ML Lab 4

DPC（密度峰值聚类）

王世炟

PB20151796

December 8, 2022

Part 1: 实验要求

本次实验要求复现现论文Clustering by fast search and find of density peaks，并在给定数据集上进行聚类。具体需要完成以下部分：

- 通读论文了解算法原理
- 实现 DPC 密度峰值聚类算法
- 调整参数，优化聚类效果
- 进行可视化
- 与其他聚类方法进行对比

Part 2: 数据集介绍

本次实验的数据集有 3 个，为 Aggregation.txt、D31.txt、R15.txt, 分布如下：

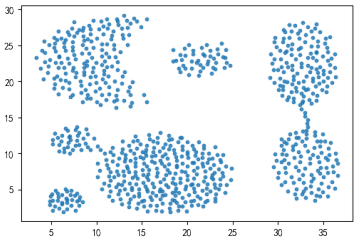

图 1: Aggregation 数据集

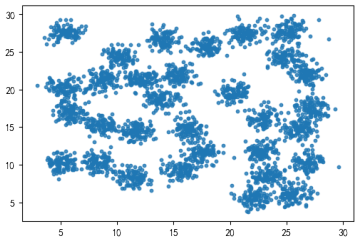

图 2: D31 数据集

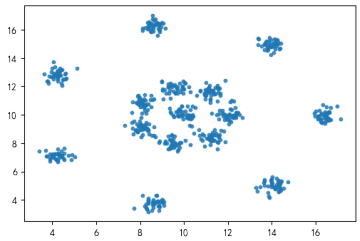

图 3: R15 数据集

可以看出，Aggregation 数据集大致可分为 7 个簇，D31 数据集大致可分为 31 个簇，R15 数据集大致可分为 15 个簇。

Part 3: 理论基础

3.1 算法简介

DPC 算法是由 Alex Rodriguez 和 Alessandro Laio 在 2014 年发表在 Science 上的一种聚类算法，它结合了 kmeans 算法与 DBSCAN 算法的优点，并且达到了更好的效果。
DPC 算法基于两个假设：

- 簇中心（密度峰值点）的局部密度大于围绕它的邻居的局部密度；
- 不同簇中心之间的距离相对较远。

这两个假设在常见的聚类问题中是很容易满足的。
在 kmeans 族算法中，一个数据点总是被分配到最近的中心，这类方法不能检测到非球形的簇。
而基于局部数据点密度的方法 (DBSCAN) 可以很容易地检测出任意形状的簇，但是对于阈值的选择却是麻烦的，且计算量（复杂度）很高。

3.2 度量定义

在 DPC 算法中，对于局部密度的定义如下：

$$\rho_i = \sum_j \chi(d_{ij} - d_c), \qquad 其中\chi(x) = \begin{cases} 1 & if \quad x < 0 \\ 0 & otherwise \end{cases}$$

事实上，这个对于密度的定义指的就是在点 i 的 d_c 邻域内的点的个数。论文中提到，d_c 的选择是由样本数决定的，要求 d_c 满足使平均每个点的 d_c 邻域内的点的个数占数据集总样本数的 $1\% \sim 2\%$。
DPC 算法中还有这样一个度量——**相对距离** δ,δ 的定义如下：
对于密度最高的样本，δ 定义为：

$$\delta_i = \max_j(d_{ij})$$

对于其余点：

$$\delta_i = \min_{j:\rho_j > \rho_i}(d_{ij})$$

这样定义的意义是：δ_i 表示样本 i 与其他密度更高的点之间的最小距离，并且 DCP 认为密度最高的点一定是一个簇中心，所以人为设定其 δ 为与其他点距离的最大值。

3.3 簇中心的选择

DPC 算法将**局部密度** ρ **较高，相对距离** δ **较大**的样本点作为聚类中心（密度峰值）。论文中给出了两种方法来选出聚类中心：

1. 画出 ρ—δ 图，选择右上方的点作为聚类中心：

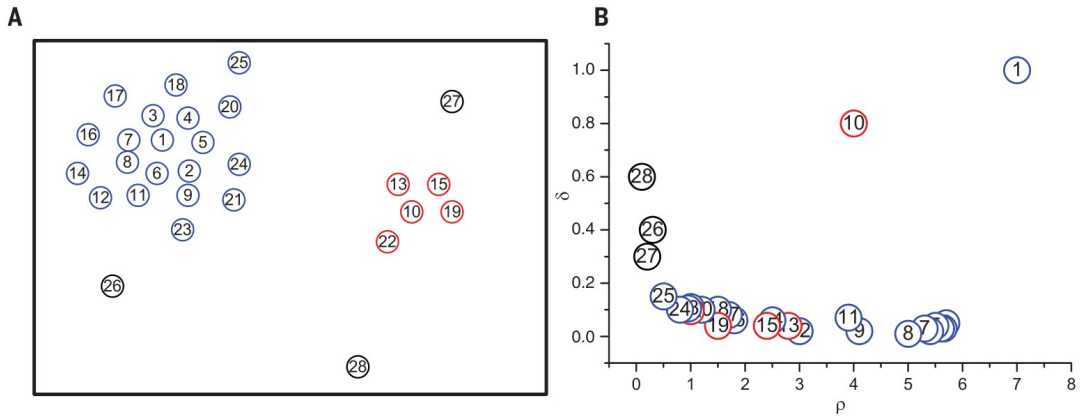

图 4: Fig. 1.The algorithm in two dimensions. (A) Point distribution. Data points are ranked in order of decreasing density. (B) Decision graph for the data in (A). Different colors correspond to different clusters.

在上图中，选择 1 号和 10 号作为簇中心进行聚类。

2. 当 ρ—δ 图不容易清晰选择时，设决策值 $\gamma_i = \rho_i \times \delta_i$, 然后将数据按 γ 排序，选择 γ 较大的点作为簇中心。

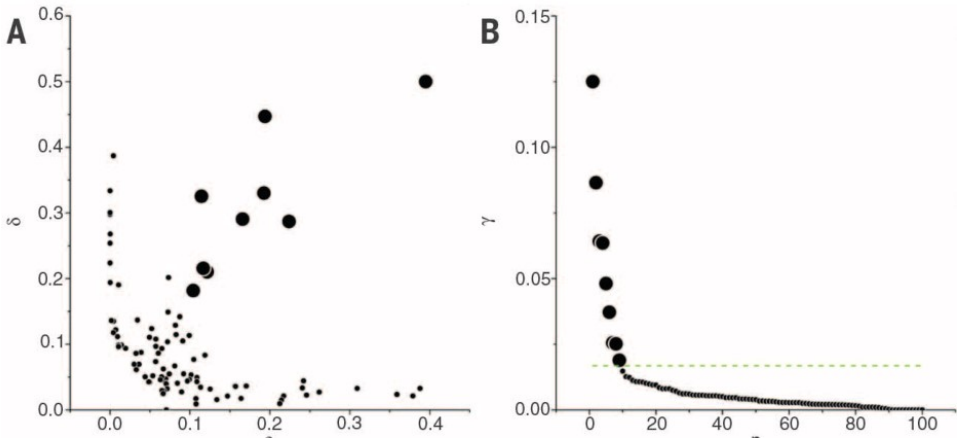

图 5: γ

在上图中选择 B 中绿线上方的点作为簇中心。

3.4 为样本点划分簇

DCP 是这样为样本点划分簇的：先对选出的每一个簇中心创建一个簇，然后对于一个不是簇中心的样本点，它与密度 (ρ) 比它大的点中与它距离最近的点同属一个簇。这样会形成多个从密度峰值出发的树状结构，每一个树状结构代表一个类簇。

3.5 噪声点

在本论文中提到了两种可能的噪声点：
1. 拥有较高的 δ 和较低的 ρ 的样本点是 “**被孤立的**” (isolated), 它们可以被视为 “**离群点**” (outliers)。
2. 一个簇中的点有两种，分别是 “**核心**” 和 “**光晕**” (halo)。核心点的密度比较大，位于簇的内部；光晕的密度比较小，位于簇的边缘。具体来说，判断是否为光晕的准则是：一个样本点的密度如果小于它所在簇的边界区域的平均密度，那么它是一个光晕点。(关于边界区域的说明详见 5.2)。

Part 4: 实验步骤

4.1 根据样本数据集计算距离矩阵 D

```
def Distance(self,data):
    m = data.shape[0]
    self.distance = np.zeros([m,m])
    self.distance = np.linalg.norm(data-data[:,None],axis=-1)
    return
```

利用 numpy 中的 linalg.norm() 函数计算距离矩阵，可以显著提高运算效率。

4.2 确定截断距离 d_c

```
def get_d_c(self,percent):
    n = int(percent * (len(self.distance)**2))
    dis_plus = np.sort(np.reshape(self.distance,len(self.distance)**2))
    self.d_c = dis_plus[n]
    return
```

为了满足对于 d_c 选取的要求，首先将距离矩阵展开，然后排序，选取第 $1\% \sim 2\%$ 的距离值作为 d_c。

4.3 计算局部密度 ρ 和相对距离 δ

```
def LocalDensity(self):
    m = self.distance.shape[0]
    self.rho = np.zeros(m)
    for i in range(m):
        self.rho[i] = np.sum(np.where(self.distance[i,:] < self.d_c, 1, 0))
    return

def Delta(self):
    m = self.distance.shape[0]
    delta = np.zeros(m)
    nearest_neighbor = np.zeros(m)
    rho_index = np.argsort(-self.rho) # 密度降序的索引
    for i in range(m):
        if i == 0:
            delta[rho_index[i]] = np.max(self.distance[rho_index[i],:])
            continue
        # 比当前点密度大的序号
        greater_index = rho_index[:i]
        # 当前点的高密度最小距离
        delta[rho_index[i]] = np.min(self.distance[rho_index[i],
            greater_index])
        # 密度高于当前点的最近点序号
        nearest_neighbor[rho_index[i]] = greater_index[np.argmin(self.
            distance[rho_index[i],greater_index])]
    self.delta = delta
    self.nearest_neighbor = nearest_neighbor
    return
```

ρ 和 δ 的计算按照定义实现即可。其中在计算 δ 时保存了样本点的高密度最近邻，用于后续簇的划分。

4.4 绘制决策图，选取聚类中心点

```
# 画出rho-delta图
def ShowRhoDelta(self):
    plt.figure(dpi = 150)
    plt.xlabel("rho")
    plt.ylabel("delta")
    plt.title('Decision Graph')
```

```
    plt.scatter(self.rho, self.delta, c='none', edgecolors='black', marker='o', alpha=0.5)
    plt.show()
    return

# 画出gamma图
def ShowGamma(self,threshold=None):
    product = self.rho * self.delta
    if threshold == None:
        threshold = np.average(product) * 8
    plt.figure(dpi = 150)
    plt.xlabel("n")
    plt.ylabel("gamma")
    plt.title("gamma-n picture")
    plt.plot(np.arange(0,len(self.rho)/2,5),np.ones([len(np.arange(0,len(self.rho)/2,5)),1]) * threshold ,color='g',ls='--',label="threshold:"+str(threshold))
    plt.scatter(list(range(len(self.rho))), np.abs(np.sort(-product)), c='black', marker='o', s=1, alpha=0.5)
    plt.legend()
    plt.show()
    return
```

4.5 对非聚类中心样本点进行归类

```
def Clustering(self):
    for i,center in enumerate(self.centers):
        self.cluster[center] = i + 1
    rho_index = np.argsort(-self.rho)
    for i in range(len(self.rho)):
        if self.cluster[rho_index[i]] == 0:
            self.cluster[rho_index[i]] = self.cluster[int(self.nearest_neighbor[rho_index[i]])]
    return
```

Part 5: 实验结果

5.1 聚类结果与评价指标

下面是三个数据集的决策图 (选择 γ 作为决策变量) 以及聚类结果 (未标记噪音):

以下结果均是在 d_c 满足使平均每个点的 d_c 邻域内的点的个数占数据集总样本数的 1% 的情况下得到。

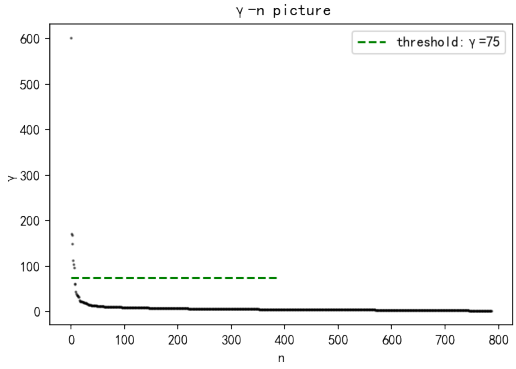

图 6: Aggregation 数据集决策图

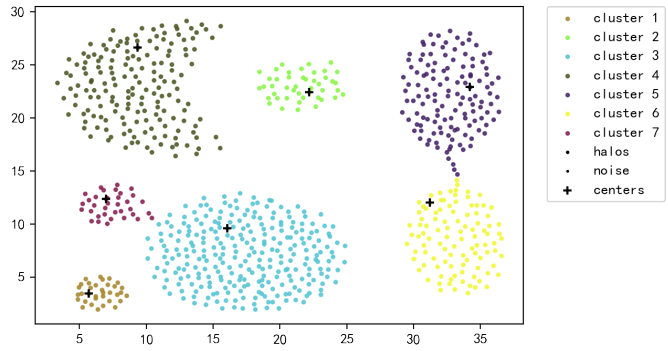

图 7: Aggregation 数据集聚类结果

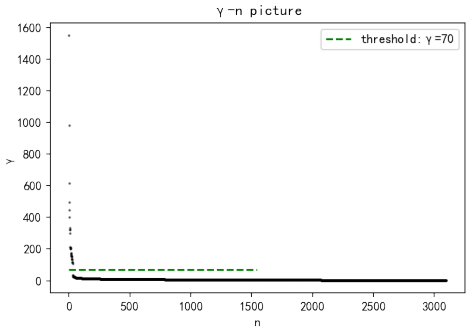

图 8: D31 数据集决策图

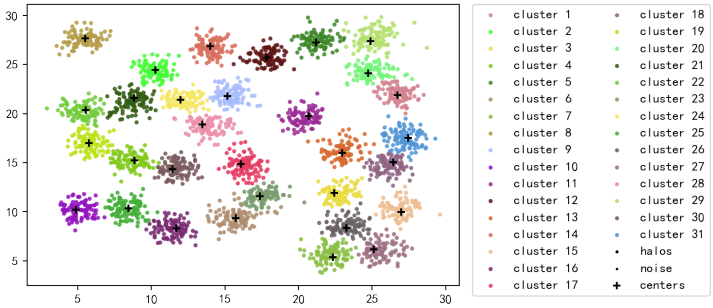

图 9: D31 数据集聚类结果

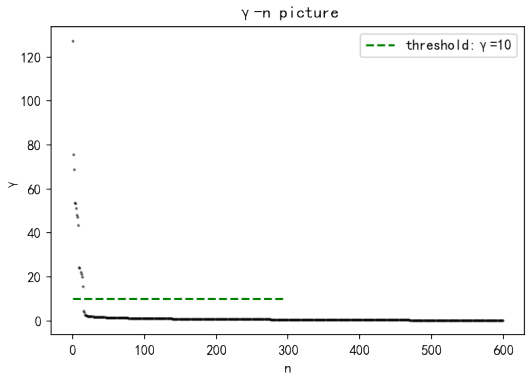

图 10: R15 数据集决策图

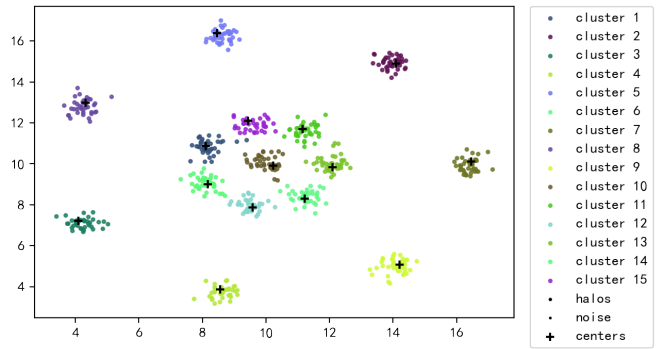

图 11: R15 数据集聚类结果

可以看出，DPC 算法将 Aggregation 数据集、D31 数据集、R15 数据集分别划分为 7、31、15 个簇，与正确结果相同，效果较好。

下面是这三个数据集聚类结果的 DBI:

表 1: 三个数据集用 DPC 聚类结果的 DBI

数据集	DBI
Aggregation	0.5066
D31	0.5510
R15	0.3143

三个 DBI 均较小，说明结果较好。

5.2 噪声

前面提到，在论文中关于噪声的定义有两种，分别是**“离群点”**和**“光晕”**，离群点的选择很容易，只需要根据 ρ—δ 图选择拥有较高的 δ 和较低的 ρ 的样本点即可，本次实验用到的三个数据集均没有特别明显的离群点，故暂不讨论，只将 ρ—δ 图展示如下：

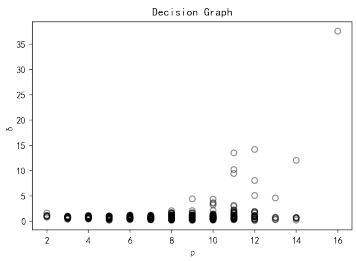

图 12: Aggregation 数据集

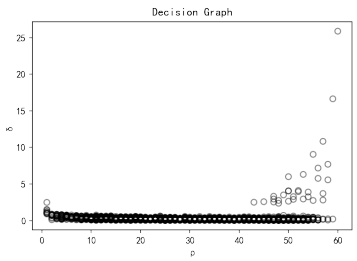

图 13: D31 数据集

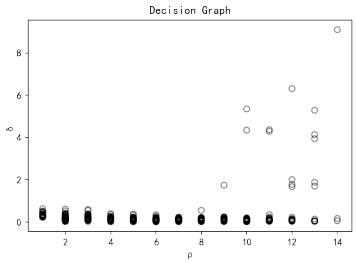

图 14: R15 数据集

对于光晕，在本实验中是这样实现的：

```
# 光晕点
def Halo(self):
    bord_rho = np.zeros(self.k)
    self.halos = np.zeros(len(self.rho))
    for i in range(len(self.distance)):
        for j in range(len(self.distance)):
            if (self.distance[i][j] < self.d_c) and (self.cluster[i] !=
                self.cluster[j]):
                rho_aver = (self.rho[i] + self.rho[j]) / 2
                if (rho_aver>bord_rho[int(self.cluster[i])-1]):
                    bord_rho[int(self.cluster[i])-1]=rho_aver
                if (rho_aver>bord_rho[int(self.cluster[j])-1]):
                    bord_rho[int(self.cluster[j])-1]=rho_aver

    for i in range(len(self.rho)):
```

```
15            if self.rho[i] < bord_rho[int(self.cluster[i])-1]:
16                self.halos[i] = 1
```

先定义位于边界的点为在其 d_c 邻域内有与它不属于同一个簇的点。这一点很容易理解，因为边界点一般位于两个簇的边界部分。然后选取一个簇边界点密度与其他簇的边界点密度的平均值的最大值作为该簇的一个阈值，如果在簇内有比这个阈值密度小的点，则被判定为光晕点。这样做一般可以将接近的两个簇的边界点找出来，若将其作为噪声，则会减小这些位于边界的点对于簇划分造成的干扰。

下面给出三个数据集画出光晕点的结果：

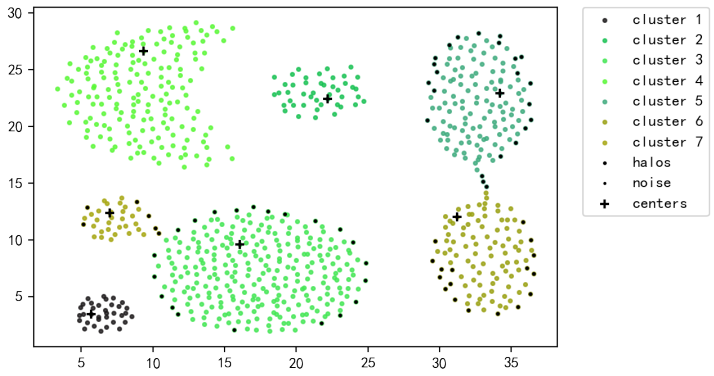

图 15: Aggregation 数据集

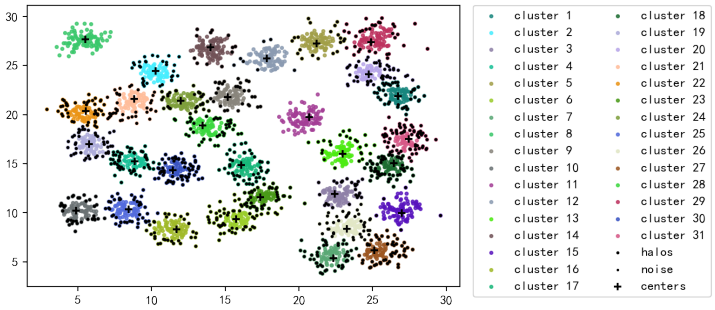

图 16: D31 数据集

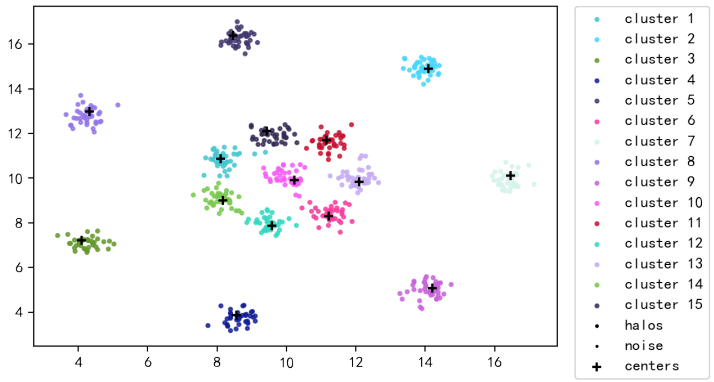

图 17: R15 数据集

可见，“光晕”确实如同光晕一样将簇包围起来，将其作为噪声也是合理的。但是本次实验所用的数据集

5.3 对比 kmeans 与 dbscan

在给出正确 k 值的 kmeans 算法对三个数据集的结果:

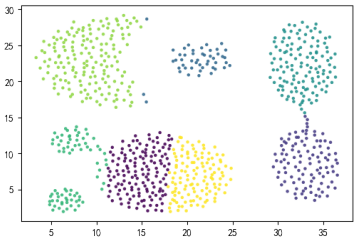

图 18: Aggregation 数据集

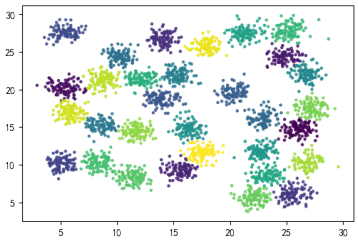

图 19: D31 数据集

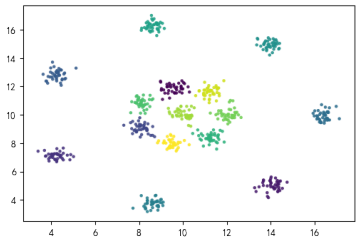

图 20: R15 数据集

可见 kmeans 算法对于 Aggregation 数据集的聚类效果不理想，因为这个数据集簇的大小有明显差别。这是由于其本身只用距离作为类别划分的标准导致的。而对于其他两个数据集，在给出正确 k 值的时候效果还不错。

在调整适当的 ϵ 值的 dbscan 算法对三个数据集的结果:

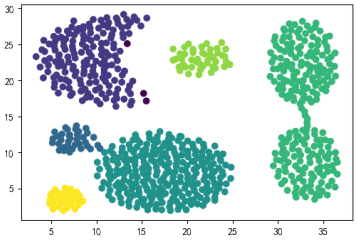

图 21: Aggregation 数据集

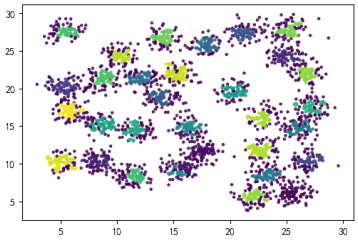

图 22: D31 数据集

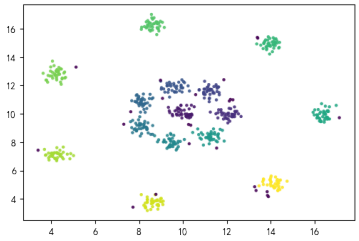

图 23: R15 数据集

dbscan 算法由于其是通过密度可达来扩展簇的，所以其对于簇之间分隔不明确的情况聚类效果不理想。
而 DPC 算法结合了 kmeans 与 dbscan，可以取得相当不错的效果。

Part 5: 实验总结

通过对于 2014 年 Science 上的一篇关于聚类算法的复现，我了解到近年聚类算法的发展，效果的提升。解决了我对于聚类算法很多的疑问与困惑。并对于无监督学习这个领域未来的发展充满期待。