【例 8.1】 实现 k-means 算法对 testSet.txt 数据集中的数据进行聚类，其中设置 4 个质心，绘制聚类结果图像。

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

def loadDataSet(fileName):                          #载入数据集
    dataMat = []
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = list(map(float, curLine))
        dataMat.append(fltLine)
    return dataMat
def distEclud(vecA, vecB):                          #数据向量计算欧式距离
    return np.sqrt(np.sum(np.power(vecA -vecB, 2)))
def randCent(dataSet, k):                           #随机初始化 k 个质心(质心位于边界之内)
    n = np.shape(dataSet)[1]                        #得到数据样本的维度
    centroids = np.mat(np.zeros((k, n)))            #初始化为一个(k,n)的全零矩阵
    for j in range(n):                              #遍历数据集的每一列
        minJ = np.min(dataSet[:, j])                #得到该列数据的最小值和最大值
        maxJ = np.max(dataSet[:, j])
        rangeJ = float(maxJ -minJ)                  #得到该列数据的范围(最大值-最小值)
        #k 个质心向量的第 j 维数据值为最小值和最大值之间的某一随机值
        centroids[:, j] = minJ +rangeJ * np.random.rand(k, 1)
    return centroids                                #返回初始化得到的 k 个质心向量

def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = np.shape(dataSet)[0]                        #获取数据集样本数
    clusterAssment = np.mat(np.zeros((m, 2)))#初始化一个 m×2 的全零矩阵
    centroids = createCent(dataSet, k)              #创建初始的 k 个质心向量
    clusterChanged = True          #判断聚类结果是否发生变化的布尔类型变量
    while clusterChanged:          #执行聚类算法,直至所有数据点的聚类结果不发生变化
        clusterChanged = False     #聚类结果是否发生变化布尔类型变量值置为 False
        for i in range(m):         #遍历数据集中的每一个样本向量
            minDist = float('inf') #初始化最小距离为正无穷,对应的索引为-1
            minIndex = -1
            for j in range(k):     #循环 k 个类的质心
                distJI = distMeas(centroids[j, :], dataSet[i, :])
                #计算数据点到质心的欧几里得距离
                if distJI < minDist:          #如果距离小于当前最小距离
                    minDist = distJI          #以当前距离为最小距离,对应的索引为 j
```

```
                minIndex = j
            if clusterAssment[i, 0] != minIndex:  #第 i 个样本的聚类结果发生变化
                clusterChanged = True             #变量值置为 True,继续执行算法
            clusterAssment[i, :] = minIndex, minDist**2
            #更新当前样本的聚类结果和平方误差
        for cent in range(k):                     #遍历每一个质心
            #将数据集中所有属于当前类的样本通过条件过滤筛选出来
            ptsInClust = dataSet[np.nonzero(clusterAssment[:, 0].A ==cent)[0]]
            #计算这些数据的均值(axis=0,求列均值),作为该类的质心向量
            centroids[cent, :] = np.mean(ptsInClust, axis=0)
    return centroids, clusterAssment              #返回 k 个聚类结果及误差
def plotDataSet(filename):                        #绘制数据集
    datMat = np.mat(loadDataSet(filename))        #导入数据
    myCentroids, clustAssing = kMeans(datMat, 4)  #执行 k-means 算法,其中 k 为 4
    clustAssing = clustAssing.tolist()
    myCentroids = myCentroids.tolist()
    xcord = [[], [], [], []]
    ycord = [[], [], [], []]
    datMat = datMat.tolist()
    m = len(clustAssing)
    for i in range(m):
        if int(clustAssing[i][0]) == 0:
            xcord[0].append(datMat[i][0])
            ycord[0].append(datMat[i][1])
        elif int(clustAssing[i][0]) == 1:
            xcord[1].append(datMat[i][0])
            ycord[1].append(datMat[i][1])
        elif int(clustAssing[i][0]) == 2:
            xcord[2].append(datMat[i][0])
            ycord[2].append(datMat[i][1])
        elif int(clustAssing[i][0]) == 3:
            xcord[3].append(datMat[i][0])
            ycord[3].append(datMat[i][1])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    #绘制样本点
    ax.scatter(xcord[0], ycord[0], s=20, c='b', marker='*', alpha=.5)
    ax.scatter(xcord[1], ycord[1], s=20, c='r', marker='D', alpha=.5)
    ax.scatter(xcord[2], ycord[2], s=20, c='c', marker='>', alpha=.5)
    ax.scatter(xcord[3], ycord[3], s=20, c='k', marker='o', alpha=.5)
    #绘制质心
    ax.scatter(myCentroids[0][0], myCentroids[0][1], s=100, c='k', marker='+',
```

```
alpha=.5)
    ax.scatter(myCentroids[1][0], myCentroids[1][1], s=100, c='k', marker='+',
alpha=.5)
    ax.scatter(myCentroids[2][0], myCentroids[2][1], s=100, c='k', marker='+',
alpha=.5)
    ax.scatter(myCentroids[3][0], myCentroids[3][1], s=100, c='k', marker='+',
alpha=.5)
    plt.title('DataSet')
    plt.xlabel('X')
    plt.show()
if __name__=='__main__':
    plotDataSet('E:\数据挖掘 &Python\第 8 章\data/testSet.txt')
```

算法输出结果如图 8.8 所示。

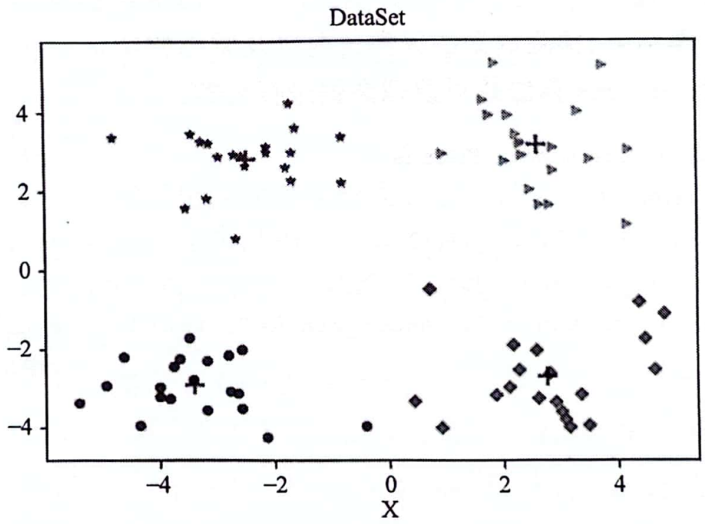

图 8.8　*k*-means 算法聚类结果

例 8.1 中的主要函数如表 8.4 所示。

表 8.4　例 8.1 中的主要函数

函　数	描　述	参　数	返　回
loadDataSet(fileName)	从文件中读取数据集	fileName：文件名	载入数据
distEclud(vecA,vecB)	计算距离，这里用的是欧几里得距离，当然其他距离也可以通过相应的设置实现	vecA 和 vecB：两个向量	两个向量之间的欧几里得距离
randCent(dataSet,k)	随机生成初始的质心，这里是随机选取数据范围内的点	dataSet：数据集 k：质心的个数	初始的 *k* 个质心向量

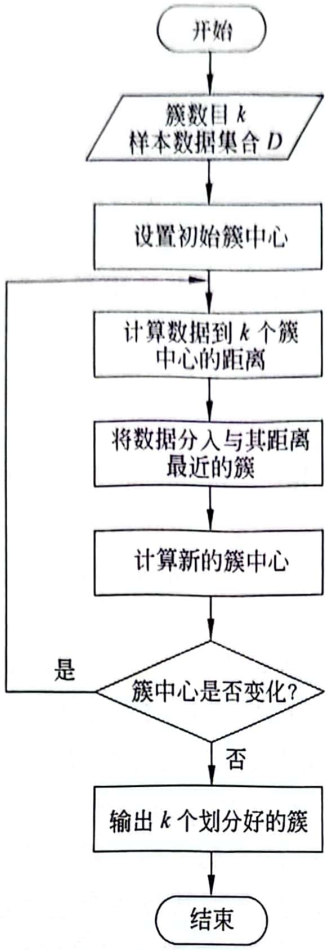

图 4.2 k-means 算法流程图

表 4.1 样本点的坐标

	x	y
P_1	0	0
P_2	1	2
P_3	3	1
P_4	8	8
P_5	9	10
P_6	10	7

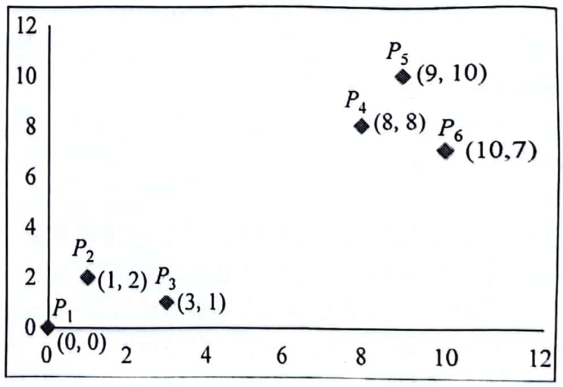

图 4.3 6 个点的分布

解析：从图 4.3 中可以看出，这 6 个点很明显地分为两个簇，P_1、P_2、P_3 为一个簇，P_4、P_5、P_6 为一个簇。现在开始使用 k-means 算法进行聚类。

首先，随机选择初始簇中心。这里随机选取 P_1 和 P_2 作为簇中心 P_a 和 P_b。

其次，计算其他几个点到初始簇中心的距离。P_3 到 P_1 的距离是 $\sqrt{10}=3.16$，P_3 到 P_2 的距离是 $\sqrt{(3-1)^2+(1-2)^2}=2.24$。那么 P_3 离 P_2 更近，将它们划分在一个簇中。