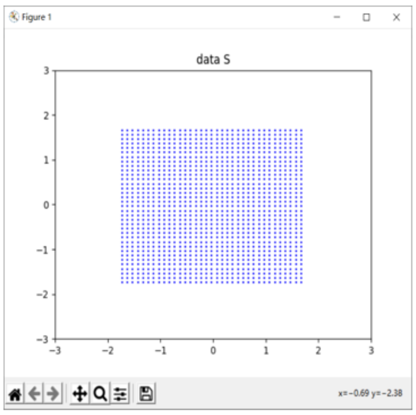
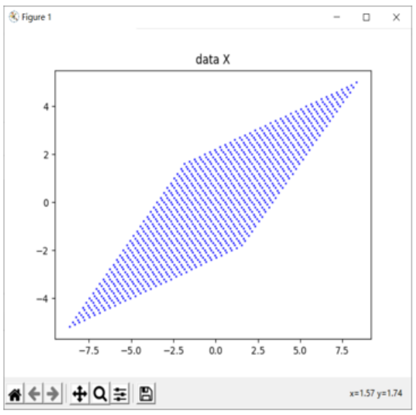
机器学习 K-means 作业

姓名： 张泽群 学号： 19049100002 班级：1班

Kmeans-1

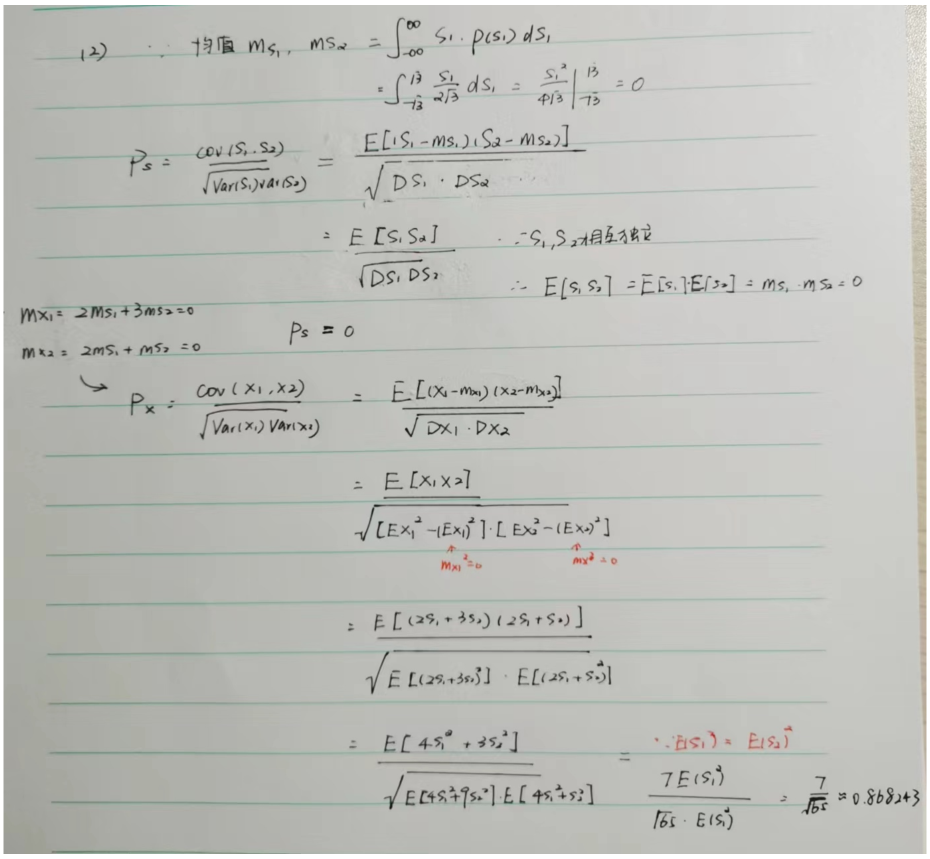
答：(1).**运行结果：**





(2). 答： $P_s = 0$ $P_x = \frac{7}{\sqrt{65}} \approx 0.868243$

数学推导：



(2) ∵ 均值 $m_{s_1}, m_{s_2} = \int_{-\infty}^{\infty} s_1 \cdot p(s_1)\, ds_1$

$= \int_{-\sqrt{3}}^{\sqrt{3}} \frac{s_1}{2\sqrt{3}}\, ds_1 = \frac{s_1^2}{4\sqrt{3}}\Big|_{-\sqrt{3}}^{\sqrt{3}} = 0$

$$P_s = \frac{\operatorname{cov}(S_1, S_2)}{\sqrt{\operatorname{Var}(S_1)\operatorname{Var}(S_2)}} = \frac{E[(S_1 - m_{s_1})(S_2 - m_{s_2})]}{\sqrt{DS_1 \cdot DS_2}}$$

$$= \frac{E[S_1 S_2]}{\sqrt{DS_1 DS_2}}$$

∵ S_1, S_2 相互独立

∴ $E[S_1 S_2] = E[S_1] \cdot E[S_2] = m_{s_1} \cdot m_{s_2} = 0$

$m_{x_1} = 2m_{s_1} + 3m_{s_2} = 0$

$m_{x_2} = 2m_{s_1} + m_{s_2} = 0$

$P_s = 0$

$$P_x = \frac{\operatorname{cov}(x_1, x_2)}{\sqrt{\operatorname{Var}(x_1)\operatorname{Var}(x_2)}} = \frac{E[(x_1 - m_{x_1})(x_2 - m_{x_2})]}{\sqrt{Dx_1 \cdot Dx_2}}$$

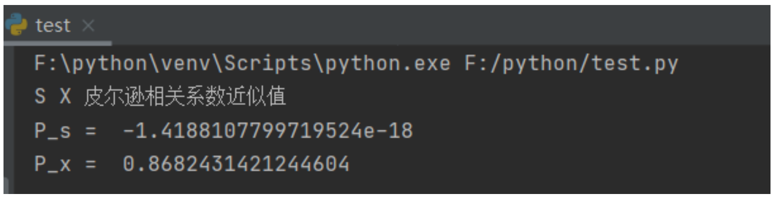
$$= \frac{E[x_1 x_2]}{\sqrt{[Ex_1^2 - \underset{m_{x_1}^2 = 0}{(Ex_1)^2}] \cdot [Ex_2^2 - \underset{m_{x}^2 = 0}{(Ex_2)^2}]}}$$

$$= \frac{E[(2s_1 + 3s_2)(2s_1 + s_2)]}{\sqrt{E[(2s_1 + 3s_2)^2] \cdot E[(2s_1 + s_2)^2]}}$$

$$= \frac{E[4s_1^2 + 3s_2^2]}{\sqrt{E[4s_1^2 + 9s_2^2] \cdot E[4s_1^2 + s_2^2]}} = \frac{7E(s_1^2)}{\sqrt{65} \cdot E(s_1^2)} = \frac{7}{\sqrt{65}} \approx 0.868243$$

∵ $E(s_1^2) = E(s_2^2)$

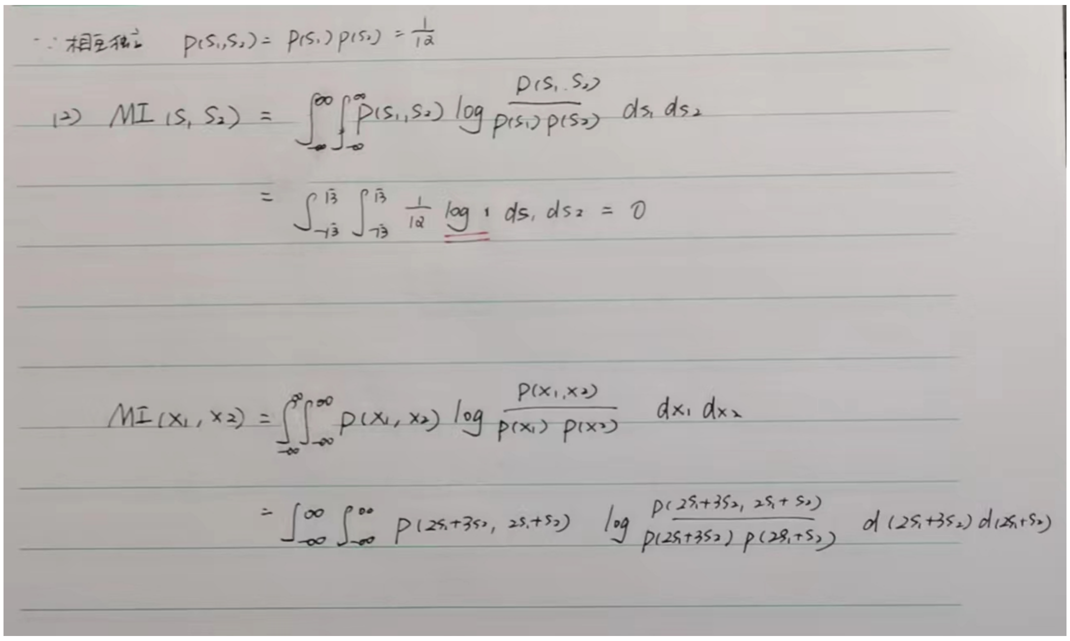
计算机模拟结果：



```
test
F:\python\venv\Scripts\python.exe F:/python/test.py
S X 皮尔逊相关系数近似值
P_s =  -1.4188107799719524e-18
P_x =  0.8682431421244604
```

(3). 答：

数学推导： data X 推导不出来



$\because$ 相互独立 $p(s_1,s_2) = p(s_1)p(s_2) = \frac{1}{12}$

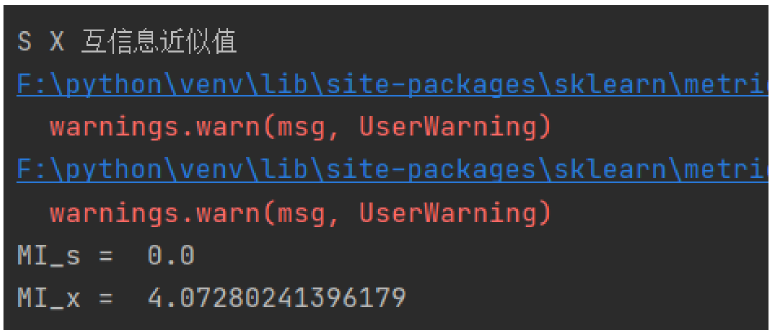
$$(2)\quad MI(s_1, s_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(s_1,s_2) \log \frac{p(s_1, s_2)}{p(s_1)p(s_2)}\, ds_1\, ds_2$$

$$= \int_{-\sqrt{3}}^{\sqrt{3}}\int_{-\sqrt{3}}^{\sqrt{3}} \frac{1}{12} \underline{\underline{\log}}\, 1\, ds_1\, ds_2 = 0$$

$$MI(x_1, x_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x_1, x_2) \log \frac{p(x_1,x_2)}{p(x_1)\,p(x_2)}\, dx_1\, dx_2$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(2s_1+3s_2,\ 2s_1+s_2) \log \frac{p(2s_1+3s_2,\ 2s_1+s_2)}{p(2s_1+3s_2)\,p(2s_1+s_2)}\, d(2s_1+3s_2)\, d(2s_1+s_2)$$

计算机模拟结果：



```
S X 互信息近似值
F:\python\venv\lib\site-packages\sklearn\metri
  warnings.warn(msg, UserWarning)
F:\python\venv\lib\site-packages\sklearn\metri
  warnings.warn(msg, UserWarning)
MI_s =  0.0
MI_x =  4.07280241396179
```

源代码：

```python
# -*- coding: utf-8 -*-

from math import *
from sklearn import metrics
import numpy as np
from matplotlib.pylab import plt


if __name__ == '__main__':
    s1 = np.arange(-1.73, 1.74, 0.01)
    s2 = np.arange(-1.73, 1.74, 0.01)
    s1_m, s2_m = np.meshgrid(s1, s1)
    A = np.array([[2, 3], [2, 1]])
    x1_m = 2*s1_m+3*s2_m
    x2_m = 2*s1_m+s2_m
    # 画散点图
    plt.title('data S')
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    plt.scatter(s1_m[:], s2_m[:], c='b', marker='o', s=1)
    plt.show()
    plt.title('data X')
    plt.scatter(x1_m[:], x2_m[:], c='b', marker='o', s=1)
    plt.show()

    # 计算相关系数
    s1_mean = np.mean(s1_m)
    s2_mean = np.mean(s2_m)
    s1_varSum, s2_varSum, s_cov = 0, 0, 0
    x1_mean = np.mean(x1_m)
    x2_mean = np.mean(x2_m)
    x1_varSum, x2_varSum, x_cov = 0, 0, 0

    for i in range(len(s1_m)):
        for j in range(len(s1_m)):
            temps1 = s1_m[i][j] - s1_mean
            temps2 = s2_m[i][j] - s2_mean
            s_cov += temps1 * temps2
            s1_varSum += pow(temps1, 2)
            s2_varSum += pow(temps2, 2)
            #
            tempx1 = x1_m[i][j] - x1_mean
            tempx2 = x2_m[i][j] - x2_mean
            x_cov += tempx1 * tempx2
            x1_varSum += pow(tempx1, 2)
            x2_varSum += pow(tempx2, 2)

    p_s = s_cov / sqrt(s1_varSum*s2_varSum)
    p_x = x_cov / sqrt(x1_varSum*x2_varSum)
    print('S X 皮尔逊相关系数近似值')
    print('P_s = ', p_s)
    print('P_x = ', p_x)

    # 计算互信息
    print()
    print('S X 互信息近似值')
    s1_i = np.reshape(s1_m, -1)
    s2_i = np.reshape(s2_m, -1)
    x1_i = np.reshape(x1_m, -1)
    x2_i = np.reshape(x2_m, -1)
    print('MI_s = ', metrics.mutual_info_score(s1_i, s2_i))
    print('MI_x = ', metrics.mutual_info_score(x1_i, x2_i))
```

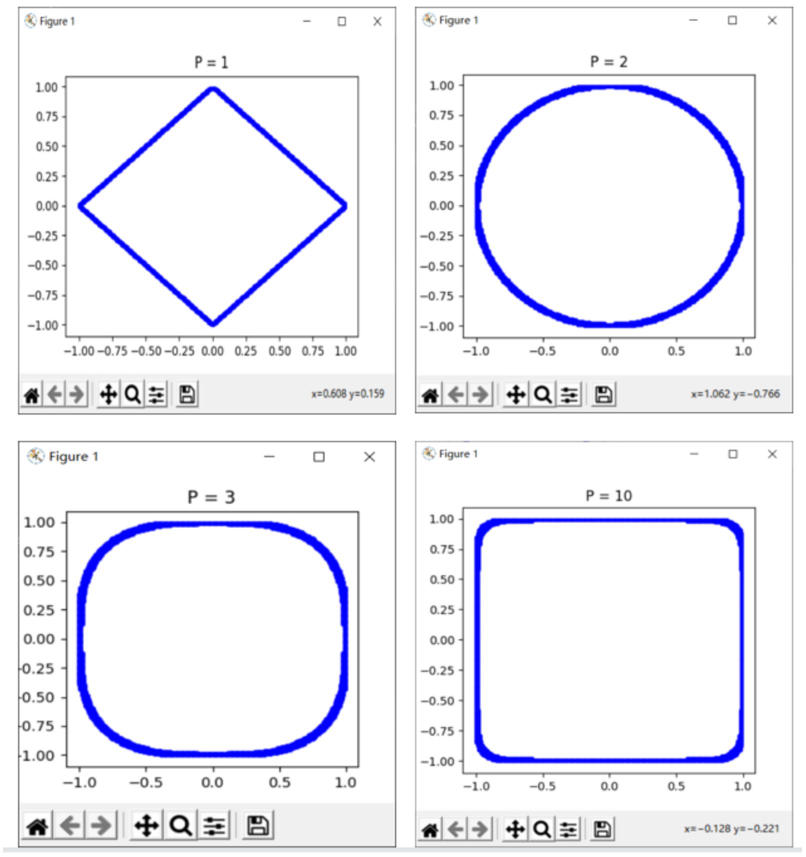
Kmeans-2

答:

```python
# -*- coding: utf-8 -*-

from math import *
from decimal import Decimal
import numpy as np
from matplotlib.pylab import plt


def nth_root(value, n_root):
    root_value = 1 / float(n_root)
    return round(Decimal(value) ** Decimal(root_value), 3)


def minkowski_distance(x, y, p_value):
    return nth_root(sum(pow(abs(a - b), p_value) for a, b in zip(x, y)), p_value)


def minkowski_pts(m_dist, p_value):
    Pts_x,Pts_y = [],[]
    x = np.arange(-1, 1, 0.01)
    y = np.arange(-1, 1, 0.01)
    x_m, y_m = np.meshgrid(x, y)
    fig = plt.figure(1, figsize=(12, 8))
    z_m = np.zeros([len(x_m),len(x_m)])
    for i in range(len(x_m)):
        for j in range(len(x_m)):
            z_m[i][j] = minkowski_distance([0, 0], [x_m[i][j], y_m[i][j]], p_value)
            if abs(z_m[i][j]-m_dist) < 0.02:
                Pts_x.append(x_m[i][j])
                Pts_y.append(y_m[i][j]);
    return Pts_x, Pts_y


if __name__ == '__main__':
    # print(minkowski_distance([0, 0], [2, 2], 3))
    Pts_x, Pts_y = minkowski_pts(1, 10)
    plt.title('P = 10')
    plt.scatter(Pts_x[:], Pts_y[:], c='b', marker='o', s=5)
    plt.show()
```

结果：

这里利用以 minkowski_dist = 1 画数据轮廓线：



结论： 参数P越大，数据轮廓线越往外扩，其四角的(x, y)值越接近原先设定闵可夫斯基距离。其轮廓形状越来越接近正方形，当 $p \to \infty$ 时，数据线轮廓变为正方形。

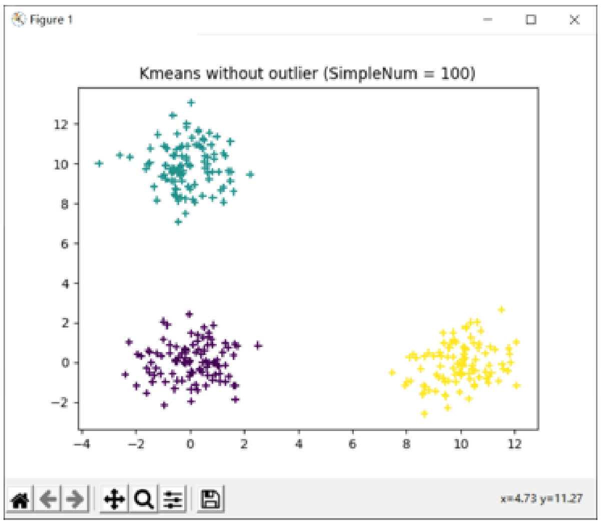
这是因为此时 minkowski_dist = $max\ |x_i - y_i|$

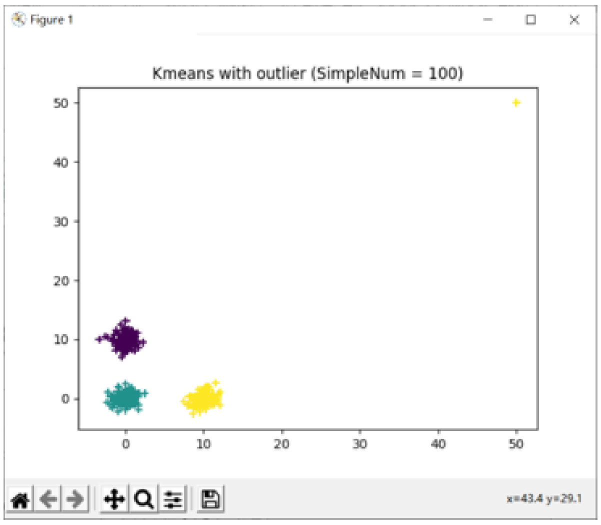
Kmeans-3

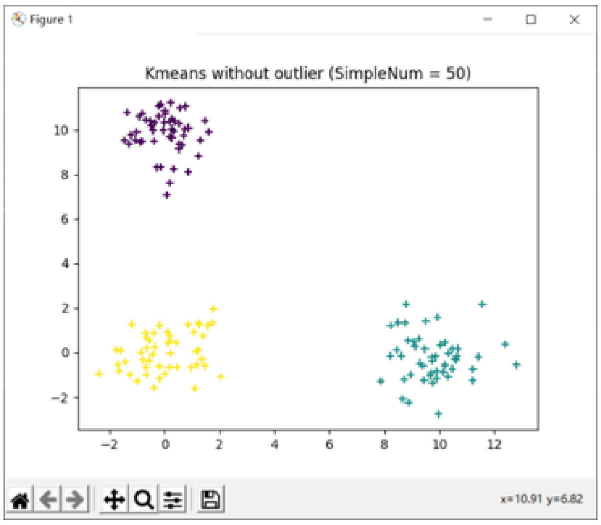
答:

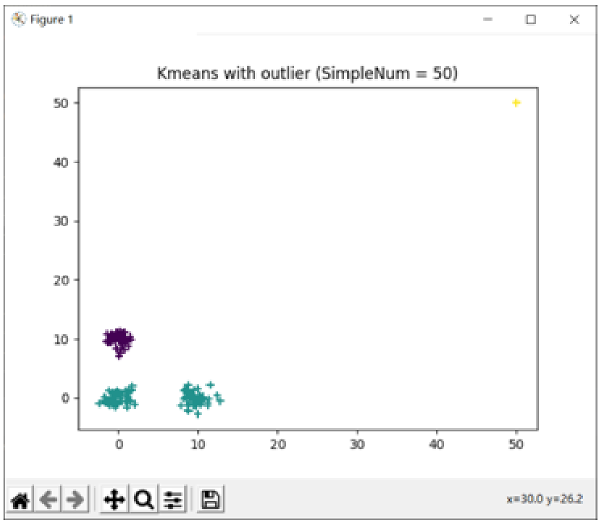
```python
# -*- coding: utf-8 -*-

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

def create_data():
    sample_num = 50
    cov = np.identity(2)
    mean = [[0, 0], [0, 10], [10, 0]]
    s1 = np.random.multivariate_normal(mean[1], cov, sample_num)
    s2 = np.random.multivariate_normal(mean[2], cov, sample_num)
    s3 = np.random.multivariate_normal(mean[0], cov, sample_num)
    outlier = np.array([50, 50])
    dataSet1 = np.vstack((s1, s2, s3))
    dataSet2 = np.vstack((dataSet1, outlier))

    return dataSet1, dataSet2


if __name__ == '__main__':
    dataSet1, dataSet2 = create_data()  # 分别是不含离群点和含有离群点的高斯分布数据集
    # 不含离群点
    k_means = KMeans(n_clusters=3, random_state=10)
    k_means.fit(dataSet1)
    l_predict = k_means.predict(dataSet1)
    plt.title('Kmeans without outlier (SimpleNum = 50)')
    plt.scatter(dataSet1[:, 0], dataSet1[:, 1], c=l_predict, marker = '+')
    plt.show()

    # 含有离群点
    k_means = KMeans(n_clusters=3, random_state=10)
    k_means.fit(dataSet2)
    l_predict = k_means.predict(dataSet2)
    plt.title('Kmeans with outlier (SimpleNum = 50)')
    plt.scatter(dataSet2[:, 0], dataSet2[:, 1], c=l_predict, marker='+')
    plt.show()
```

运行结果：









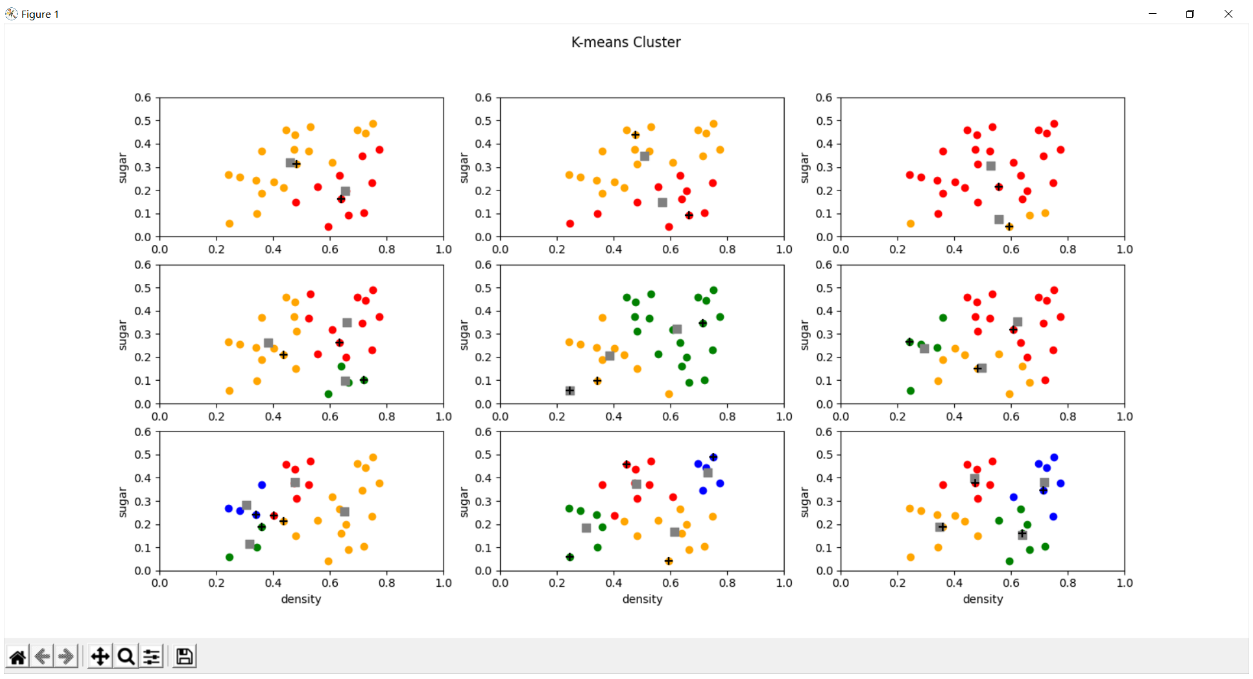
结论： 由实验结果可见，当k值合适（如图k值等于3时），能对无离群点的数据集进行正确的聚类，而当数据集存在离群点时，仅有在数据集样本量较大的情况下才能正确聚类。

从而可以得出结论，Kmeans算法抗噪声能力较弱，但由于Kmeans算法只有一个超参数K，所以当k值合适并且数据集噪声较小的情况总能获得正确的聚类结果，所以其鲁棒性较强。

Kmeans-C-1

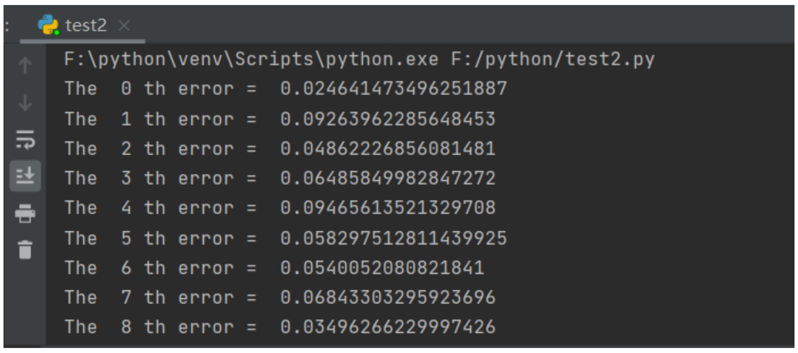
```python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt


def kmeans(data, center_ids, max_err=0.0001, max_round=30):
    init_centers = []
    n = len(center_ids)
    for ids in center_ids:
        init_centers.append(data[ids, :])
    error, rounds = 1.0, 0
    while error > max_err and rounds < max_round:
        rounds += 1
        clusters = []
        for _ in range(n):
            clusters.append([])
        for j in range(len(data)):
            dist = []
            for i in range(n):
                vector = data[j, :] - init_centers[i]
                d_ji = np.dot(vector, vector) ** 0.5
                dist.append(d_ji)
            near_id = sorted(enumerate(dist), key=lambda x: x[1])[0][0]
            clusters[near_id].append(j)

        new_center = [0] * n
        error = 0
        for i in range(n):
            new_center[i] = np.sum(data[clusters[i], :], axis=0)
            new_center[i] /= len(clusters[i])
            vec = new_center[i] - init_centers[i]
            err = np.dot(vec, vec) ** 0.5
            if err:
                init_centers[i] = new_center[i]
                error += err
        yield clusters, new_center, err  # 用yield可以得到每一轮训练后的聚类情况，最终返
回的是一个生成器
        return error


# 西瓜数据集4.0: 密度 含糖率
data = np.array([
    [0.697, 0.460], [0.774, 0.376], [0.634, 0.264], [0.608, 0.318], [0.556, 0.215],
    [0.403, 0.237], [0.481, 0.149], [0.437, 0.211], [0.666, 0.091], [0.243, 0.267],
    [0.245, 0.057], [0.343, 0.099], [0.639, 0.161], [0.657, 0.198], [0.360, 0.370],
    [0.593, 0.042], [0.719, 0.103], [0.359, 0.188], [0.339, 0.241], [0.282, 0.257],
    [0.748, 0.232], [0.714, 0.346], [0.483, 0.312], [0.478, 0.437], [0.525, 0.369],
    [0.751, 0.489], [0.532, 0.472], [0.473, 0.376], [0.725, 0.445], [0.446, 0.459]])

if __name__ == '__main__':
    # k=2,3,4
    centerPts = [[12, 22], [8, 23], [4, 15],
                 [2, 7, 16], [10, 11, 21], [3, 6, 9],
                 [5, 7, 17, 18],  # 相互靠近的点
                 [29, 15, 10, 25],  # 分散而外周的点
                 [27, 17, 12, 21]]  # 对应的是选择的初始中心样本的id，这也同时代表了选择
的聚类数目
    fig, axes = plt.subplots(3, 3, figsize=(5, 5))
    ax = axes.flatten()
    for i in range(0, len(ax)):
        ax[i].set_xlim(0, 1)
        ax[i].set_ylim(0, 0.6)
        ax[i].set_ylabel('sugar')
        if i == 6 or i == 7 or i == 8:
            ax[i].set_xlabel('density')

    plt.suptitle('K-means Cluster')

    for k in range(0, len(centerPts)):
        for cluster, center, error in kmeans(data, centerPts[k]):  # 对各轮聚类的结果进
行保存，存入imgs
            pics, dye = [], ['red', 'orange', 'green', 'blue', 'pink']
            for i, item in enumerate(cluster):
                # 聚类 和 中心点
                pics.append(ax[k].scatter(data[item, 0], data[item, 1], c=dye[i]))
        print('The ', k, 'th error = ', error)
        Center_x = np.zeros(len(center))
        Center_y = np.zeros(len(center))
        init_x = np.zeros(len(center))
        init_y = np.zeros(len(center))
        for i in range(0, len(center)):
            Center_x[i], Center_y[i] = center[i][0], center[i][1]
            init_x[i], init_y[i] = data[centerPts[k]][i][0], data[centerPts[k]][i][1]

        pics.append(ax[k].scatter(Center_x, Center_y, s=45, c='gray', marker='s'))
        pics.append(ax[k].scatter(init_x, init_y, s=45, c='black', marker='+'))
    plt.show()
```

运行结果：



正方形表示kmeans后的聚类中心，'+'号表示原先的初始中心点，不同颜色代表不同聚类。

K=2，3，4，每个k值测试了不同的三组不同中心点，其SSE如下：



```
test2
F:\python\venv\Scripts\python.exe F:/python/test2.py
The  0 th error =  0.024641473496251887
The  1 th error =  0.09263962285648453
The  2 th error =  0.04862226856081481
The  3 th error =  0.06485849982847272
The  4 th error =  0.09465613521329708
The  5 th error =  0.058297512811439925
The  6 th error =  0.054005208082184l
The  7 th error =  0.06843303295923696
The  8 th error =  0.03496266229997426
```

讨论： 在k=2和3时，不同的聚类中心所得结果相差不明显，所得结果相近；k=4时，初始中心选择分散时，最终的平方误差(按9.24式计算)更小一些。因此，就当前所尝试的有限情况而言，貌似可以得到结论：当k较小时，不同的初始中心选取差别不大，当k较大时，选取，选择较分散的初始中心更有利于取得较好结果。

结论： k-means算法的好坏与初始样本的选取有很大关系。k-means算法选的初始点离得越远越容易收敛，聚类效果也越好。