**第4章 聚类**

**一、**简述**Kmeans、DBSCAN、AGNES聚类算法**的原理、关键操作点（包括各自关键参数的调参说明等，不要超过**300字**，逻辑要通，**要有小标题**，要排版清晰，请不要网络大幅摘抄）

**K-means聚类算法 基于距离（经典欧式距离）**

核心思想：对于给定的样本集，按照样本之间的距离大小，将样本划分为 k 个簇，使得簇内的样本点之间的距离足够紧密，而不同簇之间的距离尽可能的大，转化为数学概念中则为最小化平方误差。而 k 则代表的是所分类成为的簇的个数。

实际操作：

1、先根据现有的数据集选择一个随机的样本点，以作为每一个不同标签类（簇）中的均值向量（也就是质心），简单解释就是将所有的选中的值作为标准点。

2、再将其他所有的点和 选定的簇中的标准点进行计算欧式距离，根据欧式距离的大小来判断该样本点的所属簇，并给予标签并且分类。

3、将所有的点进行标签并且聚类之后，再次计算其簇的质心，将所有的簇内样本点的不同特征值进行求均值，得到新的簇的均值向量（也就是多维值，根据特征值的个数来决定其向量的维度）

4、再根据新得出的簇的均值向量，将所有的样本点，再次根据这个新得出的簇的均值向量来进行新的分类聚类将样本点进行标签，最后再 计算新的均值向量，进行下一次迭代的。

**DBSCAN 聚类（基于密度聚类）**

核心思想：根据其选定的核心点，来计算后来的样本点，并且与之前预先设定邻域值做比较，如果符合其之前设置的邻域范围则将其分为同一个簇中，否则不进行合并。

**关键概念**

**ε-邻域**：以点p为中心，半径为ε的区域内的所有点

**核心点**：其ε-邻域内包含至少MinPts个点的点

**直接密度可达**：如果点q在点p的ε-邻域内，且p是核心点，则q从p直接密度可达

**密度可达**：存在点链p1,p2,...,pn，其中p1=p，pn=q，pi+1从pi直接密度可达

**密度相连**：如果点p和q都从点o密度可达，则p和q密度相连

1. 找出所有核心点
2. 从任意未处理的核心点出发，找出其所有密度可达的点，形成一个簇
3. 重复步骤2，直到所有核心点都被处理

这种聚类方法特别适合处理含有噪声的数据，且不需要预先指定簇的数量。

**AGNES算法思想（层次聚类）**

1. 层次聚类算法的一种，自底向上的聚类方法。

* 初始时将每个样本视为一个独立的簇
* 逐步合并最相似的簇，直到达到预设的簇数量k

1. **距离计算方法**：

* 计算两点间的欧氏距离
* 单链接法，两簇之间的最小距离
* 均链接法，两簇之间所有点对距离的平均值

1. **聚类过程**：

* 初始化：每个样本点作为一个单独的簇
* 计算所有簇对之间的距离
* 合并距离最小的两个簇
* 更新距离矩阵
* 重复上述过程直到簇的数量达到k

1. **输出结果**：

* 返回k个簇，每个簇包含多个样本点
* 通过散点图可视化不同簇，使用不同颜色和标记形状区分

这个算法与先前看到的K-means不同，AGNES是一种层次聚类方法，不需要预先指定初始中心点，且能够处理不规则形状的簇。

# 二、利用Kmeans算法对鸢尾花数据进行聚类，不要调用sklearn中kmeans。

要求：

1）针对当前数据、模型，**详叙数据预处理、模型参数调整过程及效果比对**；

2）代码要列出，重点代码加注释说明，特别是自己调试过程中的自我理解；

3）运行结果要截图，结果要文字说明；

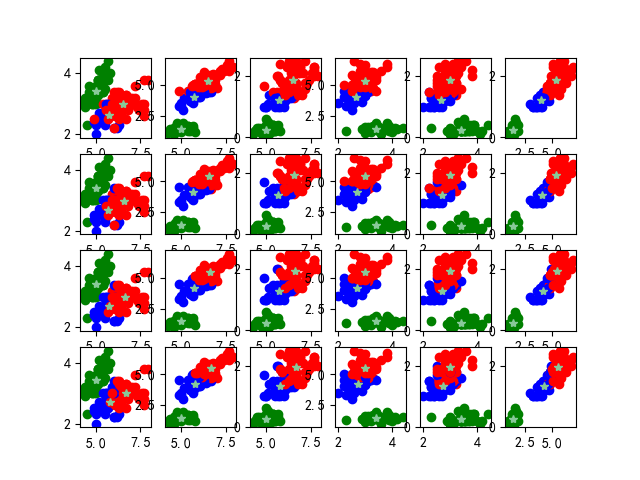
4）注意排版

数据预处理、参数调整和

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
from numpy import \*

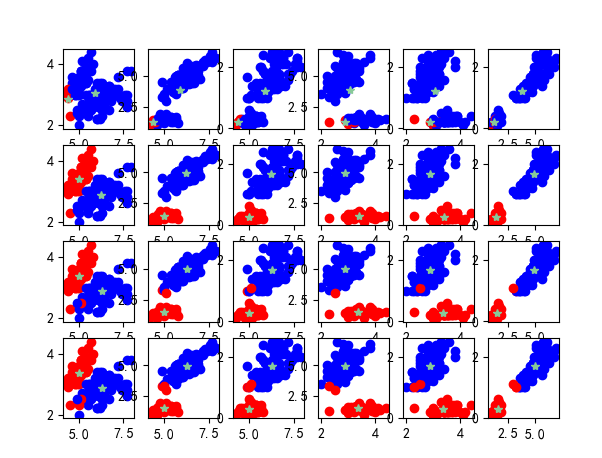
# 数据预处理  
def open\_csvgetdata() :  
 dataTrain = pd.read\_csv("./iris.csv")  
 dataTrain = dataTrain.values  
 dataTrain = np.array(dataTrain)  
 ylabel = dataTrain[:,5]  
 dataTrain = dataTrain[:,1:5]  
  
 y = pd.Categorical(ylabel).codes  
 return dataTrain,y  
  
def dist\_eclud(v1,v2) :  
 return sqrt(sum(power(v1-v2,2)))  
# 就是正常的距离公式  
# v1 的各个元素 和 v2 进行减法操作，然后对其所有进行平方  
# sum 则对于所有的元素加和 (x1 - x2)^2 + (y1 - y2)^2  
  
def update\_cluster(k,mu,X,ylabel) :  
  
 for i in range(X.shape[0]):  
 min\_dist = float('inf')  
 for index in range(k):  
 dist = dist\_eclud(mu[index], X[i])  
 # print("第1个距离：",dist)  
 if dist < min\_dist:  
 min\_dist = dist  
 ylabel[i] = index  
 # 根据距离 标签分类  
 return ylabel  
  
  
  
  
  
def show\_figure(plt,iters,dataSet, k, centroids, clusters):  
  
 num\_samples, dim = dataSet.shape  
 cnt = 0  
 for i in range(4) :  
  
 for j in range(i+1,4):  
 # print("i is : "+str(i)+" j is : "+str(j) + " cnt si : " +str(cnt))  
 # print(iters)  
 # print(iters \* 6 + cnt)  
 plt.subplot(4, 6, iters \* 6 + cnt + 1, frameon=True)  
 cnt += 1  
 # t = '第' + str(iters + 1) + '次迭代后'  
 # plt.legend(title=t)  
 marker = ['or', 'ob', 'og', 'ok']  
 marker2 = ['\*r', '\*b', '\*g', '\*k']  
  
 for kk in range(num\_samples):  
 mark\_index = int(clusters[kk])  
 plt.plot(dataSet[kk, i], dataSet[kk, j], marker[mark\_index])  
  
 for kk in range(k):  
 plt.plot(centroids[kk, i], centroids[kk, j], marker2[0],color='#88c999')  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.rcParams['axes.unicode\_minus'] = False  
  
def update\_centroids(k, mu, X, y\_label):  
 # k 代表的是簇的个数  
 # mu 代表的是 不同的簇的 质心向量  
 # X[] 代表的是 样本点 集合 存储的是样本点的特征值  
 # 更新 质心向量 mu[] 数组  
 for i in range(k):  
 sum = np.array([0.0, 0.0])  
 num = np.sum(y\_label == i)  
 cluster\_index= np.where(y\_label == i)  
 # print(cluster\_index)  
 # 根据簇的分类找出相关的变量 的 索引  
 # print("cluster\_index:", cluster\_index)  
 # 求其平均值 得到当前的质心向量  
 centroid = np.mean(X[cluster\_index], axis=0)  
  
 mu[i] = centroid  
 return mu  
  
  
if \_\_name\_\_ == '\_\_main\_\_' :  
 # print(\_\_name\_\_)  
 dataTrain , y\_label = open\_csvgetdata()  
 len = dataTrain.shape[0]  
  
 k = 3  
 alldatainedx = [ i for i in range(len)]  
 mu\_index = np.random.choice(alldatainedx,k)  
 # 在数组 alldataindex中选择 k个不同的随机数  
 # print(mu\_index)  
 # mu=np.zeros(k,dataTrain.shape[1])  
 mu = dataTrain[mu\_index]  
 # print(mu)  
 iters = 4  
  
 for i in range(iters) :  
 ylabel = np.empty(len)  
 # ylabel = np.array(ylabel)  
 ylabel = update\_cluster(k,mu,dataTrain,ylabel)  
 mu = update\_centroids(k,mu,dataTrain,ylabel)  
 show\_figure(plt,i,dataTrain,k,mu,ylabel)  
  
 plt.show()  
  
  
# x1 =np.array([1,2])  
# x2 =np.array([3,2])  
# print( sum( power(x1-x2,2)) )

**实验结果截图（分为三类）：**



该结果：每一行的图片是根据鸢尾花的四个特征值 根据不同的两个特征值来排列组合进行。不同行代表着对于簇的质心向量不同的迭代次数，分别为一次、两次、三次和四次。我们可以明显看出，每一次迭代都对于其进行聚类，并且对于所聚类的个数和密集程度变得越来越紧密了。

**（分为两类）：**



说明：建议用Kmeans、DBscan、AGNES分别对鸢尾花数据（已删除类别标签）进行聚类，聚类结果与原始类别进行比对，找出最佳的聚类算法及最优参数组合。

**三、[附加题].**利用聚类算法（Kmeans、DBscan、AGNES三种选其一）解决**自己遇到的问题**。

[解题要求同题2]

1. **AGNES算法思想**：层次聚类算法的一种，自底向上的聚类方法。

* 初始时将每个样本视为一个独立的簇
* 逐步合并最相似的簇，直到达到预设的簇数量k

1. **距离计算方法**：

* dist\_eclud: 计算两点间的欧氏距离
* get\_min\_dist: 单链接法，两簇之间的最小距离
* get\_average\_dist: 均链接法，两簇之间所有点对距离的平均值

1. **聚类过程**：

* 初始化：每个样本点作为一个单独的簇
* 计算所有簇对之间的距离
* 合并距离最小的两个簇
* 更新距离矩阵
* 重复上述过程直到簇的数量达到k

1. **输出结果**：

* 返回k个簇，每个簇包含多个样本点
* 通过散点图可视化不同簇，使用不同颜色和标记形状区分

这个算法与先前看到的K-means不同，AGNES是一种层次聚类方法，不需要预先指定初始中心点，且能够处理不规则形状的簇。在本例中，西瓜数据被分为4类，基于密度和含糖率的特征分布。

*# 计算两个样本点间的欧氏距离*

def dist\_eclud(*x1*, *x2*):

*return* math.sqrt(math.pow(x1[0]-x2[0], 2)+math.pow(x1[1]-x2[1], 2))

*# 计算两个簇之间的最小距离（单链接）*

def get\_min\_dist(*C\_x*, *C\_y*):

*return* min(dist\_eclud(x, y) *for* x *in* C\_x *for* y *in* C\_y)

*# 计算两个簇之间的平均距离（均链接）*

def get\_average\_dist(*C\_x*, *C\_y*):

*return* sum(dist\_eclud(x, y) *for* x *in* C\_x *for* y *in* C\_y)/(len(C\_x)\*len(C\_y))

*# 在距离矩阵中找到最小距离对应的簇索引*

def find\_min\_index(*D*):

    min = float('inf')

    min\_i = 0

    min\_j = 0

*for* i *in* range(len(D)):

*for* j *in* range(len(D[i])):

*if* i != j and D[i][j] < min:

                min = D[i][j]

                min\_i = i

                min\_j = j

*return* min\_i, min\_j, min

*# 初始化簇集合和距离矩阵*

*# 初始时每个样本点作为一个独立的簇*

def get\_dist\_set(*data*):

    C = []  *# 簇集合*

    D = []  *# 距离矩阵*

*# 初始化簇集合，每个样本作为一个簇*

*for* x *in* data:

        C\_x = []

        C\_x.append(x)

        C.append(C\_x)

*# 计算初始的距离矩阵*

*for* x *in* C:

        D\_x\_start = []

*for* y *in* C:

            d = dist\_eclud(x[0], y[0])

            D\_x\_start.append(d)

        D.append(D\_x\_start)

*return* C, D

**层次聚类算法主函数**

*# AGNES层次聚类算法主函数*

*# data: 数据集*

*# dist: 距离计算函数*

*# k: 目标簇的数量*

def AGNES\_clustering(*data*, *dist*, *k*):

    n = len(data)  *# 初始簇的数量等于样本数*

    C, D = get\_dist\_set(data)  *# 初始化簇和距离矩阵*

*# 当簇的数量大于目标k时，继续合并*

*while* n > k:

*# 找到距离最近的两个簇*

        min\_i, min\_j, min = find\_min\_index(D)

        print(min\_i, min\_j, min)  *# 打印待合并的簇索引和距离*

*# 合并这两个簇*

        C[min\_i].extend(C[min\_j])

        C.remove(C[min\_j])

*# 更新距离矩阵*

        D = []

*for* x *in* C:

            D\_x\_start = []

*for* y *in* C:

                D\_x\_start.append(dist(x, y))

            D.append(D\_x\_start)

        n -= 1  *# 簇的数量减1*

*return* C

**图形化：**

*# 可视化聚类结果*

def show\_figure(*C*):

    color\_value = ['r', 'g', 'b', 'y', 'm', 'k', 'c']  *# 颜色列表*

    marker\_value=['o', 's', '\*', 'd', 'x', 'v', 'p']   *# 标记形状列表*

*# 逐类绘制散点图*

*for* i *in* range(len(C)):

        crd\_x = []  *# 横坐标(密度)*

        crd\_y = []  *# 纵坐标(含糖率)*

*for* j *in* range(len(C[i])):

            crd\_x.append(C[i][j][0])

            crd\_y.append(C[i][j][1])

        class\_name='类'+str(i)

        plt.scatter(crd\_x, crd\_y, *marker*=marker\_value[i%len(marker\_value)],

*color*=color\_value[i%len(color\_value)], *label*=class\_name)

*# 设置图表属性*

    plt.xlim(0.1,0.9)

    plt.ylim(0,0.8)

    plt.xlabel('密度')

    plt.ylabel('含糖率')

    plt.legend(*loc*='upper right')

    plt.rcParams['font.sans-serif'] = ['SimHei']  *# 显示中文*

    plt.rcParams['axes.unicode\_minus'] = False

plt.show()

**层次聚类实验结果**：

