Kmeans-1

Consider two independent variables, $s1$ and $s2$, which are distributed by

$$p(s_i)=\begin{cases}\dfrac{1}{2\sqrt{3}} & \text{if } |s_i|\le\sqrt{3}\\ 0 & \text{otherwise}\end{cases}$$

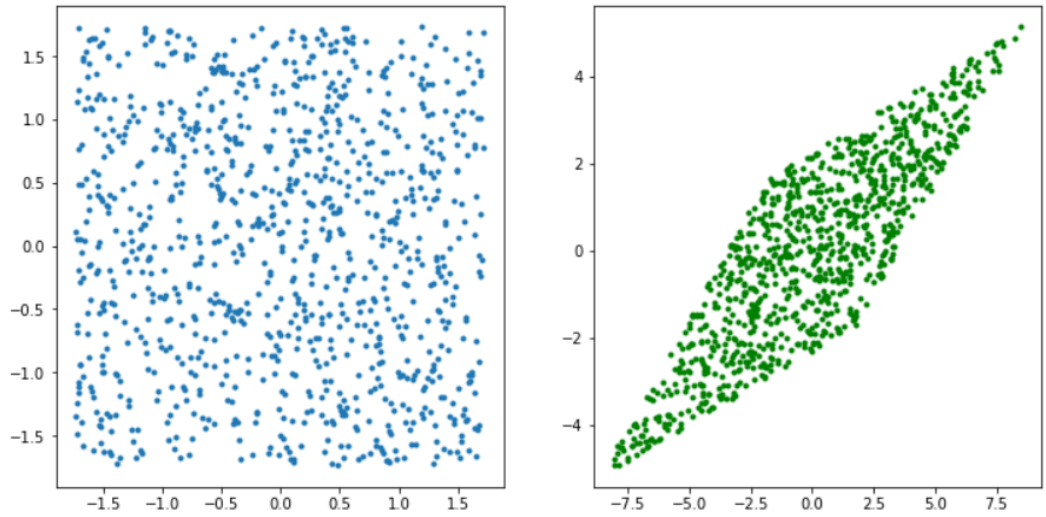
Their linear combinations are $x=As$, where $x=[x1,x2]^T$, and $s=[s1,s2]^T$,and A is the following $A0$:

$$A_0=\begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix}$$

(1) Draw the scatter plot of data s in 2-D space, and that of data x in 2-D space;

绘制散点图的代码如下:

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt

s1 = random.uniform(low=-np.sqrt(3), high=np.sqrt(3), size=1000)
s2 = random.uniform(low=-np.sqrt(3), high=np.sqrt(3), size=1000)
x1 = 2 * s1 + 3 * s2
x2 = 2 * s1 + 1 * s2
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(s1, s2, '.')
plt.subplot(1, 2, 2)
plt.plot(x1, x2, '.', color='green')
```



结论:

原本两个离散随机变量s_1, s_2独立，且各自服从均匀分布，它们联合起来服从二维空间中的二维均匀分布。经过矩阵A的线性变换后得到的x_1, x_2变得相关了。

(2) Compute the correlation coefficient of $s1$ and $s2$, and that of $x1$ and $x2$ to understand the influence of A on correlation coefficient;

```
print('s1和s2的相关系数：{:.4f}'.format(np.corrcoef(s1, s2)[0,1]))
print('x1和x2的相关系数：{:.4f}'.format(np.corrcoef(x1, x2)[0,1]))
```

结果:

```
s1和s2的相关系数：-0.0222
x1和x2的相关系数：0.8622
```

结论:

经过矩阵A的线性变换后，原本几乎无线性相关关系的两个离散变量变得强线性相关了，也即不再独立了。

(3) Compute the mutual information of $s1$ and $s2$, and that of $x1$ and $x2$ to examine the influence of A on mutual information. Compare your result on (2) and (3).

先将Python中得到的ndarray数组导出至csv文件中，再导入Matlab计算互信息：

```
function [MI,mi] = calc_mi(u1,u2,wind_size)
if nargin<2
disp('Error: please input at least two parameters:u1, u2');
return
end
if nargin==2
wind_size=floor(power(length(u1),1/3)+0.5);
end
x = [u1,u2];
n = wind_size;
[xrow, xcol] = size(x);
bin = zeros(xrow,xcol);
pmf = zeros(n, 2);
for i = 1:2
    minx = min(x(:,i));
    maxx = max(x(:,i));
    binwidth = (maxx - minx) / n;
    edges = minx + binwidth*(0:n);
    histcEdges = [-Inf edges(2:end-1) Inf];
    [occur,bin(:,i)] = histc(x(:,i),histcEdges,1);
    pmf(:,i) = occur(1:n)./xrow;
end

jointOccur = accumarray(bin,1,[n,n]);
jointPmf = jointOccur./xrow;
Hx = -(pmf(:,1))'*log2(pmf(:,1)+eps);
Hy = -(pmf(:,2))'*log2(pmf(:,2)+eps);
Hxy = -(jointPmf(:))'*log2(jointPmf(:)+eps);
MI = Hx+Hy-Hxy;
mi = MI/sqrt(Hx*Hy);
```

```
s1 = csvread('s1_data.csv')
s2 = csvread('s2_data.csv')
x1 = csvread('x1_data.csv')
x2 = csvread('x2_data.csv')
[MI,zMI] = calc_mi(s1,s2,5)
[MI,zMI] = calc_mi(x1,x2,5)
```

结果:

```
>> [MI, zMI] = calc_mi(s1, s2, 5)

MI =

    0.0059


zMI =

    0.0025

>> [MI, zMI] = calc_mi(x1, x2, 5)

MI =

    0.7769


zMI =

    0.3636
```

结论:

互信息是信息论里一种有用的信息度量，它可以看成是一个随机变量中包含的关于另一个随机变量的信息量，或者说是一个随机变量由于已知另一个随机变量而减少的不肯定性。简单说，就是两个事件集合之间的相关性。原来两个离散型随机变量的互信息值为0.0025，经过矩阵A的线性变换后得到的随机变量的互信息值为0.3636，远大于原来的值，这表明经过矩阵A的线性变换后，两个随机变量之间的相关性增强了。而在(2)中我们计算了经过A变换前后两个离散变量的相关系数，也是变换后相关系数远大于变换前，这也侧面反映出互信息计算结果的正确性。

Kmean-2

Draw contour curve of data which are same distant from the origin in 2-D space when Minkowski distance is utilized, which is defined by

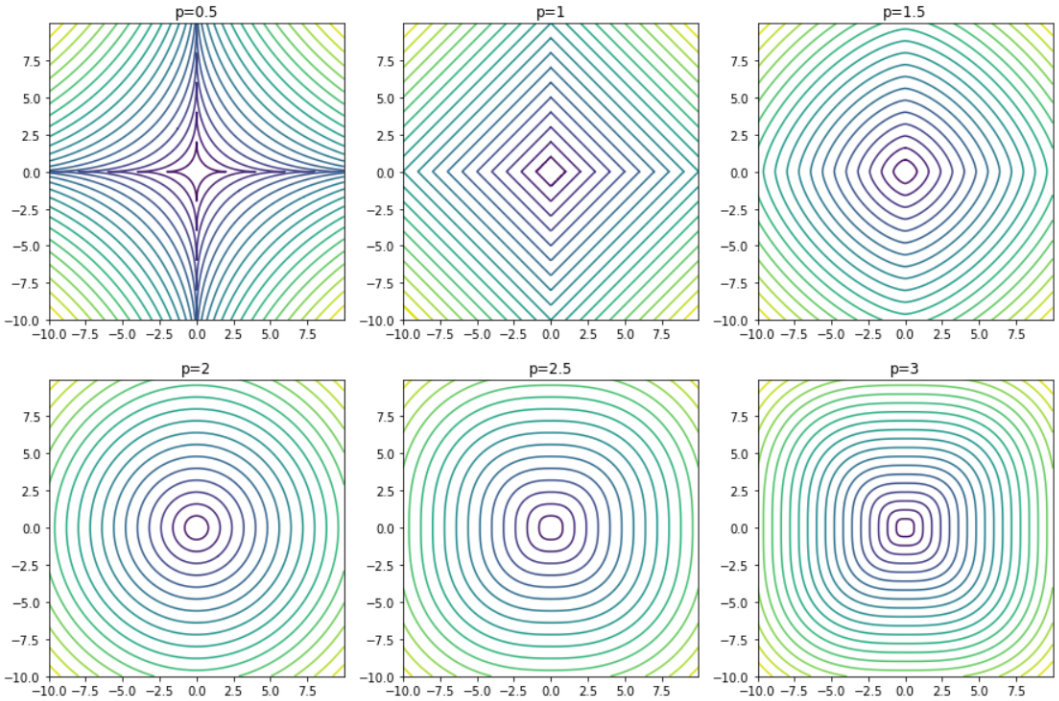
$$\mathrm{dist}_{\mathrm{Minc}}(x,y)=\|x-y\|_p=\left(\sum_i|x_i-y_i|^p\right)^{\frac{1}{p}}$$

Examine the influence of the parameter P on the contour shape.

绘制等高线图的代码如下:

```
# 计算Minkovski距离
def Min_dist(X, Y, p):
    return np.power((np.power(np.abs(X), p) + np.power(np.abs(Y), p)), 1/p)

x = np.arange(-10, 10, 0.01)          # x轴每隔0.01取一个点
y = np.arange(-10, 10, 0.01)          # y轴每隔0.01取一个点
X, Y = np.meshgrid(x, y)              # 将原始数据变成网格数据形式
for p in [0.5, 1, 1.5, 2, 2.5, 3]:
    Z = Min_dist(X, Y, p)             # 给定p计算数据点到原点的Minkovski距离
    plt.contour(X, Y, Z, 20)          # 画等高线图，等高线的数目为10
    plt.show()
```

结果如下:



分析:

当 $p=0.5$ 时，距离等高线向原点方向内凸，当 $p=1$ 时，距离等高线呈现以原点为中心的菱形分布，当 $p>1$ 时，距离等高线向原点四周外凸，而当p的值较大时，距离等高线则趋向于呈现一种以原点为中心的矩形框分布。这背后的原理不难推导：当y=[0,0]时：
$Minkovski\_dist(x,[0,0])=(|x_1|^p+|x_2|^p)^{\frac{1}{p}}$**。当 p 趋于正无穷时，若固定x_1，即x_1可视为常数，且将x_2限制为0附近的较小值，则此时**$Minkovski\_dist(x,[0,0])\approx(|x_1|^p+|0|^p)^{\frac{1}{p}}=|x_1|$**；同理，若固定$x_2$且将$x_1$限制为0附近的较小值，则此时**$Minkovski\_dist(x,[0,0])\approx|x_2|$**。**

Kmeans-3

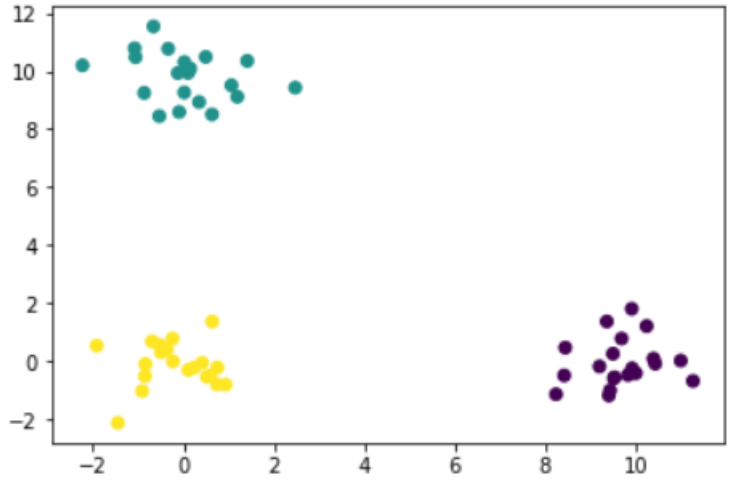
Simulate a 3-clusters dataset in 2-D space, each being gaussian distributed with the mean of [0,0], [0,10], and [10,0] with an identity covariance matrix, with an addition of an outlier standing at the point [50,50]. please do experiments for clustering analysis on the dataset with and without the outlier respectively, to study the robustness of the k-means algorithm.

编写python程序对聚类结果进行仿真：

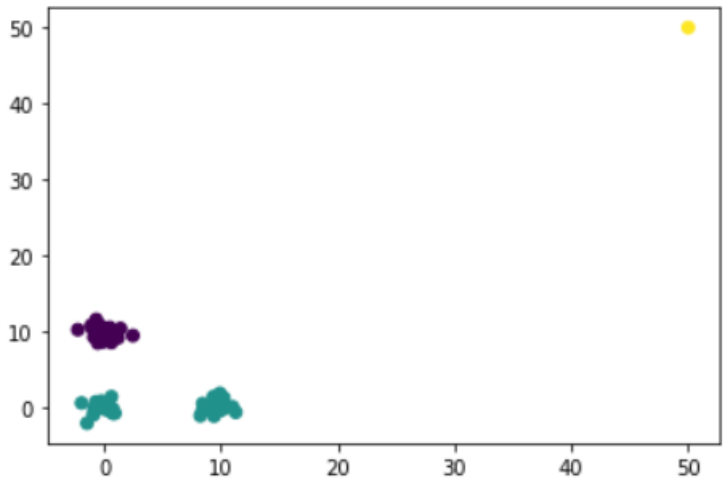
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

mu_x = [0, 0, 10]
mu_y = [0, 10, 0]
delta_x = [1, 1, 1]
delta_y = [1, 1, 1]
x0 = mu_x[0] + delta_x[0] * np.random.randn(20)
y0 = mu_y[0] + delta_y[0] * np.random.randn(20)
x1 = mu_x[1] + delta_x[1] * np.random.randn(20)
y1 = mu_y[1] + delta_y[1] * np.random.randn(20)
x2 = mu_x[2] + delta_x[2] * np.random.randn(20)
y2 = mu_y[2] + delta_y[2] * np.random.randn(20)
plt.plot(x0, y0, '.')
plt.plot(x1, y1, '.')
plt.plot(x2, y2, '.')
plt.plot(50, 50, '+')
plt.axis([-10,60,-10,60])
plt.show()

k_means = KMeans(n_clusters=3, random_state=0)
x = np.concatenate((x0, x1, x2), axis=0)
x = x.reshape(len(x), 1)
y = np.concatenate((y0, y1, y2), axis=0)
y = y.reshape(len(y), 1)
# 不带(50, 50)点
data1 = np.concatenate((x, y), axis=1)
k_means.fit(data1)
pred_cls = k_means.predict(data1)
plt.scatter(x, y, c=pred_cls)
plt.show()

# 带(50, 50)点
x = np.concatenate((x0, x1, x2, [50]), axis=0)
x = x.reshape(len(x), 1)
y = np.concatenate((y0, y1, y2, [50]), axis=0)
y = y.reshape(len(y), 1)

data2 = np.concatenate((x, y), axis=1)
k_means.fit(data2)
pred_cls = k_means.predict(data2)
plt.scatter(x, y, c=pred_cls)
plt.show()
```

聚类数 $k=3$，仿真结果如下：

1. 不带(50, 50)点的聚类结果：



2.带(50, 50)点的聚类结果：



对比以上两者的kmeans聚类仿真结果，我们可以发现，kmeans算法对于离群点、噪声点比较敏感，当样本中存在离群点时，聚类效果差；且kmeans算法无法解决簇分布差别比较大的情况（比如不均衡样本）。

上机作业

Kmeans-C-1

试编程实现K-means算法，设置三组不同的k值、三组不同的初始中心点，在西瓜数据集上进行实验比较，并讨论什么样的初始中心有利于取得好结果。

我实现了kmeans算法，核心函数如下：

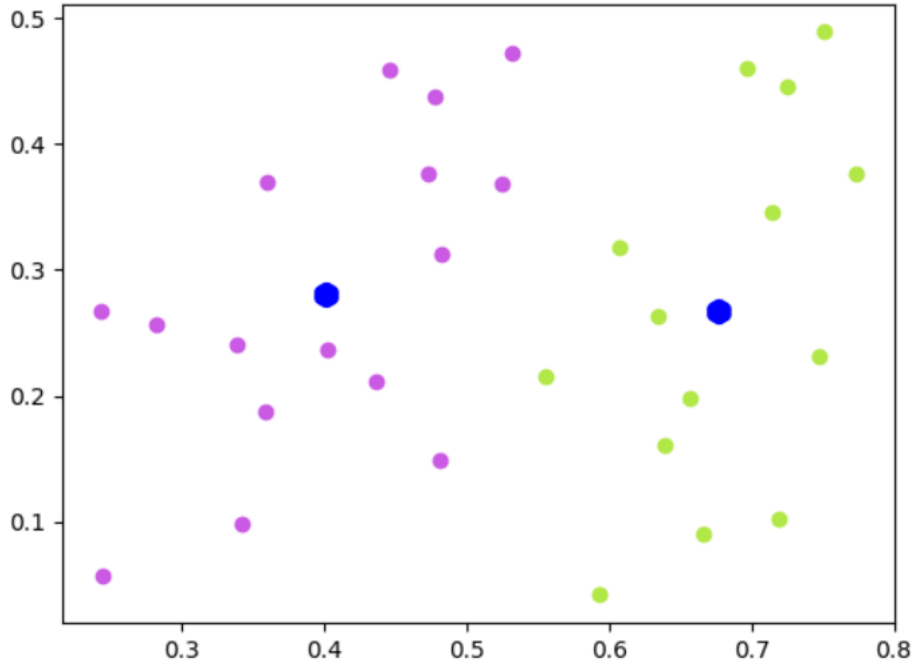
```
def calcDist(x, y):
    d = math.sqrt( ((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2) )
    return d

def calcNewCentroid(cls_pos):
    x_centroids_new = []
    y_centroids_new = []
    for centroid in cls_pos.keys():
```

```
        # 初始化新聚类中心坐标
        x_new = 0.0
        y_new = 0.0
        for pos in cls_pos[centroid]:
            x_new += pos[0]
            y_new += pos[1]
        # 以各类中点坐标的均值作为新的聚类中心点坐标
        x_new /= len(cls_pos[centroid])
        y_new /= len(cls_pos[centroid])
        # 将新中心坐标添加到列表中
        x_centroids_new.append(x_new)
        y_centroids_new.append(y_new)
    return x_centroids_new, y_centroids_new


def kmeans(x, y, num_centroids, x_centroids, y_centroids):
    n = len(x)
    epoch = 0  # 迭代次数
    while(1):
        epoch += 1
        dist = {}
        cls_pos = {}
        cls = {}
        for i in range(n):
            d_list = []
            for j in range(num_centroids):
                d = calcDist([x[i], y[i]], [x_centroids[j], y_centroids[j]])
                d_list.append(d)
            dist[i] = d_list
            # 找到距离当前点最近的中心点作为该点的聚类中心
            c = dist[i].index(min(dist[i]))
            # 在聚类字典中以c为键的值列表中添加该点
            cls.setdefault(c,[]).append(i)
            # 在聚类坐标字典中以c为键的值列表中添加该点坐标
            cls_pos.setdefault(c,[]).append([x[i], y[i]])
        # 所有点均已分到相应的类中。下求新的聚类中心坐标
        x_centroids_new, y_centroids_new = calcNewCentroid(cls_pos)
        # 当聚类中心不再变化时，停止迭代
        if (x_centroids_new == x_centroids) & (y_centroids_new ==
y_centroids):
            break
        # 更新聚类中心为新的聚类中心
        x_centroids, y_centroids = x_centroids_new, y_centroids_new
    # 返回聚类的最终结果、聚类中心坐标及迭代次数
    return epoch, x_centroids_new, y_centroids_new, cls
```

实验结果：

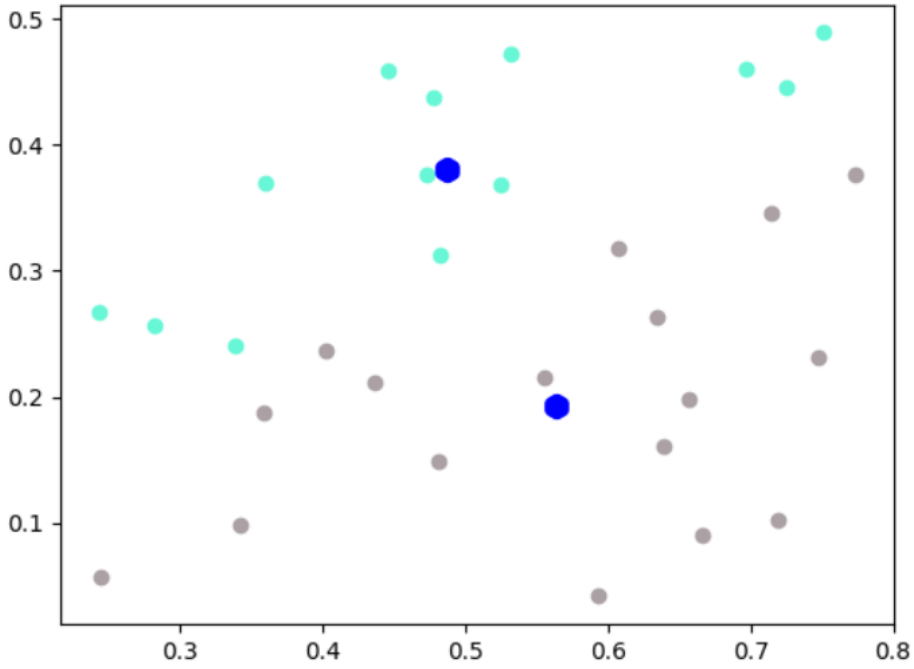
1.1 k = 2，初始聚类中心为(0.40, 0.35)和(0.65, 0.20)：

```
processing...
done! 迭代次数：3， 耗时0.0005774秒
共2类，聚类结果如下：
{0: [0, 3, 4, 5, 6, 9, 10, 12, 14, 18, 19, 20, 23, 24], 1: [1, 2, 7, 8, 11, 13, 15, 16, 17, 21, 22, 25, 26, 27, 28, 29]}
最终的聚类中心坐标：
横坐标列表：[0.6772142857142859, 0.4018125]，纵坐标列表：[0.26714285714285707, 0.28131249999999997]
```



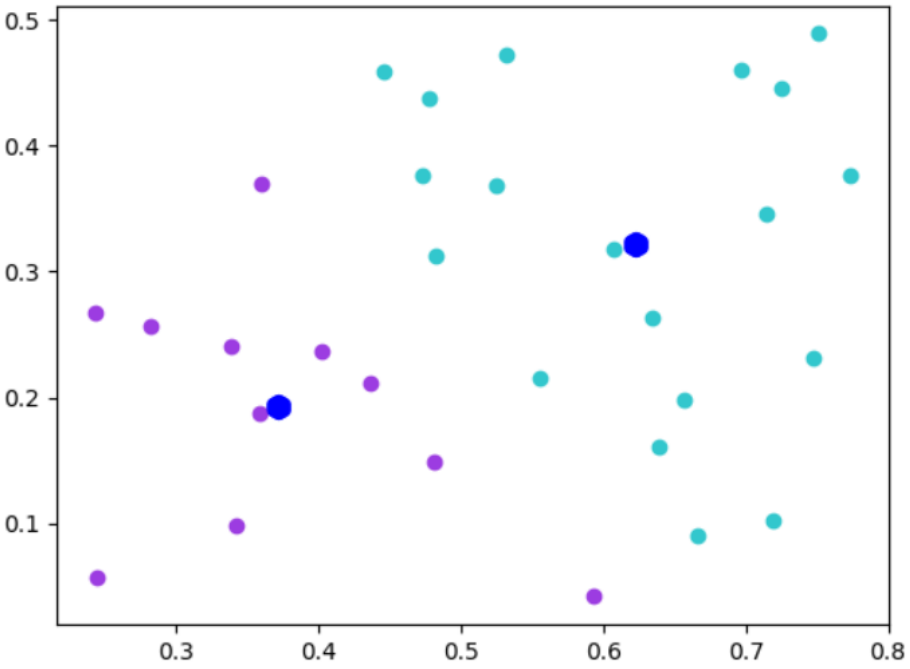
1.2 k = 2，初始聚类中心为(0.40, 0.40)和(0.42, 0.36)：

```
processing...
done! 迭代次数：4， 耗时0.0009213秒
共2类，聚类结果如下：
{0: [0, 5, 11, 16, 17, 21, 22, 23, 25, 26, 27, 28, 29], 1: [1, 2, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 24]}
最终的聚类中心坐标：
横坐标列表：[0.4872307692307692, 0.5632941176470588]，纵坐标列表：[0.381076923076923, 0.19335294117647062]
```



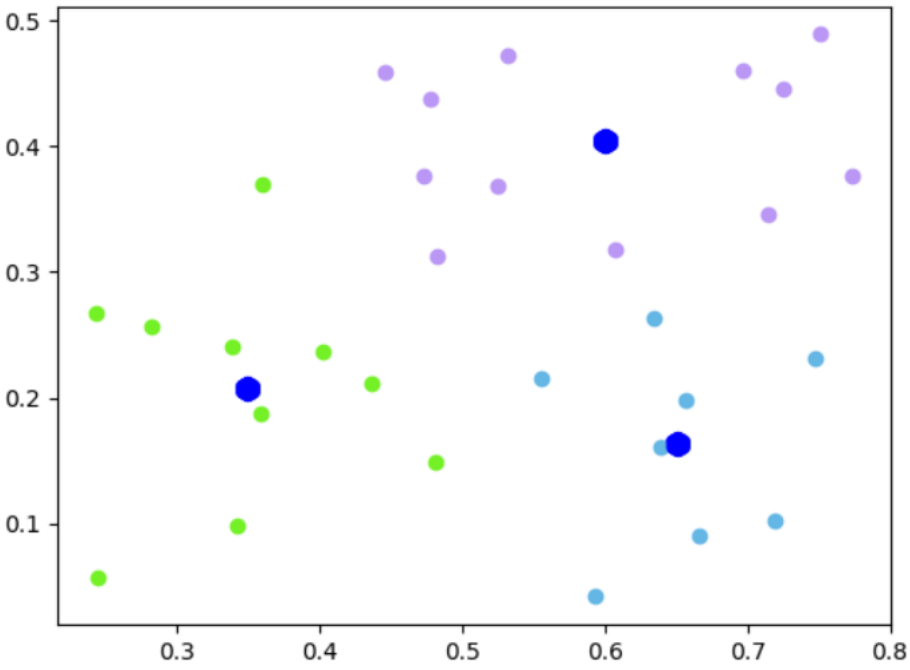
1.3 k = 2，初始聚类中心为(0.30, 0.10)和(0.70, 0.45)：

```
processing...
done! 迭代次数：2， 耗时0.0005487秒
共2类，聚类结果如下：
{0: [0, 4, 5, 6, 9, 10, 11, 12, 14, 16, 17, 18, 19, 20, 22, 23, 24, 28, 29], 1: [1, 2, 3, 7, 8, 13, 15, 21, 25, 26, 27]}
最终的聚类中心坐标：
横坐标列表：[0.6223684210526316, 0.3713636363636363]，纵坐标列表：[0.32226315789473686, 0.19254545454545455]
```



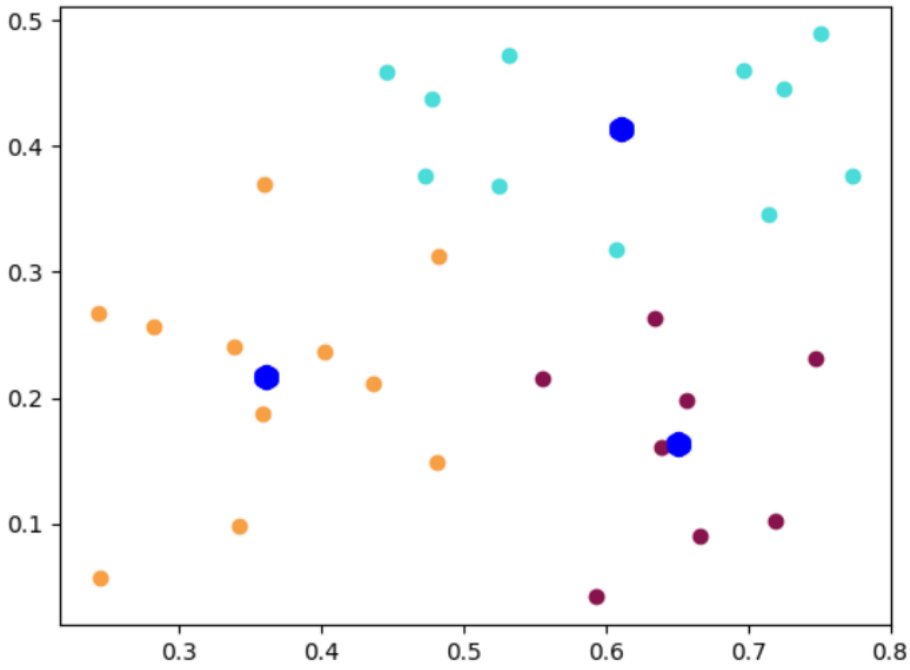
2.1 k = 3，初始聚类中心为(0.30, 0.15)、(0.60, 0.45)、(0.70, 0.20)：

```
processing...
done! 迭代次数：3， 耗时0.000977秒
共3类，聚类结果如下：
{0: [0, 5, 6, 10, 11, 16, 17, 18, 22, 23, 28, 29], 1: [1, 2, 7, 8, 13, 15, 21, 25, 26, 27], 2: [3, 4, 9, 12, 14, 19, 20, 24]}
最终的聚类中心坐标：
横坐标列表：[0.6004999999999999, 0.34919999999999995, 0.6515000000000001]，纵坐标列表：[0.4049166666666666, 0.2076, 0.16325]
```



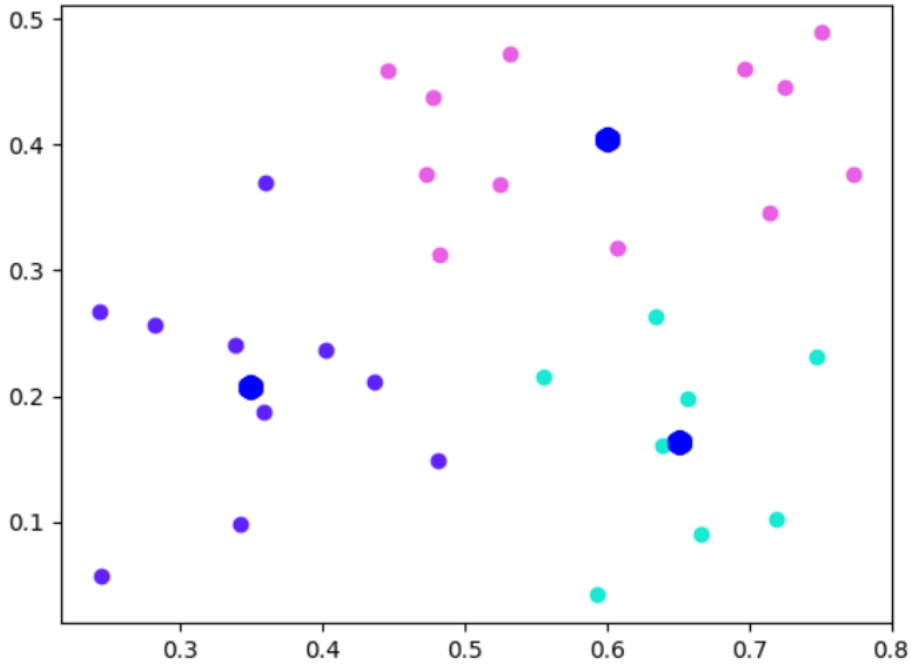
2.2 k = 3，初始聚类中心为(0.50, 0.30)、(0.55, 0.35)、(0.45, 0.35)：

```
processing...
done! 迭代次数：12， 耗时0.002793秒
共3类，聚类结果如下：
{0: [0, 5, 6, 10, 11, 17, 18, 22, 23, 28, 29], 1: [1, 2, 7, 8, 13, 15, 16, 21, 25, 26, 27], 2: [3, 4, 9, 12, 14, 19, 20, 24]}
最终的聚类中心坐标：
横坐标列表：[0.611181818181818, 0.36136363636363633, 0.6515000000000001]，纵坐标列表：[0.4133636363636363, 0.21709090909090908, 0.16325]
```



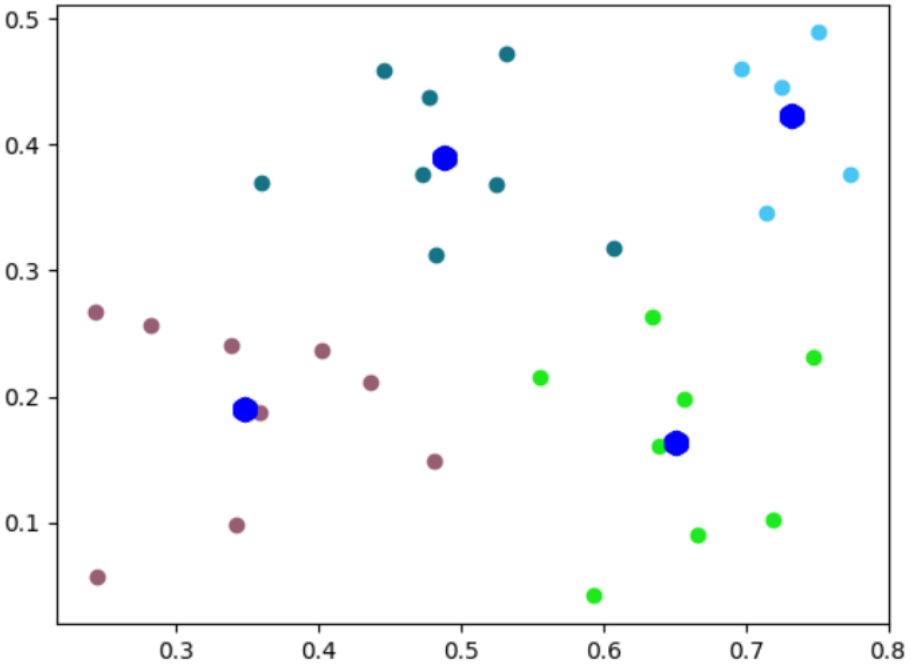
2.3 k = 3，初始聚类中心为(0.20, 0.15)、(0.50, 0.45)、(0.75, 0.10)：

```
processing...
done! 迭代次数：3， 耗时0.0005183秒
共3类，聚类结果如下：
{0: [0, 5, 6, 10, 11, 16, 17, 18, 22, 23, 28, 29], 1: [1, 2, 7, 8, 13, 15, 21, 25, 26, 27], 2: [3, 4, 9, 12, 14, 19, 20, 24]}
最终的聚类中心坐标：
横坐标列表：[0.6004999999999999, 0.34919999999999995, 0.6515000000000001]，纵坐标列表：[0.4049166666666666, 0.2076, 0.16325]
```



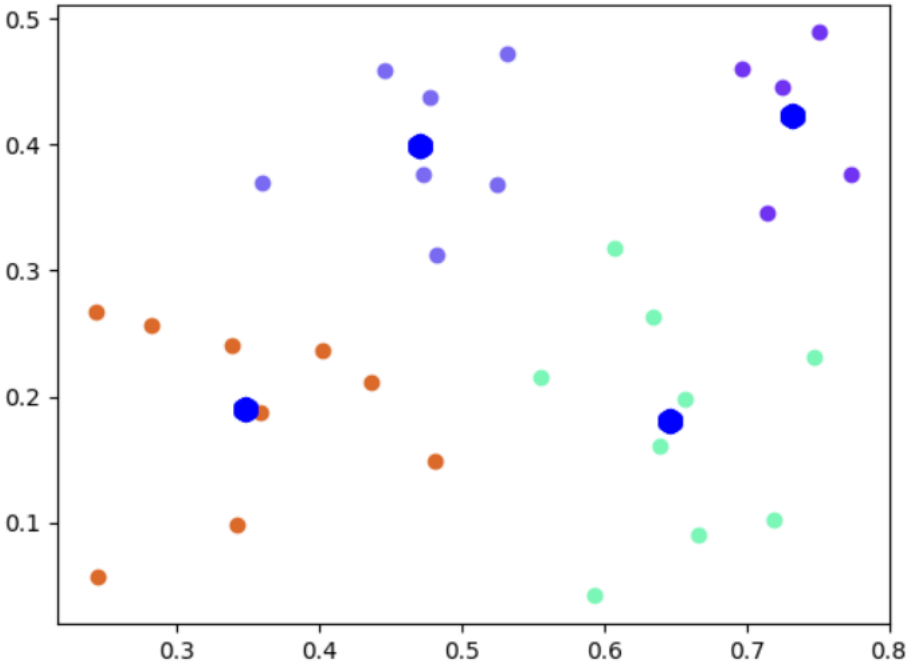
3.1 k = 4，初始聚类中心为(0.35, 0.20)、(0.50, 0.40)、(0.65, 0.15)、(0.75, 0.45)：

```
processing...
done! 迭代次数：2， 耗时0.0010966秒
共4类，聚类结果如下：
{0: [0, 5, 6, 10, 23], 1: [1, 2, 7, 8, 13, 15, 21, 25, 27], 2: [3, 4, 9, 12, 14, 19, 20, 24], 3: [11, 16, 17, 18, 22, 26, 28, 29]}
最终的聚类中心坐标：
横坐标列表：[0.7322, 0.348, 0.6515000000000001, 0.488125]，纵坐标列表：[0.4232, 0.18955555555555556, 0.16325, 0.389125]
```



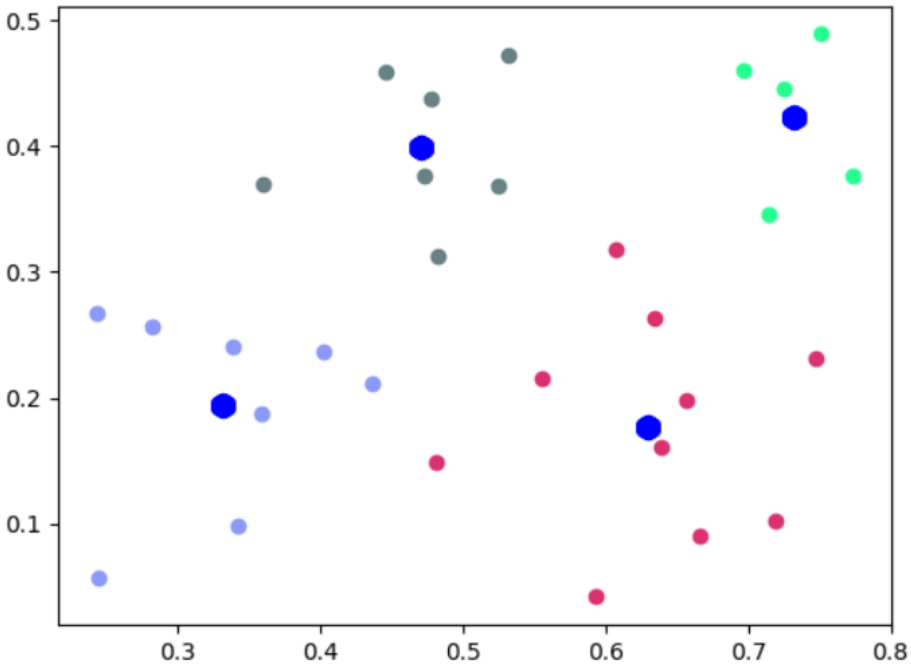
3.2 k = 4，初始聚类中心为(0.35, 0.20)、(0.50, 0.40)、(0.65, 0.15)、(0.75, 0.45)：

```
processing...
done! 迭代次数：6， 耗时0.0013104秒
共4类，聚类结果如下：
{0: [0, 5, 6, 10, 23], 1: [1, 2, 7, 8, 13, 15, 21, 25, 27], 2: [3, 4, 9, 12, 14, 18, 19, 20, 24], 3: [11, 16, 17, 22, 26, 28, 29]}
最终的聚类中心坐标：
横坐标列表：[0.7322, 0.348, 0.6466666666666667, 0.4710000000000003]，纵坐标列表：[0.4232, 0.18955555555555556, 0.18044444444444446, 0.39928571428571435]
```



3.3 k = 4，初始聚类中心为(0.25, 0.10)、(0.45, 0.50)、(0.65, 0.10)、(0.78, 0.50)：

```
processing...
done! 迭代次数：2， 耗时0.000902999999999999秒
共4类，聚类结果如下：
{0: [0, 5, 6, 10, 23], 1: [1, 2, 8, 13, 15, 21, 25, 27], 2: [3, 4, 7, 9, 12, 14, 18, 19, 20, 24], 3: [11, 16, 17, 22, 26, 28, 29]}
最终的聚类中心坐标：
横坐标列表：[0.7322, 0.33137500000000003, 0.6301, 0.47100000000000003]，纵坐标列表：[0.4232, 0.194625, 0.1773, 0.39928571428571435]
```



实验结论：

k-means算法的聚类效果与初始聚类中心位置的选取有很大关联，在实验中我发现选的初始聚类中心相互之间离得越远算法越容易收敛，聚类效果也相对较好；而当初始的聚类中心靠得很近时，算法往往需要更多次的迭代才能收敛，效率较低。