西安电子科技大学

模式识别 课程实验报告

人工智能学院 1920032 班

姓名 付凯文 学号 19170100004

实验日期 2021 年 11 月 04 日

成绩:
实验报告内容基本要求及参考格式 一、实验目的 二、实验基本原理及步骤 三、实验仿真结果与分析 四、实验中遇到的问题及解决方法（至少 3 个，每人至少写 1 个，写清楚谁的问题和解决方法）

-

实验二　K 均值聚类和模糊 C 聚类方法的比较

一、　实验目的

1、学习并掌握 K-means 算法
2、学习并掌握 FCM 算法
3、分别利用 K-means 算法和 FCM 算法实现对 iris 数据集的聚类
4、比较 K-means 算法和 FCM 算法的异同

二、　实验基本原理及步骤

实验基本原理：

Kmeans 算法基本原理：

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。

（1）选取 K 个初始聚类中心,$z_1(1)$，$z_2(1)$，……，$z_k(1)$，其中括号内的序号为寻找聚类中心的迭代运算的次序号。聚类中心的向量值可任意设定，例如，可选开始的 K 个模式样本的向量值作为初始聚类中心。

（2）根据最小距离标准将要分类的模式样本 X={x}分配给 K 个簇中心中的某一个$z_j(1)$,则 x 与各聚类中心的最小距离:

$$D_i(t) = min||x - z_j||, j = 1,2,\ldots\ldots,K$$

则 x∈ $S_j(t)$，其中 t 表示迭代次数，S_j表示第 j 个聚类簇，因此其聚类中心是z_j。

（3）计算各个聚类中心的新的向量值，$z_j(t+1)$,j=1,2,……，K，并计算各聚类簇中样本数据的均值向量

$$z_j(t+1) = \frac{1}{m_j}\sum_{x\in S_j(t)} x, j = 1,2,\ldots\ldots,K$$

其中，m_j为第 j 个聚类簇S_j的样本数目。最终的聚类准则函数为：

$$J_j = \sum_{x\in S_j(t)} ||x - z_j(t+1)||^2, j = 1,2,\ldots\ldots,K$$

(4)若$z_j(t+1) \neq z_j(t)$，$j = 1,2,\ldots\ldots,K$，则返回第二步，逐个重新分类模式样本，重复迭代操作；若$z_j(t+1) = z_j(t)$，$j = 1,2,\ldots\ldots,K$，则算法收敛，计算结束。

FCM 算法基本原理：

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。

下面介绍隶属度函数的概念。隶属度函数是描述样本 x 隶属于集合 A 的程度的函数，记做$\mu_A(x)$，其自变量范围是所有样本点，值域范围为[0，1]，即$0 \leq \mu_A(x) \leq 1$时，样本 x

完全隶属于集合 A,这种情况相当于硬聚类集合概念中的 x∈A。我们用定义在样本空间 X={x}上的隶属度函数表示一个模糊集合 A，或者叫作定义在论域 X={x}上的一个模糊子集。对于有限个对象$x_1, x_2, \ldots\ldots, x_m$模糊集合可以表示为：

$$A = \{(\mu_A(x_i), | x_i \in X)\}$$

利用模糊集的概念，一个样本隶属于哪一个聚类就不再是硬性的。该样本属于每个聚类的隶属度是区间[0，1]中的值。

FCM 的目标函数是把 m 个样本x_i，i=1，2，……,m 分为 c 个模糊集合，并给出每组的聚类中心，使得代价函数的值为最小。这里使用非相似性指标作为代价函数。FCM 加上归一化约束以后，样本数据属于所有类的隶属度的和总应等于 1:

$$\sum_{j=1}^{c} u_{ij} = 1, \qquad \forall i = 1, \ldots\ldots, m$$

FCM 的目标函数为：

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

这里u_i介于 0，1 之间；z_j为模糊集合 j 的聚类中心;为第 i 个数据与第 j 个聚类中心之间的欧氏距离，这里作为一个加权指数;U 为隶属矩阵。构建一个新的目标函数，也就是上式达到最小值的必要条件如下:

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

$$= \sum_{j=1}^{c} \sum_{i=1}^{m} u_{ij}^{a} d_{ij}^2 + \sum_{i=1}^{m} \lambda_i \left(\sum_{j=1}^{c} u_{ij} - 1 \right)$$

其中，α 为柔性参数，由上述两个条件可以看出，FCM 算法是一种简单的迭代算法。实践采用 FCM 确定聚类中心 z，和隶属矩阵 U 的步骤如下:

(1)对隶属矩阵 U 使思(0;1)之间进行均匀分布的初始化，使其如下约束：

$$\sum_{j=1}^{c} u_{ij} = 1, \qquad \forall i = 1, \ldots\ldots, m$$

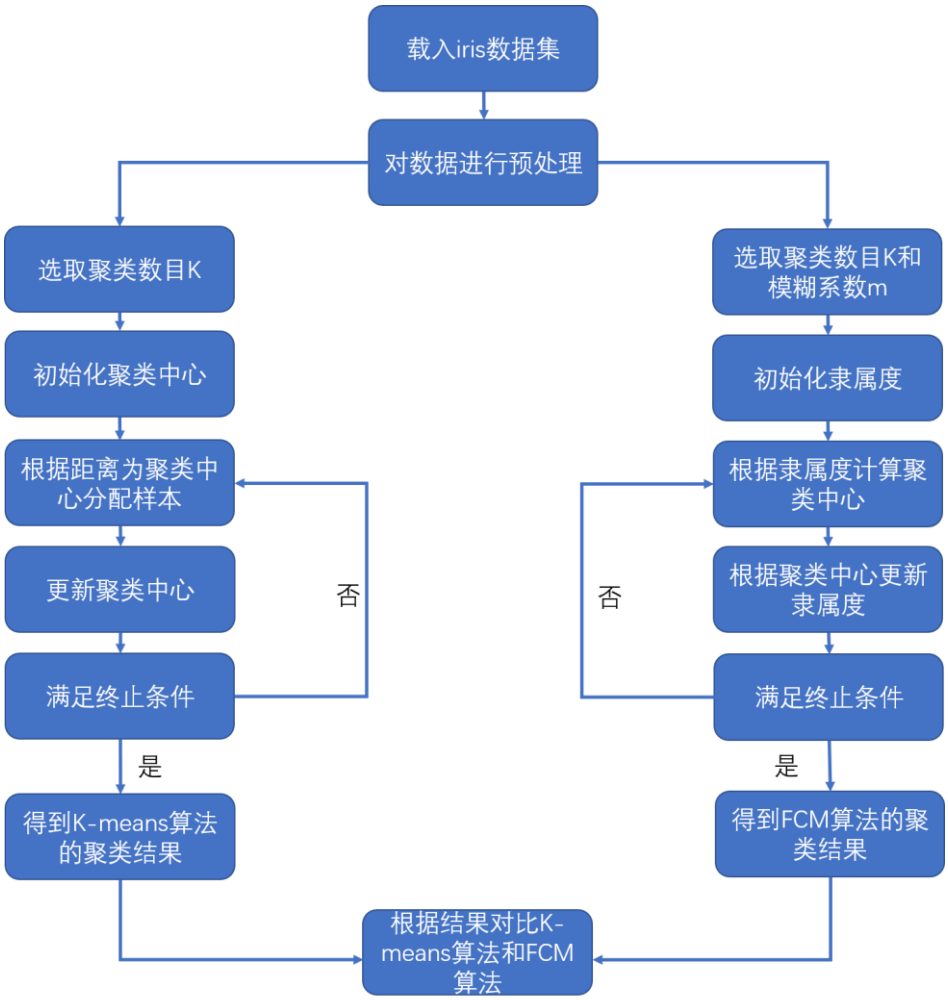
(2）计算 c 个聚类中心 z ， j=1,……,c

(3）计算代价函数。如果其结果小于某个阈值，或者与上一个成本函数值的变化小于某个阈值,则算法停止。

(4）计算新的矩阵 U。返回步骤(2）。

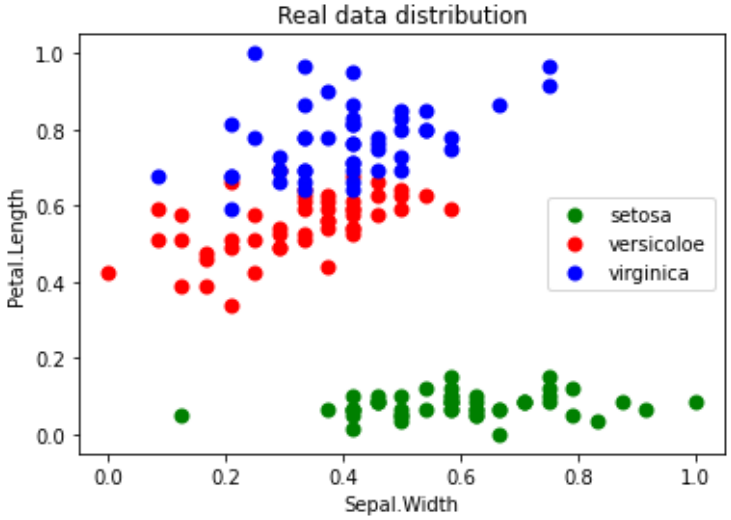
实验步骤：

实验步骤如下图所示，数据预处理时，利用化 $Xi = (Xi - Xmin)/(Xmax - Xmin)$将特征向量归一化。对于 K-means 算法的聚类数目 K，我们选取 iris 数据集的数目类别，即 K=3，对于 FCM 算法，我们选取 K=3,模糊系数 m=2。



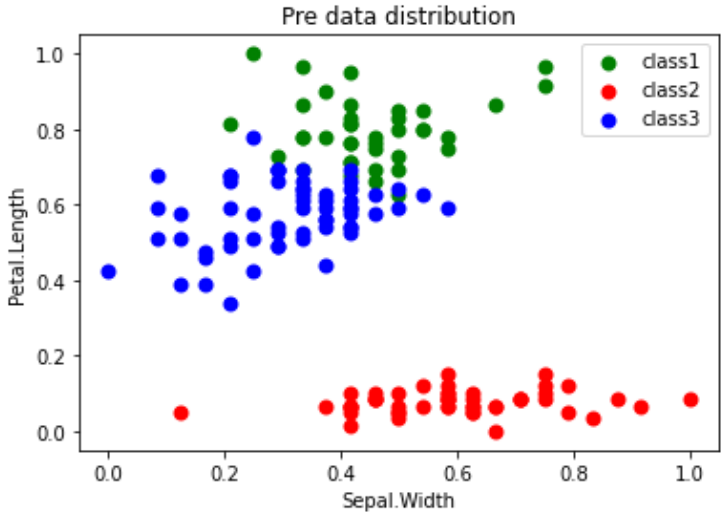
三、 实验仿真结果及分析

我们在 python 环境中对实验进行仿真。首先我们选取 Sepal Width 和 Petal.length 作为 X 轴和 Y 轴绘制出数据的真实分布图。图中蓝色，红色和绿色的点分别代 virginica，versicoloe, setosa. 我们可以看出，三类数据在空间中大致呈现簇状分布。

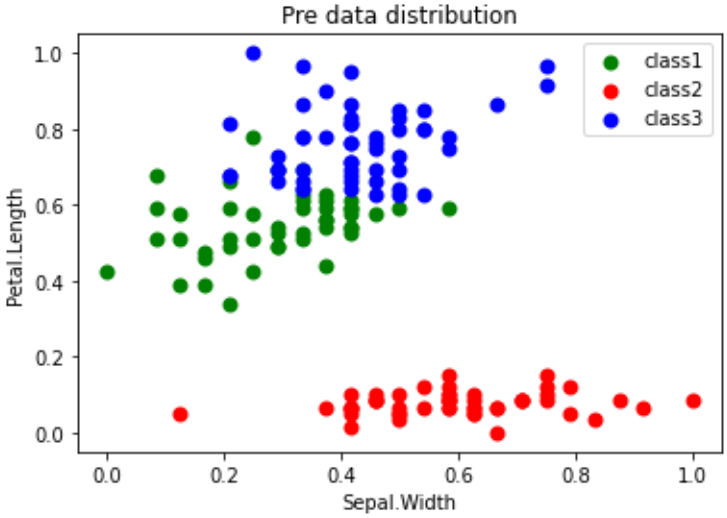


首先，我们编写 K-means 算法，设置终止条件为每个样本的类别不再改变。在经过计算

后，程序将样本分为了 3 类，分布如下图所示。从数据的大致分布中得到每个 Class 代表的类别，可知 Class1 对应 setosa， Class2 对应 versicoloe，Class3 对应 virginica，从而计算出 K-means 算法对 iris 数据集聚类的正确率为 0.86



利用 FCM 算法对数据进行聚类，设置终止条件为达到最大迭代次数或者每个样本的类别不再发生改变。最终得到以下聚类结果。从数据的大致分布图中可得到映射关系，class1 对应 virginica，class2 对应 setosa，class3 对应 versicoloe，从而计算出 FCM 算法对 iris 数据集聚类的正确率为 0.94



在所选定的数据集和参数的情况下，相比于 K-means，FCM 有着更好的聚类效果。

K-means 与 FCM 的比较：

1、K-Means 算法的原理很简单，是一种硬聚类算法，隶属度只有两个取值：0 或 1，其提出的基本根据是“类内误差平方和最小化” 的准则。对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。K-means 的主要优点：原理简单，实现容易，收敛速度快，算法的可解释度比较强。但是，对于 K-means 算法，K 值的选取不好把握；对于不是凸的数据集比较难收敛；采用迭代方法，得到的结果只是局部最优；对噪音和异常点比较的敏感。

2、FCM 算法是由 K-均值算法发展而来的一种重要的聚类分析算法，是一种模糊聚类算法，是 k 均值聚类算法的推广形式，隶属度取值为[0，1]区间内的任何数，提出的基本根据是“类内加权误差平方和最小化”准则。这种算法对特性比较复杂而人们又缺少认识的对象

进行分类，可以有效地实施人工干预,加入人脑思维信息,使分类结果更符合客观实际,可以给出相对的最优分类结果,因而具有一定的实用性。然而该算法同样存在不足之处，主要有：需要设定一些参数，若参数的初始化选取的不合适，可能影响聚类结果的正确性；当数据样本集合较大并且特征数目较多时，算法的实时性不太好。

3、速度比较：从上述两种聚类结果分析中可得，FCM 聚类代码运行时间大约为 K-means 聚类代码运行时间的两倍。因为 FCM 聚类需要经过更多次的迭代，计算量大，数据更加细腻，因此速度会稍慢些，而 K-means 聚类数据简单，速度快。

4、精度比较：由于本次实验所找的数据集本身带有标签，已做好分类，所以将实验结果同原有数据做比较，可得 K-means 聚类准确率大约在 0.81-0.92，FCM 聚类准确度大约在 0.88-0.96，在所选数据集和给定参数的请款下，FCM 有着更好的聚类精度。

四、 实验中遇到的问题及解决方法

遇到的问题：在实现 FCM 聚类算法时，选定算法终止条件为所有样本的类别不再变化。结果导致程序始终不满足终止条件，陷入死循环。

解决方法：首先尝试增加终止条件，当聚类中心的变化小于 1e-4 时停止迭代，即 $(center-center\_old)^2.sum()<1e-4$，但是依然始终无法满足终止条件。在后续探讨过程中发现聚类中心一直在两组点之间震荡。最终设置了最大迭代次数 Maxiter，问题得以解决。

五、 附录

```
#author:fkw
#utf-8
#creat time:2021/11/04
#FCM
#dataset=iris
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
#load the dataset
iris = datasets.load_iris()
# 特征值
diris = iris.data
# 标签
tiris = iris.target
labels = ['setosa', 'versicolor', 'virginica']
# 数组归一化 x=(x-min)/(max-min)
for i in range(4):
    diris[:, i] = (diris[:, i]-np.min(diris[:, i]))/(np.max(diris[:, i])-np.min(diris[:, i]))
#绘制 dataset 散点图,选取 2,3 列作为 Xlable,ylabel
x_axis=diris[:,1]
y_axis=diris[:,2]
plt.scatter(x_axis[tiris==0],y_axis[tiris==0],c='green',marker='.',linewidths=5)
```

```
plt.scatter(x_axis[tiris==1],y_axis[tiris==1],c='red',marker='.',linewidths=5)
plt.scatter(x_axis[tiris==2],y_axis[tiris==2],c='blue',marker='.',linewidths=5)
plt.legend(["setosa","versicoloe","virginica"])
plt.title("Real data distribution")
plt.xlabel('Sepal.Width')
plt.ylabel('Petal.Length')
plt.show()
#计算欧氏距离
def caldis(center,data):
    len_center=center.shape[0]
    len_data=diris.shape[0]
    #print(len_data,len_center)
    dis=np.zeros((len_data,len_center))
    for i in range(0,len_data):
        for j in range(0,len_center):
            dis[i,j]=np.math.sqrt(sum((data[i]-center[j])**2))
    return dis
#随机生成隶属度矩阵
def randU(k,data):
    len_data=data.shape[0]
    U=np.random.random((len_data,k))
    sumU=1/np.sum(U,axis=1)
    #print((sumU.T).shape)
    U = np.multiply(U.T, sumU).T
    return U
#计算 k 个聚类中心
def calcen(U,data,k):
    center=np.zeros((k,data.shape[1]))
    for i in range(0,k):
        center[i]=np.dot(U[:,i].T,data)/sum(U[:,i])
    return center
#print(calcen(randU(3,diris),diris,3))
#更新隶属度,m 为模糊系数
def computeU(data,center,m):
    dis=caldis(center,data)
    U=np.zeros((data.shape[0],center.shape[0]))
    for i in range(0,data.shape[0]):
        for j in range(0,center.shape[0]):
            for k in range(0,center.shape[0]):
                U[i,j]=U[i,j]+(dis[i,j]/dis[i,k])**(2/(m-1))
            U[i,j]=1/U[i,j]
    return U
```

```
#ans=computeU(diris,calcen(randU(3,diris),diris,3),2)
#返回分类的类别
def calans(U):
    return np.argmax(U,axis=1)
def train(data,k,m):
    U=randU(k,data)#初始化隶属度
    ans_old=calans(U)#初始化 ans
    center_old=np.zeros((k,data.shape[1]))#初始化 center_old
    item=0
    Maxitem=500#最大迭代次数
    while(1):
        center=calcen(U,data,k)#计算中心
        U=computeU(data,center,m)#更新 U
        ans=calans(U)#更新 ans
        item+=1
        if(((center_old-center)**2).sum()<=1):
            return ans #center 几乎不再改变则停止更新
        if(item==Maxitem):
            return ans #达到最大迭代次数
        if((ans-ans_old).any()==0):
            return ans #如果 ans 不再改变停止更新
ans=train(diris,3,2)
print(ans)
plt.scatter(x_axis[ans==0],y_axis[ans==0],c='green',marker='.',linewidths=5)
plt.scatter(x_axis[ans==1],y_axis[ans==1],c='red',marker='.',linewidths=5)
plt.scatter(x_axis[ans==2],y_axis[ans==2],c='blue',marker='.',linewidths=5)
plt.legend(["class1","class2","class3"])
plt.title("Pre data distribution")
plt.xlabel('Sepal.Width')
plt.ylabel('Petal.Length')
plt.show()
```

```
#author:fkw
#utf-8
#creat time:2021/11/04
#K-means
#dataset=iris
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np
#load the dataset
iris = datasets.load_iris()
# 特征值
```

```
diris = iris.data
# 标签
tiris = iris.target
labels = ['setosa', 'versicolor', 'virginica']
# 数组归一化 x=(x-min)/(max-min)
for i in range(4):
    diris[:, i] = (diris[:, i]-np.min(diris[:, i]))/(np.max(diris[:, i])-np.min(diris[:, i]))
#绘制 dataset 散点图,选取 2,3 列作为 Xlable,ylabel
x_axis=diris[:,1]
y_axis=diris[:,2]
plt.scatter(x_axis[tiris==0],y_axis[tiris==0],c='green',marker='.',linewidths=5)

plt.scatter(x_axis[tiris==1],y_axis[tiris==1],c='red',marker='.',linewidths=5)
plt.scatter(x_axis[tiris==2],y_axis[tiris==2],c='blue',marker='.',linewidths=5)

plt.legend(["setosa","versicoloe","virginica"])
plt.title("Real data distribution")
plt.xlabel('Sepal.Width')
plt.ylabel('Petal.Length')
plt.show()
#kmeans
#随机生成 K 个 center
def randcenter(k,data):
    len_data=data.shape[1]
    center=np.random.random((k,len_data))
    #print(center)
    return center
#print(randcenter(3,diris))
#计算所有样本和 center 之间的距离
def caldis(center,data):
    len_center=center.shape[0]
    len_data=diris.shape[0]
    #print(len_data,len_center)
    dis=np.zeros((len_data,len_center))
    for i in range(0,len_data):
        for j in range(0,len_center):
            dis[i,j]=np.math.sqrt(sum((data[i]-center[j])**2))
    return dis
#print(caldis(randcenter(3,diris),diris))
dis=caldis(randcenter(3,diris),diris)
#对样本进行分类
def cla(dis):
    ans=np.zeros((dis.shape[0],1))
```

```
    #选出每行最小的值
    ans=np.argmin(dis,axis=1)
    return ans
#print(cla(dis))
#训练
def train(k,data):
    #初始化 center
    center=randcenter(k,data)
    ans=np.zeros((data.shape[0],1))
    ans_old=np.ones((data.shape[0],1))
    while(1):
        #print(ans_old,ans)
        if((ans_old-ans).any()==0):
            return ans
        ans_old=ans
        #计算新的 ans(分类结果)
        ans=cla(caldis(center,data))
        #更新 center
        for i in range(0,center.shape[0]):
            center[i]=np.mean(data[ans==i],axis=0)
        #print(center)
#计算分类结果
ans=train(3,diris)
print(ans)
plt.scatter(x_axis[ans==0],y_axis[ans==0],c='green',marker='.',linewidths=5)
plt.scatter(x_axis[ans==1],y_axis[ans==1],c='red',marker='.',linewidths=5)
plt.scatter(x_axis[ans==2],y_axis[ans==2],c='blue',marker='.',linewidths=5)
plt.legend(["class1","class2","class3"])
plt.title("Pre data distribution")
plt.xlabel('Sepal.Width')
plt.ylabel('Petal.Length')
plt.show()
```