ML

LAB 3 report

王一鸣

PB21000024

目录

实验目的

实现逻辑斯蒂回归

实验环境

python3.11

windows11H23

实验步骤

- 设计算法
- 验证数据

算法的解析

对于本算法，我们首先考虑的是找到一系列聚类中心，这些聚类中心有一些特点，首先这些聚类中心是某个簇的代表元，由于其在簇中，所以必然地这些聚类中心的密度很高。但考虑到一个簇里面有很多个密度很高的点，我们应该怎么样从一个簇里面只挑一个代表元呢，答案是我们选取一个簇里面密度最高的点作为代表元，如何做到这一点？由于每一个簇中都会有一个点处于密度的极大值，其它点密度都要小于这个点，那么必然地比这个极大值还要大的点一定会离这个极大值比较远，而簇中其它点由于簇中存在极大值，又离这个值很近，于是自然而然的就被区分开来了，这就是本算法的设计思路

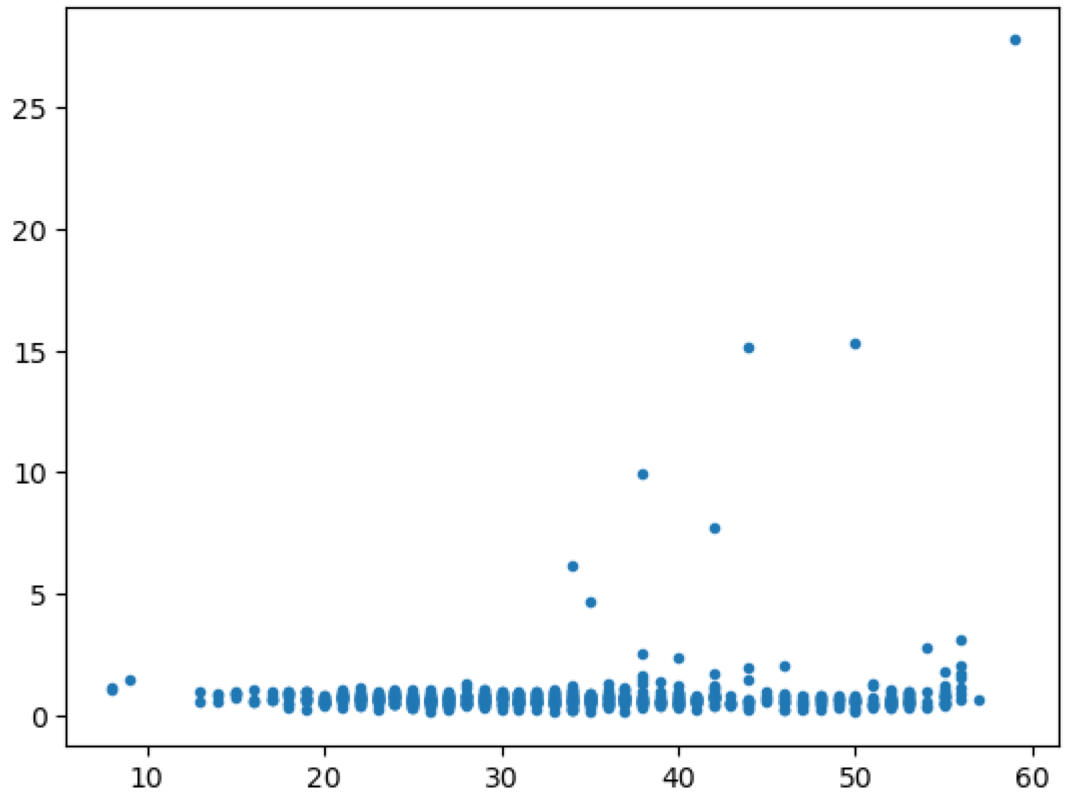
算法的实现

```
class DensityBasedCluster:
    def __init__(self, vectors, distance_threshold,C):
        self.C=C
        self.vectors = np.array(vectors)
        self.threshold = distance_threshold
        self.densities = self.calculate_densities()
        self.distances = self.find_density_and_distance()
        plt.plot(self.densities,self.distances,'.')
        plt.show()
        plt.close()
        self.center=self.select_center()
        self.labels=self.cluster()
        self.plot()
    def refresh(self):
        self.densities = self.calculate_densities()
        self.distances = self.find_density_and_distance()

        self.center=self.select_center()
        self.labels=self.cluster()
        plt.plot(self.densities,self.distances,'.')
        plt.show()
        plt.close()
        self.plot()
    def calculate_densities(self):
        densities = []
        for point in self.vectors:
            count = sum(np.linalg.norm(point - other_point) <= self.threshold
                        for other_point in self.vectors)
            densities.append(count)
        return np.array(densities)

    def find_density_and_distance(self):

 ################################################################################
        #重点在这里
        #########################################################################
        #对密度进行排序得到索引
        sorted_indices = np.argsort(-self.densities)
        results = [None] * len(self.vectors)
        distances = [None] * len(self.vectors)
        # 对于密度最大的点，找到最远的点
        max_density_index = sorted_indices[0]
        farthest_distance = max(np.linalg.norm(self.vectors[max_density_index] -
point) for point in self.vectors)
        results[max_density_index] = (self.densities[max_density_index],
farthest_distance)
        distances[max_density_index] = farthest_distance

        #对于剩下的点，检查比其密度高的点
        for i, index in enumerate(sorted_indices[1:], 1):
            min_distance = float('inf')
            nearest_index = None
            for higher_density_index in sorted_indices[:i]:
                distance = np.linalg.norm(self.vectors[index] -
self.vectors[higher_density_index])
                if distance < min_distance:
                    #找到那个距离最近的点，算出距离
                    min_distance = distance
                    nearest_index = higher_density_index

            results[index] = (self.densities[index], min_distance) if
nearest_index is not None else (self.densities[index], None)
            distances[index] = min_distance if nearest_index is not None else
None
        return distances

        pass
    def cluster(self):
        labels=np.zeros(len(self.vectors))
        # 先为中心点标记
        flag=1
        for i in self.center:
            labels[i]=flag
            flag+=1
        # 为每个点标记
        for i in range(len(self.vectors)):
            if labels[i]==0:
                # 找到最近的中心点
                min_distance=float('inf')
                nearest_center=None
                for center in self.center:
                    distance=np.linalg.norm(self.vectors[i]-self.vectors[center])
                    if distance<min_distance:
                        min_distance=distance
                        nearest_center=center
                #print(nearest_center)
                labels[i]=labels[nearest_center]
        return labels
    def is_center(self,distance,density):
        #观察分布图对二元值进行计算确定聚类中心的选择
        if distance>self.C:
            return True
        else:
            return False
    def select_center(self):
        center=[]
        for i in range(len(self.vectors)):
            #print(self.distances[i],self.densities[i])
            if self.is_center(self.distances[i],self.densities[i]):
                center.append(i)
        return center
```

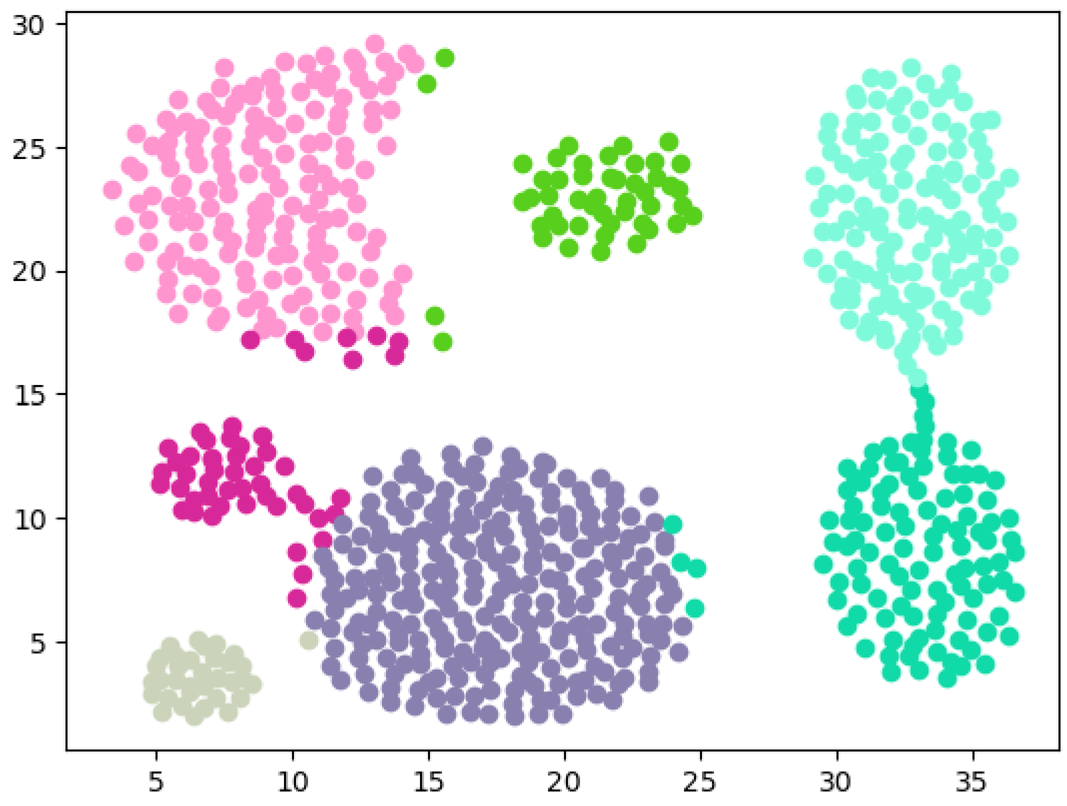
实验结果与分析

Aggregation.txt

```
with open('./Datasets/Aggregation.txt', 'r') as f:
    lines = f.readlines()
    vectors = []
    for line in lines:
        vector = [float(x) for x in line.strip().split(' ')]
        vectors.append(vector)
cluster = DensityBasedCluster(vectors, 2.8075,4)
#results = cluster.find_density_and_distance()
print(davies_bouldin_score(vectors, cluster.labels))
```

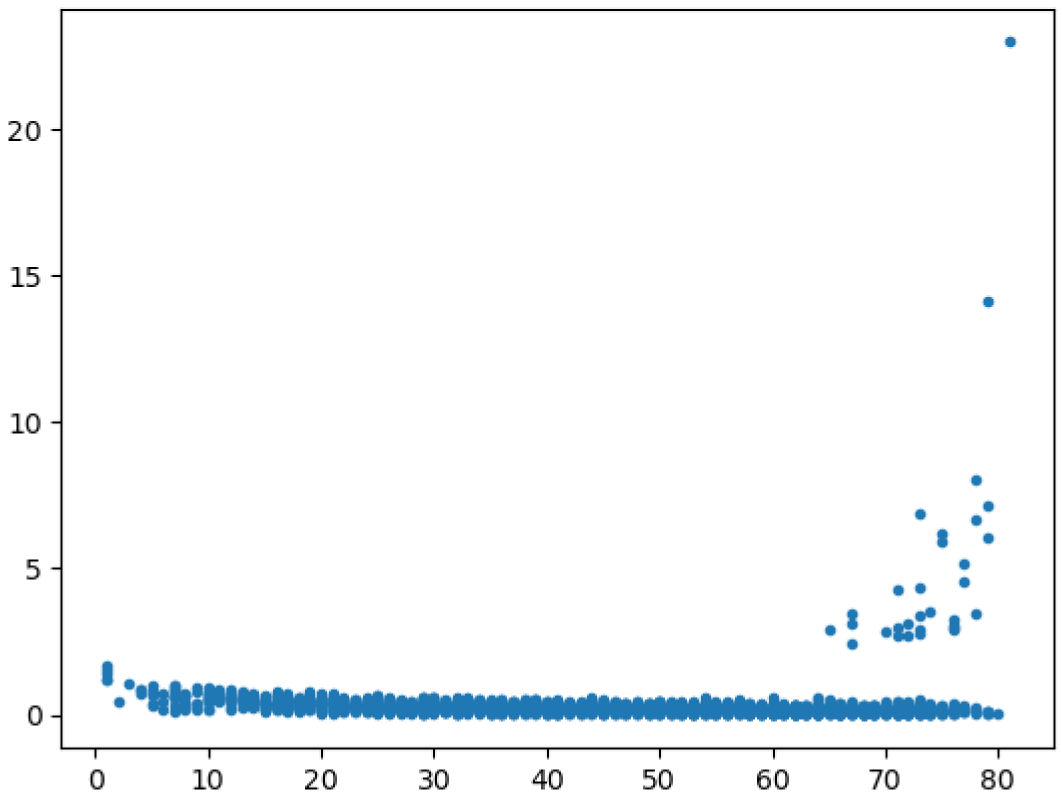


```
0.572229572344525
```

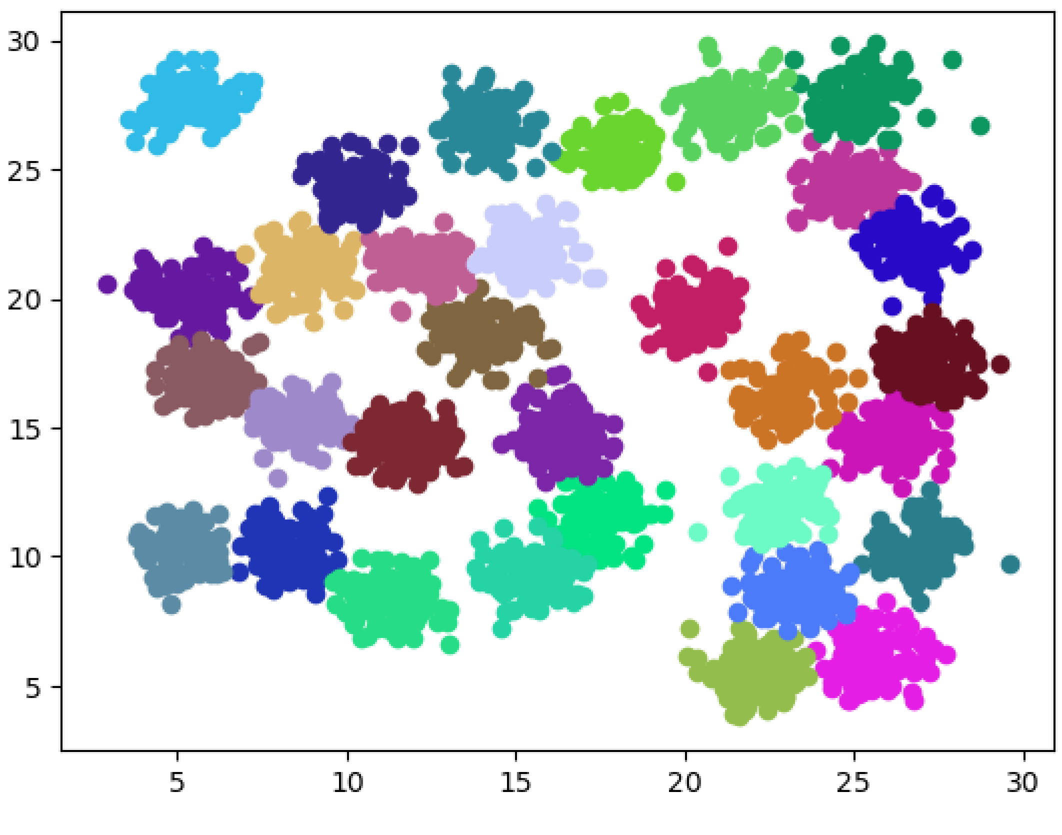


D31.txt

```
with open('./Datasets/D31.txt', 'r') as f:
    lines = f.readlines()
    vectors = []
    for line in lines:
        vector = [float(x) for x in line.strip().split(' ')]
        vectors.append(vector)
cluster = DensityBasedCluster(vectors, 1.2,2)
#results = cluster.find_density_and_distance()
print(davies_bouldin_score(vectors, cluster.labels))
```

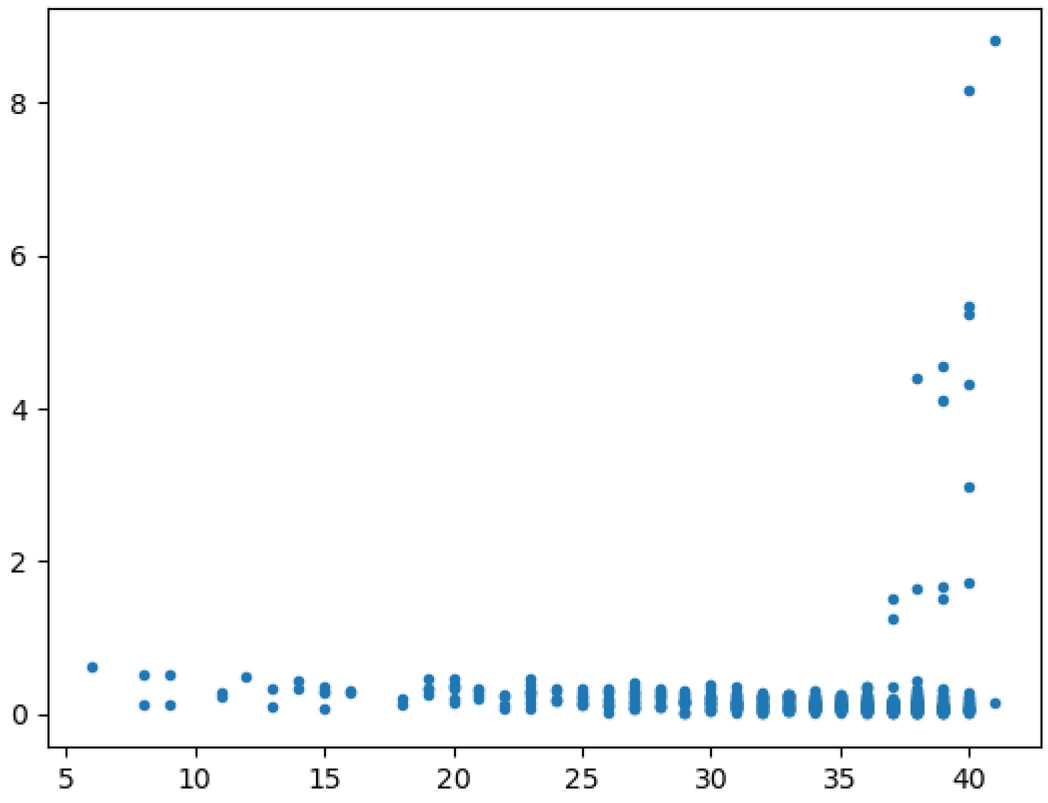


```
0.5470575289832145
```

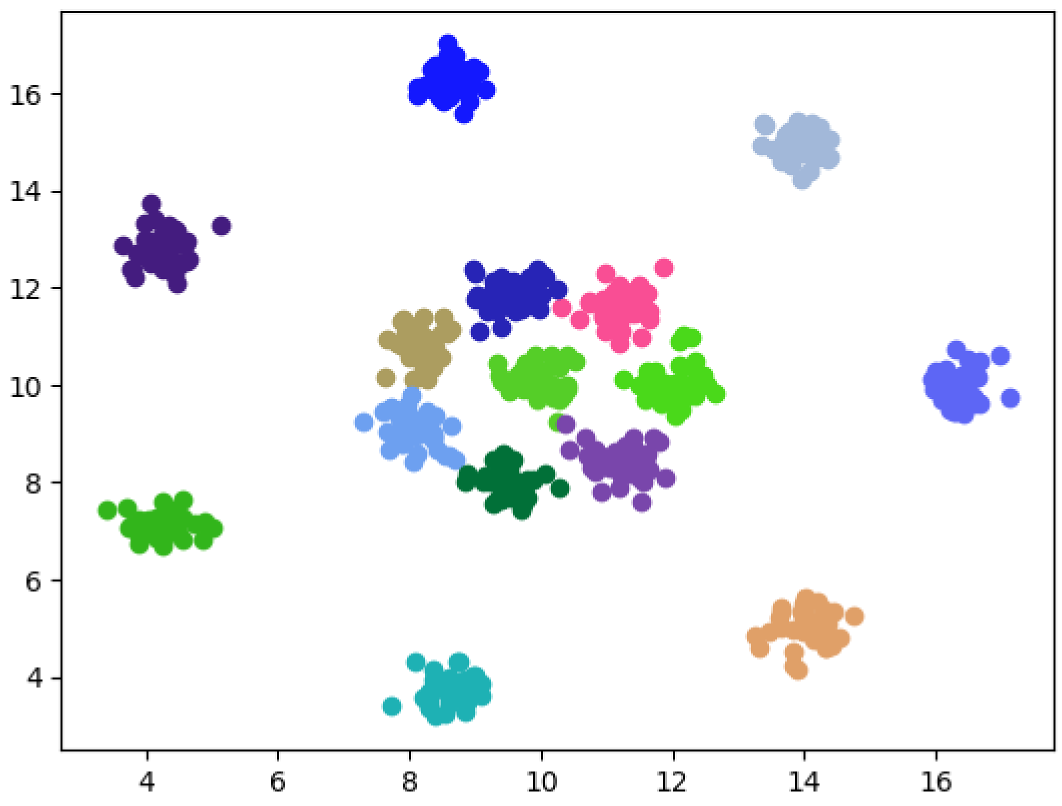


R15.txt

```
with open('./Datasets/R15.txt', 'r') as f:
    lines = f.readlines()
    vectors = []
    for line in lines:
        vector = [float(x) for x in line.strip().split(' ')]
        vectors.append(vector)
cluster = DensityBasedCluster(vectors, 0.8,1)
#results = cluster.find_density_and_distance()
print(davies_bouldin_score(vectors, cluster.labels))
```



```
0.31611506739556305
```



算法的一些局限性

由于计算密度时需要依赖选取的范围，所以聚类的效果严重依赖于所取的参数，而这依赖于我们对于数据分布的观察。同时，确定聚类中心的算法也是难以直接定义的，需要手动的去调整来判断一个点是否足够显著的作为聚类中心，这导致在进行实验的时候需要人来观察数据的分布，以此来确定两个超参数的取值。

可以看到，算法对于簇的大小比较一致的时候工作地比较好，对于第一个数据样本，簇大小很不均匀地时候我们很难把大的簇和小地簇用同一种方法进行划分，这提示我们可以改进密度地计算方法，从而更自然地得到计算簇密度，提升聚类效果。其次是动态地选择一种挑选聚类中心地算法，能够在二维图上把稀疏地，处于右上方地点和稠密地聚集在坐标轴附近地点区分开来，这样这个算法应该就会有更好地使用价值。