Report

实验目的

编写DPC算法，实现聚类

实验原理

DPC算法整合了**k-means** 算法和 **DBSCAN**算法，中心思想是：

- 每个聚类的边缘密度较低，中间密度高
- 聚类中心的点通常离其他高密度的点远

所以基于此，采取如下策略：

- 设置距离阈值变量d_c
- 对于每个数据点i，计算两个值：
  - 局部密度：$\rho_i = \sum_j \chi(d_{ij} - d_c)$，其中 $\chi(x) = 1 \text{ if } x < 0 \text{ and } \chi(x) = 0 \text{ if } x \geq 0$;
  - 更高密度距离：$\delta_i = \min_{j:\rho_j > \rho_i} d_{ij}$。特别的，若当前数据点 m_i 为 ρ 最大点，则定义 $\delta_i = \max_j d_{ij}$
- 绘制决策图，把ρ_i 和δ_i均较大的点设为聚类中心，把δ_i大而ρ_i小的点设置为OOD点
- 按照密度从高到低排序，把每个点分配给最近的有更高密度的聚类中心的类

实验步骤

Load and Process Data

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df = {}
df[0] = pd.read_csv('Datasets\Aggregation.txt', header=None, sep=" ")
df[1] = pd.read_csv('Datasets\D31.txt', header=None, sep=" ")
df[2] = pd.read_csv('Datasets\R15.txt', header=None, sep=" ")
```

```
for i in range(3):
    min_val = df[i].min()
    max_val = df[i].max()

    df[i] = (df[i] - min_val) / (max_val - min_val)

array = {}
for i in range(3):
    array[i] = df[i].to_numpy()
```

Fit

Calculate distance

计算ρ_i 和δ_i都要用到点之间的距离，这里用矩阵乘法来代替循环的计算

```
    def _calculate_distance(self, X):
        m, n = X.shape
        X_square = np.square(X)
        ones = np.ones((m, n))
        D_square = X_square @ ones.T + ones @ X_square.T - 2 * X @ X.T
        self.D = np.sqrt(np.abs(D_square))
```

Calculate_delta

按实验原理的公式计算

```
    def _calculate_delta(self, X):
        m = X.shape[0]
        self.delta = np.zeros(m)

        for i in range(m):
            if self.rho[i] == self.rho.max():       # highest density
                self.delta[i] = np.max(self.D[i, :])
            else:
                min_distance = np.inf
                for j in range(m):
                    if self.rho[j] > self.rho[i]:
                        if min_distance > self.D[i][j]:
                            min_distance = self.D[i][j]
                self.delta[i] = min_distance
```

Calculate_local_density

按公式计算

```
    def _calculate_local_density(self, X):
        m = X.shape[0]
        self.rho = np.zeros(m)

        for i in range(m):
            for j in range(m):
                distance = self.D[i][j]
                if distance < self.dc:
                    self.rho[i] += 1
            self.rho[i] = self.rho[i] + np.random.rand()/1000
```

Pridict

确定聚类中心和OOD点，然后按照密度从高到低排序，把每个点分配给最近的有更高密度的聚类中心的类

```
    def predict(self, X, rho_bound, delta_bound):
        # center
```

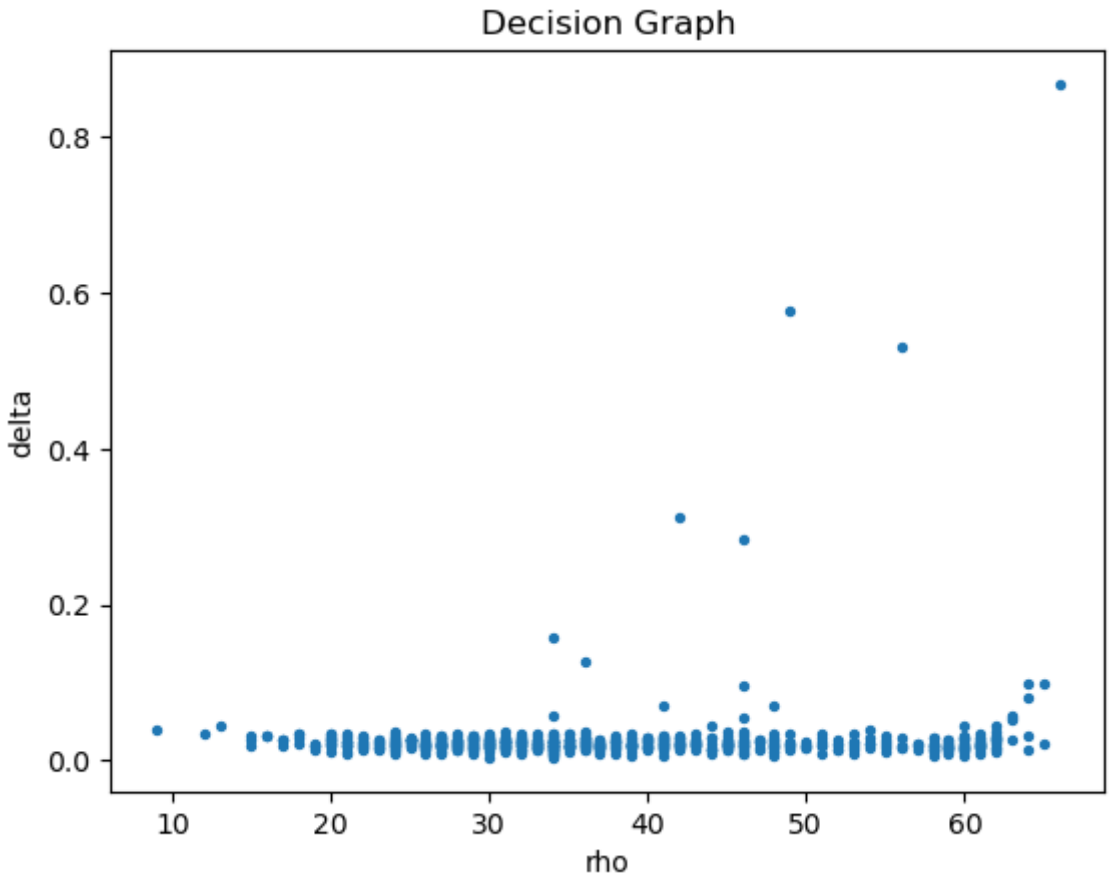
```
        self.center = []
        self.outliers = []
        m = X.shape[0]
        for i in range(m):
            if self.rho[i] >= rho_bound and self.delta[i] >= delta_bound:
                self.center.append(i)
            elif self.rho[i] < rho_bound and self.delta[i] >= delta_bound:
                self.outliers.append(i)

        self.labels = np.zeros(m)
        for i in range(len(self.center)):
            self.labels[self.center[i]] = i + 1

        rho_index = np.argsort(-self.rho)

        for i in range(m):
            if (not rho_index[i] in self.outliers) and self.labels[rho_index[i]]
== 0:
                cluster_index = rho_index[np.argmin(self.D[rho_index[i]]
[rho_index[:i]])]
                self.labels[rho_index[i]] = self.labels[cluster_index]

        plt.figure()
        plt.title('DPC')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.scatter(X[:, 0], X[:, 1], c = self.labels, cmap = "jet")
        plt.scatter(X[self.center][:, 0], X[self.center][:, 1], c = 'black',
marker = 'x')
        plt.show()

        return self.labels
```

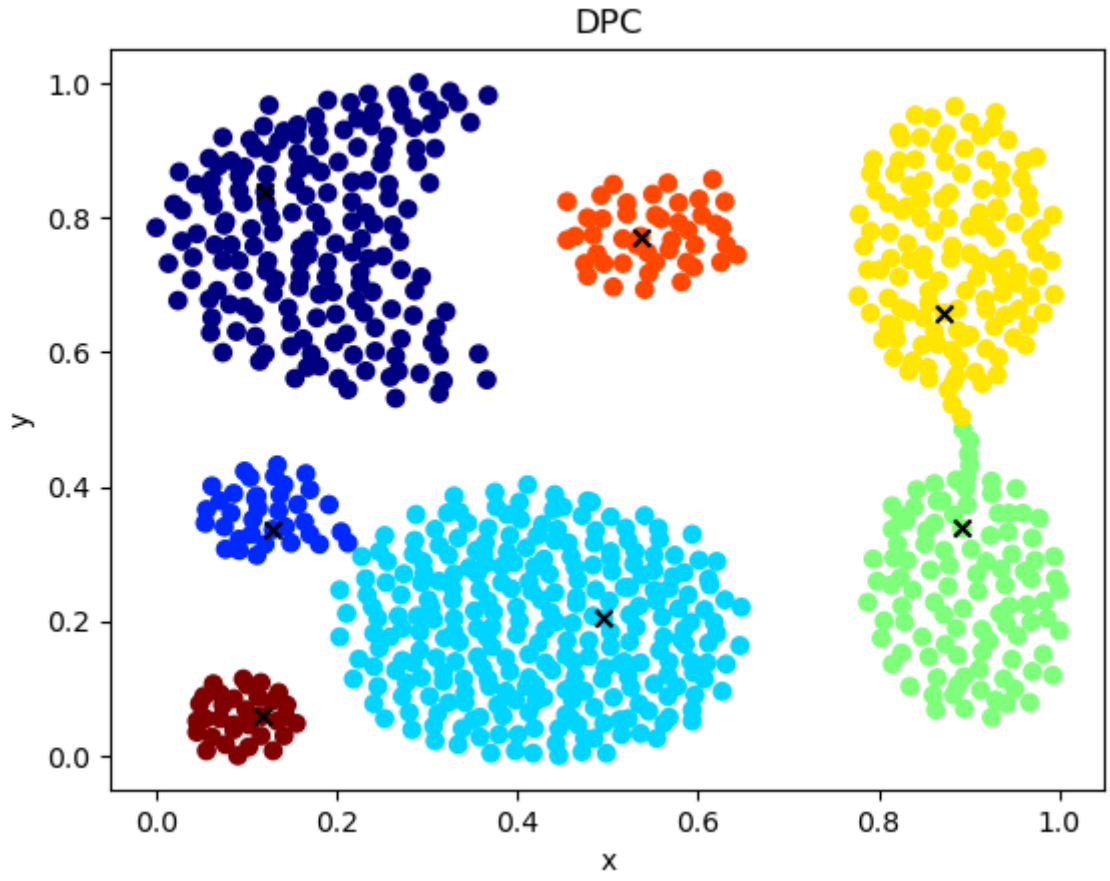
实验结果

Aggregation.txt数据集

参数 dc = 0.1, rho_bound = 30, delta_bound = 0.115

DBI : 0.5461285787085287

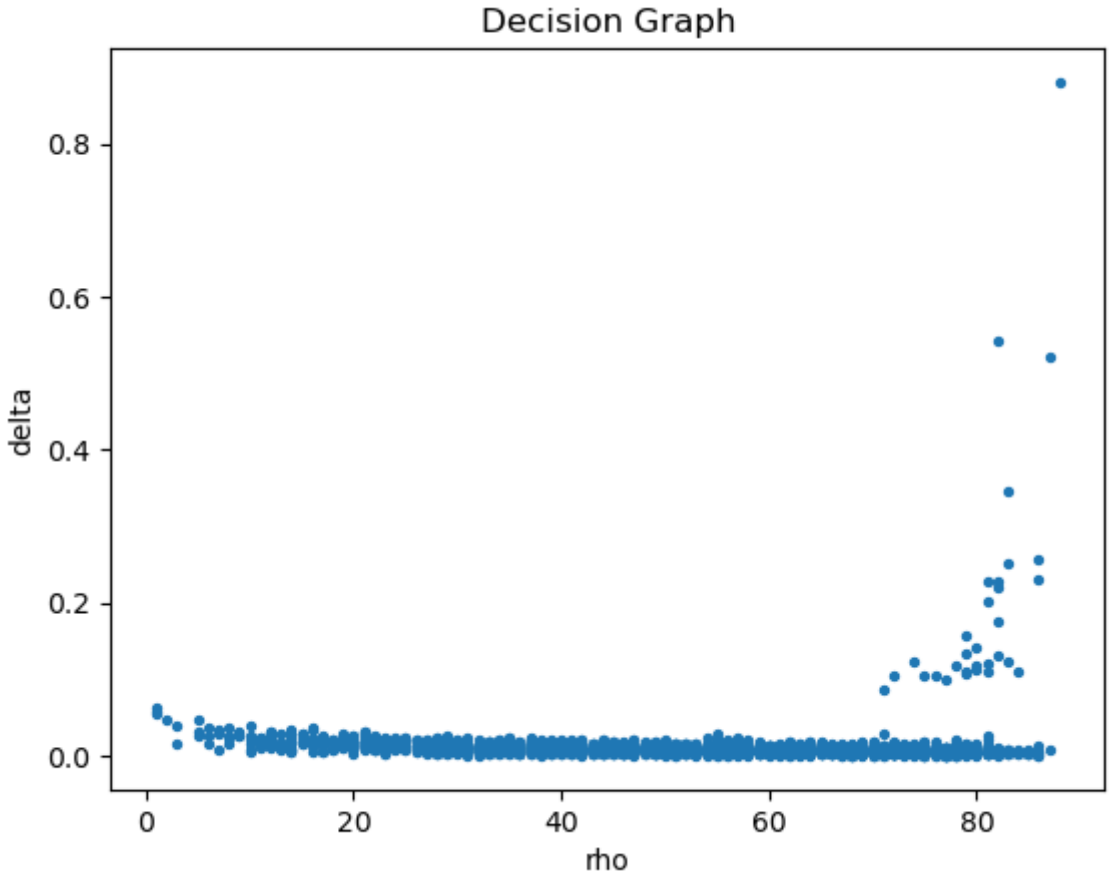

Decision Graph
delta
rho
0.0
0.2
0.4
0.6
0.8
10
20
30
40
50
60

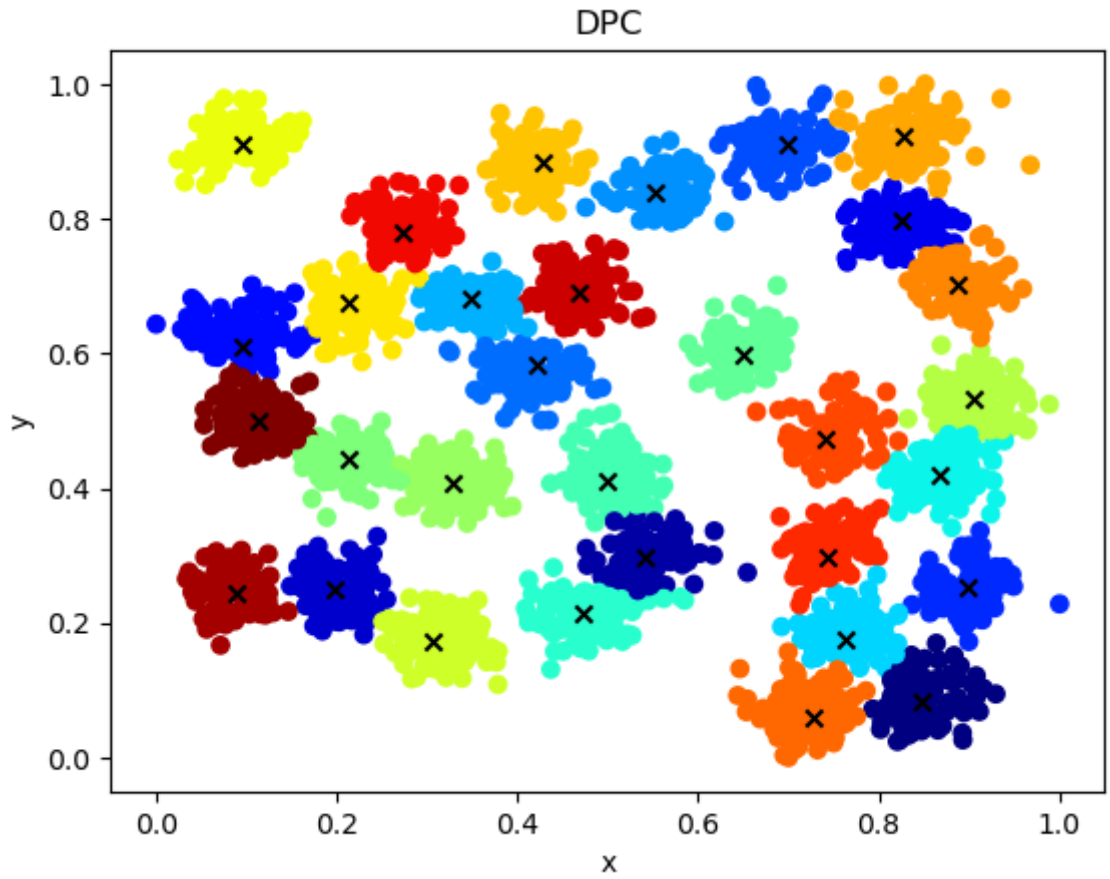

DPC
y
x
0.0
0.2
0.4
0.6
0.8
1.0

D31.txt数据集

参数 dc = 0.05, rho_bound = 60, delta_bound = 0.08

DBI : 0.5521069688311097

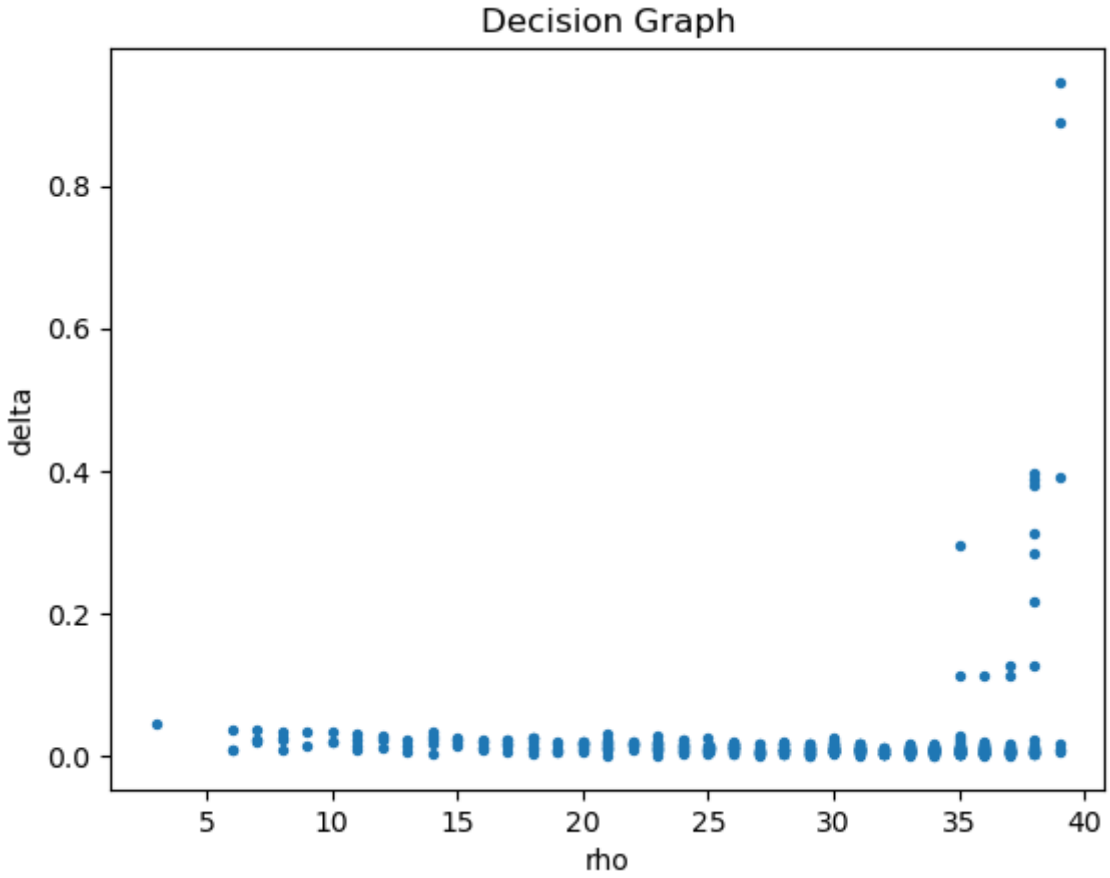


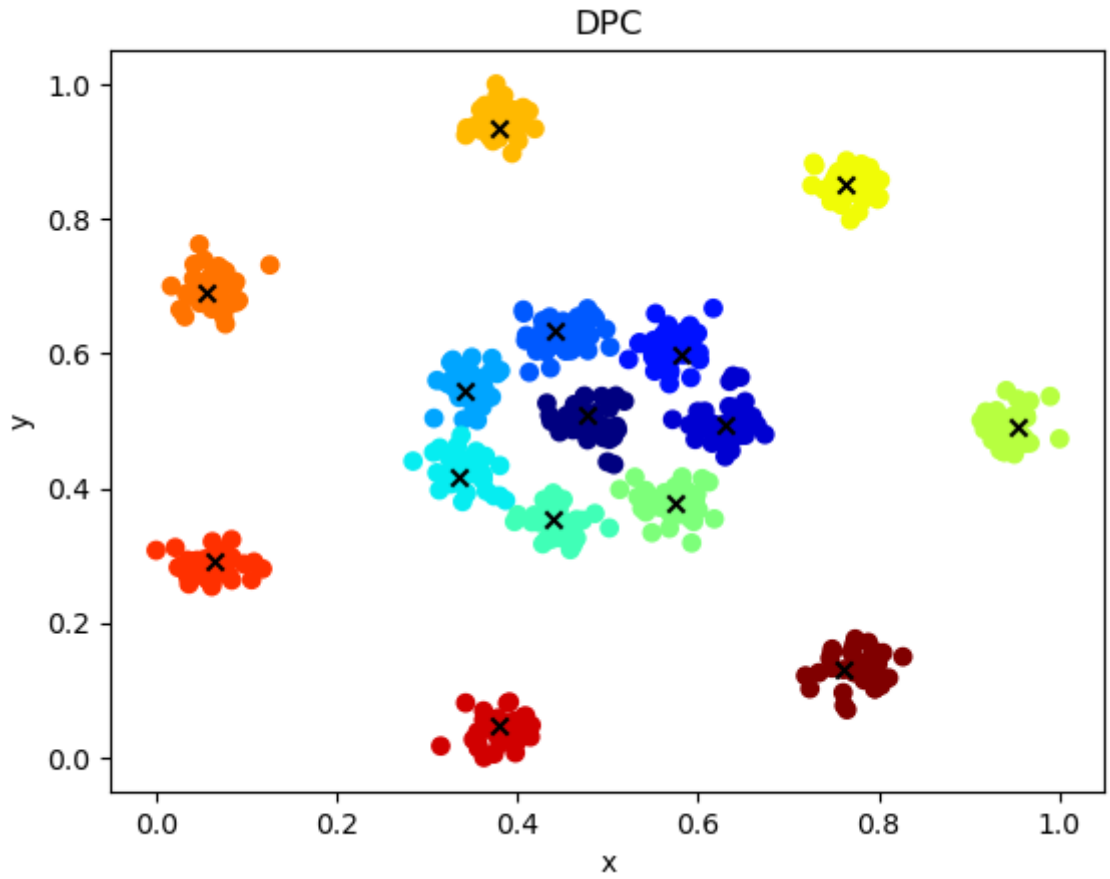


R15.txt数据集

参数 dc = 0.05, rho_bound = 30, delta_bound = 0.08

DBI : 0.31471445116423397





和K-means比较

	DPC	K-means
Aggregation	0.5461285787085287	0.7050277591438034
D31	0.5521069688311097	0.5470674995427992
R15	0.31471445116423397	0.31471445116423397

可以看出，DPC在一些特定的数据集上表现明显好于K-means算法，比如Aggression中，样本点分布的并不均匀且形成的聚类不是圆。在D31和R15上这两个算法的表现则相近

实验结果分析

DPC算法整合了k-means 算法和DBSCAN算法，可以处理非圆形(球形)的簇以及重叠度高的簇，通过选取合适的参数可以达到非常好的效果，比传统的聚类算法更加优越