**哈尔滨工业大学**

**数据挖掘理论与算法实验报告**

**(2015年度秋季学期)**

**课程编码** S1300019C

**授课教师 邹兆年**

**学生姓名 白辰甲**

**学 号 15S003002**

**学 院 计算机科学与技术**

**一、实验内容**

**1.问题：**使用层次聚类方法对数据进行聚类。

**2.目标：**使用层次聚类方法，从个体点作为簇开始，相继合并两个最接近的簇，直至只剩下一个簇。这样构建出了聚类的树形结构。根据用户输入的希望得到的簇的个数，可以在达到指定个数时终止聚类的执行。

**3.算法：**

|  |  |
| --- | --- |
|  | 基本凝聚层次聚类 |
| 1 | 计算邻近度矩阵 |
| 2 | Repeat： |
| 3 | 合并最接近的两个簇 |
| 4 | 更新邻近性矩阵，以反映新的簇与原来簇的邻近性 |
| 5 | Until 仅剩下一个簇 |

**4.期望结果：**

根据用户指定的参数N，层次聚类算法在当前有N个簇时停止聚类，得到N个簇。

**二、实验设计**

**1.程序流程图**

如右图所示

**2.数据结构**

（1）数据点

数据点用list中的二维元组来存储：

|  |
| --- |
| points = []  for line in fp.readlines():  line = line.strip()  point = re.split(r'\s+', line)  points.append((float(point[0]),float(point[1]))) |

（2）组标号

标号list存储，序号代表数据点的标号，值代表该数据点所属的类别。初始化时每个数据点都是一个簇，簇标号标记为自身的序号。

|  |
| --- |
| # 初始化簇，存储每个簇所属的类标号  group = []  for index in range(len(points)):  group.append(index) |

（3）距离矩阵

距离矩阵用字典dict存储，字典的键由两个点的标号连接而成，字典的值这两个点之间的欧式距离。可以通过有序字典将字典按照欧式距离大小排序，则每次可以提取出当前距离最小的两个数据点。

|  |
| --- |
| # 初始化两点间的距离矩阵，用字典表示，键为两个点的序号，值为两个点之间的距离  dis\_matrix = {}  for idx1, point1 in enumerate(points): # 顺序遍历，返回点的序号和坐标  for idx2, point2 in enumerate(points):  if idx1 < idx2:  dis = sqrt(pow((point1[0] - point2[0]), 2) + pow((point1[1] - point2[1]), 2))  keystr = str(idx1) + '#' + str(idx2)  dis\_matrix[keystr] = dis  # 对字典按照距离大小排序 按值排序  dis\_matrix = collections.OrderedDict(sorted(dis\_matrix.items(), key=lambda t: -t[1])) |

**3.关键函数及说明**

（1）读取测试数据集

对测试数据进行简单预处理后存储在Points中，每个数据点的序号用来代表该数据点。

|  |
| --- |
| fp = open("dataset2.txt")  points = []  for line in fp.readlines():  line = line.strip()  point = re.split(r'\s+', line)  points.append((float(point[0]), float(point[1]))) |

（2）初始化类标号

对数据集中的每个数据点，初始化类别为该数据点自身的标号，即每个数据初始化为一个簇。

|  |
| --- |
| group = []  for index in range(len(points)):  group.append(index) |

（3）初始化距离矩阵

构建距离矩阵，距离矩阵用字典存。键代表两个点，值代表着两个点之间的距离。注意，距离矩阵为对称矩阵，因此，只需要计算该矩阵规模一般的距离即可。

|  |
| --- |
| # 初始化两点间的距离矩阵，用字典表示，键为两个点的序号，值为两个点之间的距离  dis\_matrix = {}  for idx1, point1 in enumerate(points): # 顺序遍历，返回点的序号和坐标  for idx2, point2 in enumerate(points):  if idx1 < idx2:  dis = sqrt(pow((point1[0] - point2[0]), 2) + pow((point1[1] - point2[1]), 2))  keystr = str(idx1) + '#' + str(idx2)  dis\_matrix[keystr] = dis  # 对字典按照距离大小排序 按值排序  dis\_matrix = collections.OrderedDict(sorted(dis\_matrix.items(), key=lambda t: -t[1])) |

（4）层次聚类（min）构建层次聚类树

循环进行，直到当前簇的个数等于用户指定的簇的个数则停止。

在循环体中，首先获取当前距离最小的两个点，如果这两个点当前处于不同的类，如分别属于cluster1和cluster2，则将cluster2簇中所有点的类标号标记为cluster1，这样便完成了簇的合并。完成合并当前簇的个数减1。

当前group中有多少个不同的元素则代表了当前有多少个簇，同一个簇中点其值相同。如果达到了用户指定的要求则树停止生长，输出。

|  |
| --- |
| group\_num = len(group)  while True:  keystr, dis = dis\_matrix.popitem() # 获取当前dis最小的点  keystr\_list = re.split(r'#', keystr)  point1\_index = int(keystr\_list[0])  point2\_index = int(keystr\_list[1]) # 两个点的序号  # 如果它们不在同一个簇中则合并  if group[point1\_index] != group[point2\_index]:  group1 = group[point1\_index]  group2 = group[point2\_index]  for val in range(len(group)): # 将当前标号为第二组时标记为第一组  if group[val] == group2:  group[val] = group1  group\_num -= 1    #对当前组内还有多少个不同的元素进行统计 有多少个不同元素就代表有多少组  commonlist = collections.Counter(group).most\_common()    if len(commonlist) == n: # 如果达到了用户指定的簇的个数则停止层次聚类  break |

（5）将每个点分配到指定的簇中

首先统计group中有多少个不同的元素，则代表有多少个不同的簇。每个簇用首先初始化为空。然后将group中不同标号的元素放到指定的簇中。

|  |
| --- |
| # 对点进行分组，分入指定的簇中  set\_group = set(group)  set\_group = list(set\_group)  resultlist = []  for i in range(len(set\_group)):  resultlist.append([])  for point\_ind in range(len(group)):  temp = group[point\_ind] # 该点所属类别  temp1 = set\_group.index(temp) # 该类别对应的set\_group的序号  resultlist[temp1].append(point\_ind) # 将该点存入相应的list中去 |

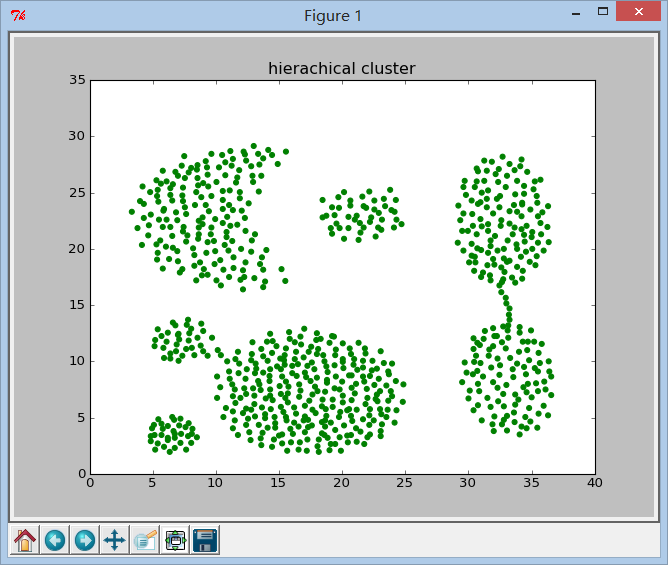
（6）绘图

每个簇中的点用不同颜色绘制散点图。

**三、测试数据**

**1.数据集描述**

原始数据集的分布如下图所示，其中每个样本包含其X坐标和Y坐标：



**2.数据样例（共788个点）**

|  |
| --- |
| 15.550000 28.650000 |
| 14.900000 27.550000 |
| 14.450000 28.350000 |
| 14.150000 28.800000 |
| 13.750000 28.050000 |
| 13.350000 28.450000 |
| 13.000000 29.150000 |

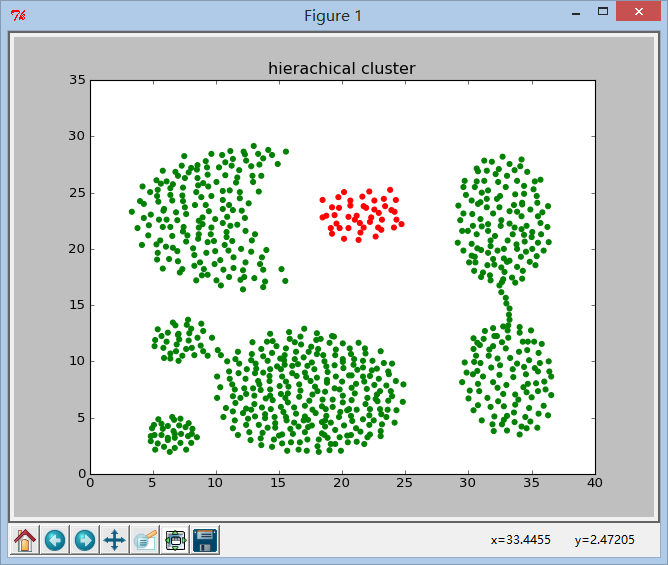
**3.预处理过程**

数据质量良好，只需要将两个方向的坐标分割即可。

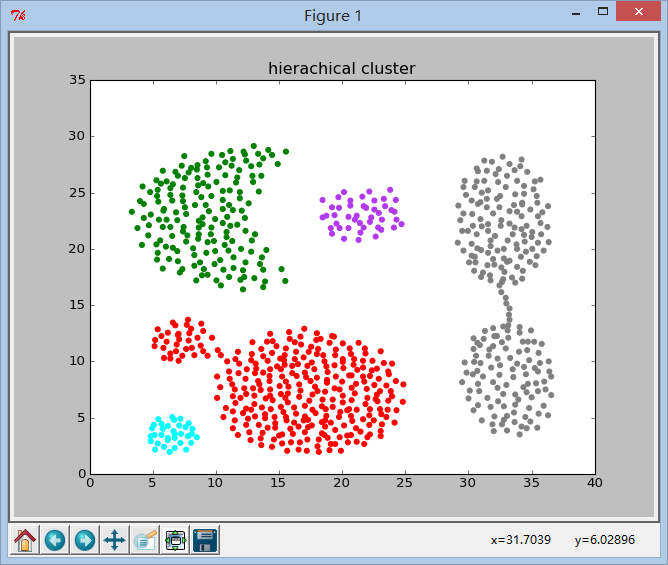
|  |
| --- |
| for line in fp.readlines():  line = line.strip()  point = re.split(r'\s+', line)  #print point[:]  points.append((float(point[0]), float(point[1]))) |

**四、实验结果**

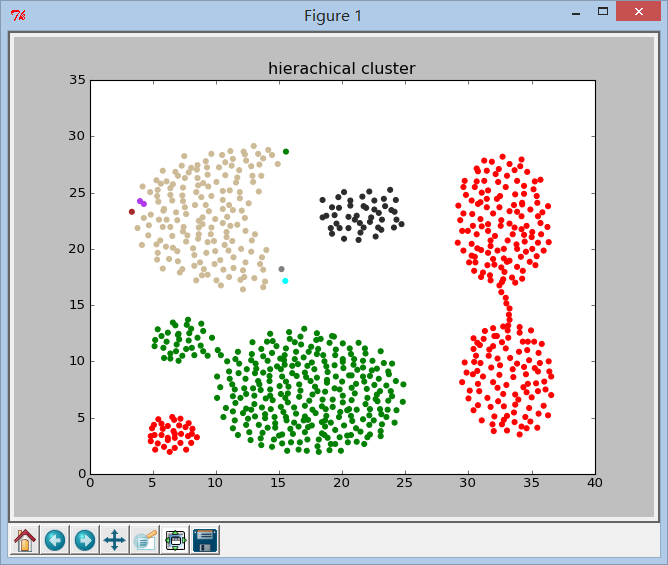
1.当N = 2时：



2.当N = 5时：



3.当N = 10时：



**五、遇到的困难及解决方法、心得体会**

层次聚类开销较大，算法从单个点构成的簇开始聚类，直到聚类成1个簇停止。但与K均值不同，K均值的初始质心位置可能对最终的聚类效果影响较大，但层次聚类不存在这样的问题。用户输入的希望得到的聚类个数N只是在达到簇个数达到N时截断了聚类树的生长，多次运行结果一致。