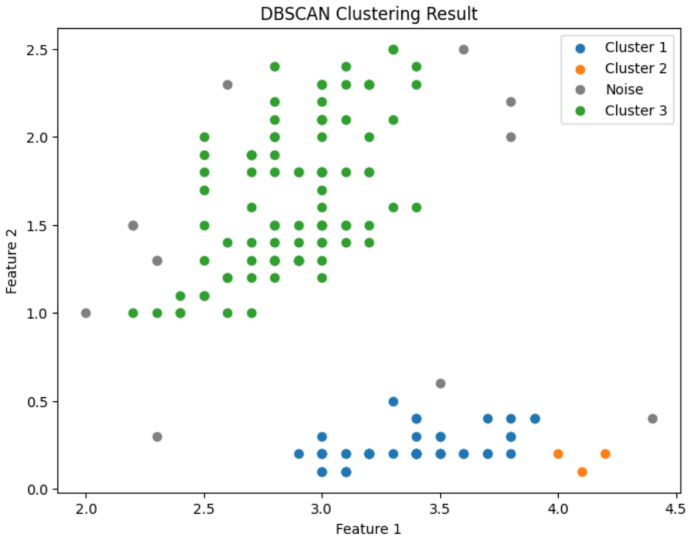
eps改变，min_pts不变



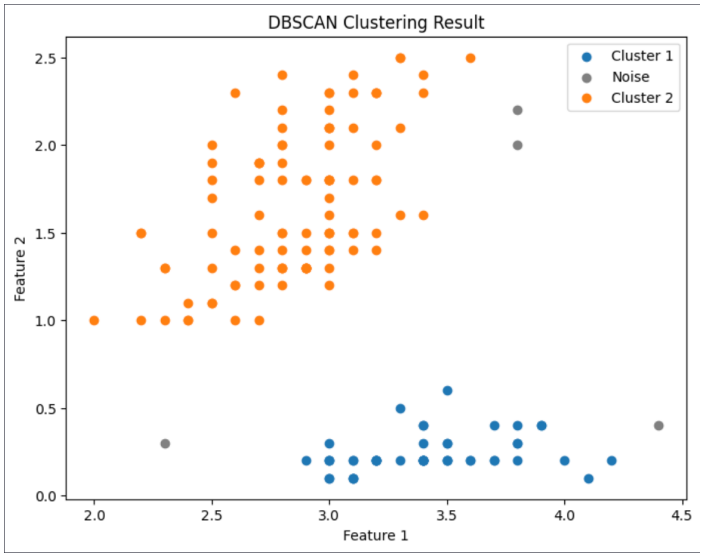
eps=0.1，min_pts=3



eps=0.15，min_pts=3



eps=0.2，min_pts=3

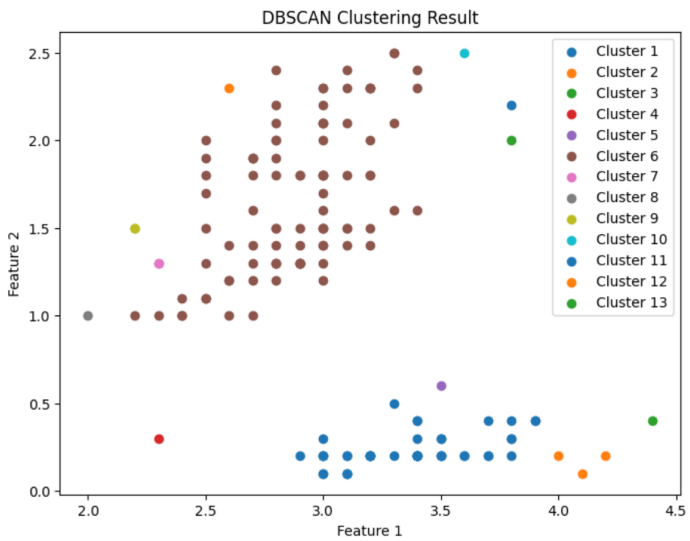


eps=0.25，min_pts=3

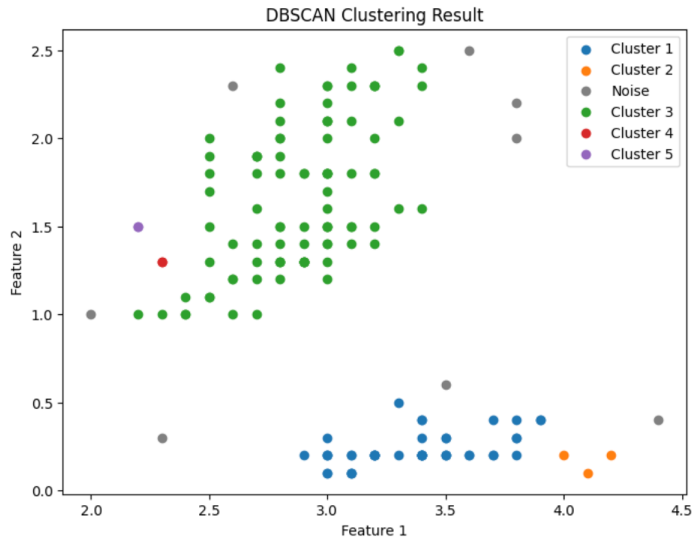
根据上面的实验结果我们可以得出结论：

1. **eps参数的影响：** 当eps值减小时，即邻域半径变小，DBSCAN算法识别出的聚类数量会增加。这意味着在较小的邻域内，数据点更容易被视为核心点，从而导致更多的小聚类形成。相反，当eps值增大时，邻域半径变大，算法倾向于将更多的点合并成一个较大的聚类。
2. **噪声点的识别：** 当eps值较小，minPts值不变时，噪声点（未被分配到任何聚类的点）的数量可能会增加，因为较小的邻域内难以找到足够的邻居点。

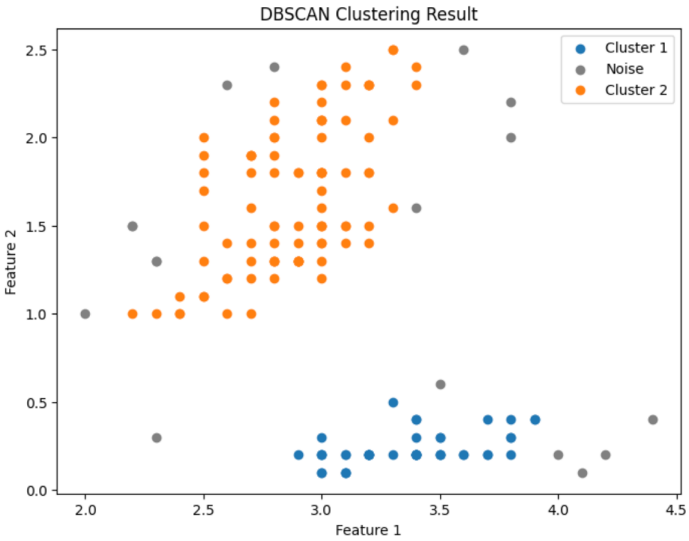
eps不变，min_pts改变



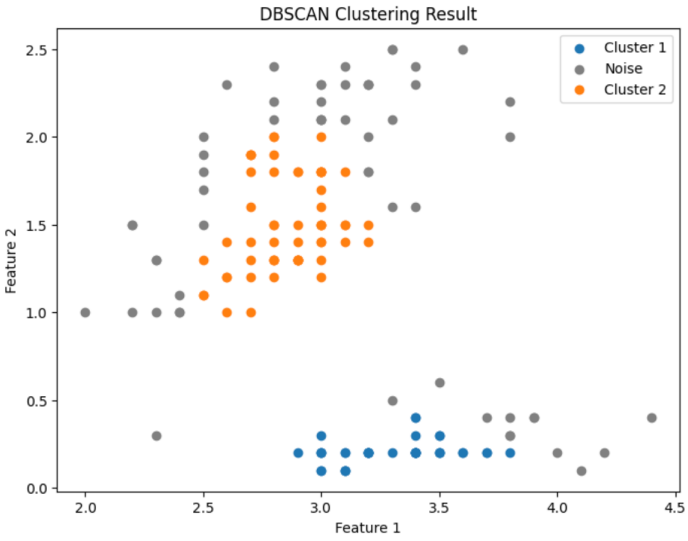
eps=0.2，min_pts=1



eps=0.2，min_pts=2



eps=0.2，min_pts=4



eps=0.2，min_pts=10

根据上面的实验结果我们可以得出结论：

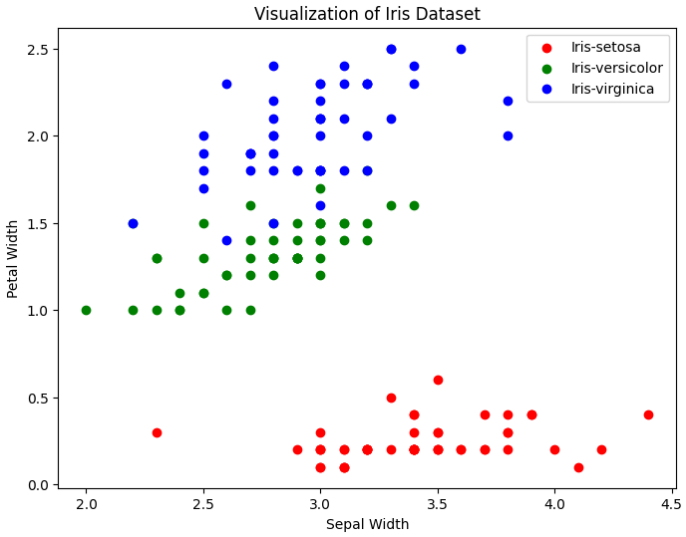
1. **minPts参数对聚类的影响：** 当eps参数保持不变，minPts参数增加时，聚类的数量会减少。这是因为较高的minPts值意味着一个点要成为“核心点”，它周围必须有更多邻居。因此，只有当一个区域的点足够密集时，才能形成一个聚类。
2. **噪声点的识别：** 当minPts设置得较高时，更多的点会被识别为噪声。这是因为在较高的minPts值下，只有少数点能够满足成为核心点的条件，导致更多的孤立点或密度较低的区域被标记为噪声。

通过综合调整 `eps` 和 `minPts` 参数，我们发现它们之间存在一种权衡关系。较小的 `eps` 值和较小的 `minPts` 值可能导致过度分割，而较大的 `eps` 值和较大的 `minPts` 值可能导致欠分割。实验中我们也注意到，聚类结果的稳定性对参数选择非常敏感。微小的参数变动可能导致聚类数目和结构的显著变化。

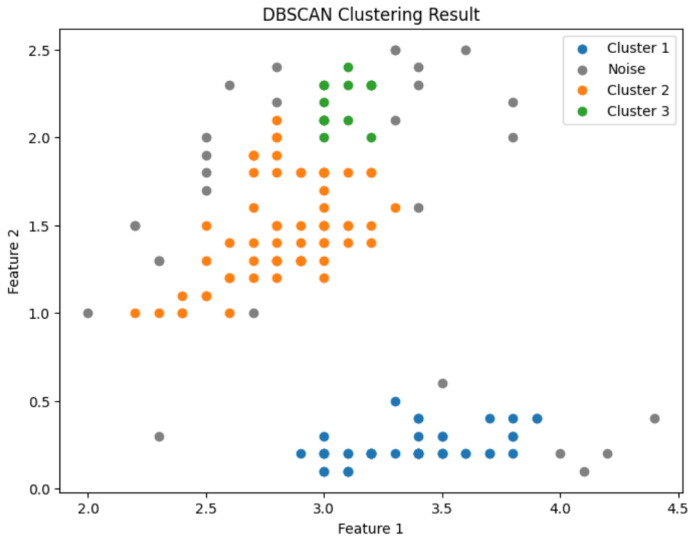
因此，参数的选择应基于对数据集特性的理解和聚类目标的具体要求。

聚类结果可视化

和标签相比，我们发现当eps=0.143且min_pts=5的时候正好可以将数据集分成3类，和原来的标签比较相近。



鸢尾花数据集的原始分类



eps=0.143, min_pts=5