**模式识别 K-means、Fuzzy C-means**

**一、动态聚类算法介绍**

**1.1 介绍**

聚类就是按照某个特定标准(如距离准则)把一个数据集分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。即聚类后同一类的数据尽可能聚集到一起，不同数据尽量分离。这种基于相似度度量的聚类方法也是实际中更常用的方法，其中，根据算法设计的不同又可分为动态聚类法和分级聚类法等。

动态聚类方法是一种普遍采用的方法，它具有以下3个要点：

1. 选定某种距离度量作为样本间的相似度度量。
2. 确定某个评价聚类结果质量的准则函数。

③ 给定某个初始分类，然后用迭代算法找出使准则函数取极值的最好聚类结果。

**1.2 K均值算法（K-means）**

**（1）算法原理**

K均值(K-means)算法是一种很常用的聚类算法，其基本思想是，通过迭代寻找k个聚类的一种划分方案，使得用这k个聚类的均值来代表相应各类样本时所得到的总体误差最小。K均值法在向量量化(例如对音频信号)和图像分隔等领域有广泛的应用，也有时被称作广义Llogd算法(GLA)。

算法原理：

K均值算法的基础是最小误差平方和准则。

将N个样本划分到k个类中，k为一正整数，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：

**-** 是误差平方和聚类准则。

**-** k为聚类个数。

**-** 是划分到类的样本。

**-** 类的质心（均值向量）：

**-**是第聚类中的样本数目。

误差平方和无法用解析的方法最小化，只能用迭代的方法，通过不断调整样本的类别归属来求解。

**（2）K-means算法流程**

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；
2. 按照最小距离法则逐个将样本划分到以聚类中心为代表的k个类中；
3. 计算聚类准则函数，重新计算k个类的聚类中心；
4. 重复2和3到聚类中心无改变或目标函数不减小。

**（3）算法优缺点**

**1、优点：**

1. 简单、快速。
2. 对处理大数据集，该算法是相对可伸缩和高效率的。
3. 当类密集，且类与类之间区别明显（比如球型聚集） 时，聚类效果很好；

**2、缺点：**

1. 结果与初始聚类中心有关；
2. 必须预先给出聚类的类别数k；
3. 对“噪声”和孤立点数据敏感，少量的这些数据对平 均值产生较大的影响；
4. 不适合发现非凸面形状的聚类

**1.2 模糊C均值算法（Fuzzy C-means）**

**（1）模糊分类**

在普通模式识别中，分类就是把样本空间(或样本集) 分成若干个子集。 在模糊模式识别中，用模糊子集代替确定子集, 从而得到模糊的分类结果，即分类结果的模糊化，其中，一个样本以不同的程度属于各个类别，而不再属于某个确定的类别。

模糊分类与确定的分类结果相比，模糊化的分类结果主要有两个显著的优点:

1. 可以反映出分类过程中的不确定性, 有利于用户根据结果进行决策；
2. 模糊化的分类结果比明确的分类结果中包含更多的信息, 有利于进一步决策。

**（2）FCM算法原理**

与K均值相比，模糊C均值聚类最大的特点在于：在C均值算法中，把硬划分变为模糊划分。

设是第个样本属于第类隶属度，利用隶属度定义的聚类准则函数为：

（1）

其中, b>1是一个可以控制聚类结果的模糊程度的常数。 约束条件为一个样本属于各个聚类的隶属度之和为1,即:

（2）

利用拉格朗日乘数法来求解在条件（2）约束下，式（1）的极小值。

令优化的目标函数为：

（3）

分别求L对、和的梯度（或偏导），并置为0，可得必要条件：

（5）

（4）

**（3）FCM算法流程**

1. 设定聚类数目C、参数b和一个适当的小数 ε>0, 通常取1<b≤5。
2. 初始化各个聚类中心，迭代次数s=0。
3. 根据式（5）更新隶属度函数：

④ 根据式（4）更新聚类中心：

⑤ 如果，则停止迭代，输出聚类中心和隶属度函数，否则, 返回步骤（3）。

⑥ 输出：将样本点划分为隶属度最大的那一类。

**二、实验数据集介绍**

**2.1 Iris数据集介绍**

|  |  |  |  |
| --- | --- | --- | --- |
| Iris 数据集 | | | |
| 类别 | 3 | | |
| 维度 | 4 | | |
| 数据长度 | 150 | 50 | Iris-setosa |
| 50 | Iris-versicolor |
| 50 | Iris-setosa |

**2.2 Sonar数据集介绍**

|  |  |  |  |
| --- | --- | --- | --- |
| Sonar 数据集 | | | |
| 类别 | 2 | | |
| 维度 | 60 | | |
| 数据长度 | 208 | 97 | R |
| 111 | M |

**三、实验设置**

对两类数据集，均分别采用K-means、FCM两种方法进行聚类，显示最终的聚类结果。

两类数据集处理方法具有相似性，因此实验设置基本一致，具体如下：

**3.1 K-means**

**1、读取数据信息。**首先从数据集文件（iris.data、sonar.all-data）中读取数据，由于是聚类的缘故，将所有数据均放在训练集中

**2、随机选取K个聚类中心,并初始化聚类名称与聚类均值。**在样本中随机选取K个标号作为初始聚类中心和聚类均值。

**3、以准则函数作为迭代结束依据，进行迭代。**

（1）按最小距离准则划分样本

（2）计算均值，更新聚类中心，计算准则函数值

**4、迭代结束，按照数据标签，统计分类结果标签并计算准确率。**

**5、重复1-4步10次，计算平均准确率，绘图显示准确率、准则函数变化情况、聚类结果。**

**3.2 FCM**

**1、读取数据信息。**首先从数据集文件（iris.data、sonar.all-data）中读取数据，由于是聚类的缘故，将所有数据均放在训练集中

**2、随机选取C个聚类中心,初始化聚类中心(均值向量)、b和终止条件。**在样本中随机选取C个标号作为初始聚类中心和聚类均值。b默认为2

**3、以作为迭代结束依据，进行迭代。**

（1）按照公式计算，更新隶属度函数

（2）按照公式计算，更新聚类中心

**4、迭代结束，根据隶属度函数划分样本。**

**5、划分结束，统计分类结果标签数目并计算准确率。**

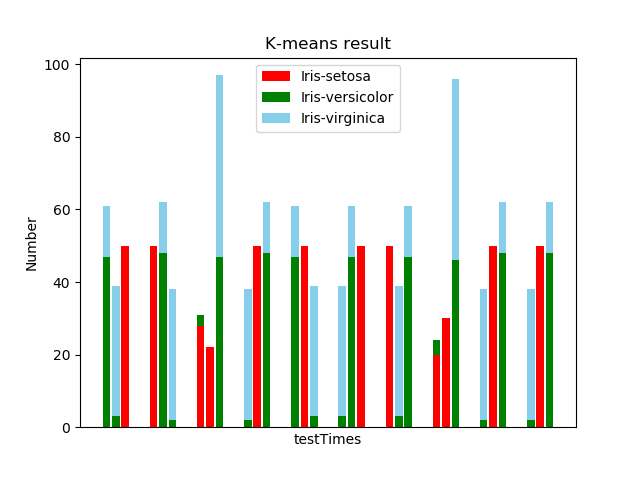
**6、重复1-5步10次，计算平均准确率，绘图显示准确率变化趋势，绘图显示10次聚类结果。**

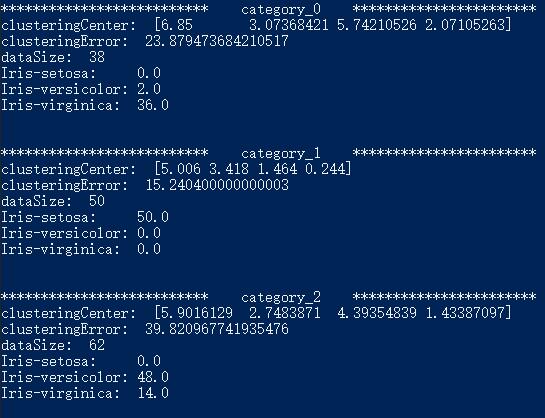
**四、实验结果展示与分析**

**4.1 K-means聚类结果与分析**

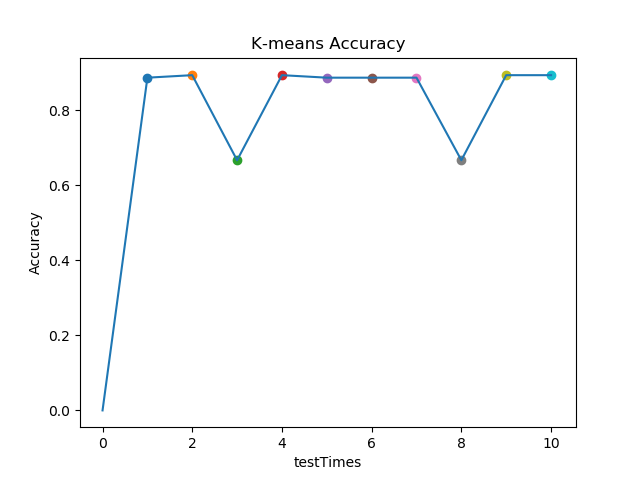
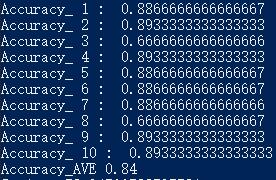
10次聚类结果绘图及最后一次聚类数据展示如下，可以看出因初始聚类中心的选取不同，每次聚类的结果并不相同。对于iris数据集，可以得到不错的聚类结果，但sonar数据集却不能很好分开：

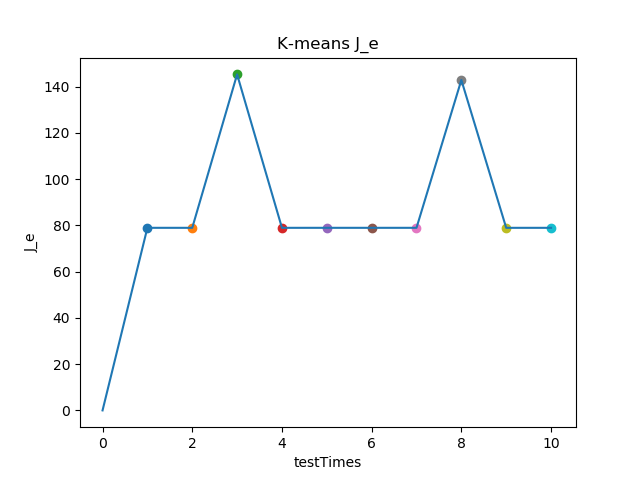
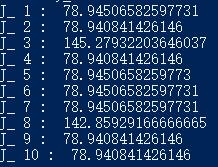
**Iris数据集**10次K均值聚类结果绘图、最后一次聚类数据展示如下：



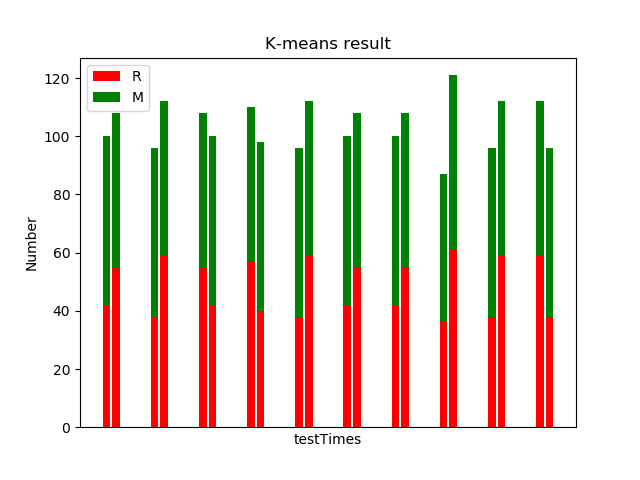


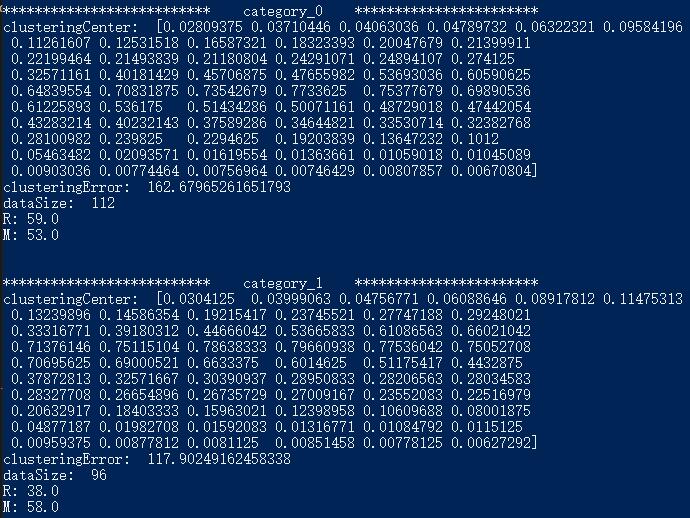
Iris数据集10次K均值聚类准确率、最终准则函数展示如下：



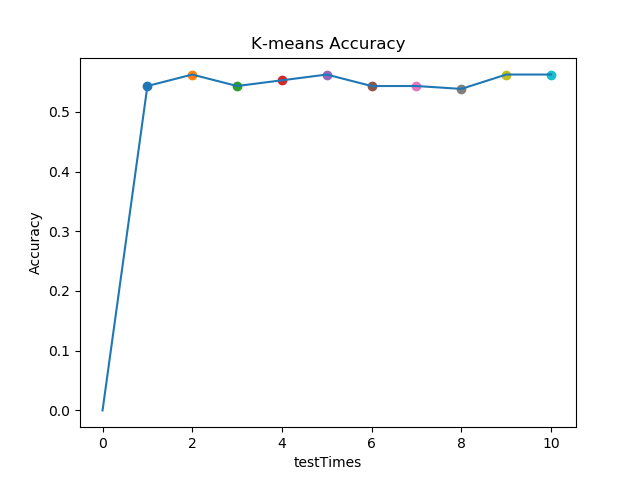
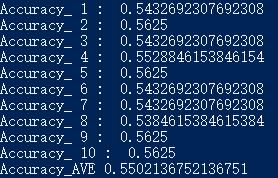


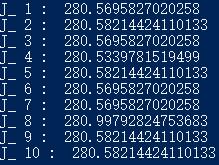
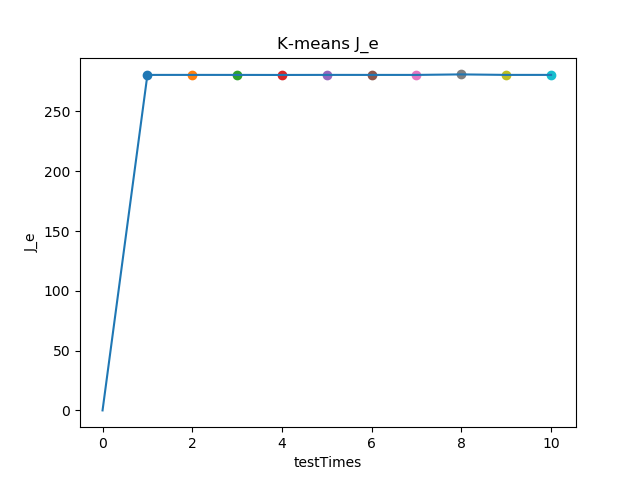
**sonar数据集**10次K均值聚类结果绘图、最后一次聚类数据展示如下：





10次K均值聚类准确率、最终准则函数展示如下：

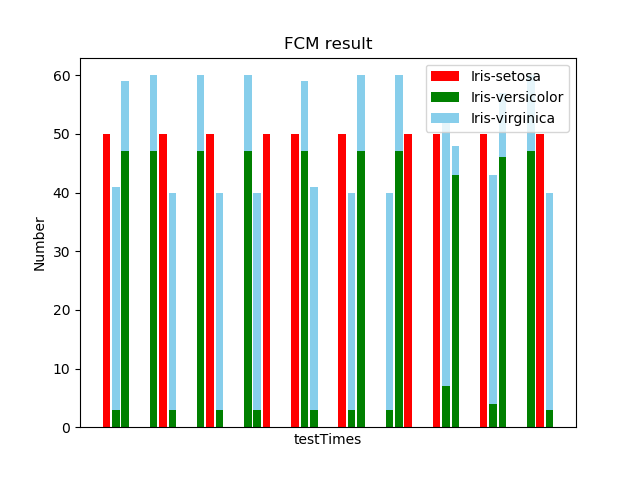


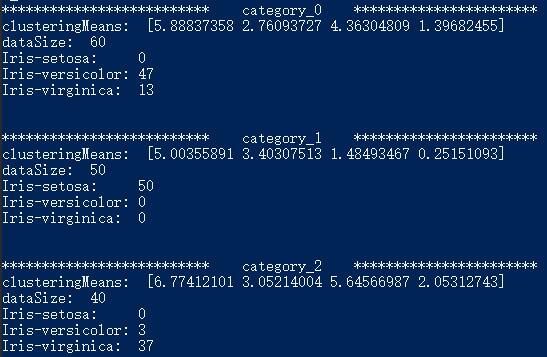
****

**4.2 FCM聚类结果与分析**

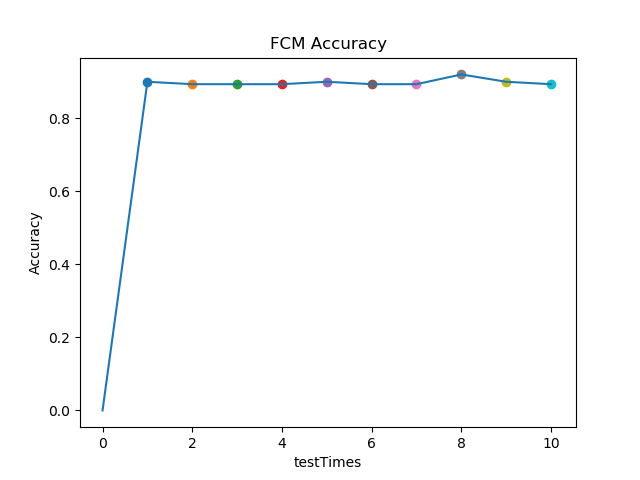
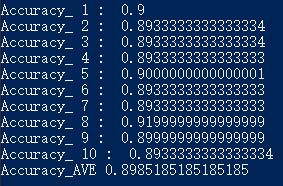
10次模糊C均值聚类结果绘图及最后一次聚类数据展示如下，可以看出因初始聚类中心的选取不同，每次聚类的结果并不相同。相比于K均值聚类，聚类结果和准确率有所改善，但和K均值聚类相似，对于iris数据集，可以得到不错的聚类结果，但sonar数据集却不能很好分开：

**Iris数据集**10次模糊C均值聚类结果绘图、最后一次聚类数据展示如下：

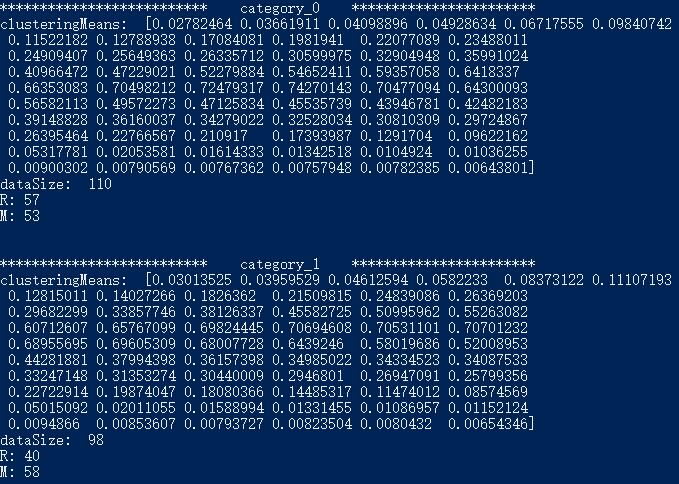
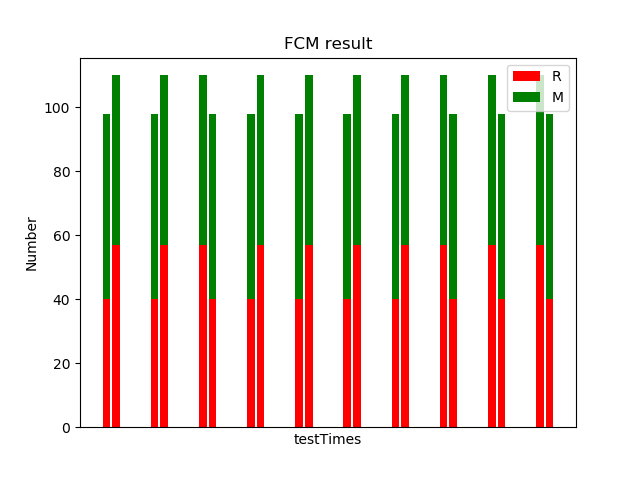


****

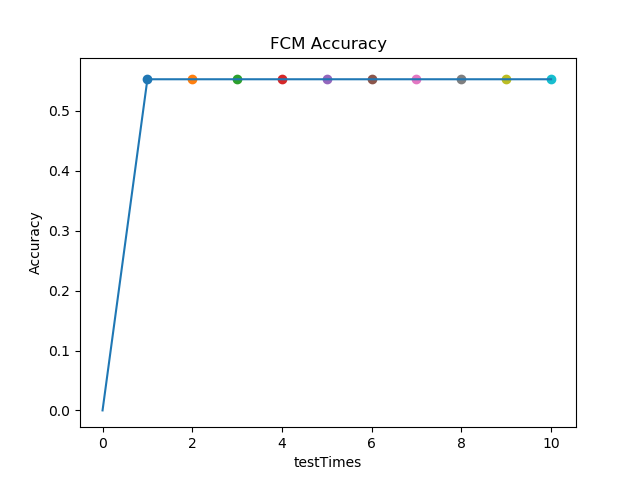
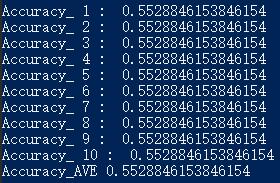
10次模糊C均值聚类准确率展示如下：



**sonar数据集**10次模糊C均值聚类结果绘图、最后一次聚类数据展示如下：



10次模糊C均值聚类准确率展示如下：



**五、Python代码**

**5.1 iris数据集**

import os

import sys

import numpy as np

from numpy import \*

import random

import matplotlib.pyplot as plt

def readData(DATA\_PATH,DATA\_SIZE,DATA\_DIMENSION,test\_rate):

    test\_Labels = random.sample(range(0,DATA\_SIZE-1),int(DATA\_SIZE\*test\_rate))

    test\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(len(test\_Labels))]

    train\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(DATA\_SIZE-len(test\_Labels))]

    #随机提取训练、测试数据

    f = open(DATA\_PATH)

    line = f.readline()

    i = 0

    train\_Label = 0

    test\_Label = 0

    while line:

        temp = line.split(",",DATA\_DIMENSION)

        if i in test\_Labels:

            test\_Data[test\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            test\_Data[test\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            test\_Label = test\_Label+1

        else:

            train\_Data[train\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            train\_Data[train\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            train\_Label = train\_Label+1

        i = i+1

        line = f.readline()

    #将要训练的数据分类保存

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-setosa\n':

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-versicolor\n':

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Num = \_3\_Num+1

    \_1\_Data = mat(np.zeros((\_1\_Num,DATA\_DIMENSION)))

    \_2\_Data = mat(np.zeros((\_2\_Num,DATA\_DIMENSION)))

    \_3\_Data = mat(np.zeros((\_3\_Num,DATA\_DIMENSION)))

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-setosa\n':

            \_1\_Data[\_1\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-versicolor\n':

            \_2\_Data[\_2\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Data[\_3\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_3\_Num = \_3\_Num+1

    f.close()

    return train\_Data,test\_Data,\_1\_Data.T,\_2\_Data.T,\_3\_Data.T

#计算欧氏距离

def calcu\_Eucli\_Dis(x,y):

    dis = np.sqrt(np.sum(np.square(np.array(x)-np.array(y))))

    return dis

#计算样本到均值向量的距离平方和

def calcu\_error(samples, mean):

    sum=0

    for i in range(len(samples)):

        e = np.sum(np.square(np.array(samples[i])-np.array(mean)))

        sum += e

    return sum

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*K-means\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#K-means类结构

class Data\_K\_means:

    def \_\_init\_\_(self):

        self.name = ''

        self.clusteringCenter = []  #聚类中心

        self.clusteringData = []    #包含的样本数据

        self.clusteringMeans = 0    #均值

        self.clusteringError = 0    #每一类中的最小误差平方和

        self.Iris\_setosa = 0

        self.Iris\_versicolor = 0

        self.Iris\_virginica = 0

    def print(self):

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   ",self.name,"   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        print("clusteringCenter: ",self.clusteringCenter)

        print("clusteringError: "  ,self.clusteringError)

        # print("clusteringData: \n"  ,np.array(self.clusteringData))

        print("dataSize: "  ,len(self.clusteringData))

#按最小距离准则划分样本到N类

def divide\_Sample(train\_Data,category,dataDimension,K):

    for i in range(K):

        category[i].clusteringData = []

    for i in range(len(train\_Data)):

        min\_dis = 1024

        min\_dis\_label = -1

        for j in range(K):

            dis = calcu\_Eucli\_Dis(train\_Data[i][:dataDimension],category[j].clusteringCenter)

            if dis<min\_dis:

                min\_dis = dis

                min\_dis\_label = j

        # print("min\_dis:",min\_dis,"label:",min\_dis\_label)

        category[min\_dis\_label].clusteringData.append(train\_Data[i])

#更新参数：计算均值与准则函数，更新聚类中心,返回准则函数

def update\_Cluster\_Center(category,dataDimension,K):

    #计算每一类的均值、误差J\_i,计算出准则函数

    J = 0

    for i in range(K):

        dataArray = np.array(np.array(category[i].clusteringData)[:,0:dataDimension],dtype=np.float64)

        #计算均值

        category[i].clusteringMeans = mean(dataArray,0)

        #更新聚类中心

        category[i].clusteringCenter = category[i].clusteringMeans

        #计算每一类内距离差

        error = calcu\_error(dataArray,category[i].clusteringMeans)

        category[i].clusteringError = error

        J += error

    return J

def k\_Means(dataPath,dataSize,dataDimension,K,testRate=0):

    #提取数据

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("train\_Data SIZE:",len(train\_Data))

    #随机选取K个聚类中心,并初始化聚类名称与聚类均值

    clustering\_center\_label = random.sample(range(0,len(train\_Data)-1),K)

    print("clustering\_center\_label:",clustering\_center\_label)

    category = [Data\_K\_means() for i in range(K)]

    for i in range(K):

        category[i].clusteringCenter = train\_Data[clustering\_center\_label[i]][:dataDimension]

        category[i].clusteringMeans = category[i].clusteringCenter

        category[i].name = "category\_"+str(i)

        # category[i].print()

    J = 1024.0

    J\_last = -1024.0

    repeat\_time = 0

    #开始迭代

    while J!=J\_last:    #以准则函数作为迭代结束依据

        J\_last = J

        repeat\_time += 1

        #按最小距离准则划分样本

        divide\_Sample(train\_Data,category,dataDimension,K)

        #计算均值，更新聚类中心，计算准则函数值

        J = update\_Cluster\_Center(category,dataDimension,K)

        print("---------------------------------------------------------------------------->Num: ",repeat\_time)

        # for i in range(K):

        #     category[i].print()

        print("J = ",J)

        print("J\_last = ",J\_last,"\n\n")

    #迭代结束，统计分类结果标签数目 并 计算准确率

    accuracy = 0

    for k in range(K):

        \_1\_Num = \_2\_Num = \_3\_Num = 0.0

        for i in range(len(category[k].clusteringData)):

            if category[k].clusteringData[i][dataDimension] == 'Iris-setosa\n':

                \_1\_Num += 1

            if category[k].clusteringData[i][dataDimension] == 'Iris-versicolor\n':

                \_2\_Num += 1

            if category[k].clusteringData[i][dataDimension] == 'Iris-virginica\n':

                \_3\_Num += 1

        category[k].print()

        print("Iris-setosa:    ",\_1\_Num)

        print("Iris-versicolor:",\_2\_Num)

        print("Iris-virginica: ",\_3\_Num,"\n\n")

        category[k].Iris\_setosa = \_1\_Num

        category[k].Iris\_versicolor = \_2\_Num

        category[k].Iris\_virginica = \_3\_Num

        if \_1\_Num>=\_2\_Num and \_1\_Num>=\_3\_Num:

            accuracy += \_1\_Num/len(category[k].clusteringData) \* len(category[k].clusteringData)/len(train\_Data)

        elif \_2\_Num>=\_1\_Num and \_2\_Num>=\_3\_Num:

            accuracy += \_2\_Num/len(category[k].clusteringData) \* len(category[k].clusteringData)/len(train\_Data)

        else:

            accuracy += \_3\_Num/len(category[k].clusteringData) \* len(category[k].clusteringData)/len(train\_Data)

    return category,J,accuracy

def test\_K\_Means():

    J = [0]

    accuracy = [0]

    testTimes = 10

    #绘制聚类结果

    plt.figure()

    for i in range(testTimes):

        category,J\_temp,accuracy\_temp = k\_Means("iris.data",150,4,K=3)    #进行K均值聚类,返回聚类结果、误差平方和、精准度

        J.append(J\_temp)

        accuracy.append(accuracy\_temp)

        for k in range(3):

            p1 = plt.bar(x=k+(i\*5), height=category[k].Iris\_setosa, label='Iris-setosa', color="red")

            p2 = plt.bar(x=k+(i\*5), height=category[k].Iris\_versicolor, label='Iris-versicolor', color="green",bottom=category[k].Iris\_setosa)

            p3 = plt.bar(x=k+(i\*5), height=category[k].Iris\_virginica, label='Iris-virginica', color="skyblue",bottom=category[k].Iris\_setosa+category[k].Iris\_versicolor)

        plt.legend((p1[0], p2[0], p3[0]), ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'))

    plt.ylabel('Number')

    plt.xlabel("testTimes")

    plt.xticks([])

    plt.title('K-means result')

    plt.show(block = False)

    #绘制聚类精确度

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,accuracy[i])

        print("Accuracy\_",i,": ",accuracy[i])

    print("Accuracy\_AVE",mean(accuracy[1:testTimes]))

    plt.title("K-means Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("Accuracy")

    plt.plot(accuracy)

    plt.show(block = False)

    #绘制聚类最终准则函数值（#最小误差平方和）

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,J[i])

        print("J\_",i,": ",J[i])

    plt.title("K-means J\_e")

    plt.xlabel("testTimes")

    plt.ylabel("J\_e")

    plt.plot(J)

    plt.show()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FCM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Data\_FCM:

    def \_\_init\_\_(self):

        self.name = ''

        self.Data = []          #最终包含的样本数据

        self.Membership = []    #隶属度

        self.Means = 0          #均值/聚类中心

        self.Iris\_setosa = 0

        self.Iris\_versicolor = 0

        self.Iris\_virginica = 0

    def print(self):

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   ",self.name,"   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        print("clusteringMeans: ",self.Means)

        # print("clusteringData: \n"  ,np.array(self.clusteringData))

        print("dataSize: "  ,len(self.Data))

#计算 更新隶属度函数

def update\_Membership(train\_Data,category,dataDimension,C,b):

    trainDataArray = np.array(np.array(train\_Data)[:,0:dataDimension],dtype=np.float64)

    for j in range(C):  #u\_j(x\_i)

        category[j].Membership = []     #清空隶属度向量

        for i in range(len(train\_Data)):

            #计算分母、分子

            denom = 0.0

            for l in range(C):

                denom += pow((1/np.square(np.linalg.norm(np.array(trainDataArray[i])-np.array(category[l].Means)))),1/(b-1))

            numer = pow((1/np.square(np.linalg.norm(np.array(trainDataArray[i])-np.array(category[j].Means)))),1/(b-1))

            #更新隶属度

            if numer == inf:    #解决自己与自己隶属度计算出错问题

                category[j].Membership.append(1.0)

            else:

                category[j].Membership.append(numer/denom)

        # print(j,np.array(category[j].Membership))

#计算 更新聚类中心

def update\_Cluster\_Center\_FCM(train\_Data,category,dataDimension,C,b):

    trainDataArray = np.array(np.array(train\_Data)[:,0:dataDimension],dtype=np.float64)

    for j in range(C):

        #计算分母、分子

        denom = 0.0

        numer = 0.0

        for i in range(len(train\_Data)):

            denom += pow(category[j].Membership[i],b)

            numer += pow(category[j].Membership[i],b) \* trainDataArray[i]

        # print(j,":",numer,denom,numer/denom)

        category[j].Means = numer/denom

#根据隶属度函数划分样本

def divide\_Sample\_FCM(train\_Data,category,C):

    for i in range(len(train\_Data)):

        membership\_max = -1

        label = -1

        for j in range(C):

            if category[j].Membership[i] > membership\_max:

                membership\_max = category[j].Membership[i]

                label = j

        category[label].Data.append(train\_Data[i])

def FCM(dataPath,dataSize,dataDimension,C,b=2,end\_condi=0.001,testRate=0):

    #提取数据

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("train\_Data SIZE:",len(train\_Data))

    #随机选取C个聚类中心,初始化聚类中心(均值向量)

    clustering\_center\_label = random.sample(range(0,len(train\_Data)-1),C)

    print("clustering\_center\_label:",clustering\_center\_label)

    category = [Data\_FCM() for i in range(C)]

    for i in range(C):

        category[i].Means = train\_Data[clustering\_center\_label[i]][:dataDimension]

        category[i].name = "category\_"+str(i)

        category[i].print()

    m = [1024 for i in range(dataDimension)]

    m\_last = [-1024 for i in range(dataDimension)]

    repeat\_time = 0

    #开始迭代

    while calcu\_Eucli\_Dis(m,m\_last) > end\_condi:

        repeat\_time += 1

        m\_last = m

        #计算，更新隶属度

        update\_Membership(train\_Data,category,dataDimension,C,b)

        #计算均值，更新聚类中心

        update\_Cluster\_Center\_FCM(train\_Data,category,dataDimension,C,b)

        m = category[0].Means

        print("---------------------------------------------------------------------------->Num: ",repeat\_time)

        print("m=",m)

        print("m\_last = ",m\_last)

    #迭代结束，划分样本

    divide\_Sample\_FCM(train\_Data,category,C)

    #划分结束，统计分类结果标签数目 并 计算准确率

    accuracy = 0

    for k in range(C):

        \_1\_Num = \_2\_Num = \_3\_Num = 0

        for i in range(len(category[k].Data)):

            if category[k].Data[i][dataDimension] == 'Iris-setosa\n':

                \_1\_Num += 1

            if category[k].Data[i][dataDimension] == 'Iris-versicolor\n':

                \_2\_Num += 1

            if category[k].Data[i][dataDimension] == 'Iris-virginica\n':

                \_3\_Num += 1

        category[k].print()

        print("Iris-setosa:    ",\_1\_Num)

        print("Iris-versicolor:",\_2\_Num)

        print("Iris-virginica: ",\_3\_Num,"\n\n")

        category[k].Iris\_setosa = \_1\_Num

        category[k].Iris\_versicolor = \_2\_Num

        category[k].Iris\_virginica = \_3\_Num

        if \_1\_Num>=\_2\_Num and \_1\_Num>=\_3\_Num:

            accuracy += \_1\_Num/len(category[k].Data) \* len(category[k].Data)/len(train\_Data)

        elif \_2\_Num>=\_1\_Num and \_2\_Num>=\_3\_Num:

            accuracy += \_2\_Num/len(category[k].Data) \* len(category[k].Data)/len(train\_Data)

        else:

            accuracy += \_3\_Num/len(category[k].Data) \* len(category[k].Data)/len(train\_Data)

    return category,accuracy

def test\_FCM():

    accuracy = [0]

    testTimes = 10

    #绘制聚类结果

    plt.figure()

    for i in range(testTimes):

        category,accuracy\_temp = FCM("iris.data",150,4,C=3,b=2,end\_condi=0.001)    #进行K均值聚类,返回聚类结果、误差平方和、精准度

        accuracy.append(accuracy\_temp)

        for k in range(3):

            p1 = plt.bar(x=k+(i\*5), height=category[k].Iris\_setosa, label='Iris-setosa', color="red")

            p2 = plt.bar(x=k+(i\*5), height=category[k].Iris\_versicolor, label='Iris-versicolor', color="green",bottom=category[k].Iris\_setosa)

            p3 = plt.bar(x=k+(i\*5), height=category[k].Iris\_virginica, label='Iris-virginica', color="skyblue",bottom=category[k].Iris\_setosa+category[k].Iris\_versicolor)

        plt.legend((p1[0], p2[0], p3[0]), ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'))

    plt.ylabel('Number')

    plt.xlabel("testTimes")

    plt.xticks([])

    plt.title('FCM result')

    plt.show(block = False)

    #绘制聚类精确度

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,accuracy[i])

        print("Accuracy\_",i,": ",accuracy[i])

    print("Accuracy\_AVE",mean(accuracy[1:testTimes]))

    plt.title("FCM Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("Accuracy")

    plt.plot(accuracy)

    plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

    # test\_K\_Means()

    test\_FCM()

**5.2 sonar数据集（与iris相差无几）**

import os

import sys

import numpy as np

from numpy import \*

import random

import matplotlib.pyplot as plt

def readData(DATA\_PATH,DATA\_SIZE,DATA\_DIMENSION,test\_rate):

    test\_Labels = random.sample(range(0,DATA\_SIZE-1),int(DATA\_SIZE\*test\_rate))

    test\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(len(test\_Labels))]

    train\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(DATA\_SIZE-len(test\_Labels))]

    #随机提取训练、测试数据

    f = open(DATA\_PATH)

    line = f.readline()

    i = 0

    train\_Label = 0

    test\_Label = 0

    while line:

        temp = line.split(",",DATA\_DIMENSION)

        if i in test\_Labels:

            test\_Data[test\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            test\_Data[test\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            test\_Label = test\_Label+1

        else:

            train\_Data[train\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            train\_Data[train\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            train\_Label = train\_Label+1

        i = i+1

        line = f.readline()

    #将要训练的数据分类保存

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'R\n':

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'M\n':

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Num = \_3\_Num+1

    \_1\_Data = mat(np.zeros((\_1\_Num,DATA\_DIMENSION)))

    \_2\_Data = mat(np.zeros((\_2\_Num,DATA\_DIMENSION)))

    \_3\_Data = mat(np.zeros((\_3\_Num,DATA\_DIMENSION)))

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'R\n':

            \_1\_Data[\_1\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'M\n':

            \_2\_Data[\_2\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Data[\_3\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_3\_Num = \_3\_Num+1

    f.close()

    return train\_Data,test\_Data,\_1\_Data.T,\_2\_Data.T,\_3\_Data.T

#计算欧氏距离

def calcu\_Eucli\_Dis(x,y):

    dis = np.sqrt(np.sum(np.square(np.array(x)-np.array(y))))

    return dis

#计算样本到均值向量的距离平方和

def calcu\_error(samples, mean):

    sum=0

    for i in range(len(samples)):

        e = np.sum(np.square(np.array(samples[i])-np.array(mean)))

        sum += e

    return sum

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*K-means\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#K-means类结构

class Data\_K\_means:

    def \_\_init\_\_(self):

        self.name = ''

        self.clusteringCenter = []  #聚类中心

        self.clusteringData = []    #包含的样本数据

        self.clusteringMeans = 0    #均值

        self.clusteringError = 0    #每一类中的最小误差平方和

        self.R = 0

        self.M = 0

    def print(self):

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   ",self.name,"   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        print("clusteringCenter: ",self.clusteringCenter)

        print("clusteringError: "  ,self.clusteringError)

        # print("clusteringData: \n"  ,np.array(self.clusteringData))

        print("dataSize: "  ,len(self.clusteringData))

#按最小距离准则划分样本到N类

def divide\_Sample(train\_Data,category,dataDimension,K):

    for i in range(K):

        category[i].clusteringData = []

    for i in range(len(train\_Data)):

        min\_dis = 1024

        min\_dis\_label = -1

        for j in range(K):

            dis = calcu\_Eucli\_Dis(train\_Data[i][:dataDimension],category[j].clusteringCenter)

            if dis<min\_dis:

                min\_dis = dis

                min\_dis\_label = j

        # print("min\_dis:",min\_dis,"label:",min\_dis\_label)

        category[min\_dis\_label].clusteringData.append(train\_Data[i])

#更新参数：计算均值与准则函数，更新聚类中心,返回准则函数

def update\_Cluster\_Center(category,dataDimension,K):

    #计算每一类的均值、误差J\_i,计算出准则函数

    J = 0

    for i in range(K):

        dataArray = np.array(np.array(category[i].clusteringData)[:,0:dataDimension],dtype=np.float64)

        #计算均值

        category[i].clusteringMeans = mean(dataArray,0)

        #更新聚类中心

        category[i].clusteringCenter = category[i].clusteringMeans

        #计算每一类内距离差

        error = calcu\_error(dataArray,category[i].clusteringMeans)

        category[i].clusteringError = error

        J += error

    return J

def k\_Means(dataPath,dataSize,dataDimension,K,testRate=0):

    #提取数据

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("train\_Data SIZE:",len(train\_Data))

    #随机选取K个聚类中心,并初始化聚类名称与聚类均值

    clustering\_center\_label = random.sample(range(0,len(train\_Data)-1),K)

    print("clustering\_center\_label:",clustering\_center\_label)

    category = [Data\_K\_means() for i in range(K)]

    for i in range(K):

        category[i].clusteringCenter = train\_Data[clustering\_center\_label[i]][:dataDimension]

        category[i].clusteringMeans = category[i].clusteringCenter

        category[i].name = "category\_"+str(i)

        # category[i].print()

    J = 1024.0

    J\_last = -1024.0

    repeat\_time = 0

    #开始迭代

    while J!=J\_last:    #以准则函数作为迭代结束依据

        J\_last = J

        repeat\_time += 1

        #按最小距离准则划分样本

        divide\_Sample(train\_Data,category,dataDimension,K)

        #计算均值，更新聚类中心，计算准则函数值

        J = update\_Cluster\_Center(category,dataDimension,K)

        print("---------------------------------------------------------------------------->Num: ",repeat\_time)

        # for i in range(K):

        #     category[i].print()

        print("J = ",J)

        print("J\_last = ",J\_last,"\n\n")

    #迭代结束，统计分类结果标签数目 并 计算准确率

    accuracy = 0

    for k in range(K):

        \_1\_Num = \_2\_Num = 0.0

        for i in range(len(category[k].clusteringData)):

            if category[k].clusteringData[i][dataDimension] == 'R\n':

                \_1\_Num += 1

            if category[k].clusteringData[i][dataDimension] == 'M\n':

                \_2\_Num += 1

        category[k].print()

        print("R:",\_1\_Num)

        print("M:",\_2\_Num,"\n\n")

        category[k].R = \_1\_Num

        category[k].M = \_2\_Num

        if \_1\_Num>=\_2\_Num:

            accuracy += \_1\_Num/len(category[k].clusteringData) \* len(category[k].clusteringData)/len(train\_Data)

        else:

            accuracy += \_2\_Num/len(category[k].clusteringData) \* len(category[k].clusteringData)/len(train\_Data)

    return category,J,accuracy

def test\_K\_Means():

    J = [0]

    accuracy = [0]

    testTimes = 10

    #绘制聚类结果

    plt.figure()

    for i in range(testTimes):

        category,J\_temp,accuracy\_temp = k\_Means("sonar.all-data",208,60,K=2)    #进行K均值聚类,返回聚类结果、误差平方和、精准度

        J.append(J\_temp)

        accuracy.append(accuracy\_temp)

        for k in range(2):

            p1 = plt.bar(x=k+(i\*5), height=category[k].R, label='R', color="red")

            p2 = plt.bar(x=k+(i\*5), height=category[k].M, label='M', color="green",bottom=category[k].R)

        plt.legend((p1[0], p2[0]), ('R', 'M'))

    plt.ylabel('Number')

    plt.xlabel("testTimes")

    plt.xticks([])

    plt.title('K-means result')

    plt.show(block = False)

    #绘制聚类精确度

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,accuracy[i])

        print("Accuracy\_",i,": ",accuracy[i])

    print("Accuracy\_AVE",mean(accuracy[1:testTimes]))

    plt.title("K-means Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("Accuracy")

    plt.plot(accuracy)

    plt.show(block = False)

    #绘制聚类最终准则函数值（#最小误差平方和）

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,J[i])

        print("J\_",i,": ",J[i])

    plt.title("K-means J\_e")

    plt.xlabel("testTimes")

    plt.ylabel("J\_e")

    plt.plot(J)

    plt.show()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FCM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class Data\_FCM:

    def \_\_init\_\_(self):

        self.name = ''

        self.Data = []          #最终包含的样本数据

        self.Membership = []    #隶属度

        self.Means = 0          #均值/聚类中心

        self.R = 0

        self.M = 0

        self.Iris\_virginica = 0

    def print(self):

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   ",self.name,"   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        print("clusteringMeans: ",self.Means)

        # print("clusteringData: \n"  ,np.array(self.clusteringData))

        print("dataSize: "  ,len(self.Data))

#计算 更新隶属度函数

def update\_Membership(train\_Data,category,dataDimension,C,b):

    trainDataArray = np.array(np.array(train\_Data)[:,0:dataDimension],dtype=np.float64)

    for j in range(C):  #u\_j(x\_i)

        category[j].Membership = []     #清空隶属度向量

        for i in range(len(train\_Data)):

            #计算分母、分子

            denom = 0.0

            for l in range(C):

                denom += pow((1/np.square(np.linalg.norm(np.array(trainDataArray[i])-np.array(category[l].Means)))),1/(b-1))

            numer = pow((1/np.square(np.linalg.norm(np.array(trainDataArray[i])-np.array(category[j].Means)))),1/(b-1))

            #更新隶属度

            if numer == inf:    #解决自己与自己隶属度计算出错问题

                category[j].Membership.append(1.0)

            else:

                category[j].Membership.append(numer/denom)

        # print(j,np.array(category[j].Membership))

#计算 更新聚类中心

def update\_Cluster\_Center\_FCM(train\_Data,category,dataDimension,C,b):

    trainDataArray = np.array(np.array(train\_Data)[:,0:dataDimension],dtype=np.float64)

    for j in range(C):

        #计算分母、分子

        denom = 0.0

        numer = 0.0

        for i in range(len(train\_Data)):

            denom += pow(category[j].Membership[i],b)

            numer += pow(category[j].Membership[i],b) \* trainDataArray[i]

        # print(j,":",numer,denom,numer/denom)

        category[j].Means = numer/denom

#根据隶属度函数划分样本

def divide\_Sample\_FCM(train\_Data,category,C):

    for i in range(len(train\_Data)):

        membership\_max = -1

        label = -1

        for j in range(C):

            if category[j].Membership[i] > membership\_max:

                membership\_max = category[j].Membership[i]

                label = j

        category[label].Data.append(train\_Data[i])

def FCM(dataPath,dataSize,dataDimension,C,b=2,end\_condi=0.001,testRate=0):

    #提取数据

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("train\_Data SIZE:",len(train\_Data))

    #随机选取C个聚类中心,初始化聚类中心(均值向量)

    clustering\_center\_label = random.sample(range(0,len(train\_Data)-1),C)

    print("clustering\_center\_label:",clustering\_center\_label)

    category = [Data\_FCM() for i in range(C)]

    for i in range(C):

        category[i].Means = train\_Data[clustering\_center\_label[i]][:dataDimension]

        category[i].name = "category\_"+str(i)

        category[i].print()

    m = [1024 for i in range(dataDimension)]

    m\_last = [-1024 for i in range(dataDimension)]

    repeat\_time = 0

    #开始迭代

    while calcu\_Eucli\_Dis(m,m\_last) > end\_condi:

        repeat\_time += 1

        m\_last = m

        #计算，更新隶属度

        update\_Membership(train\_Data,category,dataDimension,C,b)

        #计算均值，更新聚类中心

        update\_Cluster\_Center\_FCM(train\_Data,category,dataDimension,C,b)

        m = category[0].Means

        print("---------------------------------------------------------------------------->Num: ",repeat\_time)

        print("m=",m)

        print("m\_last = ",m\_last)

    #迭代结束，划分样本

    divide\_Sample\_FCM(train\_Data,category,C)

    #划分结束，统计分类结果标签数目 并 计算准确率

    accuracy = 0

    for k in range(C):

        \_1\_Num = \_2\_Num = 0

        for i in range(len(category[k].Data)):

            if category[k].Data[i][dataDimension] == 'R\n':

                \_1\_Num += 1

            if category[k].Data[i][dataDimension] == 'M\n':

                \_2\_Num += 1

        category[k].print()

        print("R:",\_1\_Num)

        print("M:",\_2\_Num,"\n\n")

        category[k].R = \_1\_Num

        category[k].M = \_2\_Num

        if \_1\_Num>=\_2\_Num:

            accuracy += \_1\_Num/len(category[k].Data) \* len(category[k].Data)/len(train\_Data)

        else:

            accuracy += \_2\_Num/len(category[k].Data) \* len(category[k].Data)/len(train\_Data)

    return category,accuracy

def test\_FCM():

    accuracy = [0]

    testTimes = 10

    #绘制聚类结果

    plt.figure()

    for i in range(testTimes):

        category,accuracy\_temp = FCM("sonar.all-data",208,60,C=2,b=2,end\_condi=0.001)    #进行K均值聚类,返回聚类结果、误差平方和、精准度

        accuracy.append(accuracy\_temp)

        for k in range(2):

            p1 = plt.bar(x=k+(i\*5), height=category[k].R, label='R', color="red")

            p2 = plt.bar(x=k+(i\*5), height=category[k].M, label='M', color="green",bottom=category[k].R)

        plt.legend((p1[0], p2[0]), ('R', 'M'))

    plt.ylabel('Number')

    plt.xlabel("testTimes")

    plt.xticks([])

    plt.title('FCM result')

    plt.show(block = False)

    #绘制聚类精确度

    plt.figure()

    for i in range(1,testTimes+1):

        plt.scatter(i,accuracy[i])

        print("Accuracy\_",i,": ",accuracy[i])

    print("Accuracy\_AVE",mean(accuracy[1:testTimes]))

    plt.title("FCM Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("Accuracy")

    plt.plot(accuracy)

    plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

    # test\_K\_Means()

    test\_FCM()