实验三：K 均值算法和模糊 C 均值算法

1 问题描述

编程实现 K-means 算法和 FCM 算法，并对比两者的性能，要求：

1. 查阅无监督聚类的评价标准有哪些，选择其中一个标准作为后续试验的验证指标。

2. 在 sonar 和 Iris 数据上分别验证两种聚类算法。

2 数据集说明

2.1 Iris 数据集

Iris 数据集中包含了 3 类鸢尾花特征数据。每一类分别有 50 个样本，每条样本有 4 个维度的特征数据（花萼长度，花萼宽度，花瓣长度，花瓣宽度）。

2.2 Sonar 数据集

Sonar 数据集，通过声纳从不同角度返回的强度来预测目标是岩石还是矿井，其中 R 类代表岩石，M 类代表矿井。共有 208 个样本，60 个维度，2 个类别。

3 K 均值算法

3.1 算法原理

K 均值聚类算法是应用最广泛的基于划分的聚类算法之一，适用于处理大样本数据。它是一种典型的基于相似性度量的方法, 目标是根据输入参数 K 将数据集划分为 K 簇。由于初始值、相似度、聚类均值计算策略的不同，因而有很多种 K 均值算法的变种。在数据分布接近球体的情况下，K 均值算法具有较好的聚类效果。

算法目标：使得各个数据与其对应聚类中心点的误差平方和最小。

$$J = \sum_{i=1}^{k} J_i = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - m_i\|^2$$

式中，J_i 为第 i 类聚类的目标函数，k 为聚类个数，x 是划分到类 C_i 的样本。

$m_1, ..., m_k$ 是类 $C_1, ..., C_k$ 的质心。

$$m_i = \frac{1}{N_i} \sum_{x \in C_i} x$$

3.2 算法流程

Step 1: 初始化：随机选择 k 个样本点，并将其视为各聚类的初始中心 $m_1, ..., m_k$；

Step 2: 按照最小距离法则逐个将样本 x 划分到以聚类中心 $m_1, ..., m_k$ 为代表的 k 个类 $C_1, ..., C_k$ 中；

Step 3: 计算聚类准则函数 J，重新计算 k 个类的聚类中心 $m_1, ..., m_k$；

Step 4: 重复 Step2 和 Step3 直到聚类中心 $m_1, ..., m_k$ 无改变或目标函数 J 不减小。

4 模糊 C 均值算法

4.1 算法原理

K 均值算法属于硬聚类算法，它把数据点划分到确切的某一聚类中。而在模糊聚类亦称软聚类中，数据点则可能归属于不止一个聚类中，并且这些聚类与数据点通过一个成员水平 (实际上类似于模糊集合中

隶属度的概念) 联系起来。成员水平显示了数据点与某一聚类之间的联系很密切。模糊聚类就是计算这些成员水平，按照成员水平来决定数据点属于哪一个或哪些聚类的过程。模糊 C 均值算法 (Fuzzy C-Means，FCM) 是模糊聚类算法中使用最广泛的算法之一。

FCM 的目标函数是把 m 个样本分为 c 个模糊集合，并给出聚类中心，使得代价函数的值最小。我们构建一个隶属矩阵 U，其中 u_{ij} 表示第 i 条样本对于第 j 个模糊集合的隶属度。FCM 进行归一化约束后，样本数据属于所有类的隶属度的总和应该等于 1，即：

$$\sum_{j=1}^{c} u_{ij} = 1$$

FCM 的目标函数定义为：

$$J\left(U, z_1, \ldots z_c\right) = \sum_{j=1}^{c} J_j = \sum_{j=1}^{c}\sum_{i=1}^{m} u_{ij}^{\alpha} d_{ij}^2$$

其中，z_j 为第 j 个模糊集合的聚类中心；d_{ij} 表示第 i 条样本与第 j 个聚类中心间的欧式距离；α 为柔性参数。

我们需要让目标函数达到最小，此时的必要条件为：

$$\bar{J}\left(U, z_1, \ldots, z_m, \lambda_1, \ldots \lambda_m\right) = \sum_{j=1}^{c}\sum_{i=1}^{m} u_{ij}^{\alpha} d_{ij}^2 + \sum_{i=1}^{m} \lambda_j \left(\sum_{j=1}^{c} u_{ij} - 1\right)$$

我们对输入参量进行求导，从而得到目标函数达到最小值的条件：

$$z_j = \frac{\sum_{i=1}^{m} u_{ij}^{\alpha} x_i}{\sum_{i=1}^{m} u_{ij}^{\alpha}} u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{ki}}\right)^{\frac{2}{\alpha-1}}}$$

4.2 算法流程

Step 1: 初始化隶属矩阵 U；

Step 2: 根据隶属矩阵 U，计算各个聚类中心 $m^{(s)}$；

Step 3: 计算代价函数 J，重新计算 c 个类的聚类中心 $m^{(s+1)}$；

Step 4: 更新隶属矩阵 U；

Step 5: 重复 Step3 和 Step4 直到聚类中心 $\left\|m^{(s)} - m^{(s+1)}\right\| < \varepsilon$；

输出：将样本点划分为隶属度最大的那一类。

5 无监督聚类的评价标准

在无监督学习中，数据没有标签。聚类之后，只能得到聚类的结果，并不知道结果是否正确，因此我们无法判断其准确率。对此，我们需要引入评价无监督聚类结果的好坏的评价指标。查阅相关资料，常用的有六种评价指标。

5.1 纯度 (Purity)

纯度：代表正确聚类的类别数占总类别数地比例。其计算公式如下：

$$purity(\Omega, C) = \frac{1}{N}\sum_{k} \max_{j} \left|\omega_k \cap C_j\right|$$

其中 N 代表总类别数，w_k 代表第 k 个聚类，C 代表类别集合，C_j 表示第 j 类。其结算结果必然在 [0,1] 之间，完全错误时其值为 0，完全正确时其值为 1。

5.2 熵 (Entropy)

对于一个聚类 i, 首先计算 P_{ij}，P_{ij} 指的是聚类 i 中的成员属于类 j 的概率

$$P_{ij} = \frac{m_{ij}}{m_i}$$

其中 m_i 是在聚类 i 中所有成员地个数，m_{ij} 是聚类 i 中成员属于 j 类的个数。
每个聚类的熵可以表示为

$$e_i = -\sum_{j=1}^{L} P_{ij} \log 2P_{ij}$$

其中 L 是类的个数。
整个聚类划分的熵为

$$e = \sum_{i=1}^{k} \frac{m_i}{m} e_i$$

其中 K 是聚类的数目，m 是整个聚类划分所涉及到的成员个数。
划分的熵越小，说明聚类效果越好。

5.3 标准化互信息 (NMI)

互信息用于衡量两个信息之间的相关性，对于两个随机变量 X 和 Y，互信息的公式如下

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

理论上，互信息的值越大越好，可是其取值范围是没有上边界的。为了更好的比较不同聚类结果，提出了标准化互信息的概念，公式如下

$$U(X,Y) = 2R = 2\frac{I(X;Y)}{H(X) + H(Y)}$$

将互信息的值归一化到 0 和 1 之间，称作标准化互信息。标准化互信息的值越接近 1，聚类效果越好。

5.4 调整互信息 (AMI)

调整互信息的公式如下

$$\mathrm{AMI} = \frac{\mathrm{MI} - E[\mathrm{MI}]}{mean(H(U), H(V)) - E[\mathrm{MI}]}$$

其中 E 表示期望值，对应的公式如下

$$E[MI(U,V)] = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \sum_{n_{ij}=(a_i+b_{j-N})^+}^{\min(a_i,b_j)} \frac{n_{ij}}{N} \log \left(\frac{N \cdot n_{ij}}{a_i b_j} \right) \frac{a_i! b_j! (N-a_i)! (N-b_j)!}{N! n_{ij}! (a_i - n_{ij})! (b_j - n_{ij})! (N - a_i - b_j + n_{ij})!}$$

互信息和归一化互信息的值都会受到聚类的类别数 K 的影响，而 AMI 则不会受到干扰，取值范围为 [-1,1]，数值越大，两种聚类结果越接近。

5.5 兰德指数 (AMI)

兰德指数公式如下

$$RI = (a+b)/\left(C_2^n\right)$$

其中 C 表示实际类别信息，K 表示聚类结果，a 表示在 C 与 K 中都是同类别的元素对数，b 表示在 C 与 K 中都是不同类别的元素对数。RI 的取值为 [0,1]，值越大表示聚类结果与真实情况越吻合。

5.6 调整兰德指数 (ARI)

调整兰德指数的公式如下

$$\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$$

ARI 的取值范围为 [-1,1]，数值越大，聚类效果越好。

6 实验结果

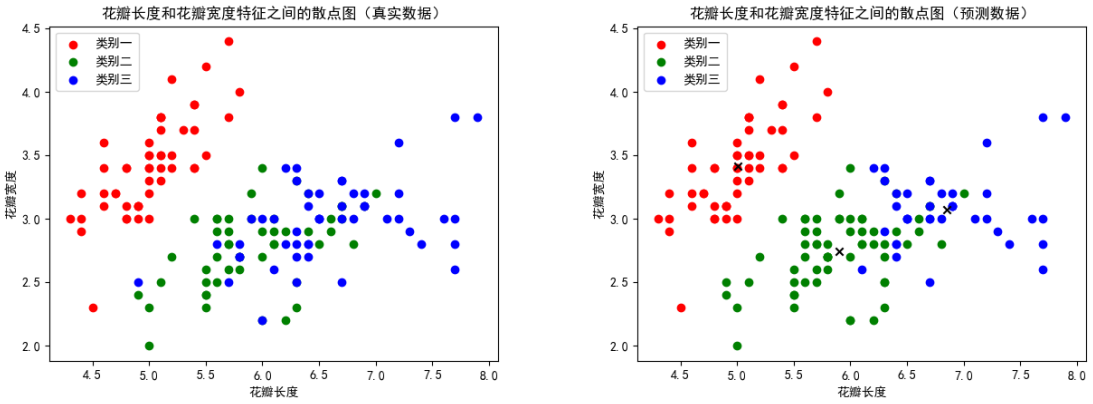
本实验选取**纯度**作为验证指标。

6.1 k 均值聚类

6.1.1 Iris 数据集

本次实验设定 k 值为 3，通过随机选取任意 3 个样本作为初始聚类中心，之后进行迭代。由于 k 均值聚类算法的结果受初始聚类中心位置的影响较大，因此，本实验采取了进行 10 次实验，求取平均值，计算出最后的结果：平均迭代次数为 **6.8**，平均聚类纯度为 **86.73%**。

为了进一步可视化聚类的效果，本实验选取了花瓣长度和花瓣宽度两个特征，作真实类别和预测类别的散点图，如下图所示：



(a) 花瓣长度和花瓣宽度特征之间的散点图（真实数据）　(b) 花瓣长度和花瓣宽度特征之间的散点图（预测数据）

图 1: 花瓣长度和花瓣宽度特征之间的散点对比图

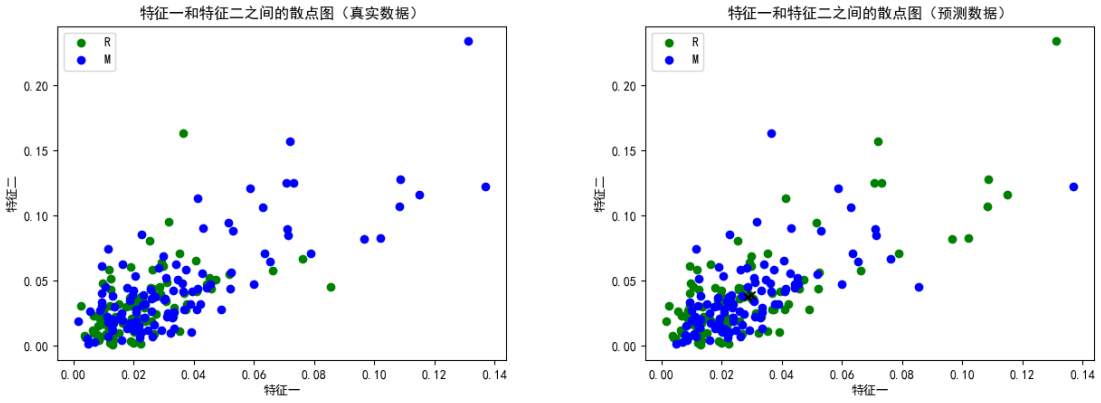
右图中的黑色“x”代表聚类中心。从图中可以发现，类别一的数据几乎完全聚类成功，类别二和类别三的样本则较难聚类，特别在两者的交界处，聚类效果较差。对比真实数据可以发现，类别二和类别三的数据较为混杂，这增大了 k 均值聚类的难度。

从图中还可以看见，类别二的聚类中心旁边出现了其它类别的样本，这是由于聚类时采用的是四个维度的特征，而在此处仅选取了两个维度特征，因此出现这种现象。

6.1.2 sonar 数据集

对于 sonar 数据集，本次实验设定 k 值为 2，通过随机选取任意 2 个样本作为初始聚类中心，之后进行迭代。和之前类似，本实验采取了进行 10 次实验，求取平均值的方式，计算出最后的结果：平均迭代次数为 **14.3**，平均聚类纯度为 **54.33%**。

为了进一步可视化聚类的效果，本实验选取了前两个维度的特征，作真实类别和预测类别的散点图，如下图所示：



(a) 特征一和特征二之间的散点图（真实数据）　(b) 特征一和特征二特征之间的散点图（预测数据）

图 2: sonar 数据两个特征之间的散点对比图

从图中可以发现，两个类别的聚类中心在特征一和特征二中离得很近，这也导致了准确率不高。因此，k 均值聚类对于大量线性不可分的数据来说，效果不佳。

6.2 模糊 C 均值聚类

6.2.1 Iris 数据集

对于 Iris 数据集，本次实验设定 c 值为 3，先随机初始化隶属矩阵，之后计算聚类中心，进行迭代。考虑到 FCM 算法结果受到柔性参数 α 的影响。因此，本实验确定阈值 ε 为 0.001，分别选取数 α 为 1.5，2，3，4，6，8，10，分别进行 10 次实验计算平均整体纯度。结果如下表所示：

表 1: Iris 数据集不同 α 的实验结果表

α	平均聚类纯度	平均迭代次数
1.5	88.67%	15.4
2	89.33%	14.4
3	90.60%	14.3
4	90.20%	13.7
6	92.00%	11.3
8	92.13%	7.5
10	86.40%	4.8

从表中可以发现，当柔性参数 α 取值为 6-8 时，平均聚类纯度最高。此外，随着 α 取值增大，平均迭代次数不断减小。这表明 α 越大，目标函数的变化量越大，从而能更快收敛到最优值，而当 α 过大时，目标函数变化过快，则会越过最优值，从而导致聚类纯度降低。

6.2.2 sonar 数据集

对于 sonar 数据集，本次实验设定 c 值为 2，先随机初始化隶属矩阵，之后计算聚类中心，进行迭代。和之前类似，本实验确定阈值 ε 为 0.001，分别选取数 α 为 1.5，2，3，4，6，8，10，分别进行 10 次实验计算平均整体纯度。结果如下表所示：

表 2: sonar 数据集不同 α 的实验结果表

α	平均聚类纯度	平均迭代次数
1.5	55.48%	15.8
2	55.00%	27.2
3	54.86%	6.5
4	54.71%	5.1
6	55.72%	4.3
8	55.19%	3.3
10	55.58%	2.8

从表中可以发现，当柔性参数 α 取值变化时，平均聚类纯度始终维持在 55% 左右，这主要是由于 FCM 算法和 K-means 算法的原理较为相似，对于 sonar 这类存在大量线性不可分的数据集，聚类效果较差。

6.3 k 均值聚类和模糊 C 均值聚类性能对比

对于 sonar 数据集来说，两类算法的效果相差不大，平均聚类纯度均在 55% 左右。

对于 Iris 数据集来说，k 均值聚类的平均聚类纯度为 86.73%，模糊 C 均值聚类的最高平均聚类纯度为 92.13%，说明模糊 C 均值聚类在 Iris 数据集上的表现略优于 k 均值聚类。

两个算法都属于无监督学习的聚类算法，在某些样本分布下具有较快的分类速度和精度，但都需要提前确定分类数量，对于坏值噪声的抗干扰能力较差。并且，两者初始聚类中心以及初始隶属矩阵的选取和构建具有随机性，对聚类结果也有较大的影响。

7 程序代码

k-means 算法代码：

```
1
2           import numpy as np
3           import matplotlib.pyplot as plt
4           import random
5
6
7           # 正常导入数据
8           def load_dataset():
9           data = np.genfromtxt('./数据集/iris.txt', delimiter=',', usecols
                =(0, 1, 2, 3))
10          target = np.genfromtxt('./数据集/iris.txt', delimiter=',',
                usecols=(4), dtype=str)
11          t = np.zeros(len(target))
```

```
12          t[target == 'setosa'] = 1
13          t[target == 'versicolor'] = 2
14          t[target == 'virginica'] = 3
15          return data, t
16
17
18          # 随机初始化k个聚类中心，从样本中随机选取
19          def randChosenCent(data, k):
20          # 样本数
21          m = data.shape[0]
22          # 初始化列表
23          centroids = []
24          # 生成类似于样本索引的列表
25          centroidsIndex = random.sample(range(0, m), k)  # 产生k个[0,60)的
                不同随机数
26          # 根据索引获取样本
27          for j in centroidsIndex:
28          centroids.append(data[j])
29          return centroids
30
31
32          def osdistance(vecA, vecB):  # 两个向量间欧式距离
33          return np.sqrt(sum(np.power(vecA - vecB, 2)))
34
35
36          def kMeans(data, k):
37          # 样本总数
38          m = len(data)
39          # 分配样本到最近的簇：存[簇序号,距离的平方]，m行2列
40          cluster = np.zeros((m, 2))
41
42          # 通过随机产生的样本点初始化聚类中心
43          centroids = np.array(randChosenCent(data, k))
44          # print('最初的中心=', centroids)
45          clusterChanged = True  # 标记每次迭代后聚类中心是否发生变化
46          iterTime = 0  # 标记迭代次数
47          # 所有样本分配结果不再改变，迭代终止
48          while clusterChanged:
49          # step2:分配到最近的聚类中心对应的簇中
50          for i in range(m):
51          # 初始定义距离为无穷大
52          minDist = float('inf')
53          # 初始化索引值
54          minIndex = -1
```

```
55          # 计算每个样本与k个中心点距离
56          for j in range(k):
57          # 计算第i个样本到第j个中心点的距离
58          distJI = osdistance(centroids[j], data[i])
59          # 判断距离是否为最小
60          if distJI < minDist:
61          # 更新获取到最小距离
62          minDist = distJI
63          # 获取对应的簇序号
64          minIndex = j
65          cluster[i, 0] = minIndex
66          cluster[i, 1] = minDist
67          iterTime += 1
68          # 更新聚类中心
69          centroids_pre = centroids.copy()  # 将之前的聚类中心做深拷贝
70          for cent in range(k):
71          cent_sum = np.zeros((1, 4))  # (1,4)维度的向量
72          num = 0  # num 用来计量簇内个数
73          for i in range(m):
74          if (cluster[i, 0] == cent):
75          cent_sum += data[i, :]
76          num += 1
77          centroids[cent, :] = cent_sum / num
78          if ((centroids_pre == centroids).all()):
79          clusterChanged = False
80          # print('迭代次数为', '%d' % iterTime)
81          return cluster, iterTime, centroids
82
83
84          # 计算分类准确率
85          def cal_accuracy(k):
86          accuracy = 0
87          for i in range(k):
88          label_list = []  # label_list 存储第i簇样本的真实标签
89          for j in range(len(cluster)):
90          if (cluster[j][0] == i):
91          label_list.append(t[j])
92          # print(label_list)
93          true_label = max(label_list, key=label_list.count)  # 选取数量最
                大的标签作为其标签
94          # 再次遍历真实样本类别，若真实样本类别=簇类别，accuracy+1
95          for n in range(len(label_list)):
96          if (label_list[n] == true_label):
97          accuracy += 1
```

```
98          accuracy = accuracy / len(data)
99          return accuracy
100
101
102         def draw(data, t):
103         x0 = data[t == 1]
104         x1 = data[t == 2]
105         x2 = data[t == 3]
106         plt.figure(1)
107         plt.scatter(x0[:, 0], x0[:, 1], c='r', marker='o', label='类别一'
                )
108         plt.scatter(x1[:, 0], x1[:, 1], c='g', marker='o', label='类别二'
                )
109         plt.scatter(x2[:, 0], x2[:, 1], c='blue', marker='o', label='类别
                三')
110         plt.xlabel('花瓣长度')
111         plt.ylabel('花瓣宽度')
112         plt.title('花瓣长度和花瓣宽度特征之间的散点图（真实数据）')
113         plt.legend(loc=2)  # 把图例放到左上角
114         plt.rcParams['font.sans-serif'] = ['SimHei']
115         plt.savefig('./iris_kmeans(yuanshi)')
116         plt.show()
117
118
119         def draw_pre(cluster, data, centroids):
120         x0 = []
121         x1 = []
122         x2 = []
123         for i in range(len(cluster)):
124         if cluster[i][0] == 0:
125         x0.append(data[i])
126         elif cluster[i][0] == 1:
127         x1.append(data[i])
128         elif cluster[i][0] == 2:
129         x2.append(data[i])
130         x0 = np.array(x0)
131         x1 = np.array(x1)
132         x2 = np.array(x2)
133         plt.figure(2)
134         plt.scatter(x0[:, 0], x0[:, 1], c='r', marker='o', label='类别一'
                )
135         plt.scatter(x1[:, 0], x1[:, 1], c='g', marker='o', label='类别二'
                )
136         plt.scatter(x2[:, 0], x2[:, 1], c='b', marker='o', label='类别三'
```

```
        )
    plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='x')
    plt.xlabel('花瓣长度')
    plt.ylabel('花瓣宽度')
    plt.title('花瓣长度和花瓣宽度特征之间的散点图（预测数据）')
    plt.legend(loc=2)  # 把图例放到左上角
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.savefig('./iris_kmeans(yuce)')
    plt.show()


    if __name__ == '__main__':
    data, t = load_dataset()
    k = 3
    cluster, iterTime, centroids = kMeans(data, k)
    # 绘制前后对比散点图
    draw(data, t)
    draw_pre(cluster, data, centroids)
    sum_iterTime = 0
    sum_accuracy = 0
    for i in range(10):
    cluster, iterTime, centroids = kMeans(data, k)
    accuracy = cal_accuracy(k)
    sum_iterTime += iterTime
    sum_accuracy += accuracy
    print("平均迭代次数为:", "{}".format(sum_iterTime / 10))
    print("平均分类纯度为:", "{:.2%}".format(sum_accuracy / 10))
```

FCM 算法代码：

```
    import numpy as np

    # 正常导入数据
    def load_dataset():
    data = np.genfromtxt('./数据集/iris.txt', delimiter=',', usecols=(0, 1, 2, 3))
    target = np.genfromtxt('./数据集/iris.txt', delimiter=',', usecols=(4), dtype=str)
    t = np.zeros(len(target))
    t[target == 'setosa'] = 1
    t[target == 'versicolor'] = 2
    t[target == 'virginica'] = 3
    return data, t

```

```
13
14          def osdistance(vecA, vecB):  # 两个向量间欧式距离
15          return np.sqrt(sum(np.power(vecA - vecB, 2)))
16
17          # 初始化U矩阵
18          def initmatU(m, c):
19          mat_u = np.random.uniform(0, 1, (m, c))  # 0,1之间均匀分布初始化
20          # 归一化——每一个样本对所有分类集合隶属度总和为1
21          for i in range(m):
22          addsum = 0
23          for j in range(c):
24          addsum += mat_u[i, j]
25          mat_u[i, :] = mat_u[i, :] / addsum
26          return mat_u
27
28
29          def FCMtrain(data, c, alpha, theta):
30          m = len(data)
31          dim = data.shape[1]  # 样本维度
32          mat_u = initmatU(m, c)
33          # 计算c个聚类中心
34          c_list = np.zeros([c, dim])
35          iterTime = 0  # 标记迭代次数
36          last_cost = 0  # 上一次的损失
37
38          while True:
39          # 计算聚类中心c_list
40          for j in range(c):
41          sum_uij = 0  # 表达式分母
42          sum_uij_x = 0  # 表达式分子
43          for i in range(m):
44          sum_uij += mat_u[i, j] ** alpha
45          sum_uij_x += mat_u[i, j] ** alpha * data[i, :]
46          c_list[j, :] = sum_uij_x / sum_uij
47          # 计算损失函数
48          cost = 0
49          for j in range(c):
50          for i in range(m):
51          vec1 = np.array(data[i, :])  # 第i条样本
52          vec2 = np.array(c_list[j, :])  # 第j个中心
53          dis = osdistance(vec1, vec2)
54          cost += mat_u[i, j] ** alpha * dis ** 2
55          if abs(last_cost - cost) < theta:
56          break
```

```
57          last_cost = cost
58          # 重新计算U
59          for j in range(c):
60          vec1 = np.array(c_list[j, :])  # 第j条样本
61          for i in range(m):
62          vec2 = np.array(data[i, :])  # 第i个中心
63          dis_ij = osdistance(vec1, vec2)
64          sumd_d = 0
65          for k in range(c):
66          vec3 = np.array(c_list[k, :])  # 第k个中心
67          dis_ki = osdistance(vec2, vec3)
68          sumd_d += (dis_ij / dis_ki) ** (2 / (alpha - 1))
69          mat_u[i, j] = 1 / sumd_d
70          # 归一化
71          for i in range(m):
72          addsum = 0
73          for j in range(c):
74          addsum += mat_u[i, j]
75          mat_u[i, :] = mat_u[i, :] / addsum
76          iterTime += 1
77          # print('迭代次数为', '%d' % iterTime)
78          # 对每一条样本进行遍历，隶属度最大的集合类别即为样本预测类别
79          pred = []
80          for i in range(m):
81          t = np.argmax(mat_u[i, :])
82          pred.append(t)
83          return c_list, pred, iterTime
84
85
86          # 计算分类准确率
87          def cal_accuracy(c, pred):
88          accuracy = 0
89          for i in range(c):
90          label_list = []  # label_list 存储第i簇样本的真实标签
91          for j in range(len(pred)):
92          if (pred[j] == i):
93          label_list.append(t[j])
94          true_label = max(label_list, key=label_list.count)  # 选取数量最
                大的标签作为其标签
95          # 再次遍历真实样本类别，若真实样本类别=簇类别，accuracy+1
96          for n in range(len(label_list)):
97          if (label_list[n] == true_label):
98          accuracy += 1
99          accuracy = accuracy / len(data)
```

```
100         return accuracy
101
102
103         if __name__ == '__main__':
104         data, t = load_dataset()
105         c = 3
106         alpha = 6
107         theta = 0.001
108         sum_iterTime = 0
109         sum_accuracy = 0
110         # 绘制图像
111         c_list, pred, iterTime = FCMtrain(data, c, alpha, theta)
112         # print(c_list)
113         for i in range(10):
114         c_list, pred, iterTime = FCMtrain(data, c, alpha, theta)
115         accuracy = cal_accuracy(c, pred)
116         sum_iterTime += iterTime
117         sum_accuracy += accuracy
118         print("平均迭代次数为:", "{}".format(sum_iterTime/10))
119         print("平均分类纯度为:", "{:.2%}".format(sum_accuracy/10))
```