数学软件第二次大作业

——基于密度峰值的聚类算法

作　　者: 张晋
学　　号: 15091060
学　　院: 数学与系统科学学院
学　　校: 北京航空航天大学
指导教师: 夏勇

目录

1 问题背景

聚类 (Clustering) 算法基于元素之间的相似度将元素分为不同的类簇。

在 K-means 和 K-medoids 算法中，每一个类簇都是由离聚类中心距离很小的数据点组成，目标函数为各样本点到对应的聚类中心的距离之和，通过反复更新聚类中心和样本点所属的聚类来进行迭代直至收敛。K-means 和 K-medoids 之间的区别在于，K-means 的聚类中心为属于该类簇的所有样本点的均值，而 K-medoids 的聚类中心为该类簇中离所有样本点的聚类之和最小的样本点。而它们的缺点也很明显：不能检测非球形 (non-spherical) 的聚类，需要事先给出 K 的值。

在基于概率分布的算法中，我们先假定各类簇是由不同的概率密度函数的混合产生的，而每个样本点以不同的权重服从这些概率分布，对于这类情况，我们通常用 EM 算法求解。而这类算法的准确度取决于数据关于预定义概率密度函数的拟合情况。

而基于局部密度的聚类算法可以轻松地求出任意形状的类簇，比如 DBSCAN(Density-based spatial clustering of applications with noise)，它将簇定义为密度相连的点的最大集合，我们需要先给定一个密度阈值和区域半径，将区域半径内密度低于阈值的点舍弃为噪声，并将剩下的非连通高概率区域划分为不同的类簇，然而选择合适的密度阈值是非常不容易的。该算法的优点是聚类速度快且能够有效处理噪声点和发现任意形状的空间聚类。

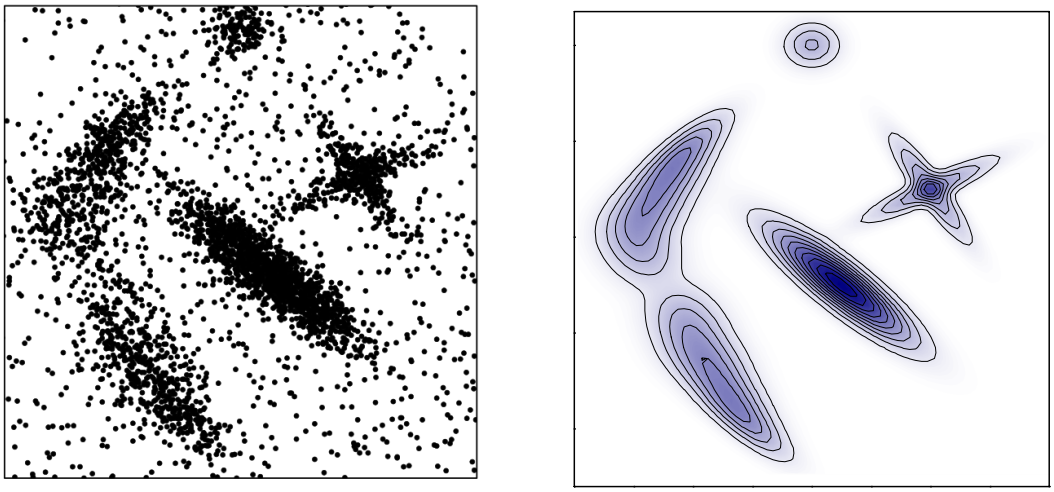


图 1: 基于密度的聚类算法

2 算法介绍

作者回顾了以上的一系列聚类算法，在基于以下两条假设的基础上，提出了新的基于密度峰值和距离的聚类算法，并宣称该算法能像 DBSCAN 和均值漂移 (mean-shift) 算法一样能检测出非球形的聚类并自动找出类簇的正确数目。

1. 聚类中心的密度要比它周围的样本点的密度更高[1]
2. 每个聚类中心离更高密度的点有较长的距离

对于每个样本点 i，我们只需要计算两个量：局部密度 ρ_i 和到更高密度点的距离 δ_i。我们注意到这两个量都只基于样本点之间的距离（该距离需满足三角不等式），这和 K-medoids 算法是类似的。

2.1 局部密度

Cutoff kernel

论文中作者指出样本点 i 的局部密度 ρ_i 定义为：

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{1}$$

其中 $\chi(x)$ 定义为：

$$\chi(x) = \begin{cases} 1, & \text{if } x < 0; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

其中 d_c 是截断距离 (cutoff distance)，也就是区域半径。实际上 ρ_i 就是在点 i 区域半径内的样本点的个数[2]。

如图 2所示，其中 $\rho(1) = 7, \rho(8) = 5, \rho(10) = 4.$

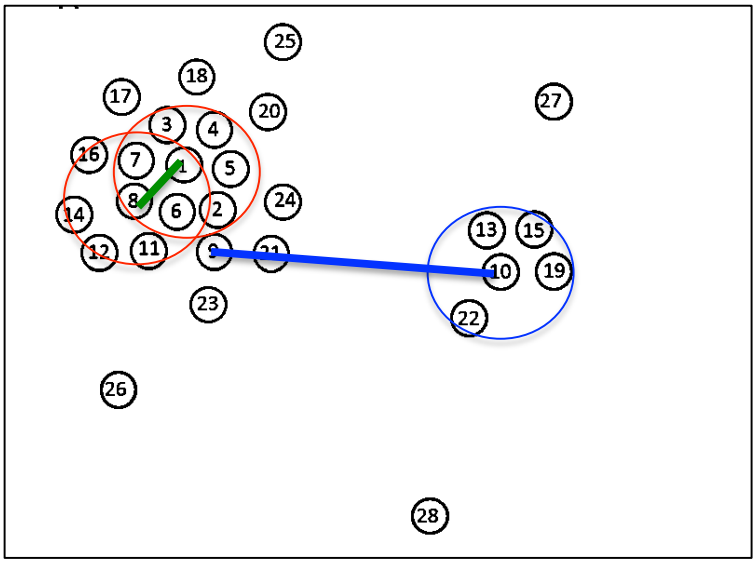


图 2: 样本点的局部密度

[1] 这一点和均值漂移算法的定义类似

[2] 在文章后面作者提到：作为经验法则，d_c 的选取最好能使得周围邻居的数目占总数的 1% ∼ 2%

作者在文章中指出该算法仅仅是对于 ρ_i 的相对大小敏感，因此在大数据集中，分析结果对于 d_c 是鲁棒的，但实际上，如果 d_c 过大，将会使得每个数据点的密度值都近似相等，从而使不同的类簇被合并，而当 d_c 过小时，将会使高密度点出现得更频繁，从而使得同一个类簇被分割成几个类簇。

Gaussian kernel

然而文章在后面的实验中还提到了可以用高斯核函数来衡量密度 ρ_i：

$$\rho_i = \sum_{j=1}^{N} \exp(-\frac{d_{ij}^2}{d_c^2}) \tag{3}$$

2.2 距离度量

而 δ_i 则度量点 i 到其他更高密度点的最小距离：

$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij}) \tag{4}$$

当 i 本身就是最高密度点时，可以令 $\delta_i = \max_j(d_i j)$，这样使得聚类中心的 δ 值远远比普通邻居点的 δ 值要大得多。

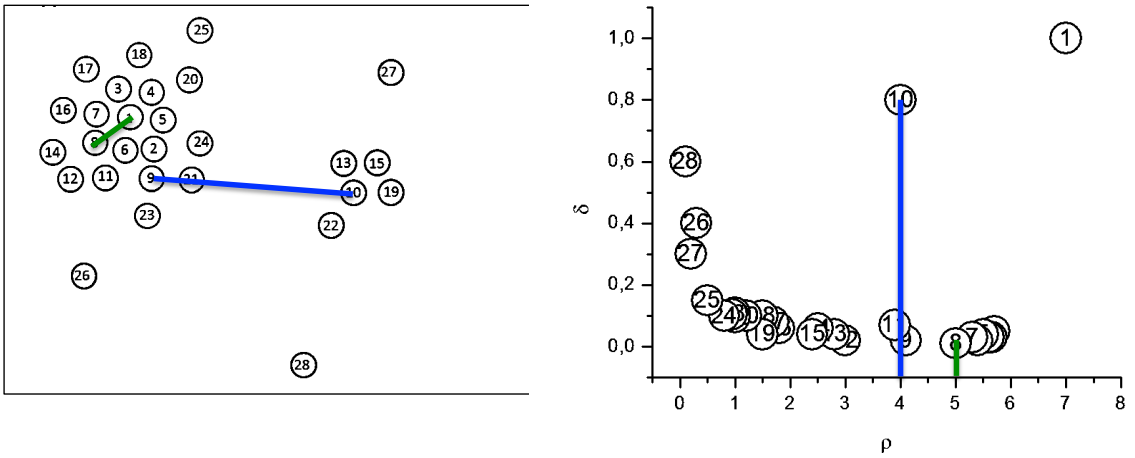


图 3: 样本点的距离 δ_i

2.3 聚类中心的选取

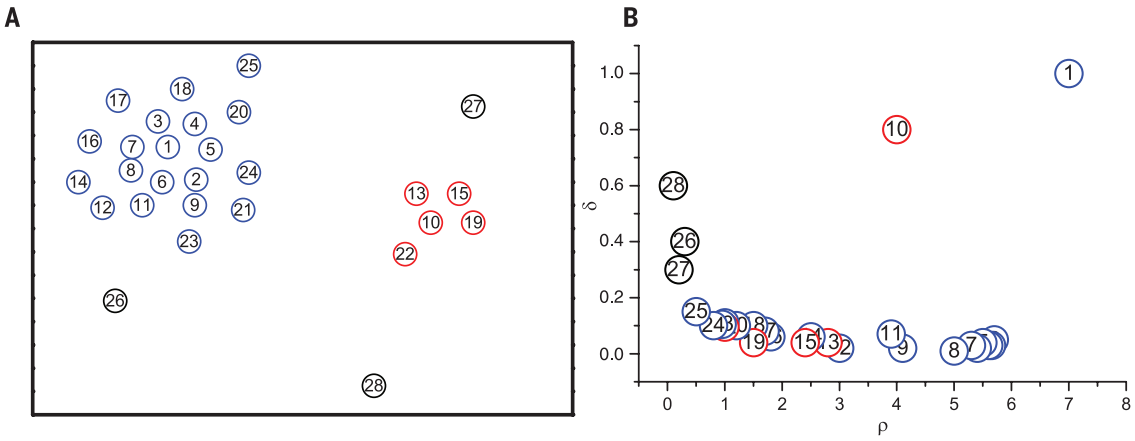


图 4: 图 A 为样本点的分布，其中点的数字按局部密度降序排列；图 B 为 A 中数据的决策图，不同的颜色对应不同的类簇

通过以上给出的公式 (1)(4)，我们可以算出所有点的局部密度 ρ_i 和距离 δ_i，然后可以绘制出基于不同样本点的 δ 关于 ρ 的函数。该算法认为高 ρ 高 δ 值的点为聚类中心，高 ρ 低 δ 值的点为其邻居。

可以看到，数据点 1 和 10 的 ρ 值和 δ 值都远远大于其他点，算法将这些点识别为聚类中心，虽然数据点 9 的密度 ρ 和点 10 的差不多，但由于其 δ 值太低，因此被识别为聚类中心周围的邻居。

那么什么样的 ρ 值和 δ 值被认为是远远大于其他点呢？作者在文章中提到，可以用 $\gamma_i = \rho_i\delta_i$ 来衡量。

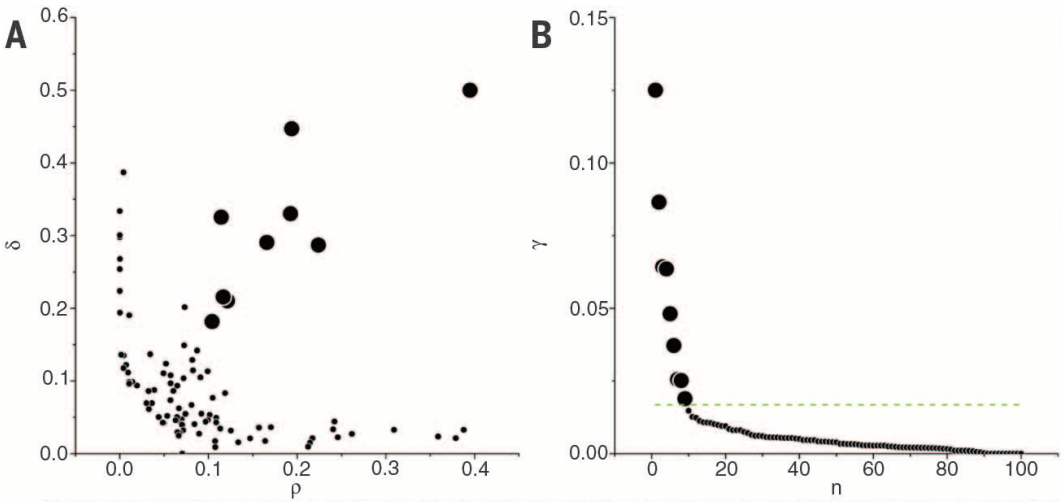


图 5: 图 A 为数据的决策图；图 B 为将 A 中 γ 值排序后的图像

作者在文中指出：对于按分布随机生成的样本数据，其 γ 值是服从幂律分布的，即 $\log(r) = a * n + b$，在图 5中可以看到：图 A 中加粗的点对应 B 中虚线以上的点，为聚类中心，同时我们可以看到在虚线以上的点的分布较稀疏，而虚线以下的点较为密集，文章正是根据这一

点来确定聚类中心点。但是这时不免要引入人为的主观因素，不太符合作者在摘要中所说的能自动确定类簇的正确数目 (automatically find the correct number of clusters)。

2.4 点的分配

聚类中心点的分配

我们通过计算 γ 值排序后输出的图像，然后人为确定聚类个数 k 并输入，然后将 γ 值靠前的 k 个点分别定为 k 个不同的聚类中心。

邻居点的分配

对于聚类中心的邻居点，我们以一个密度从大到小的顺序，分别进行如下操作：

1. 如果该点已经被分配到某个聚类中，那么跳过该点，对下一个点进行操作。
2. 如果该点还未被分配到某个聚类中，那么找到密度比该点高的点中离它最近的点，将这个点分配到那个点所属的聚类中。

在执行第二步操作时，因为是按密度从高到低的顺序执行操作，所以所有密度比当前点高的点必然已经被分配了聚类，而密度最高的点必然为聚类中心，也被分配了聚类。因此在执行完以上操作完后，一定能将所有的点都分配好聚类。

3 仿真实验

我们先通过 3 条螺旋曲线组成的图像来进行测试，其中局部密度 ρ 通过 Gaussian kernel 公式 (3) 计算，截断距离 d_c 选取的准则是使得周围邻居的数目占总数的 2%。

原图像见图 6 :

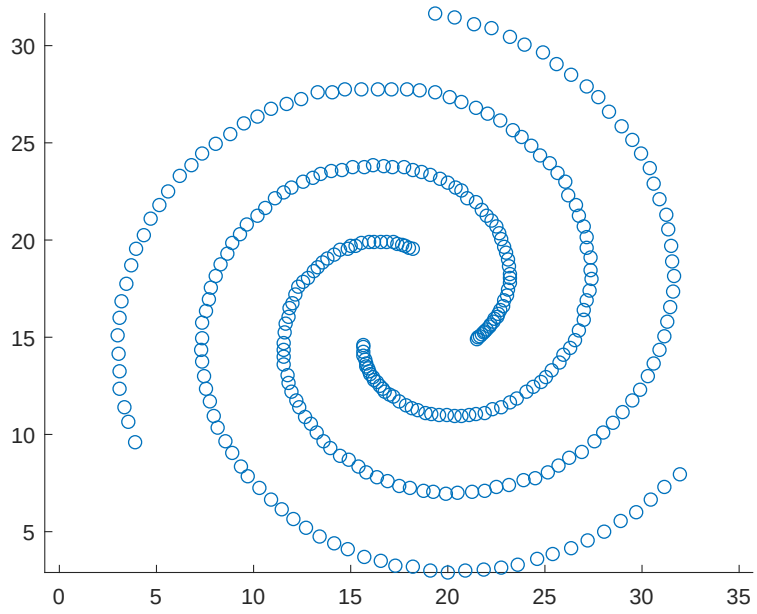
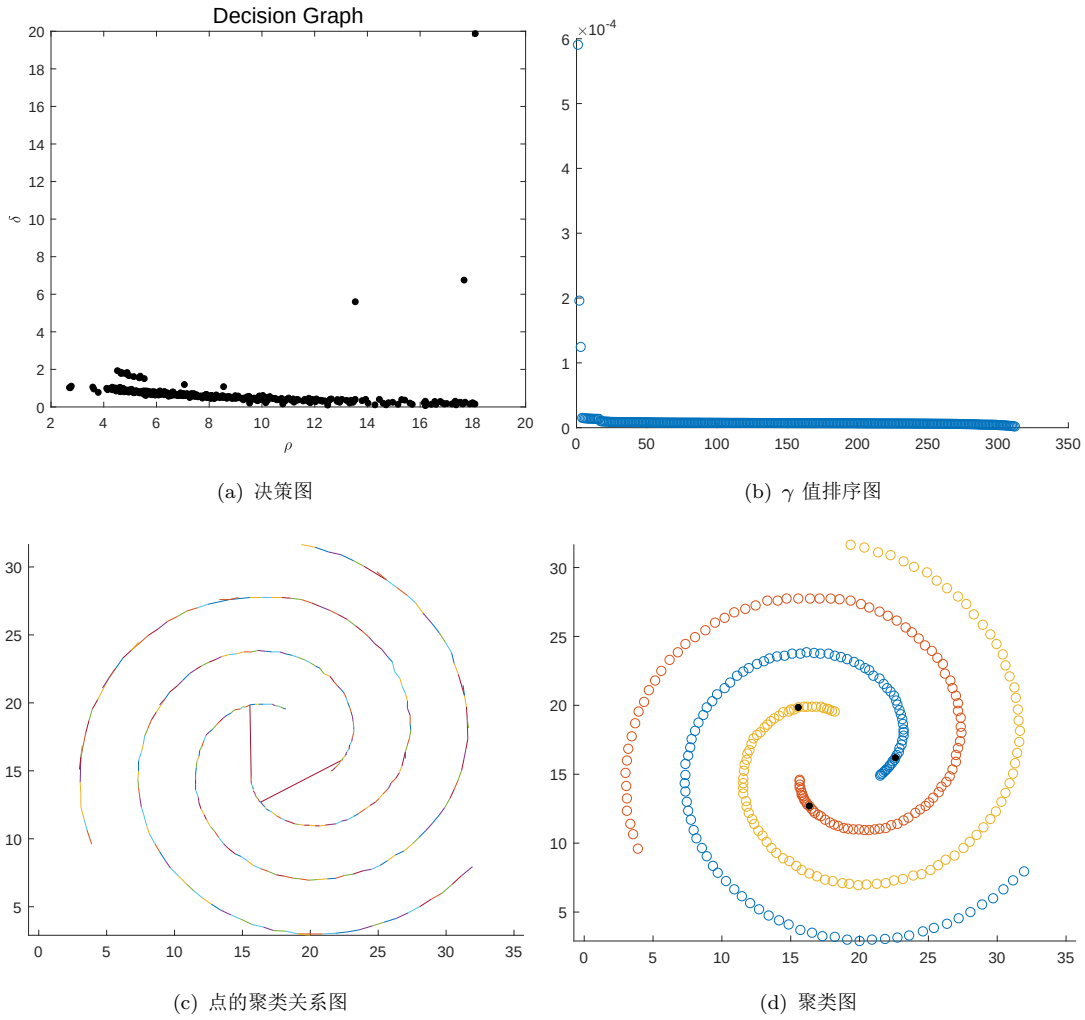


图 6: Spiral



(a) 决策图

(b) γ 值排序图

(c) 点的聚类关系图

(d) 聚类图

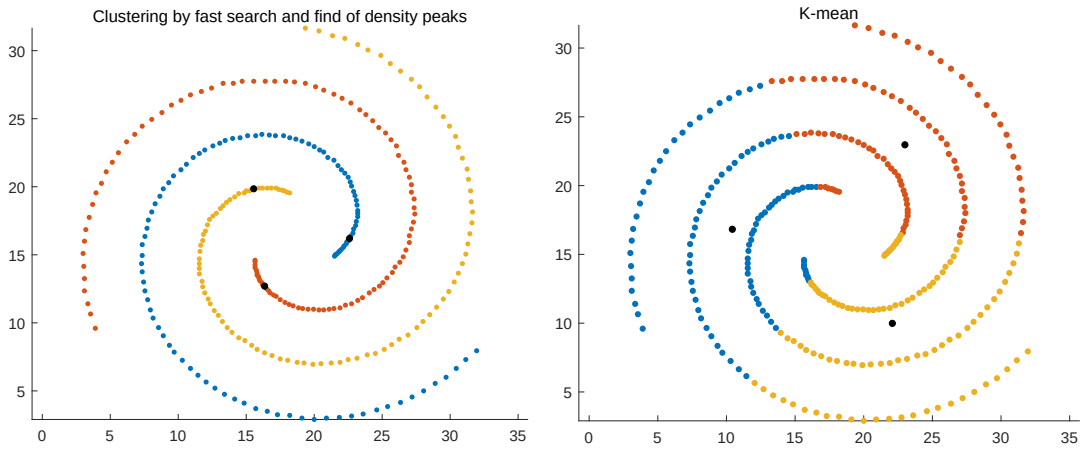
图 7: 实验结果

通过图 (a) 的决策图可以看出，右上角有三个点特别突出，可以将这三个点作为聚类中心，从图 (b) 中也能看出 γ 值排名特别高的点有三个，其他的点都基本上在同一范围内。图 (c) 将每个点及其最近的更高密度点连了起来，代表着该点的聚类由其相连的点决定。从图 (d) 中的结果可以看出，我们的算法还是比较成功的。

3.1 对比测试

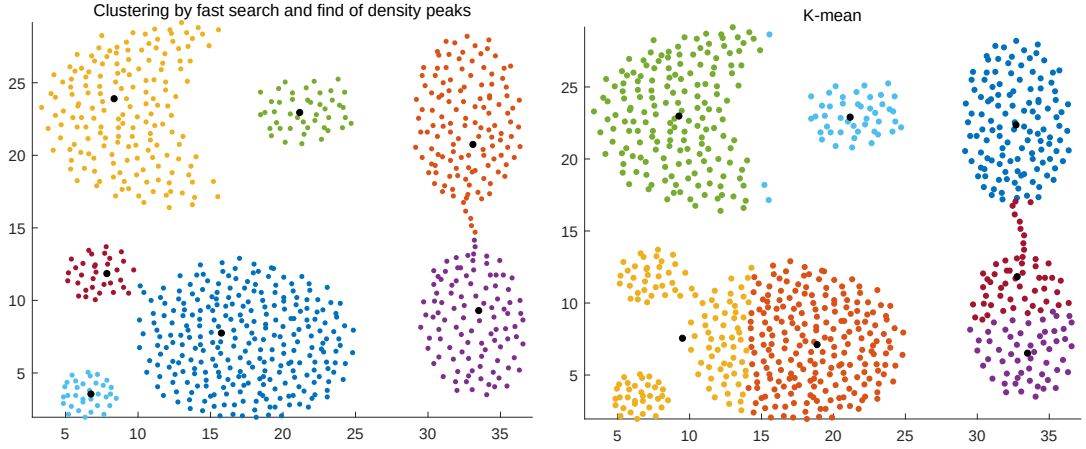
我们使用 K-mean 算法对同一个数据集和原算法进行对比测试，具体比较可见图 8,9,10,11,12,13。

综合各种情况来看，论文中给出的算法效果在大部分情况下都比 K-mean 算法要好。



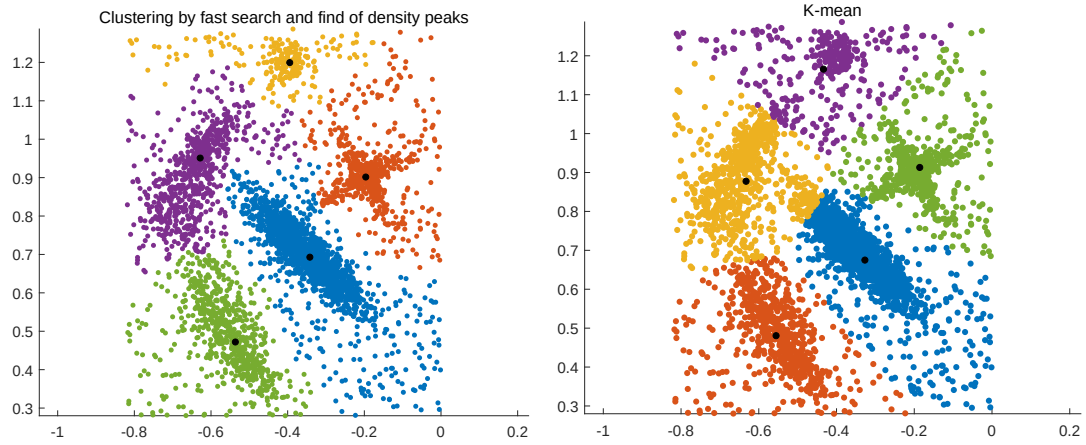
(a) 原算法 (b) K-mean 算法

图 8: spiral



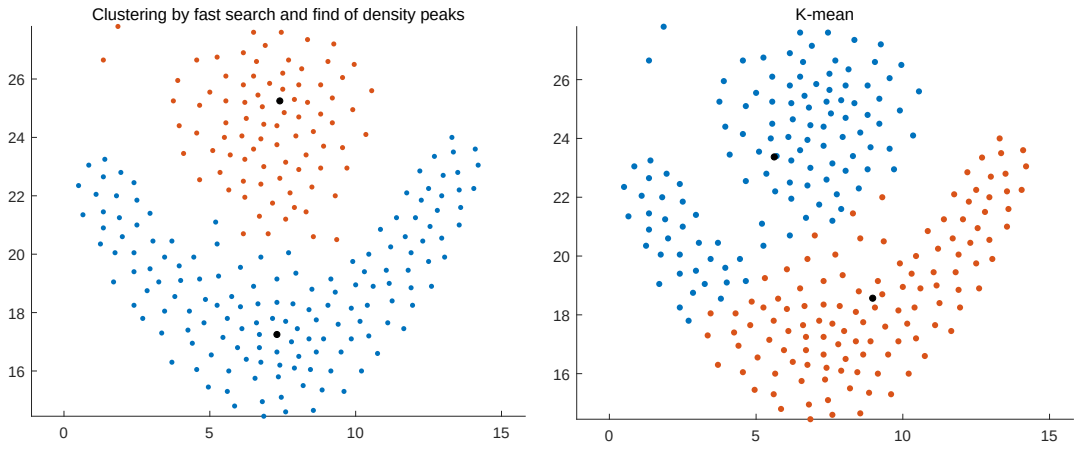
(a) 原算法 (b) K-mean 算法

图 9: Aggregation



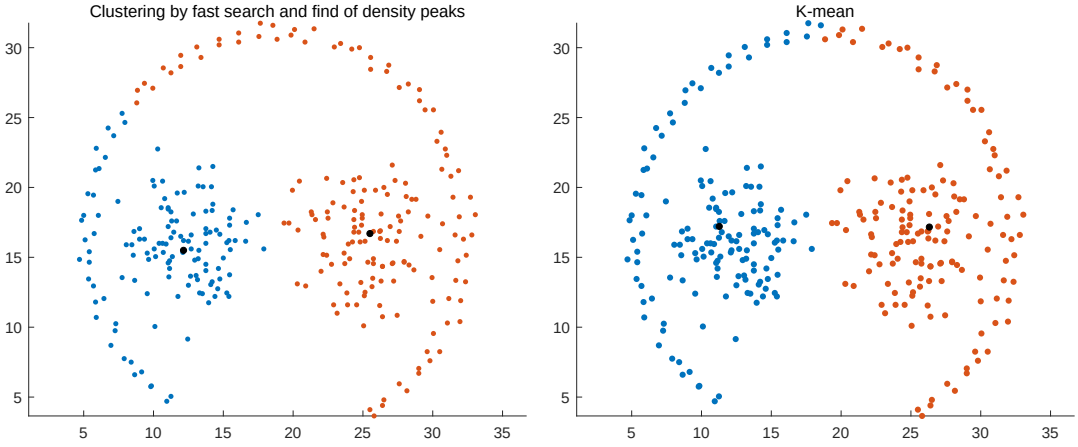
(a) 原算法 (b) K-mean 算法

图 10: panelB



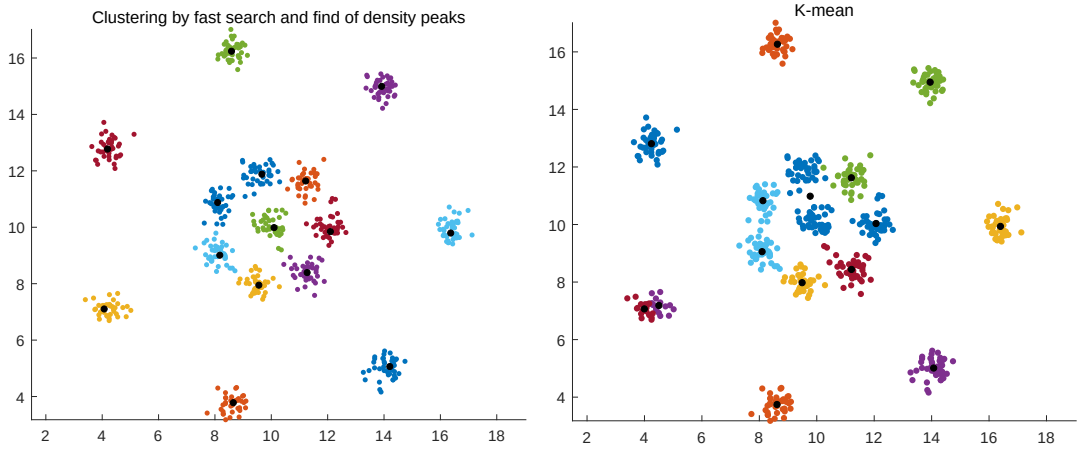
(a) 原算法 (b) K-mean 算法

图 11: flame



(a) 原算法 (b) K-mean 算法

图 12: pathbased



(a) 原算法 (b) K-mean 算法

图 13: R15

4 应用——人脸区分

4.1 准备阶段

我们通过调用旷世科技公司的人脸比较 API 来计算每张人脸之间的相似度。
这是官网给出的 face compare API 调用模板：

```
curl -X POST "https://api-cn.faceplusplus.com/facepp/v3/compare" \
-F "api_key=<api_key>" \
-F "api_secret=<api_secret>" \
-F "face_token1=c2fc0ad7c8da3af5a34b9c70ff764da0" \
-F "face_token2=ad248a809408b6320485ab4de13fe6a9"
```

我们首先需要申请一个开发者账号，获得api_key 和api_secret 后就可以进行调用 API 了。

4.2 简单测试

首先我们选择川普的两张照片 (图 14) 进行一下测试：

图 14: 川普照片

打开命令行，输入以下代码：

```
set WLD1=http://chuantu.biz/t6/310/1526109459x-1404817808.jpg
set WLD2=http://chuantu.biz/t6/310/1526109944x-1404817808.jpg
curl -X POST "https://api-cn.faceplusplus.com/facepp/v3/compare" -F
↪ "api_key=MSEuYIibv2ldXKRoNTq61tNmqlT4POaN" -F
↪ "api_secret=vlhOUyZOPKNJLIJYWX9yKUOHj47Rc5XF" -F "image_url1=%WLD1%" -F
↪ "image_url2=%WLD2%"
```

返回的结果如下：


```
{
    "faces1": [
        {
            "face_rectangle": {
                "width": 212,
                "top": 99,
                "left": 346,
                "height": 212
            },
            "face_token": "2f12c93a92b679acb6118a2508e4543a"
        }
    ],
    "faces2": [
        {
            "face_rectangle": {
                "width": 113,
                "top": 47,
                "left": 69,
                "height": 113
            },
            "face_token": "3421b3220d32822c49fb2177b726703a"
        }
    ],
    "time_used": 1753,
    "thresholds": {
        "1e-3": 62.327,
        "1e-5": 73.975,
        "1e-4": 69.101
    },
    "confidence": 79.489,
    "image_id2": "b6jg6AVo0p03F/oZRXJagw==",
    "image_id1": "iBAxxO9T/xdcCEXM6XhZgA==",
    "request_id": "1527076087,a3d65104-9386-4879-9890-9fdd75e57153"
}
```


服务器返回了 json 格式的人脸信息，其中我们需要的关键信息为`confidence`，这表明这两张图片为同一个人的可能性高达 79.489%。

4.3 正式实验

对于较多数量的图片时，一个一个用命令行输入就显得过于繁琐，并且也每次都要手动提取关键信息，很不方便，于是我用 python 写了一个程序，将文件夹下的图片传送到服务器，并提取相似度信息，返回一个相似度矩阵。

其数据集来源于我们宿舍三个人无 (chou) 私 (bu) 分 (yao) 享 (lian) 的自拍：

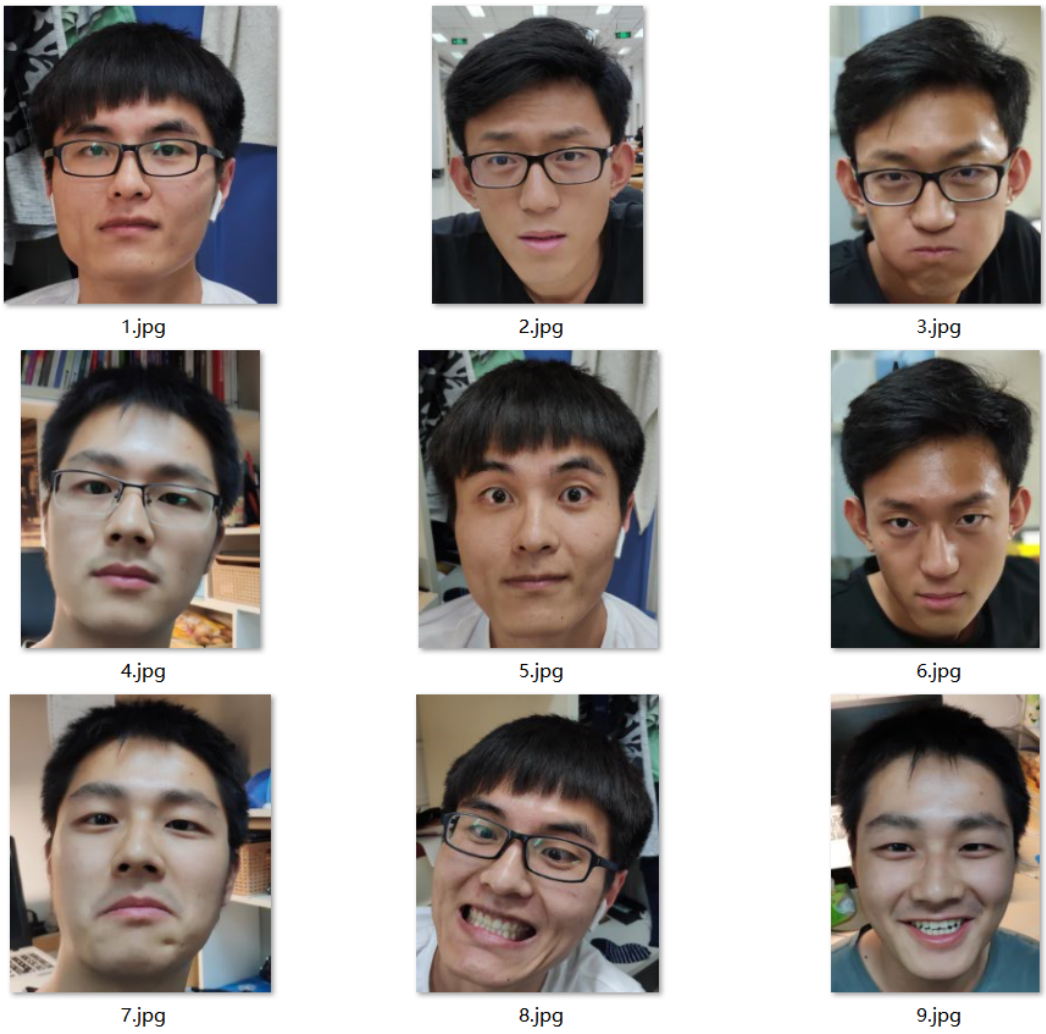


图 15: 6 公寓 109 众美男表情包

程序返回的相似度矩阵见表 1

其中第 i 行 j 列的值代表第 i 张图片与第 j 张图片之间的相似度，颜色的不同代表相似度的差异：

- 表中 80 ~ 100 区间被标记为了红色，说明图片大概率为同一个人
- 40 ~ 80 区间被标记为了黄色，说明图片中这两个人有点像
- 40 以下的数据标为绿色，说明图片中这两个人长相差异很大

表 1: 相似度矩阵

100	24.384	28.526	59.233	91.302	33.867	55.257	91.156	56.509
24.384	100	87.353	38.75	22.701	87.971	32.421	23.132	21.015
28.526	87.353	100	42.088	26.372	89.814	37.974	31.582	30.32
59.233	38.75	42.088	100	41.12	39.282	91.967	57.067	89.618
91.302	22.701	26.372	41.12	100	34.896	44.818	88.443	46.267
33.867	87.971	89.814	39.282	34.896	100	38.229	30.205	29.49
55.257	32.421	37.974	91.967	44.818	38.229	100	57.561	91.604
91.156	23.132	31.582	57.067	88.443	30.205	57.561	100	51.633
56.509	21.015	30.32	89.618	46.267	29.49	91.604	51.633	100

4.4 距离映射

如果要通过上面的算法进行聚类，我们首先需要给不同的图片相互之间定义一个距离，也就是说我们需要构造一个映射 $f:[0,1]\mapsto[0,\infty]$ 将相似度关系转化为距离关系以方便程序计算，其中相似度越高，距离越小，比如线性关系：

$$f(x)=100(1-x)$$

或者倒数关系：

$$f(x)=1/x$$

但是从实际情况出发，我们需要考虑到**边界效应**：在不同的值处相似度差异给人带来的直观感受是不同的。比如将相似度从 99% 提升到 100% 就明显比从 49% 到 50% 要难得多，同理，将相似度从 1% 降到 0% 也比从 50% 到 49% 要难得多。

因此在我们的映射中：$\left|f(1)-f(1-\Delta x)\right|>\left|f(0.5)-f(0.5-\Delta x)\right|$，即 $\left|f'(1)\right|>\left|f'(0.5)\right|$。也就是说要找到一个在两端导数绝对值大，中间导数绝对值小的函数映射。

在此我们可以考虑将 Beta 分布 (5) 的概率密度函数作为其导数：

$$\begin{aligned}Beta(x;\alpha,\beta)&=\frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\,du}\\&=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\,x^{\alpha-1}(1-x)^{\beta-1}\\&=\frac{1}{\mathrm{B}(\alpha,\beta)}\,x^{\alpha-1}(1-x)^{\beta-1}\end{aligned}\tag{5}$$

对于不同 Beta 分布的概率密度函数见图 16，经过观察图像后，我们选取 $\alpha=\beta=0.5$，得：

$$Beta(x;0.5,0.5)=6/\sqrt{x(1-x)}\tag{6}$$

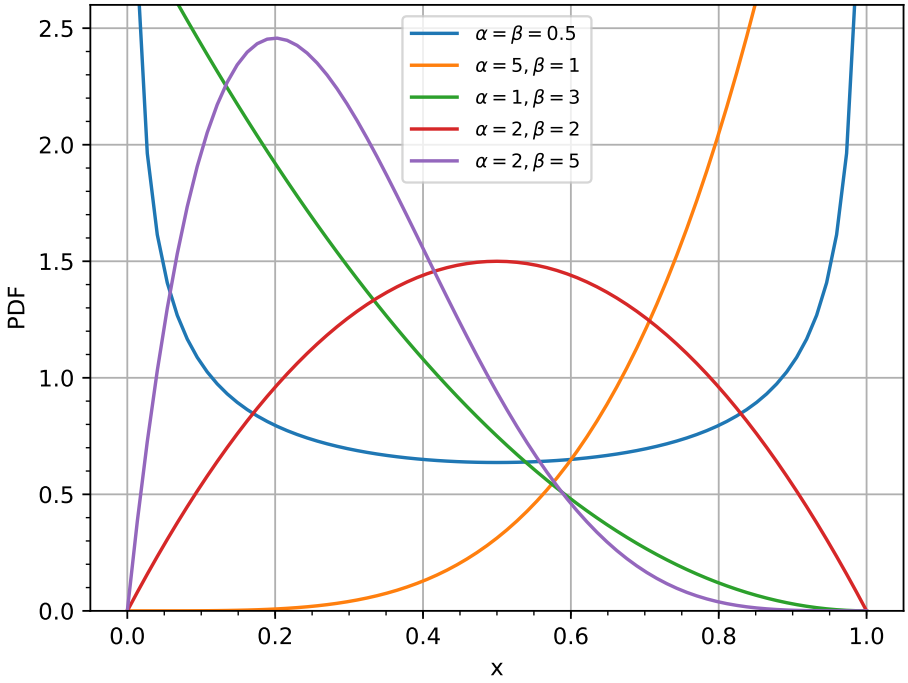


图 16: Beta distribution pdf

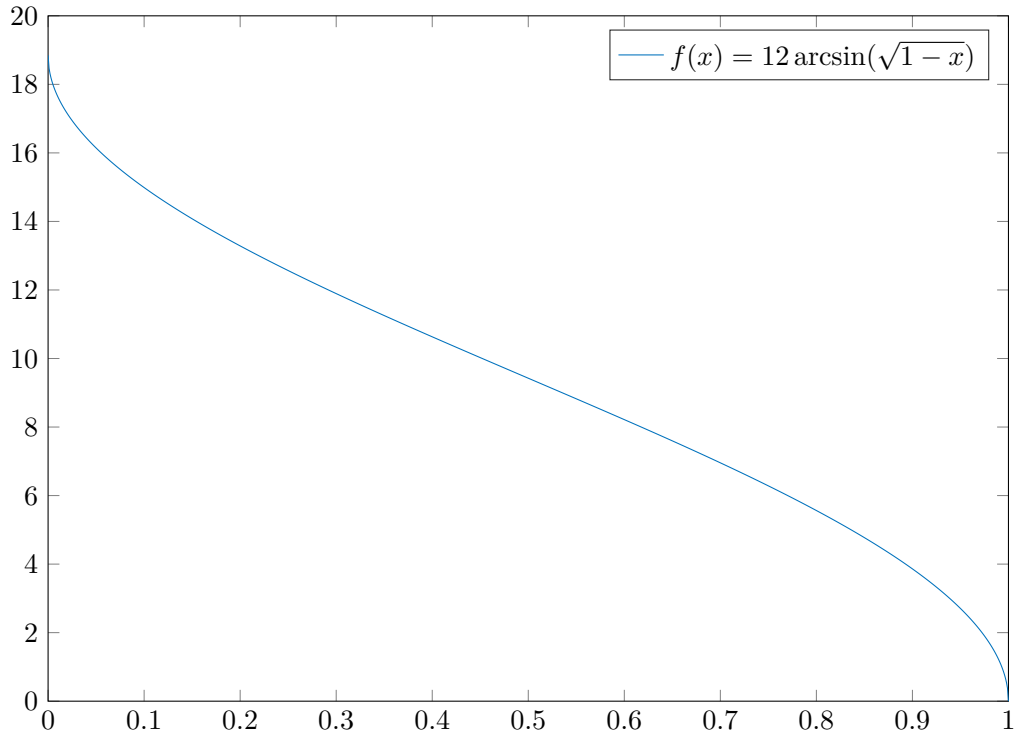
将式 (6) 积分得:

$$f(x) = 12 \arcsin(\sqrt{x}) \tag{7}$$

由于相似度越高，距离越小，所以我们需要将 x 换成 $1-x$ 得：

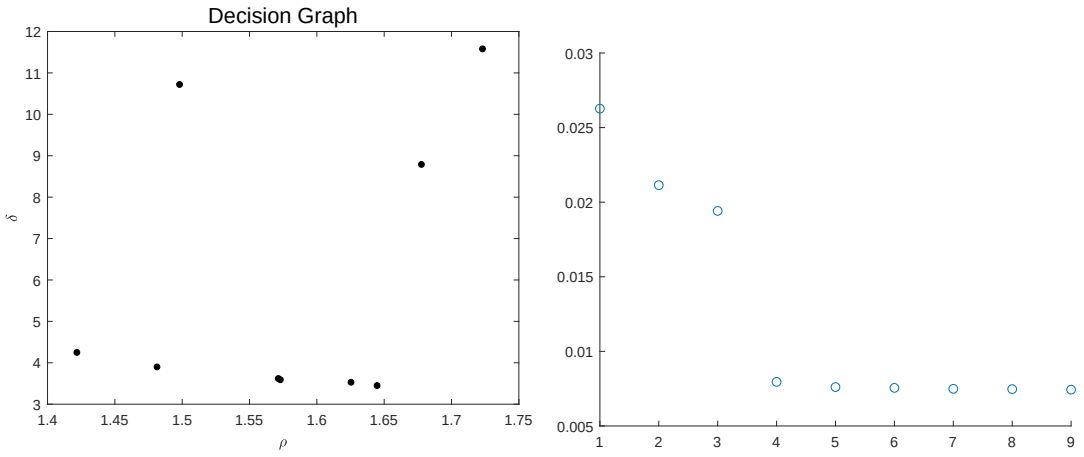
$$f(x) = 12 \arcsin(\sqrt{1-x}) \tag{8}$$

最终定义的映射函数 (8) 的图像如下：



4.5 实验结果

数据经过转化后，我们就可以使用原算法进行聚类，得到的决策图如下：



(a) 决策图　　(b) γ 值排序图

图 17: 实验结果

从图 17中的 γ 值排序图可以清晰地看出程序有三个特别明显的聚类中心，于是我们手动输入聚类数 $k = 3$，得到了程序的聚类输出结果如下：

```
Cluster 1 :

     4     7     9


Cluster 2 :

     2     3     6


Cluster 3 :

     1     5     8
```

回顾图 15，我们可以发现这三个聚类刚好将我们三个人的照片正确地分类好了，所以说最终的实验还是挺成功的。

5 附录：程序代码

5.1 Clustering by fast search and find of density peaks 聚类算法代码

```
%% Clustering by density peaks
% by ZHANG JIN

clc;clear;
% 输入数据的距离矩阵
load('R15.txt');
P=R15;
n=size(P,1);
N=n*n;
D=zeros(n,n);
for i=1:n
    for j=1:n
         D(i,j)=get_distance(P(i,1),P(i,2),P(j,1),P(j,2));
    end
end

% 升序排列 distance，然后取 1% 到 2% 的那个 distance 作为 dc
p=2;
position=round(N*p*2/100);
sort_d=sort(D(:));
dc=sort_d(position);

%% 计算密度 rho
% cutoff
% rho=sum(D(:,:)<dc,2);

% Gaussian kernel
rho=sum(exp(-D.^2./(dc^2)),2);

%% 计算距离 delta
% 将 rho 按降序排列，ordrho 保持序
[rho_sorted,ordrho]=sort(rho,'descend');

% 处理 rho 值最大的数据点
delta(ordrho(1))=max(D(ordrho(1),:));

% 计算 delta 矩阵
for i=2:n
```

```
    [delta(ordrho(i)),idx_delta]=min(D(ordrho(i),ordrho(1:i-1)));
    neigh(ordrho(i))=ordrho(idx_delta);
end
%% 画出决策图
figure
tt=plot(rho,delta,'o','MarkerSize',4,'MarkerFaceColor','k','MarkerEdgeColor','k');
title ('Decision Graph','FontSize',15.0)
xlabel ('\rho')
ylabel ('\delta')

figure
xx=1:n;
gamma=delta/sum(delta).*rho'/sum(rho);
[gamma,ordgamma]=sort(gamma,'descend');
scatter(xx,gamma)

%% 计算分配矩阵
% 人为确定聚类个数
k=input(' 请输入聚类的个数: \n')
assign=zeros(n,1);
for i=1:k
    assign(ordgamma(i))=i;
end

[M,I]=min(D(ordrho(1),ordgamma(1:k)));
neigh(ordrho(1))=ordgamma(I);

for i=1:n
    if(~assign(ordrho(i)))
        assign(ordrho(i))=assign(neigh(ordrho(i)));
    end
end
%% 画出聚类后的图
figure
for i=1:k
    T=find(assign==i);
    hold on
    plot(P(T,1),P(T,2),'.','Markersize',10)
end

% 标出聚类中心
title('Clustering by fast search and find of density peaks')
for i=1:k
```

```
    ↪ plot(P(ordgamma(i),1),P(ordgamma(i),2),'o','MarkerSize',4,'MarkerFaceColor','k','Mark
end
axis equal

%% 画出 neighbor 关系图
figure
hold on
axis equal
for i=1:n
 plot([P(ordrho(i),1);P(neigh(ordrho(i)),1)],
  ↪ [P(ordrho(i),2),P(neigh(ordrho(i)),2)])
 %pause(0.02)
end


figure
hold on
axis equal
plot(P(:,1),P(:,2),'.','Markersize',10)
%% 距离函数
function z=get_distance(x1,y1,x2,y2)
    z=sqrt((x1-x2)^2+(y1-y2)^2);
end
```

5.2 K-mean 聚类算法代码

```
clc;clear;
% 输入数据的距离矩阵
load('Aggregation.txt');
N=7;
P=Aggregation;
n=size(P,1);
X=P(:,1:2);
[idx,C] = kmeans(X,N);

figure;
hold on;
axis equal

for i=1:N
    plot(X(idx==i,1),X(idx==i,2),'.','MarkerSize',12)
```

```
end

plot(C(:,1),C(:,2),'o','MarkerSize',4,'MarkerFaceColor','k','MarkerEdgeColor','k')
title('K-mean')
hold off
```

5.3 Face++ API 调用程序

```
# -*- coding: utf-8 -*-
# 调用Face++ API计算fig文件夹下图片的相似度矩阵
import urllib2
import urllib
import time
import numpy as np
http_url = 'https://api-cn.faceplusplus.com/facepp/v3/compare'
key = "MSEuYIibv2ldXKRoNTq61tNmqlT4POaN"
secret = "vlhOUyZ0PKNJLIJYWX9yKUOHj47Rc5XF"


def fun(i, j):
    filepath1 = r"C:\Users\80693\Desktop\fig\%s.jpg" % (i + 1)
    filepath2 = r"C:\Users\80693\Desktop\fig\%s.jpg" % (j + 1)
    boundary = '----------%s' % hex(int(time.time() * 1000))
    data = []
    data.append('--%s' % boundary)
    data.append('Content-Disposition: form-data; name="%s"\r\n' %
                'api_key')
    data.append(key)
    data.append('--%s' % boundary)
    data.append('Content-Disposition: form-data; name="%s"\r\n' %
                'api_secret')
    data.append(secret)
    data.append('--%s' % boundary)
    fr1 = open(filepath1, 'rb')
    data.append('Content-Disposition: form-data; name="%s"; filename=" "' %
                'image_file1')
    data.append('Content-Type: %s\r\n' % 'application/octet-stream')
    data.append(fr1.read())
    fr1.close()
    data.append('--%s' % boundary)
    fr2 = open(filepath2, 'rb')
    data.append('Content-Disposition: form-data; name="%s"; filename=" "' %
```

```
                'image_file2')
    data.append('Content-Type: %s\r\n' % 'application/octet-stream')
    data.append(fr2.read())
    fr2.close()
    data.append('--%s--\r\n' % boundary)
    http_body = '\r\n'.join(data)
    # buld http request
    req = urllib2.Request(http_url)
    # header
    req.add_header(
        'Content-Type', 'multipart/form-data; boundary=%s' % boundary)
    req.add_data(http_body)
    try:
        # req.add_header('Referer','http://remotserver.com/')
        # post data to server
        resp = urllib2.urlopen(req, timeout=5)
        # get response
        qrcont = resp.read()
        # print qrcont
        return eval(qrcont)["confidence"]
    except urllib2.HTTPError as e:
        print e.read()
        print "第%s张图与第j张图中的相似度未算出来" % (i + 1, j + 1)


N = 9
A = np.ones((N, N)) * 100
for i in range(N):
    for j in range(i + 1, N):
        A[i][j] = A[j][i] = fun(i, j)
        time.sleep(3)
```