《机器学习基础》实验报告

实验题目	聚类算法实践

一、实验目的

掌握聚类算法原理。

二、实验项目内容

1. 理解并描述各一种原型聚类算法、密度聚类算法的原理。

2. 编程实践，将 k 均值算法应用于合适的数据集（西瓜数据集、鸢尾花数据集或其它合适数据集），设置三组以上不同的 k 值，分别使用三组不同初始中心点，对比实验结果，分析聚类结果的优劣。

三、实验过程或算法（源程序）

1.聚类算法原理

（1）原型聚类：k-means 聚类算法

K 均值算法是一种聚类无监督学习算法，是原型聚类的一种。其核心思想是通过不断迭代优化聚类中心的位置，使得每个样本点都被分配到最近的聚类中心所属的簇中，并且使得簇内的样本相互之间的距离尽可能小，而不同簇之间的距离尽可能大。这样就实现了对数据集的聚类分析，将相似的样本归为一类，从而发现数据集中的内在结构。

其主要步骤为：

1）确定 k 值的大小，选取初始中心点

2）对于每个样本，计算其与各个聚类中心的距离，并将其分配到距离最近的聚类中心所属的簇中

3）针对每个簇，计算其中所有样本的均值，将该均值作为新的聚类中心

4）重复以上步骤，直到迭代优化到指定条件获得达到了迭代次数为止

（2）密度聚类：DBSCAN 算法

DBSCAN（Density-Based Spatial Clustering of Applications with Noise），它是一种基于密度的聚类算法。DBSCAN 算法将样本点分为核心点、边界点和噪声点，通过对样本点的密度进行判断来实现聚类。与原型算法不同的是，DBSCAN 算法可以有效地发现具有不同密度的聚类，对异常值具有一定的鲁棒性，并且不需要预先指定聚类数量。密度聚类算法可以在处理非凸形状的簇和噪声点方面表现得更好，适用于不规则分布的数据集。

其主要步骤如下：

1）设定两个参数：邻域半径 ε 和最小样本数 MinPts。

2）对数据集中的每个样本点进行遍历，如果一个样本点的 ε-邻域内包含至少 MinPts 个样本点，则将该样本点标记为核心点

3）对核心点进行扩展，找出密度可达的样本点，并将它们归为同一个簇

4）继续处理未被访问的样本点，直至所有样本点都被访问。

5）标记剩余的未分类点为噪声点

2.源程序

```
import numpy as np
import random
import matplotlib.pyplot as plt
import csv

def loadDataSet(filename):
    dataset = []
    labelset = []
    with open(filename, 'r', encoding='utf-8') as csvfile:
        csv_reader = csv.reader(csvfile)
        header = next(csv_reader)
        for row in csv_reader:
            data_row = [float(row[1]),
float(row[2]),float(row[3]),float(row[4])]
            dataset.append(data_row)
    return dataset


def findCentroids(datMat,k):    #获取三组不同的中心点
    data_set=np.array(datMat)
    init_centers_1 = random.sample(data_set.tolist(), k)
    init_centers_2 = random.sample(data_set.tolist(), k)
    init_centers_3 = random.sample(data_set.tolist(), k)

    centroidsList=[]
    centroidsList.append(np.array(init_centers_1))
    centroidsList.append(np.array(init_centers_2))
    centroidsList.append(np.array(init_centers_3))
    return centroidsList

def distEclud(vecA,vecB):     #计算欧式距离
    return np.sqrt(sum(np.power(vecA-vecB,2)))

def Kmeans(data_get,centroidsList):   #划分k个簇
#    np.array(data_get)     #将数据集转为数组
    distculde = {}   #建立一个字典
    flag = 0        #元素分类标记，记录与相应聚类距离最近的那个类
    for data in data_get:
        vecA = np.array(data)
        minDis = float('inf')     #始化为最大值
        for i in range(len(centroidsList)):
            vecB = centroidsList[i]
            distance = distEclud(vecA, vecB)    #计算距离
            if distance < minDis:     #直至找出距离最小的质点
                minDis = distance
                flag = i
        if flag not in distculde.keys():
            distculde[flag] = list()
        distculde[flag].append(data)
    return distculde

def getCentroids(distculde):   #得到新的质心
```

```
    newcentroidsList = []     #建立新质点集
    for key in distculde:
        cent = np.array(distculde[key])
        newcentroid = np.mean(cent,axis=0)      #计算新质点
        newcentroidsList.append(newcentroid.tolist())
    return np.array(newcentroidsList)     #返回新质点数组
```

```
def calculate_Var(distculde, centroidsList):
    #计算均方误差
    item_sum = 0.0
    for key in distculde:
        vecA = centroidsList[key]
        dist = 0.0
        for item in distculde[key]:
            vecB = np.array(item)
            dist += distEclud(vecA, vecB)
        item_sum += dist
    return item_sum
```

```
def showCluster(distculde, centroidsList, initcentrol):
    # 画聚类图像
    plt.figure()
    x = []
    y = []
    x.append(centroidsList[:,0].tolist())
    y.append(centroidsList[:,1].tolist())
    init_centers = np.array(initcentrol)
    x_centers = init_centers[:, 0]
    y_centers = init_centers[:, 1]
    '''
    new_centers = np.array(centroidsList)
    x_centers_new = new_centers[:, 0]
    y_centers_new = new_centers[:, 1]
    '''
```

```
    colourList = ['g^', 'bo', 'r^', 'yo']
    for i in distculde:
        # 获取每簇
        centx = []
        centy = []
        for item in distculde[i]:
            centx.append(item[0])
            centy.append(item[1])
        plt.plot(centx, centy, colourList[i])   # 画簇
```

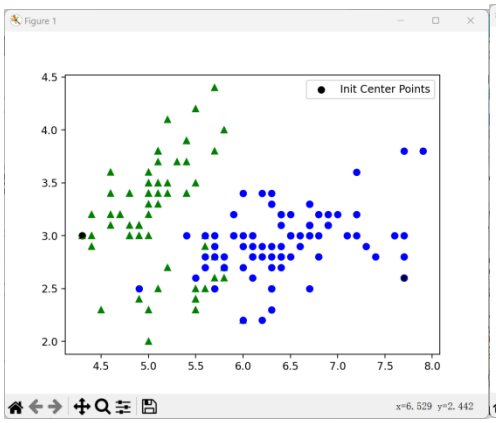
```
    plt.scatter(x_centers, y_centers, color='black', label='Init
Center Points')
    #plt.scatter(x_centers_new, y_centers_new, color=(1,0,1),
label='New Center Points')
    plt.plot(x, y, 'k*')  # 画质点，为黑色*
    plt.legend()
    plt.show()
```

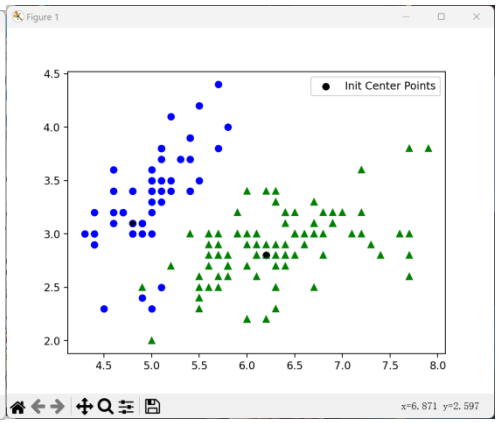
```
def kmeansfork(k):
    datMat = loadDataSet('D:\zjw\demo\machine
learning\Iris-data.csv')
    centroidsList = findCentroids(datMat, k)      #随机获得k 个聚类中
心
    for centrolist in centroidsList:
        initcentrol=[]
        initcentrol = centrolist
        distculde = Kmeans(datMat,centrolist)      #第一次聚类迭代
        newVar = calculate_Var(distculde,centrolist)
        oldVar = -0.0001   #初始化均方误差
        while abs(newVar - oldVar) >= 0.0001:
            centroidsList = getCentroids(distculde)
            distculde = Kmeans(datMat, centrolist)
            oldVar = newVar
            newVar = calculate_Var(distculde, centrolist)
        showCluster(distculde,centrolist,initcentrol)
```

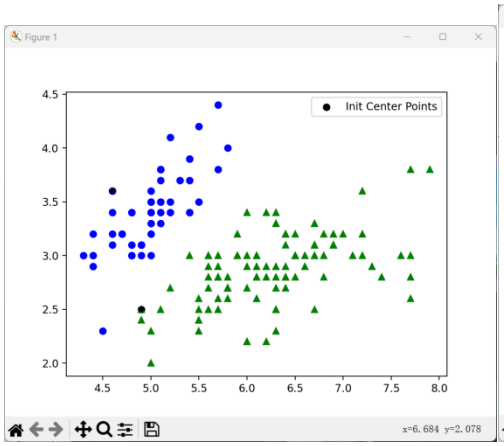
```
if __name__ == "__main__":
    k_values = [2, 3, 4]
    for k in k_values:
        kmeansfork(k)
```

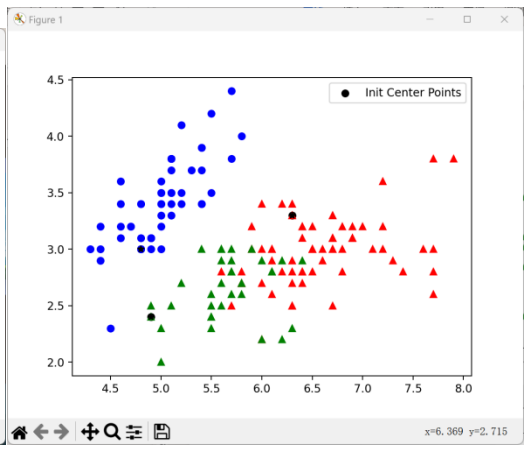
四、实验结果及分析

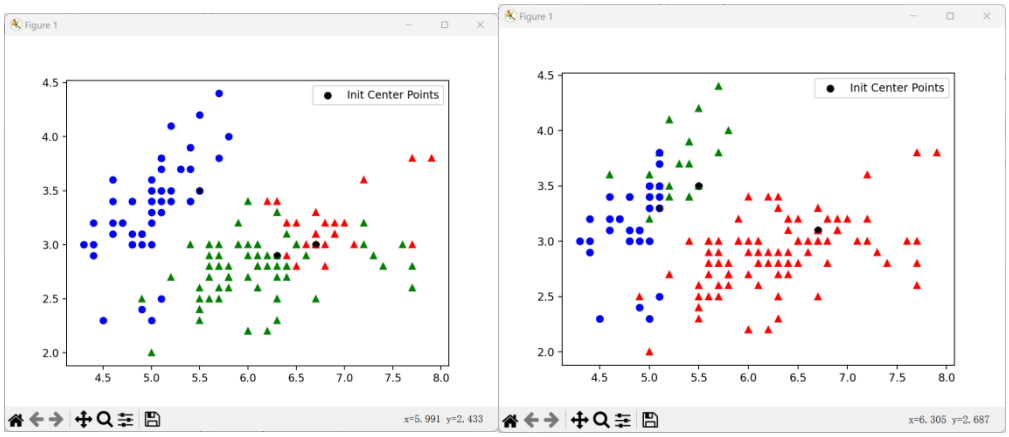
1.运行结果

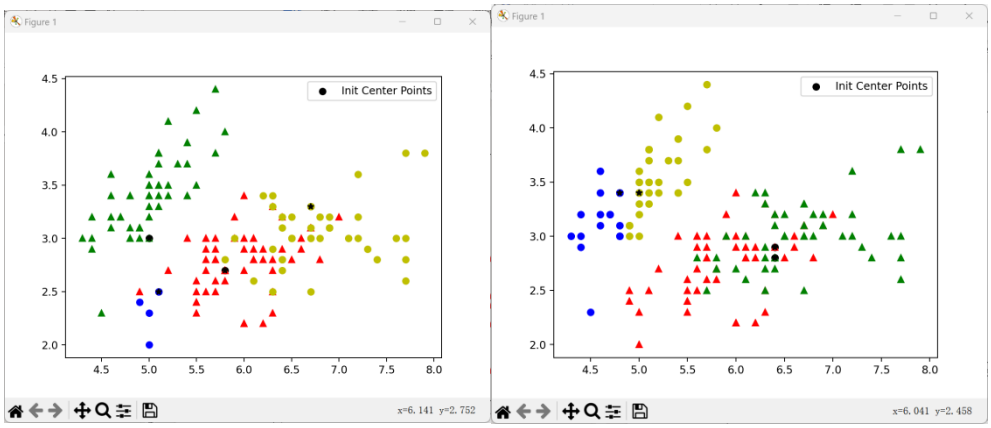


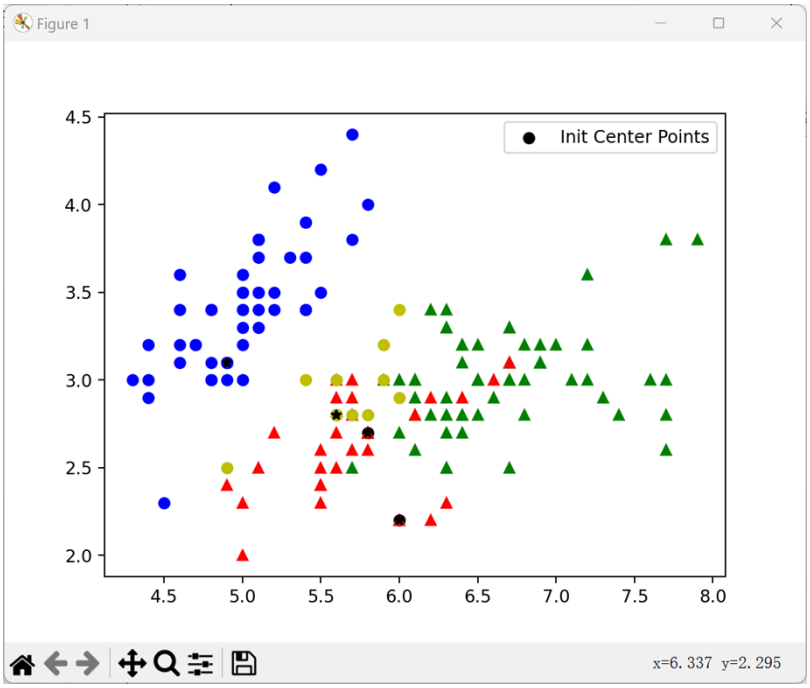












2.实验结果分析

由结果可以看出，使用 k-means 聚类算法，最后聚类的结果与选取的初始中心点有关。这是因为 k-means 算法是根据初始中心点找的局部最优解，结果受初始中心点的影响较大。

同时，不同的 k 值对于结果的影响也较大，较大的 k 值增大了计算复杂度，通常会导致更细粒度的聚类，而较小的 k 值则可能导致聚类过于粗糙。