# 算法的实现

### 1、算法简介

算法的介绍：

K-means算法是聚类分析中使用最广泛的算法之一。它把n个对象根据他们的属性分为k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小

### 2、算法描述

输入：数据集D，中心点数目k

算法的流程（伪代码）：

**1 function** **K-Means**(输入数据，中心点个数K)

2 获取输入数据的维度Dim和个数N

3 随机生成K个Dim维的点

4 **while**(算法未收敛)

5 对N个点：计算每个点属于哪一类。

6 对于K个中心点：

7 1，找出所有属于自己这一类的所有数据点

8 2，把自己的坐标修改为这些数据点的中心点坐标

9 **end**

10 输出结果：

**11 end**

输出：聚类结果图及对应文件

### 3、参数列表

**Class KMeans (n\_clusters=8, n\_init=10, max\_iter=300, …)**

表 1参数列表

|  |  |
| --- | --- |
| Parameters | 参数介绍 |
| n\_clusters | int, optional, default: 8，  产生簇的数目 |
| n\_init | int, default: 10，  使用不同质心运行k-均值算法的次数 |
| max\_iter | int, default: 300  单次运行k-均值算法的最大迭代次数 |
| 。。。 | 。。。 |

### 4、方法列表

表 2方法列表

|  |  |
| --- | --- |
| Methods | 方法介绍 |
| \_init\_(a=1, b=true, c=0.01) | 使用初始化算法 |
| fit(X, y) | 使用训练集训练模型 |
| predict(X) | 使用测试集进行预测 |
| 。。。 | 。。。 |

### 5、方法参数列表

表 3 方法参数列表

|  |  |  |
| --- | --- | --- |
| Methods | 方法参数介绍 | 返回值 |
| \_init\_(a=1, b=true, c=0.01) |  |  |
| fit(X, y) | X: 类数组或稀疏矩阵（训练集数据）  y: X数据的对应值 |  |
| predict(X) | X: 类数组或稀疏矩阵（测试集数据） | labels: array  每个样本属于某个簇的索引 |
| 。。。 | 。。。 |  |

### 6、使用示例

**（1）问题描述**

假设我们的n个样本点分布在图中所示的二维空间。从数据点的大致形状可以看出它们大致聚为三个cluster，其中两个紧凑一些，剩下那个松散一些。我们的目的是为这些数据分组，以便能区分出属于不同的簇的数据。如果按照分组给它们标上不同的颜色，就是像图 1右边的图那样。

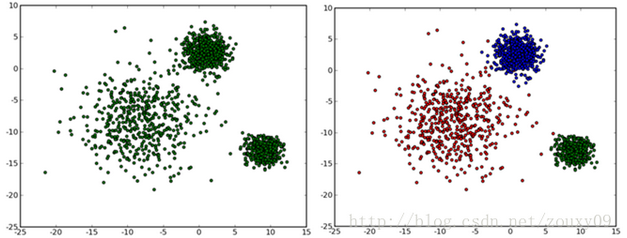


图 1 Kmeans算法问题描述

**（2）数据集**

测试数据集名为testSet.txt，数据为二维数据，共80个样本，有4个类。

表4 数据集说明

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集特征** | 多变量 | **记录数** | 80 |
| **属性特征** | 实数，浮点型 | **属性数目** | 2 |
| **相关应用** | 分类、聚类 | **缺失值** | 无 |

**（3）实验设置**

直接对上述80条记录进行kmeans聚类，聚类数目k设置为4。

**（4）应用程序**

算法程序kmeans.py

**from** numpy **import** \*  
**import** matplotlib.pyplot **as** plt  
  
  
# calculate Euclidean distance  
**def euclDistance**(vector1, vector2):  
 **return** sqrt(sum(power(vector2 - vector1, 2)))  
# init centroids with random samples  
**def initCentroids**(dataSet, k):  
 numSamples, dim = dataSet.shape  
 centroids = zeros((k, dim))  
 **for** i **in** range(k):  
 index = int(random.uniform(0, numSamples))  
 centroids[i, :] = dataSet[index, :]  
 **return** centroids  
  
# k-means cluster  
**def kmeans**(dataSet, k):  
 numSamples = dataSet.shape[0]  
 # first column stores which cluster this sample belongs to,  
 # second column stores the error between this sample and its centroid  
 clusterAssment = mat(zeros((numSamples, 2)))  
 clusterChanged = True  
  
 ## step 1: init centroids  
 centroids = initCentroids(dataSet, k)  
  
 **while** clusterChanged:  
 clusterChanged = False  
 ## for each sample  
 **for** i **in** xrange(numSamples):  
 minDist = 100000.0  
 minIndex = 0  
 ## for each centroid  
 ## step 2: find the centroid who is closest  
 **for** j **in** range(k):  
 distance = euclDistance(centroids[j, :], dataSet[i, :])  
 **if** distance < minDist:  
 minDist = distance  
 minIndex = j  
  
 ## step 3: update its cluster  
 **if** clusterAssment[i, 0] != minIndex:  
 clusterChanged = True  
 clusterAssment[i, :] = minIndex, minDist\*\*2  
  
 ## step 4: update centroids  
 **for** j **in** range(k):  
 pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]  
 centroids[j, :] = mean(pointsInCluster, axis = 0)  
  
 **print** 'Congratulations, cluster complete!'  
 **return** centroids, clusterAssment  
  
# show your cluster only available with 2-D data  
**def showCluster**(dataSet, k, centroids, clusterAssment):  
 numSamples, dim = dataSet.shape  
 **if** dim != 2:  
 **print** "Sorry! I can not draw because the dimension of your data is not 2!"  
 **return** 1  
  
 mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']  
 **if** k > len(mark):  
 **print** "Sorry! Your k is too large! please contact Zouxy"  
 **return** 1  
  
 # draw all samples  
 **for** i **in** xrange(numSamples):  
 markIndex = int(clusterAssment[i, 0])  
 plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])  
  
 mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']  
 # draw the centroids  
 **for** i **in** range(k):  
 plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize = 12)  
  
 plt.show()

测试程序testKmeans.py

**from** numpy **import** \*  
**import** kmeans  
**import** time  
**import** matplotlib.pyplot **as** plt  
  
## step 1: load data  
**print** "step 1: load data..."  
dataSet = []  
fileIn = open('C:/Users/Administrator/Desktop/testSet.txt')  
**for** line **in** fileIn.readlines():  
 lineArr = line.strip().split('\t')  
 dataSet.append([float(lineArr[0]), float(lineArr[1])])  
  
## step 2: clustering...  
**print** "step 2: clustering..."  
dataSet = mat(dataSet)  
k = 4  
centroids, clusterAssment = kmeans.kmeans(dataSet, k)  
  
## step 3: show the result  
**print** "step 3: show the result..."  
kmeans.showCluster(dataSet, k, centroids, clusterAssment)

**（5）运行结果**

程序运行之后80条记录被分为了4大类，不同的类别用不同的颜色和形状来区分，程序运行后的坐标图如图 2所示。

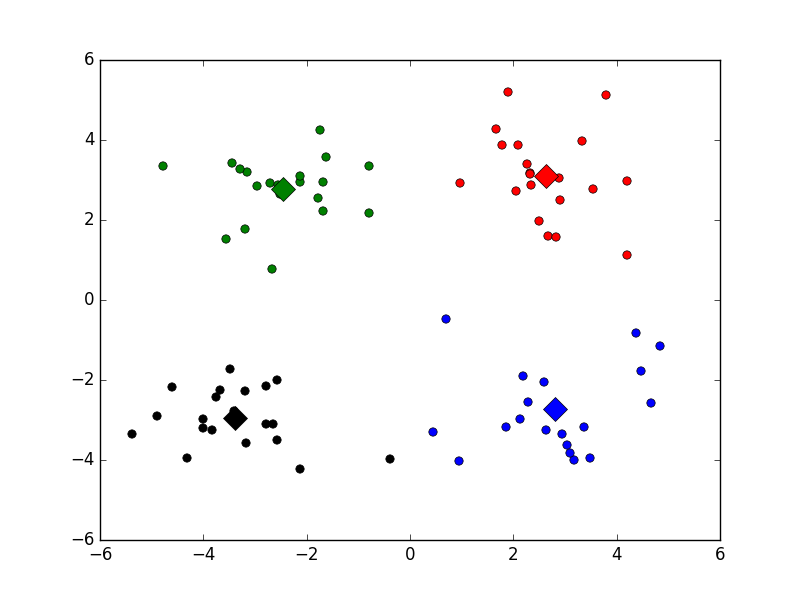


图 2算法运行后坐标图