第5章

聚　　类

CHAPTER 5

聚类(clustering)是一种无监督的学习方法,它是按照某一个特定的标准(比如距离),把一个数据集分隔成不同的类或簇,使得同一个簇内的数据对象的相似性尽可能大,同时不在同一个簇内的数据对象的差异性也尽可能地大。与分类算法类似,它也是要确定一个事物的类别,即把相似的对象归为一类;但它与分类的区别是,聚类不知道真实的样本类别,分类是利用已知的样本类别训练学习器来预测未知样本的类别。聚类分析常被用于图像分割、离群点检测,广泛应用于图像处理、推荐系统、数据挖掘等领域。

5.1 聚类算法简介

聚类分析是一种无监督学习的算法,它是将数据对象按照相似性划分为多个子集的过程,每个子集被称为一个"簇"(cluster)。同一个簇中的数据相似性高,不同簇中的数据相似性低。在利用无监督学习方法划分数据时,数据是没有类别标记的,它需要利用样本间的相似性去划分类别,而相似性是根据样本间的距离进行度量的。

聚类任务的形式化描述如下。

假设样本集合为 $D=\{x_1,x_2,\cdots,x_N\}$,通过聚类把样本划分到不同的簇,使得具有相似特征的样本在同一个簇中,不具有相似特征的样本在不同的簇中,最终形成 k 个不同簇 $C=\{C_1,C_2,\cdots,C_k\}$,若各个簇互不相交,即对任意两个簇 $C_i \cap C_j=\varnothing$,则称为硬聚类,否则称为软聚类。

5.1.1 聚类算法的分类

聚类本质上是集合划分问题。它的基本原则是使簇内的样本尽可能相似,通常的做法是根据簇内样本之间的距离,或是样本点在数据空间中的密度来确定。对簇的不同定义可以得到各种不同的聚类算法。聚类分析的算法可以分为基于划分的方法(partitioning method)、基于层次的方法(hierarchical-based method)、基于密度的方法(density-based method)、基于网格的方法(grid-based method)、基于模型的方法(model-based method)。

1. 基于划分的方法

基于划分的方法是基于距离作为判断依据,将数据对象划分为不重叠的簇,使每个数据对象属于且只属于一个簇。首先要确定这些样本点最后聚成几类,然后挑选几个样本点作为初始中心点,通过不断迭代,直到达到"类(簇)内的样本点都足够近,类(簇)间的样本点都足够远"的目标。基于划分的距离算法有 K-means、K-medoids、kernel K-means 等算法。

2. 基于层次的方法

基于层次的聚类可分为两种:凝聚法和分裂法。凝聚法采用的是一种自底向上的方法,从最底层开始,每一次通过合并最相似的聚类来形成上一层次中的聚类,当全部数据都合并到一个簇或者达到某个终止条件时,算法结束。分裂法采用的是一种自顶向下的方法,从一个包含全部样本数据的簇开始,逐层分裂为若干簇,每个簇继续不断地往下分裂,直到每个簇中仅包含一个样本数据。

3. 基于密度的方法

在基于密度的聚类方法中,簇被看作是由低密度区域分隔开来的高密度对象区域。基于密度的聚类方法定义了领域的范围,当邻近区域的密度超过某个阈值,就继续聚类,即某区域内的对象个数超过一个给定范围,则将其添加到簇中。基于密度的聚类方法可以对不规则形状的数据样本点进行聚类,同时过滤噪声数据的效果比较好。DBSCAN(Density-

Based Spatial Clustering of Applications with Noise)就是典型的代表。

4. 基于网格的方法

基于网格的聚类方法将数据空间划分为由若干有限的网格单元(Cell)组成的网格结构,将数据对象集映射到网格单元中,所有聚类操作都在该结构上进行。该方法的处理与数据对象的个数无关,只依赖于每个量化空间中每一维上的单元数,处理速度快,但算法效率的提高是以聚类结果的准确率为代价的,经常与基于密度的聚类算法结合使用。

5. 基于模型的方法

基于模型的方法包括基于概率模型的方法和基于神经网络模型的方法。概率模型主要指概率生成模型,同一"类"的数据属于同一种概率分布,即假设数据是根据潜在的概率分布生成的。高斯混合模型(Gaussian Mixture Models,GMM)就是最典型、常用的基于概率模型的聚类方法。自组织映射(Self Organized Maps,SOM)则是一种常见的基于神经网络模型的方法。

5.1.2　距离度量的方法

无论是基于划分的聚类,还是基于层次的聚类,核心问题都是度量两个样本之间的距离,常用的距离度量方法有闵可夫斯基距离、马氏距离、汉明距离、夹角余弦等。

1. 闵可夫斯基距离

闵可夫斯基距离(Minkowski distance)将样本看作高维空间中的点进行距离度量。对于 n 维空间中任意两个样本点 $\boldsymbol{P}=(x_1,x_2,\cdots,x_n)^{\mathrm{T}}$ 和 $\boldsymbol{Q}=(y_1,y_2,\cdots,y_n)^{\mathrm{T}}$,$\boldsymbol{P}$ 和 $\boldsymbol{Q}$ 的闵可夫斯基距离的定义为

$$d_{\boldsymbol{PQ}}=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}} \tag{5-1}$$

其中,$p\geqslant 1$,$|x_i-y_i|^p$ 为 x_i-y_i 的 p 范数。当 $p=1$ 时,有

$$d_{\boldsymbol{PQ}}=\sum_{i=1}^{n}|x_i-y_i| \tag{5-2}$$

此时,$d_{\boldsymbol{PQ}}$ 的取值就是两个点的绝对值之和,称为曼哈顿距离(Manhattan distance)。几何意义就是沿水平方向从 $\boldsymbol{P}$ 到 $\boldsymbol{Q}$ 的距离。

当 $p=2$ 时,$\boldsymbol{P}$ 和 $\boldsymbol{Q}$ 的距离为

$$d_{\boldsymbol{PQ}}=\left(\sum_{i=1}^{n}|x_i-y_i|^2\right)^{\frac{1}{2}} \tag{5-3}$$

此时,$d_{\boldsymbol{PQ}}$ 就表示二维空间中两个点 $\boldsymbol{P}$ 和 $\boldsymbol{Q}$ 之间的直线距离,称为欧几里得距离或欧氏距离(Euclidean distance)。

2. 马氏距离

与欧氏距离、曼哈顿距离一样,马氏距离(Mahalanobis distance)常被用于评定数据之

间的相似度指标,它可以看作是欧氏距离的修正,修正了欧氏距离中各维度尺度不一致且相关的问题。单个数据点的马氏距离的定义为

$$d_x = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{5-4}$$

数据点 $\boldsymbol{P}$ 和 $\boldsymbol{Q}$ 之间的马氏距离定义为

$$d_{\boldsymbol{PQ}} = \sqrt{(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{y})} \tag{5-5}$$

其中,$\boldsymbol{\Sigma}$ 是多维随机变量的协方差矩阵,$\boldsymbol{\mu}$ 为样本均值。当样本的各个特征向量相互独立时,协方差是单位向量,马氏距离就成了欧氏距离。

3. 汉明距离

汉明距离(Hamming distance)需要将处理的样本数据转换为 0 和 1 表示的二进制串,样本中各分量的取值只能是 0 或 1,例如字符串"1110"与"1001"之间的汉明距离为 3。对于任意样本特征 x 和 y,有 $x,y\in\{0,1\}$,其汉明距离为

$$d_{ij} = \sum_{i=1}^{n} 1\{x_i \neq y_i\} \tag{5-6}$$

汉明距离常应用在信息论、编码理论、密码学等领域。

4. 夹角余弦

夹角余弦(cosine)度量将样本看成是高维空间中的向量进行度量,度量方法就是计算两个向量间的余弦夹角。对于任意两个 n 维样本 $\boldsymbol{P}=(x_{i1},x_{i2},\cdots,x_{in})$ 和 $\boldsymbol{Q}=(y_{i1},y_{i2},\cdots,y_{in})$,其夹角余弦为

$$\cos_{\boldsymbol{PQ}} = \frac{\sum_{k=1}^{n} x_{ik}y_{ik}}{\sqrt{\sum_{k=1}^{n} x_{ik}^2}\sqrt{\sum_{k=1}^{n} y_{ik}^2}} \tag{5-7}$$

当 $n=2$ 时,夹角余弦计算的就是二维空间中两条直线的夹角余弦值。夹角余弦的取值范围为[-1,1]。夹角余弦值越大,表示两个向量的夹角越小;夹角余弦越小,表示两个向量的夹角越大。当两个向量的方向重合时,夹角余弦取最大值 1;当两个向量的方向完全相反时,夹角余弦取最小值-1。


视频讲解

5.2 K-means 聚类

K-means 聚类,也称为 k 均值聚类,是聚类分析中使用最广泛的聚类算法之一。K-means 算法的思想很简单,对于给定的样本集,按照样本之间的距离大小,将样本集划分为 k 个簇,使簇内的样本点的连接尽可能紧密,不同簇内的样本点的距离尽量大。

5.2.1 K-means 聚类算法的思想

假设簇划分为$(C_1,C_2,\cdots,C_k)$,则目标就是最小化平方误差

$$E=\sum_{i=1}^{k}\sum_{x\in C_i}\|\boldsymbol{x}-\boldsymbol{\mu}_i\|_2^2 \tag{5-8}$$

其中，$\boldsymbol{\mu}_i$ 为聚类簇 C_i 的均值向量，也称为质心(centroid)，其计算公式为

$$\boldsymbol{\mu}_i=\frac{1}{|C_i|}\sum_{x\in C_i}\boldsymbol{x} \tag{5-9}$$

直接求最小误差是一个 NP 难问题，因此只能采用启发式的迭代方法求解。

k 均值聚类算法的描述如下：

输入：训练数据集 $D=\{x_1,x_2,\cdots,x_N\}$，聚类个数 k。

过程：

(1) 从 D 中随机选择 k 个样本作为初始的均值向量：$\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\cdots,\boldsymbol{\mu}_k$。

(2) 重复执行以下过程，直至当前均值向量不再更新：

① 令 $C_i=\varnothing$，其中 $1\leqslant i\leqslant k$。

② 对于 $i=1,2,\cdots,N$，选择每个样本 $\boldsymbol{x}_i$ 与各均值向量 $\boldsymbol{\mu}_j$ ($1\leqslant j\leqslant k$)的距离：$\mathrm{dist}_{ij}=\|\boldsymbol{x}_i-\boldsymbol{\mu}_j\|_2^2$，根据离均值距离最小的 $\boldsymbol{x}_i$ 确定其聚类标记：$\lambda_i=\underset{j\in\{1,2,\cdots,k\}}{\operatorname{argmin}}\mathrm{dist}_{ij}$，将样本 $\boldsymbol{x}_i$ 划入相应的聚类 $C_{\lambda_i}=C_{\lambda_i}\cup\{\boldsymbol{x}_i\}$。

③ 对于 $i=1,2,\cdots,k$，计算新的均值向量：$\boldsymbol{\mu}_i'=\frac{1}{|C_i|}\sum_{\boldsymbol{x}\in C_i}\boldsymbol{x}$，如果新的均值向量与之前的均值向量不相等，则更新，即 $\boldsymbol{\mu}_i'=\boldsymbol{\mu}_i$，否则不更新。

输出：聚类划分 $C=\{C_1,C_2,\cdots,C_k\}$。

1. k 均值聚类算法的计算过程

下面以西瓜数据集为例，说明 k 均值聚类算法的计算过程，并对该过程进行算法验证。这里的西瓜数据集共 30 条记录，包含密度和含糖率两个特征，如表 5-1 所示。

表 5-1　西瓜数据集

序号	密度	含糖率	序号	密度	含糖率	序号	密度	含糖率
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

下面使用聚类算法演示聚类过程：

(1) 假设聚类个数为 $k=3$，首先随机选取 3 个样本 $\boldsymbol{x}_3=(0.634,0.264)$、$\boldsymbol{x}_8=(0.437,0.211)$、$\boldsymbol{x}_9=(0.666,0.091)$作为初始的均值向量，即 $\boldsymbol{\mu}_1=\boldsymbol{x}_3$、$\boldsymbol{\mu}_2=\boldsymbol{x}_8$、$\boldsymbol{\mu}_3=\boldsymbol{x}_9$，分别对应于 3 个聚类 C_1、C_2、C_3 中的均值向量，初始时每个聚类中的元素为空。

(2) 考察样本 $\boldsymbol{x}_1=(0.697,0.460)$,它与当前的均值向量 $\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$、$\boldsymbol{\mu}_3$ 距离(欧氏距离)分别为 0.206、0.360、0.370,因此 $\boldsymbol{x}_1$ 被划入聚类 C_1 中,类似地,对 $\boldsymbol{x}_2,\boldsymbol{x}_3,\cdots,\boldsymbol{x}_{30}$ 所有样本都执行类似的过程,将每个样本进行了划分,故有。

$$C_1=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3,\boldsymbol{x}_4,\boldsymbol{x}_5,\boldsymbol{x}_{14},\boldsymbol{x}_{21},\boldsymbol{x}_{22},\boldsymbol{x}_{25},\boldsymbol{x}_{26},\boldsymbol{x}_{27},\boldsymbol{x}_{29}\}$$

$$C_2=\{\boldsymbol{x}_6,\boldsymbol{x}_7,\boldsymbol{x}_8,\boldsymbol{x}_{10},\boldsymbol{x}_{11},\boldsymbol{x}_{12},\boldsymbol{x}_{15},\boldsymbol{x}_{18},\boldsymbol{x}_{19},\boldsymbol{x}_{20},\boldsymbol{x}_{23},\boldsymbol{x}_{24},\boldsymbol{x}_{28},\boldsymbol{x}_{30}\}$$

$$C_3=\{\boldsymbol{x}_9,\boldsymbol{x}_{13},\boldsymbol{x}_{16},\boldsymbol{x}_{17}\}$$

(3) 根据得到的 C_1、C_2、C_3 更新均值向量:

$$\boldsymbol{\mu}_1=(0.660,0.349)$$

$$\boldsymbol{\mu}_2=(0.384,0.261)$$

$$\boldsymbol{\mu}_3=(0.654,0.0993)$$

2. k 均值聚类的算法实现

根据前面的 k 均值算法思想实现 k 均值算法,并在西瓜数据集上验证其正确性。这里同样使用表 5-1 所示的西瓜数据集。

k 均值聚类的算法实现如下。

```
import pandas as pd
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
def dist_eclud(v1, v2):
    return sqrt(sum(power(v1 - v2, 2)))
def update_clusters(k,mu,X,y_label):
    for i in range(X.shape[0]):
          min_dist = float('inf')
          for index in range(k):
                dist = dist_eclud(mu[index], X[i])
                if dist < min_dist:
                     min_dist = dist
                     y_label[i] = index
    return y_label

def update_centroids(k,mu,X,y_label):
    for i in range(k):
          sum = np.array([0.0,0.0])
          num = np.sum(y_label == i)
          cluster_index,label = np.where(y_label == i)
          print("cluster_index:",list(cluster_index))
          for j in cluster_index:
                     sum = sum + X[j]
          print("sum:",sum)
          centroid = sum/num
          centroid = np.mean(X[cluster_index],axis = 0)
          mu[i] = centroid
    return mu

def show(dataset, k, centroids, clusters):
    num_samples, dim = dataset.shape
```

```
    marker = ['or', 'ob', 'og', 'ok']
    marker2 = ['*r', '*b', '*g', '*k']
    for i in range(num_samples):
        mark_index = int(clusters[i])
        plt.plot(dataset[i, 0], dataset[i, 1], marker[mark_index])
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], marker2[i], markersize = 10)
    plt.xlim(0.1, 0.9)                                  #把 x 轴的刻度范围设置为 0.1～0.9
    plt.ylim(0, 0.8)                                    #把 y 轴的刻度范围设置为 0～0.8
    plt.xlabel('密度')
    plt.ylabel('含糖率')
    plt.rcParams['font.sans-serif'] = ['SimHei']        #显示中文
    plt.rcParams['axes.unicode_minus'] = False

if __name__ == '__main__':
    k = 3
    dataTrain = pd.read_csv("xiguadata.csv")
    dataTrain = [
        [0.697, 0.460], [0.774, 0.376],[0.634, 0.264],[0.608, 0.318],
        [0.556, 0.215],[0.403, 0.237],[0.481,0.149],[0.437, 0.211],
        [0.666, 0.091],[0.243, 0.267],[0.245, 0.057],[0.343, 0.099],
        [0.639, 0.161],[0.657, 0.198],[0.360, 0.370],[0.593, 0.042],
        [0.719, 0.103],[0.359,0.188],[0.339,0.241],[0.282,0.257],
        [0.748, 0.232],[0.714,0.346],[0.483,0.312],[0.478,0.437],
        [0.525, 0.369],[0.751,0.489],[0.532,0.472],[0.473,0.376],
        [0.725, 0.445],[0.446,0.459]]
    dataTrain = np.array(dataTrain)
    y_label = np.zeros((dataTrain.shape[0],1))

    mu1 = np.array([0.403, 0.237])
    mu2 = np.array([0.343, 0.099])
    mu3 = np.array([0.478,0.437])
    #mu = np.zeros((k,dataTrain.shape[1]))
    mu = np.array([[0.403, 0.237],[0.343, 0.099],[0.478,0.437]])
    iters = 4
    for iter in range(iters):
      y_label = update_clusters(k,mu,dataTrain,y_label)
      mu = update_centroids(k,mu,dataTrain,y_label)
      print("mu = ",mu)
      plt.subplot(2, 2, iter + 1, frameon = True)       #两行、两列的子图
      t = '第' + str(iter + 1) + '次迭代后'
      plt.legend(title = t)
      show(dataTrain, k, mu, y_label)
    plt.show()
```

利用以上 K-means 算法对西瓜集数据进行聚类，经过 4 次迭代后，每一次聚类结果其散点图及聚类结果如图 5-1 所示。

图 5-1 中，不同颜色的五角星表示各样本的聚类中心，圆点表示各类样本数据。

针对三文鱼和鲈鱼数据的聚类，仍然使用 K-means 聚类的算法实现如下：

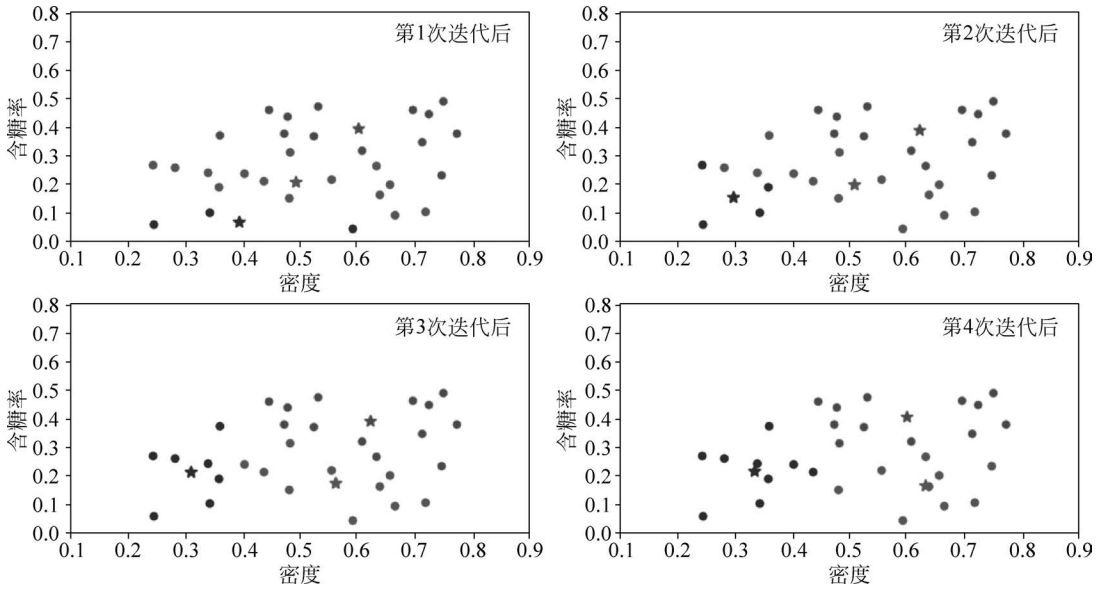

图 5-1　利用 K-means 聚类算法的迭代结果(见彩插)

```
from numpy import *
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
import numpy as np
import xlrd
def load_dataset(file_name):
    xlbook = xlrd.open_workbook(file_name)
    sheet = xlbook.sheet_by_index(3)                    #索引的方式,从 0 开始
    sheet = xlbook.sheet_by_name('joint_normal')        #名字的方式
    nrows = sheet.nrows                                 #行
    ncols = sheet.ncols                                 #列
    fish = []
    y = []
    for i in range(1, nrows):
        v1 = sheet.cell(i, 0).value                     #获取 i 行 3 列的表格值
        v2 = sheet.cell(i, 1).value                     #获取 i 行 3 列的表格值
        print(v1, v2)
        fish.append([v1, v2])
        y.append(int(sheet.cell(i,2).value))
    train0 = fish[0:500]
    train1 = fish[1000:1500]
    train = train0 + train1
    y0 = y[0:500]
    y1 = y[1000:1500]
    y = y0 + y1
    return train,y

#利用欧几里得距离公式计算两个向量的距离
def dist_eclud(v1, v2):
    return sqrt(sum(power(v1 - v2, 2)))
```

```
#随机生成初始的质心
def make_rand_centroids(dataset, k):
    n = shape(dataset)[1]
    centroids = mat(zeros((k, n)))
    for i in range(n):
        min_data = min(dataset[:, i])
    data_range = float(max(array(dataset)[:, i]) - min_data)
    centroids[:, i] = min_data + data_range * random.rand(k, 1)
    return centroids

#初始时随机选择 k 个质心
def select_rand_centroids(X, k):
    n,cols = shape(X)
    centroids = np.zeros((k, cols))
    for i in range(k):
        centroid = X[np.random.choice(range(n))]
        centroids[i] = centroid
    return centroids

def divide_closest_centroid(feature,centroids):
    min_index = 0
    min_dist = float('inf')
    for index,centroid in enumerate(centroids):
        dist = dist_eclud(feature,centroid)
        if dist < min_dist:
            min_index = index
            min_dist = dist
    return min_index

def init_cluster(centroids,k,X):
    clusters = [[] for i in range(k)]
    for i, feature in enumerate(X):
        centroid_i = divide_closest_centroid(feature,centroids)
        clusters[centroid_i].append(i)
    return clusters

def update_centroids(clusters,k,X):
    cols = np.shape(X)[1]
    centroids = np.zeros((k,cols))
    pre_clusters = clusters
    for i,cluster in enumerate(clusters):
        centroid = np.mean(X[cluster],axis = 0)
        centroids[i] = centroid
    return centroids

def k_means(X, k):
    m = shape(X)[0]
    centroids = select_rand_centroids(X,k)
    iter = 0
    while iter < 10:
        clusters = init_cluster(centroids,k,X)
        centroids = update_centroids(clusters,k,X)
        iter += 1
```

```
    y_pre = np.zeros(np.shape(X)[0])
    for i,cluster in enumerate(clusters):
        for x_index in cluster:
            y_pre[x_index] = i
    return centroids, y_pre

def show(X, k, centroids, clusters):
    num_samples, dim = X.shape
    marker = ['or', 'sb', '+g', 'pk', '^r', '+r']
    marker2 = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b']
    for i in range(num_samples):
        mark_index = int(clusters[i])
        plt.plot(dataSet[i, 0], dataSet[i, 1], marker[mark_index])
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], marker2[i], markersize = 12)
    legend_label = 'k = ' + str(k)
    plt.legend(title = legend_label)
    plt.xlabel('长度')
    plt.ylabel('亮度')
    plt.rcParams['font.sans-serif'] = ['SimHei']        #显示中文
    plt.rcParams['axes.unicode_minus'] = False
    plt.show()

if __name__ == '__main__':
    X_train,y = load_dataset(r'fish.xls')
    data_mat = mat(X_train)
    cluster_number = 3
    my_centroids, cluster_labels = k_means(data_mat, cluster_number)
    show(dataMat, cluster_number, my_centroids, cluster_labels)
```

这里聚类结束条件设置为迭代次数,有可能循环结束时,未达到较好的聚类效果;若将迭代次数设置得过大,则会使程序运行时间较长。为了在合适的时间内达到较好的聚类效果,可将聚类中心是否改变作为聚类的结束条件。为此需要修改 update_centroids()的实现,具体代码如下。

```
def update_centroids(clusters,centroids,k,X):
    cols = np.shape(X)[1]
    centroids = np.zeros((k,cols))
    pre_clusters = centroids
    no_same = False
    for i,cluster in enumerate(clusters):
        centroid = np.mean(X[cluster],axis = 0)
        centroids[i] = centroid
        if centroids[i].any() != pre_clusters[i].any():
            no_same = True
            break
    return centroids,no_same
```

当 $k=2$ 和 $k=3$ 时,聚类结果如图 5-2 所示。

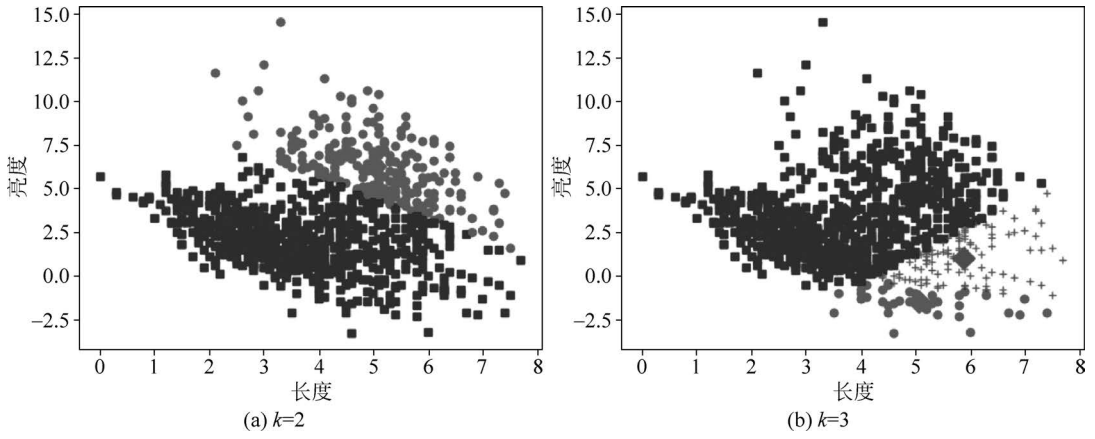

图 5-2　三文鱼和鲈鱼簇为 2 和 3 时的聚类结果(见彩插)

5.2.2　通过调用库函数实现聚类

1. K-means 聚类的函数原型及参数介绍

scikit-learn 模块提供的 K-means 聚类函数原型如下：

```
KMeans(n_clusters = 8, init = 'k - means++', n_init = 10, max_iter = 300, tol = 0.0001,
precompute_distances = 'auto', verbose = 0, random_state = None,
copy_x = True, n_jobs = None, algorithm = 'auto')
```

主要参数说明如下。

(1) n_clusters：整型，生成的聚类数，默认值为 8。

(2) init：指定初始化聚类中心的方法，有 3 个可选值：k-means++、random 或 ndarray。默认值为 k-means++。

- k-means++：用一种特殊的方法选定初始聚类，可加速迭代过程的收敛。
- random：随机从训练数据中选取 k 个样本作为初始质心。
- ndarray：指定一个形如 k×n_features 的数组作为初始质心。

(3) n_init：整型，使用不同的初始聚类中心执行算法的次数，以选择最好的聚类结果，默认值为 10。

(4) max_iter：整型，执行一次 K-means 算法所进行的最大迭代数，默认值为 300。

(5) tol：float 类型，最小容忍误差，与 inertia 结合来确定收敛条件，默认值为 1×10^{-4}。

(6) precompute_distances：提前计算样本距离，计算速度更快但占用更多内存。有 3 个可选值：auto、true 或 false。

- auto：如果样本数×聚类数>12MB，则不提前计算样本距离。
- true：总是预先计算样本距离。
- false：永远不预先计算样本距离。

(7) n_jobs：整型，指定计算所用的进程数。若值为−1，则用所有的 CPU 进行运算；若值为 1，则不进行并行运算；若值为−2，则用到的 CPU 数为总 CPU 数减 1。

(8) random_state：整型，NumPy. RandomState 类型或 None。用于初始化质心的生

成器(Generator)。如果值为整数,则确定一个 seed。默认值为 NumPy 的随机数生成器。

(9) copy_x:布尔型,默认值为 True。当设置 precomputing_distances 为 True 时,该参数才有效。如果此参数值设为 True,则原始数据不会被改变;如果参数设置为 False,则原始数据会改变。

(10) verbose:整型,默认值为 0。verbose 表示是否输出详细信息,0 表示不输出日志信息;1 表示每隔一段时间打印一次日志信息;大于 1 表示打印次数频繁。

(11) algorithm:可选参数为 auto、full、elkan,默认值为 auto。auto 表示采用的是 K-means 算法,full 表示采用的是经典的 EM-style 算法,elkan 是 elkan K-Means 算法,利用了两边之和大于第三边及两边之差小于第三边的三角形性质,来减少距离的计算。

常用属性说明如下。

- cluster_centers_:类别的均值向量。
- labels_:每个样本所属的类别标记。
- inertia_:每个样本与其各个簇的质心的距离之和。

常用方法说明如下。

- fit(X[,y]):K-means 聚类训练模型。
- fit_predictt(X[,y]):计算簇质心并预测每个样本的类别。
- transform(X):在 fit 的基础上,进行标准化,降维,归一化等操作。
- fit_transform(X[,y]):fit_transform 是 fit 和 transform 的组合,既包括了训练又包含了转换。
- predict(X):预测每个样本所属的簇类别标签。
- score(X[,y]):计算聚类误差。

2. 调用 K-means 聚类函数对鸢尾花进行聚类处理

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

iris = load_iris()
X_data = iris.data
km = KMeans(n_clusters = 3)
km.fit(X_data)
kc = km.cluster_centers_
y_kmeans = km.predict(X_data)

print(y_kmeans,kc)
print(kc.shape,y_kmeans.shape,X_data.shape)
markers = ['o','p','*','s','D']
colors = ['r','g','b','y']
for i,j in enumerate(km.labels_):
    plt.scatter(X_data[i, 0], X_data[i, 1],marker = markers[j],color = colors[j])
plt.scatter(kc[:,0],kc[:,1],color = 'r',s = 80)
plt.xlabel('花萼长度')
plt.ylabel('花萼宽度')
plt.title('K-means 在鸢尾花数据集上的聚类效果 k = 3')
plt.show()
```

程序运行结果如图 5-3 所示。

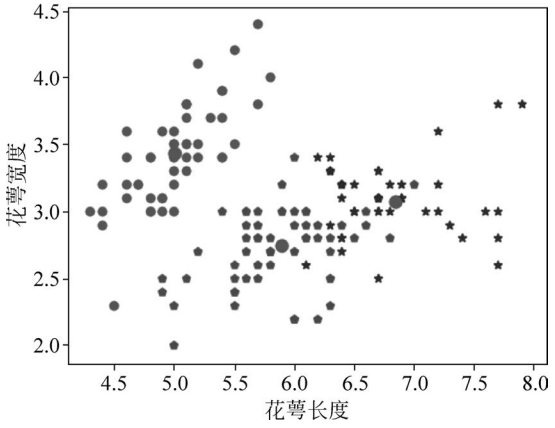

图 5-3　K-means 聚类函数在鸢尾花数据集上的运行结果($k=3$)

提　示

K-means 聚类函数运用了 Lloyd 算法，平均计算复杂度是 $O(knt)$，其中 n 是样本数量，t 是迭代次数。在最坏的情况下，计算复杂度为 $O(n^{\wedge}(k+2/p))$，其中 p 是特征个数。一般情况下，K-means 算法运算速度非常快，但是其局限性在于聚类结果是由特定初始值所产生的局部解。为了使聚类结果更准确，需要尝试使用不同的初始值反复实验。

3. 轮廓系数——K-means 评价方法

当文本类别未知时，可以选择轮廓系数作为聚类性能的评估指标。轮廓系数的取值范围为[−1,1]，取值越接近 1，则说明聚类性能越好；反之，取值越接近−1，则说明聚类性能越差。轮廓系数越大，表明簇内样本之间越紧凑，簇间距离越大。

下面通过对鸢尾花数据设置不同的 k 值，观察其轮廓系数的取值变化，确定对数据进行聚类时 k 的取值何时为最优。

```
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
silhouette_score_set = []
n = 15
for i in range(2,n):
    kmeans = KMeans(n_clusters = i,random_state = 100).fit(X_data)
    score = silhouette_score(X_data,kmeans.labels_)
    silhouette_score_set.append(score)
    plt.plot(range(2,n), silhouette_score_set,linewidth = 2,linestyle = '-')
    plt.show()
```

k 的取值从 2 到 14，对应的轮廓系数得分如下：

```
[0.681046169211746, 0.5528190123564091, 0.4980505049972867, 0.4887488870931048,
0.3648340039670018, 0.34750423280461507, 0.35745369258527043, 0.3203121081683388,
0.31784160872963685, 0.3151992769724403, 0.3026804038545623, 0.28457124261699057,
0.2889218914644372]
```

执行结果如图 5-4 所示。

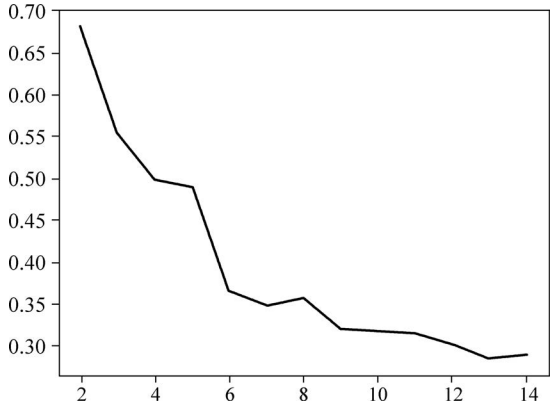

图 5-4 不同簇个数取值时的轮廓系数

轮廓系数就是看这条曲线的畸变程度,也就是斜率变化,变化快的部分就是分类的最佳选择。从图 5-4 中可以看出,在 2～3、5～6 两段之间的变化比较快,结合实际情况判断,还是分 3 类比较好。


视频讲解

5.3 基于密度的聚类——DBSCAN 聚类

基于密度的聚类假设聚类结构可通过样本分布的紧密程度来确定。同一样本中,样本之间是紧密相连的。根据样本之间的紧密程度,可将不同的样本划分到不同的聚类簇中。DBSCAN 就是一种著名的基于密度的聚类算法。

5.3.1 DBSCAN 算法的原理及相关概念

DBSCAN 通过一组邻域参数(ϵ,minPts)刻画样本分布的紧密程度,其中,ϵ 表示某一样本的邻域距离阈值,minPts 表示某一样本的距离为 ϵ 的邻域中样本个数的阈值。

给出样本数据集合 $D=(x_1,x_2,\cdots,x_N)$,相关概念描述如下:

(1) ϵ-邻域:对于任一样本 $x_j\in D$,其 ϵ-邻域包含的样本集是 D 中与 x_j 的距离不大于 ϵ 的样本,即 $N_\epsilon(x_j)=\{x_i\in D\mid \text{distance}(x_i,x_j)\leqslant\epsilon\}$,其样本集个数记作 $|N_\epsilon(x_j)|$。

(2) 核心对象:对于任一样本 $x_j\in D$,如果其 ϵ-邻域包含至少 minPts 个样本,即当 $|N_\epsilon(x_j)|\geqslant$minPts 时,则 x_j 为一个核心对象。

(3) 密度直达:如果 x_i 位于 x_j 的 ϵ-邻域中,且 x_j 是核心对象,则称 x_i 由 x_j 密度直达。

(4) 密度可达:对于 x_i 和 x_j,如果存在样本序列 $p_1,p_2,\cdots,p_n$,满足 $p_1=x_i$、$p_n=x_j$,且 $1\leqslant i<n$,其 p_{i+1} 由 p_i 密度直达,则称 x_j 由 x_i 密度可达。也就是说,$p_1,p_2,\cdots,p_{n-1}$ 是核心对象组成的链,链的起点 p_1 和经过的点 $p_i(1<i<n)$ 必须是核心对象,链的最后一个点 p_n 可以是任意对象。也就是说,密度可达满足传递性。

(5) 密度相连:对于 x_i 和 x_j,如果存在核心对象样本 x_k,使 x_i 和 x_j 均由 x_k 密度可达,则称 x_i 和 x_j 密度相连。密度相连关系满足对称性。

例如，图 5-5 中给出了一系列样本点，设邻域参数 minPts＝3，虚线圆圈表示 ϵ 邻域，y、x、r、p、q、m 是其中一条由核心对象组成的链，q 由 p 密度直达，r 由 p 密度直达，m 由 p 密度可达，m 与 y 密度相连。

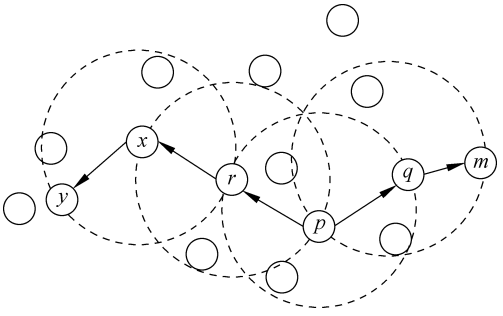

图 5-5　DBSCAN 聚类中相关概念的表示

5.3.2　DBSCAN 聚类算法

DBSCAN 聚类算法根据设置的邻域参数 ϵ、minPts 先确定核心对象集合 Ω，然后随机选择一个核心对象确定相应的聚类簇，直到所有的核心对象都被访问过为止。

DBSCAN 聚类算法描述如下：

输入：训练数据集 $D=\{x_1,x_2,\cdots,x_N\}$、邻域参数 ϵ、minPts。

过程：

(1) 初始化核心对象集合 $\Omega=\varnothing$、聚类簇数 $k=0$、未访问样本集合 $\Gamma=D$。

(2) 对于 $j=1,2,\cdots,N$：

① 通过距离度量方式，找到样本 x_j 的 ϵ-邻域集合 $M_\epsilon(x_j)$。

② 如果样本集中的样本个数满足 $|N_\epsilon(x_j)|\geqslant$minPts，则将样本 x_j 加入核心对象样本集合 $\Omega=\Omega\cup\{x_j\}$。

(3) 如果核心对象集合 $\Omega=\varnothing$，则算法结束，否则执行第(4)步。

(4) 在核心对象集合 Ω 中，随机选择一个核心对象 o，初始化当前簇核心对象队列 $Q_\Omega=\{o\}$，初始化类别序号 $k=k+1$，初始化当前簇样本集合 $C_k=\{o\}$，更新未访问样本集合 $\Gamma=\Gamma-\{o\}$。

(5) 如果当前簇核心对象队列 $Q_\Omega=\varnothing$，则当前聚类簇 C_k 生成完毕，更新簇划分 $C=\{C_1,C_2,\cdots,C_k\}$，更新核心对象集合 $\Omega=\Omega-C_k$，转到第(3)步，否则更新核心对象集合 $\Omega=\Omega-C_k$。

(6) 在当前簇核心对象队列 Q_Ω 中取出一个核心对象 o′，通过邻域距离阈值 ϵ 找出所有的 ϵ-邻域样本集 $M_\epsilon(o')$，令 $\Delta=M_\epsilon(o')\cap\Gamma$，更新当前簇样本集合 $C_k=C_k\cup\Delta$，更新未访问样本集合 $\Gamma=\Gamma-\Delta$，更新 $Q_\Omega=Q_\Omega\cup(\Delta\cap\Omega)-o'$，转到第(5)步执行。

输出：簇划分 $C=\{C_1,C_2,\cdots,C_k\}$。

依据以上算法思想，利用 DBSCAN 算法对西瓜数据集进行聚类，其算法实现如下。

```
import numpy as np
from numpy import *
```

```
import matplotlib.pyplot as plt
import math
def load_dataset(file_name):
    X_dataset = []
    f = open(file_name)
    for x in f.readlines():
        x_data = x.strip().split()
        X_dataset.append([float(x_data[0]), float(x_data[1])])
    return X_dataset

def get_dist(vec1, vec2):
    vec1 = np.array(vec1)
    vec2 = np.array(vec2)
    return sqrt(sum(power(vec1 - vec2, 2)))


def DBSCAN_clustering(X_data, max_dist, min_pts):
    core_object = []
    cluster = []
    N = len(X_data)
    #得到核心对象集合
    for i in range(N):
        M= get_divide_set(X_data[i], X_data, max_dist)
        if(len(M) >= min_pts):
            core_object.append(X_data[i])
    #生成聚类簇
    no_access_data = copy(X_data).tolist()
    while(len(core_object) != 0):
        old_no_access_data = copy(no_access_data).tolist()
        index = random.randint(0, len(core_object))
        Q = []
        Q.append(core_object[index])
        core_object.remove(core_object[index])
        while len(Q) != 0:
            e = copy(Q[0]).tolist()
            Q.remove(Q[0])
            K = get_divide_set(e, X_data, max_dist)
            if len(K) >= min_pts:
                delta = get_comm_set(K, no_access_data)
                for i in range(len(delta)):
                    Q.append(delta[i])
                no_access_data = del_aggregate(no_access_data, delta)
        clu_k = del_aggregate(old_no_access_data, no_access_data)
        cluster.append(clu_k)
        core_object = del_aggregate(core_object, clu_k)
    return cluster

def get_divide_set(x0, X_data, max_dist):
    N = len(X_data)
    M = []
    for i in range(N):
        d = get_dist(x0, X_data[i])
        print(X_data[i])
```

```
            if(d <= max_dist):
                M.append(X_data[i])
        return M
    def set_compare(set1, set2):
        len1 = len(set1)
        len2 = len(set2)
        flag = True
        if(len1 != len2):
            return False
        for i in range(len1):
            if(set1[i] != set2[i]):
                flag = False
        return flag
    def get_comm_set(set1, set2):
        m = len(set1)
        n = len(set2)
        delta = []
        for i in range(m):
            for j in range(n):
                if(set_compare(set1[i], set2[j]) == True):
                    delta.append(set1[i])
        return delta
    def del_aggregate(X_data, x):
        m = len(x)
        N = len(X_data)
        lst_deleted = []
        for i in range(m):
            for j in range(N):
                if(set_compare(x[i], X_data[j]) == True):
                    lst_deleted.append(X_data[j])
        for i in range(len(lst_deleted)):
            print("lst_deleted[", i, "] = ", lst_deleted[i])
            X_data.remove([lst_deleted[i][0], lst_deleted[i][1]])
        return X_data
    def show_figure(X_data):
        C1 = mat(array(X_data[0]))
        C2 = mat(array(X_data[1]))
        C3 = mat(array(X_data[2]))
        C4 = mat(array(X_data[3]))
        fig,ax = plt.subplots(1,1)
        l1, = ax.plot(C1[:, 0], C1[:, 1], "Dr")
        l2, = ax.plot(C2[:, 0], C2[:, 1], "sg")
        l3, = ax.plot(C3[:, 0], C3[:, 1], "*b")
        l4, = ax.plot(C4[:, 0], C4[:, 1], "oc")
    plt.legend(handles = [l1,l2,l3,l4],labels = ['C1','C2','C3','C4'],loc = 0)
        ax.set_xlabel("密度")
        ax.set_ylabel("含糖率")
        plt.title('基于密度的聚类')
        plt.rcParams['font.sans-serif'] = ['SimHei']      #显示中文
        plt.rcParams['axes.unicode_minus'] = False
        plt.show()
    if __name__ == "__main__":
        data_set = load_dataset("xiguadata.txt")
```

```
max_dist = 0.1
cluster = DBSCAN_clustering(data_set, max_dist, 5)
print(data_set)
print(len(cluster))
show_figure(cluster)
```

当 minPts=5 时,程序运行结果如图 5-6 所示。

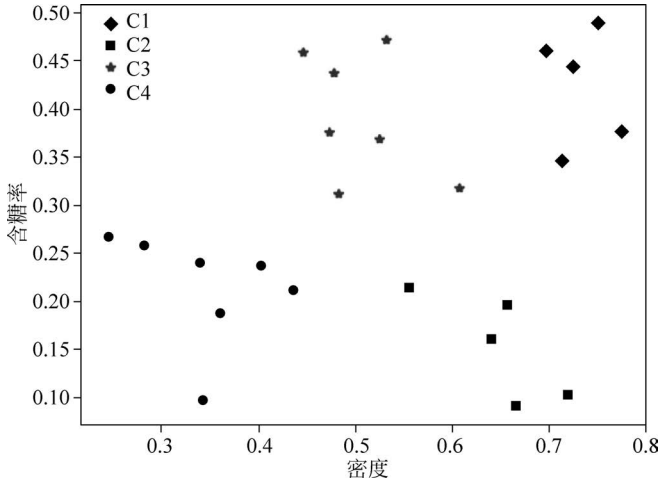

图 5-6 DBSCAN 聚类(minPts=5)

5.4 基于层次的聚类——AGNES 聚类

基于层次的聚类是一种很直观的算法,顾名思义,就是要逐层地进行聚类。按照层次聚类策略,可分为自底而上的合并聚类和自顶而下的分割聚类。其中 AGNES(agglomerative nesting)聚类是一种最为常见的自底而上的合并聚类算法。

5.4.1 AGNES 聚类算法的思想

AGNES 采用自底而上合并聚类簇,每次找到距离最短的两个聚类簇,然后合并成一个大的聚类簇,以此类推,直到全部样本数据合并为一个聚类簇。整个聚类过程就形成了一个树状结构,如图 5-7 所示。

如何计算两个聚类簇之间的距离呢? 初始时,每个样本数据点作为一个类,它们的距离就是这两个点之间的距离。对于一个包含不止一个数据样本的聚类簇,有 3 种聚类簇度量方法:最小距离、最大距离和平均距离。

最小距离:

$$d_{\min}(C_i,C_j)=\min_{x\in C_i,y\in C_j}\text{dist}(x,y) \tag{5-10}$$

最大距离:

$$d_{\max}(C_i,C_j)=\max_{x\in C_i,y\in C_j}\text{dist}(x,y) \tag{5-11}$$

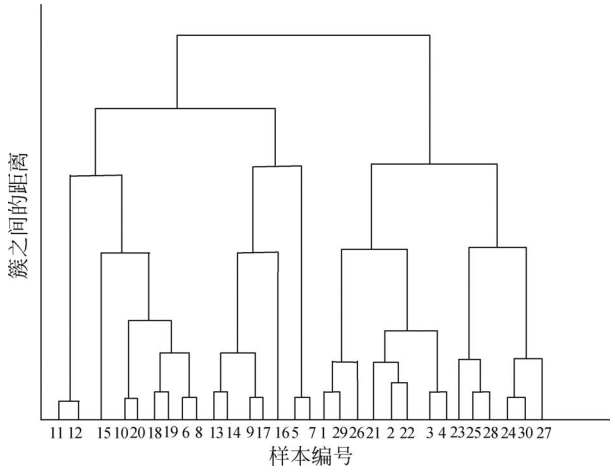

图 5-7　AGNES 算法聚类结构

平均距离：

$$d_{\text{average}}(C_i,C_j)=\frac{1}{|C_i||C_j|}\sum_{x\in C_i}\sum_{y\in C_j}\text{dist}(x,y) \tag{5-12}$$

AGNES 聚类算法描述如下。

输入：样本数据集 $D=\{x_1,x_2,\cdots,x_N\}$、聚类簇个数 k、聚类簇度量函数 get_dist。

过程：

(1) 将每个对象看成一个聚类簇，即对于任意的 $1\leqslant j\leqslant N$，有 $C_j=\{x_j\}$。

(2) 根据聚类簇度量函数 get_dist 确定各个簇之间的距离。

(3) 设置当前簇个数 $q=N$。

(4) 当 $q>k$ 时，重复执行以下步骤：

① 找出距离最近的两个簇 C_i 和 C_j，合并 C_i 和 C_j：$C_i=C_i\cup C_j$。

② 对编号为 $j+1,j+2,\cdots,q$ 的簇重新编号，依次为 $j,j+1,\cdots,q-1$。

③ 对 $j=1,2,\cdots,q-1$，更新聚类簇之间的距离。

④ 根据约束条件，确定新参数 λ_2 的上下界。

输出：簇划分 $C=\{C_1,C_2,\cdots,C_k\}$。

根据 AGNES 聚类算法思想，对西瓜数据集进行聚类，当 $k=4$ 时，聚类结果如下。

簇 1：[[0.697，0.46]，[0.725，0.445]，[0.751，0.489]，[0.774，0.376]，[0.714，0.346]]

簇 2：[[0.666，0.091]，[0.719，0.103]，[0.639，0.161]，[0.657，0.198]，[0.748，0.232]，[0.593，0.042]，[0.634，0.264]，[0.608，0.318]，[0.556，0.215]，[0.481，0.149]]

簇 3：[[0.243，0.267]，[0.282，0.257]，[0.403，0.237]，[0.437，0.211]，[0.359，0.188]，[0.339，0.241]，[0.245，0.057]，[0.343，0.099]]

簇 4：[[0.36，0.37]，[0.483，0.312]，[0.525，0.369]，[0.473，0.376]，[0.478，0.437]，[0.446，0.459]，[0.532，0.472]]

5.4.2 AGNES 算法的实现

根据 AGNES 聚类算法的思想,针对西瓜数据集,其算法实现如下。

```
import math
import matplotlib.pyplot as plt
#data_train 包含 30 个西瓜样本(密度,含糖量)
data_train = [
     [0.697, 0.460], [0.774, 0.376],[0.634, 0.264],[0.608, 0.318],
     [0.556, 0.215],[0.403, 0.237],[0.481,0.149],[0.437, 0.211],
     [0.666, 0.091],[0.243, 0.267],[0.245, 0.057],[0.343, 0.099],
     [0.639, 0.161],[0.657, 0.198],[0.360, 0.370],[0.593, 0.042],
     [0.719, 0.103],[0.359,0.188],[0.339,0.241],[0.282,0.257],
     [0.748, 0.232],[0.714,0.346],[0.483,0.312],[0.478,0.437],
     [0.525, 0.369],[0.751,0.489],[0.532,0.472],[0.473,0.376],
     [0.725, 0.445],[0.446,0.459]]
#计算样本点之间的欧几里得距离
def dist_eclud(x1, x2):
   return math.sqrt(math.pow(x1[0] - x2[0], 2) + math.pow(x1[1] - x2[1], 2))
#计算簇之间的最小距离
def get_min_dist(C_x, C_y):
   return min(dist_eclud(x, y) for x in C_x for y in C_y)
#计算簇之间的平均距离
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))
#找到距离最小的下标
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
def get_dist(data):                          #获得样本点距离
   C = []
   D = []
   for x in data:
      C_x = []
      C_x.append(x)
      C.append(C_x)
   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

#AGNES 层次聚类算法
def AGNES_clustering(data, dist, k):
   n = len(data)
   C,D = get_dist(data)
```

```
    #合并各簇
    while n > k:
        min_i, min_j, min = find_min_index(D)
        print(min_i,min_j,min)
        C[min_i].extend(C[min_j])
        C.remove(C[min_j])
        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
    return C
#显示聚类结果
def show_figure(C):
    color_value = ['r', 'g', 'b', 'y', 'm', 'k', 'c']
    marker_value = ['o', 's', '*', 'd', 'x', 'v', 'p']
    for i in range(len(C)):
        crd_x = []                                          #x 坐标
        crd_y = []                                          #y 坐标
        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)
    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()
if __name__ == "__main__":
    k = 4
    C = AGNES_clustering(data_train, get_average_dist, k)
    show_figure(C)
```

AGNES 聚类算法在西瓜数据集上的运行结果如图 5-8 所示。

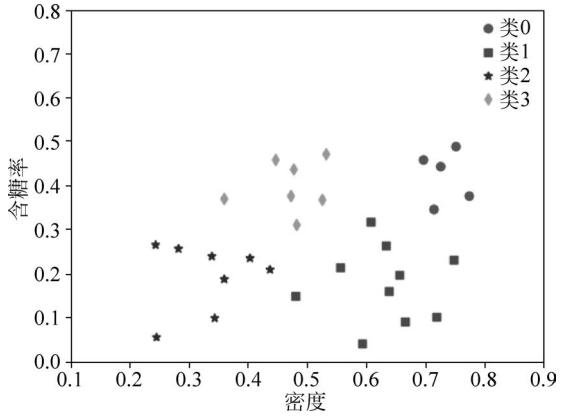

图 5-8　基于层次的聚类在西瓜数据集上的运行结果（$k=4$）

视频讲解

5.5 高斯混合聚类

高斯混合聚类是一种采用高斯混合模型(Gaussian Mixture Model,GMM)的聚类算法,主要采用概率统计的方法进行聚类。高斯混合聚类假设样本服从不同的独立的高斯分布,通过采用 EM(Expectation-Maximization)算法实现聚类。

5.5.1 概率密度函数

对于样本 X 服从均值为 μ、方差为 σ^2 的正态分布 $p(x) \sim N(\mu,\sigma^2)$,即

$$p(X)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(X-\mu)^2}{2\sigma^2}} \tag{5-13}$$

其概率密度函数如图 5-9 所示。

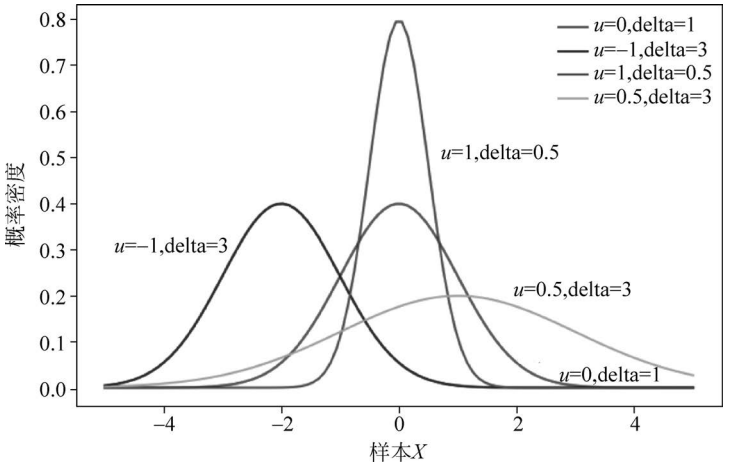

图 5-9 高斯分布的概率密度函数(见彩插)

服从正态分布的概率密度函数具有以下特征:

(1) 当自变量 $x=\mu$ 时,$f(x)$取最大值。

(2) 概率密度函数的图像(曲线图像)关于 $x=\mu$ 对称。

(3) 标准差 σ 越大,则图像峰值(峰值也就是概率最大值,即峰值$=f(x=\mu)$)越小。

这是样本服从单个分布的情况,对于 n 维样本空间 χ 中的随机向量 $\boldsymbol{X}_i$,若 $\boldsymbol{X}_i$ 服从高斯分布,其概率密度函数为

$$p(\boldsymbol{X})=\frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}}}e^{-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})} \tag{5-14}$$

其中,$\boldsymbol{\mu}$ 是 n 维均值向量,$\boldsymbol{\Sigma}$ 是 $n\times n$ 的协方差矩阵。一个混合高斯概率密度函数如图 5-10 所示。

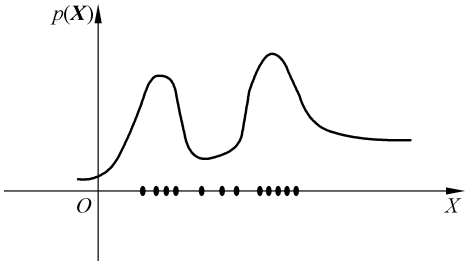

图 5-10 高斯分布的混合概率密度函数

提　示
高斯混合模型其实就是若干单高斯混合模型的线性叠加，直观上看，高斯混合概率密度函数是由若干单高斯概率密度函数组合而成的。

5.5.2　高斯混合聚类算法的推导过程

若这些数据是由服从若干高斯分布的模型生成，根据最大似然估计和贝叶斯公式可得 EM 算法的 E 步：

$$\gamma_j^{(i)} = p(z^{(i)} = j \mid x^{(i)}, \alpha, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{5-15}$$

该公式的含义为：每个样本的隐含类别 $z^{(i)}$ 可通过各混合成分的后验概率得到。基于此求解 M 步，利用最大似然估计求以下函数的参数：

$$\begin{aligned} f(\Theta, Z) &= \sum_{i=1}^{n}\sum_{j=1}^{z} Q_i(Z_j) \ln \frac{p(X_i, Z_j \mid \Theta)}{Q_i(Z_j)} = \sum_{i=1}^{n}\sum_{j=1}^{z} Q_i(Z_j) \ln \frac{p(X_i, Z_j \mid \gamma, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{Q_i(Z_j)} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{z} Q_i(Z_j^{(i)}) \ln \frac{p(X_i \mid Z_j^{(i)}, \gamma, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(Z_j^{(i)} \mid \gamma, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{Q_i(Z_j^{(i)})} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{z} \gamma_j^{(i)} \ln \frac{\dfrac{1}{(2\pi)^{\frac{n}{2}} \lvert \boldsymbol{\Sigma}_j \rvert} \exp\left(-\frac{1}{2}\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\right)^{\mathrm{T}} \boldsymbol{\Sigma}_j^{-1} \left(-\frac{1}{2}\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\right) \alpha_j}{\gamma_j^{(i)}} \end{aligned} \tag{5-16}$$

固定参数 α_j 和 $\boldsymbol{\Sigma}_j$，对 $\boldsymbol{\mu}_j$ 求导可得

$$\frac{\partial f(\Theta, Z)}{\partial \boldsymbol{\mu}_j} = \frac{1}{2}\sum_{i=1}^{n} \gamma_j^{(i)} (\boldsymbol{\Sigma}_j^{-1} \boldsymbol{x}^{(i)} - \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\mu}_j) \tag{5-17}$$

令式(5-17)为零，即得参数 $\boldsymbol{\mu}_j$ 的更新公式 $\boldsymbol{\mu}_j' = \dfrac{\sum_{i=1}^{n} \gamma_j^{(i)} \boldsymbol{x}^{(i)}}{\sum_{i=1}^{n} \gamma_j^{(i)}}$。同理，分别固定参数 $\boldsymbol{\mu}_j$ 和 α_j、$\boldsymbol{\mu}_j$ 和 $\boldsymbol{\Sigma}_j$，对 $\boldsymbol{\Sigma}_j$、α_j 求导，可分别得到参数 $\boldsymbol{\Sigma}_j$ 和 α_j 的更新公式：

$$\boldsymbol{\Sigma}_j' = \frac{\sum_{i=1}^{n} \gamma_j^{(i)} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j^{(i)})^{\mathrm{T}} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j^{(i)})}{\sum_{i=1}^{n} \gamma_j^{(i)}}, \quad \alpha_j' = \frac{\sum_{i=1}^{k} \gamma_j^{(i)}}{k} \tag{5-18}$$

5.5.3　高斯混合聚类算法思想

类似于单高斯模型，高斯混合分布定义如下。

$$p_M(X) = \sum_{i=1}^{k} \alpha_i p(X \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{5-19}$$

该公式表示该高斯分布由 k 个服从高斯分布的成分构成。其中，α_i 是混合成分的系数，$\sum_{i=1}^{k}\alpha_i=1$ 且 $\alpha_i>0$；$\boldsymbol{\mu}_i$ 和 $\boldsymbol{\Sigma}_i$ 分别是第 i 个高斯混合成分的均值和方差。

对于随机变量 $X=\{X_1,X_2,X_3,\cdots,X_m\}$来说，每个样本 X_j 属于 $z_j=i$ 的概率可由贝叶斯定理获得，即隐变量 z_j 的后验概率为

$$p_M(z_j=i\mid X_j)=\frac{P(z_j=i)p_M(X_j\mid z_j=i)}{p_M(X_j)}=\frac{\alpha_i p(X_j\mid\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)}{\sum_{i=1}^{k}\alpha_i p(X\mid\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)} \tag{5-20}$$

高斯混合聚类的算法描述如下：

输入样本集为 $D=\{X_1,X_2,X_3,\cdots,X_m\}$，高斯混合成分个数为 k。

步骤如下：

初始化高斯混合分布的各模型参数 k、α_i、$\boldsymbol{\mu}_i$ 和 $\boldsymbol{\Sigma}_i$。

while 迭代次数 iter < MaxIter

for j=1 to m do

根据式(5-19)计算 X_j 由各混合成分生成的后验概率 $r_{ji}=P_M(z_j=i\mid X_j)$，$1\leqslant j\leqslant K$，表示属于哪个聚类。

for i=1 to k do

计算并更新均值向量，即 $\boldsymbol{\mu}_i=\dfrac{\sum_{j=1}^{m}r_{ji}x_j}{\sum_{j=1}^{m}r_{ji}}$。

计算并更新协方差矩阵，即 $\boldsymbol{\Sigma}_i=\dfrac{\sum_{j=1}^{m}r_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i)(\boldsymbol{x}_j-\boldsymbol{\mu}_i)^{\mathrm{T}}}{\sum_{j=1}^{m}r_{ji}}$。

计算并更新混合系数，即 $\alpha_i=\dfrac{\sum_{j=1}^{m}r_{ji}}{m}$。

令 $C=\{\}$：

for j=1 to m do

根据式(5-19)确定 X_j 所属的聚类标记 C_j。

将 X_j 划入相应的聚类 $C=C\cup\{X_j\}$。

输出：聚类划分 $C=\{C_1,C_2,C_3,\cdots,C_k\}$。

5.5.4 高斯混合聚类应用举例

【例 5-1】 在表 5-2 中，有 30 条西瓜数据，包括密度和含糖率，根据其特征将其相似的

水果聚集为一类，并用散点图绘制最终的聚类结果。

表 5-2　西瓜数据的密度和含糖率

编号	密度	含糖率	编号	密度	含糖率	编号	密度	含糖率
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

【分析】

如何利用 EM 算法将所给西瓜数据具有相似特征的数据聚为一类，需要先确定数据中各变量分别是什么，包括隐变量 k、α_i、$\boldsymbol{\mu}_i$ 和 $\boldsymbol{\Sigma}_i$。这里的隐变量就是每条数据属于哪一类的概率，其他变量的取值可以随机设置。

初始时，设 $k=3$，即有 3 个聚类，权值 $\alpha_1=\alpha_2=\alpha_3=\frac{1}{3}$，选择第 6、22、27 条数据分别作为 3 个聚类的均值，即 $\boldsymbol{\mu}_1=(0.403,0.237)$、$\boldsymbol{\mu}_2=(0.714,0.346)$、$\boldsymbol{\mu}_3=(0.532,0.472)$，$\boldsymbol{\Sigma}_1=\boldsymbol{\Sigma}_2=\boldsymbol{\Sigma}_3=\begin{pmatrix}0.1 & 0\\ 0 & 0.1\end{pmatrix}$。然后开始第一轮迭代，根据上述高斯混合模型计算出第 1 条数据分别属于聚类 1、聚类 2、聚类 3 的概率密度为 0.726、1.56、1.134，其属于聚类 1、聚类 2、聚类 3 的后验概率分别是 0.213、0.456、0.332，即 E 步。从而由后验概率更新均值、协方差和系数，即 M 步。

【算法实现】

1. 数据的加载

数据存储在 EMData.xls 文件中，通过 open_workbook 打开 Excel 文件，读取工作表中的数据到 myData[]列表中。

```
import os
import xlrd
input_file = u"EMData.xls"
from xlrd import open_workbook
workbook = open_workbook(input_file)
#输出此工作簿中有多少个工作表 workbook.nsheets
print('Number of worksheets: ', workbook.nsheets)
#遍历工作簿中的每张工作表
for worksheet in workbook.sheets():
    #分别输出每张工作表的名称、行数、列数
    print( 'Worksheet name: ', worksheet.name, '\tRows: ', worksheet.nrows, '\tColumns: ',
worksheet.ncols)
```

```
    for row_index in range(worksheet.nrows):
        for column_index in range(worksheet.ncols):
                    print(worksheet.cell_value(row_index, column_index),end = ' ')
        print()
worksheet = workbook.sheet_by_index(0)
print(worksheet)
nrows = worksheet.nrows
nclos = worksheet.ncols
myData = [[0.0 for i in range(nclos)]for j in range(nrows)]
print(len(myData))
for row_index in range(nrows):
    myData[row_index][0] = worksheet.cell(row_index, 0).value
    myData[row_index][1] = worksheet.cell(row_index,1).value
    myData[row_index][2] = worksheet.cell(row_index,2).value
```

2. 数据的初始化

初始时,设 k=3,系数 a1=a2=a3=1/3,均值 u1= myData[5][1:3]、u2=myData[21][1:3]、u3=myData[26][1:3],方差 sigma1=np.array([[0.1,0], [0,0.1]])、sigma2=np.array([[0.1,0],[0,0.1]])、sigma3=np.array([[0.1,0],[0,0.1]]),利用 multivariate_normal.pdf()可求出由聚类 1、聚类 2、聚类 3 生成的后验概率和属于各聚类的后验概率。

```
k = 3
a1 = 1/3
a2 = 1/3
a3 = 1/3
a = [a1,a2,a3]
u1 = myData[5][1:3]                          #均值
u2 = myData[21][1:3]
u3 = myData[26][1:3]
u = [u1,u2,u3]
u = np.array(u)
sigma1 = np.array([[0.1,0], [0,0.1]])    #协方差
sigma2 = np.array([[0.1,0],[0,0.1]])
sigma3 = np.array([[0.1,0],[0,0.1]])
sigma = np.vstack((sigma1,sigma2,sigma3))
i = 1
p1 = st.multivariate_normal.pdf(myData[i][1:3],u1,sigma1)
p2 = st.multivariate_normal.pdf(myData[i][1:3],u2,sigma2)
p3 = st.multivariate_normal.pdf(myData[i][1:3],u3,sigma3)
p = a1 * p1 + a2 * p2 + a3 * p3
r11 = a1 * p1/p
r12 = a2 * p2/p
r13 = a3 * p3/p
print(r11,r12,r13)
```

3. E 步和 M 步

利用初始设置的系数、均值和方差求后验概率,并更新均值和方差,经过若干次迭代直至收敛,得到最终的均值和方差,以求出后验概率,从而完成聚类。

```
for it in range(50):
    for i in range(0,nrows):
        pp = 0
        for j in range(0,k):
            print(x[i,:])
            p[i][j] = st.multivariate_normal.pdf(x[i, 0:2], u[j,0:2], sigma[2 * j:2 * j + 2,0:2])
            pp = pp + a[j] * p[i,j]
        for j in range(k):
            r[i,j] = a[j] * p[i,j]/pp
    #更新 a、u 和 sigma 的值
    for j in range(k):
        y = np.zeros([1,2])
        for i in range(nrows):
            y = y + r[i,j] * x[i,:]
        a[j] = np.sum(r[:,j])/nrows
        u[j,:] = y/np.sum(r[:,j])
    oo1 = np.zeros([2, 2])
    sigma = np.zeros([6, 2])
    for j in range(k):
        s1 = 0
        s2 = 0
        s3 = 0
        s4 = 0
        for i in range(nrows):
            s1 = s1 + r[i, j] * ((x[i, 0] - u[j, 0]) * (x[i, 0] - u[j, 0]))
            s2 = s2 + r[i, j] * ((x[i, 0] - u[j, 0]) * (x[i, 1] - u[j, 1]))
            s3 = s3 + r[i, j] * ((x[i, 1] - u[j, 1]) * (x[i, 0] - u[j, 0]))
            s4 = s4 + r[i, j] * ((x[i, 1] - u[j, 1]) * (x[i, 1] - u[j, 1]))
        oo1 = [[s1 / np.sum(r[:, j]), s2 / np.sum(r[:, j])],[s3 / np.sum(r[:, j]), s4 / np.sum
(r[:, j])]]
        set_segama(sigma,j,oo1)
ones = np.ones(30)
x = np.c_[x,ones]
idx = np.ones(nrows)
for i in range(nrows):
    max,index = get_max_index(r[i,:])
    print(max,index)
    idx[i] = index
for i in range(nrows):
    x[i,2] = idx[i]
```

4. 绘图

根据聚类结果标签绘制散点图，算法实现如下。

```
d = dict(x = x[:,0], y = x[:,1], label = labels)
df = pd.DataFrame(d)
groups = df.groupby('label')
markers = ['x', 'o', '^']
fig, ax = plt.subplots()
ax.margins(0.05)
for (name, group), marker in zip(groups, cycle(markers)):
```

```
    ax.plot(group.x, group.y, marker = marker, linestyle = '', ms = 12, label = name)
ax.legend()
plt.show()
```

分别经过迭代 50 次和 100 次后,程序运行结果如图 5-11 所示。

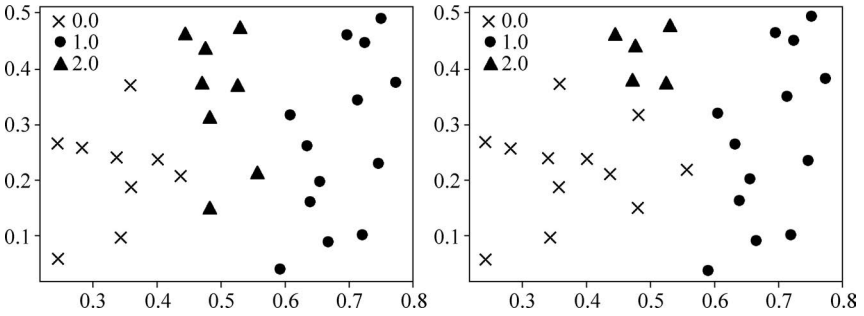

图 5-11 利用 EM 算法对西瓜数据集进行聚类的散点图(迭代 50 次和 100 次)

在此程序中,主要用到了多元正态分布函数 multivariate_normal.pdf(x, mean=None, cov=1),其主要功能是根据给定的均值和方差求解样本的概率密度。

函数原型:pdf(x, mean=None, cov=1)。

- mean:均值,维度为 1,必选参数。
- cov:协方差矩阵,必选参数。

例如,下面代码利用 st.multivariate_normal.pdf()函数生成概率密度并绘制概率密度图。

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
x = np.linspace(0, 10, 20, endpoint = False)          #样本
y = st.multivariate_normal.pdf(x, mean = 5, cov = 1)  #样本的概率密度函数
plt.plot(x, y)
plt.show()
```

绘制的概率密度图如图 5-12 所示。

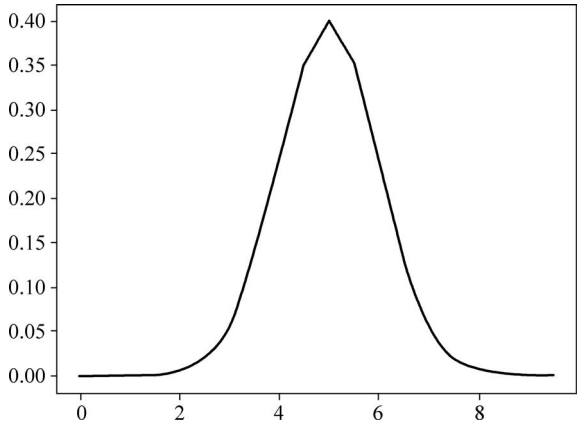

图 5-12 利用多元概率密度函数绘制的概率密度图

此外，multivariate_normal(mean，cov，size＝None，check_valid＝None，tol＝None)可用于生成多元正态分布数据。除了 mean 和 cov 外，各参数含义如下。

- size：指定生成矩阵的维度，若 size＝(1，1，2)，则输出矩阵 shape，即形状为 1×1×2×N(N 为 mean 的长度)的矩阵。
- check_valid：可取值 warn、raise 以及 ignore。
- tol：检查协方差矩阵奇异值时的公差，float 类型。

5.6　各种聚类算法的比较

不同的聚类算法具有各自的优缺点和适用情况。衡量一个算法的优劣主要从数据的属性、算法模型的预设、模型的处理能力上来分析。

(1) 算法的处理能力：处理大的数据集的能力(即算法复杂度)、处理数据噪声的能力、处理任意形状的能力。

(2) 算法是否需要预设条件：是否需要预先知道聚类个数，是否需要用户给出领域知识。

(3) 算法的数据输入属性：算法处理的结果与数据输入的顺序是否相关，算法处理是否对数据的维度敏感，对数据的类型有无要求。

各种聚类算法的比较情况如表 5-3 所示。

表 5-3　各种聚类算法的比较

聚类方法	优　点	缺　点	常用算法
基于划分的方法	对于大型数据集简单高效，时间复杂度及空间复杂度低	容易局部最优，需要预先设置 k 值，对初值 k 敏感，只能处理数值型数据，不能解决非凸数据	K-means、k-means + +、k-medoids、k-medians、kernel K-means
基于密度的方法	对噪声不敏感，能发现任意形状的聚类	聚类的结果依赖参数的设置，较稀的聚类会被划分为多个类，或密度较大且离得较近的类会被合并成一个聚类	DBSCAN、OPTICS
基于层次的方法	可解释性好，这些算法能产生高质量的聚类，能解决非球形聚类	时间复杂度高	适用于小数量级，BIRCH
基于网络的方法	聚类速度快，其运算效率与数据个数无关，只依赖于数据空间中每一维上单元的个数	参数敏感、无法处理不规则分布的数据、维数灾难等	经常与基于密度的算法结合使用，常见的算法有 STING、WAVE -CLUSTER、CLIQUE
基于模型的方法	对“类”的划分以概率形式出现	执行效率不高，尤其是由多种分布并且数据量很少的情况	GMM、SOM
基于模糊的聚类方法	对于满足正态分布的数据聚类效果会很好	算法的性能依赖于初始聚类中心	FCM、HCM

5.7 本章小结

本章主要介绍了聚类算法的思想、聚类算法的分类及具有代表性的聚类算法,包括K-means、DBSCAN、AGNES算法、高斯混合聚类原理及其实现。通过实现各种聚类算法,提高对算法思想的理解和算法的实现能力。K-means算法思想比较简单,适用于符合高斯分布的样本聚类,算法运行速度快。DBSCAN算法对噪声不敏感,能对任意形状的样本聚类。AGNES算法可解释性好,能产生高质量的聚类,能解决非球形聚类,适用于小规模样本数据。EM算法属于自收敛的分类算法,需要事先初始化模型参数,能可靠地找到最优的收敛值。只要给定一些训练数据,再定义一个最大化函数,采用EM算法,利用计算机经过若干次迭代,就可以得到所需的模型。缺点是对初始值敏感,而参数Q的选择直接影响收敛效率以及能否得到全局最优解,当所要优化的函数不是凸函数时,EM算法容易给出局部最佳解,而不是最优解。

EM算法可用于0-1二项分布、隐马尔可夫模型、高斯混合聚类等问题。每种聚类算法都有其优势和不足之处,应根据实际场合选择合适的聚类算法。

5.8 习题


在线测试

一、选择题

1. K-means聚类属于(　　)。
   A. 基于密度的方法　　B. 基于层次的方法
   C. 基于划分的方法　　D. 基于网格的方法
2. DBSCAN算法属于(　　)。
   A. 基于密度的方法　　B. 基于层次的方法
   C. 基于网格的方法　　D. 基于划分的方法
3. (　　)解释性好,且能解决非球形聚类。
   A. 基于层次的方法　　B. 基于模型的方法
   C. 基于划分的方法　　D. 基于密度的方法
4. (　　)聚类需要考虑不同簇之间的距离。
   A. 基于模型的方法　　B. 基于划分的方法
   C. 基于密度方法　　D. 基于层次的方法
5. (　　)算法的目标是过滤密度低区域的样本。
   A. AGNES算法　　B. K-means算法
   C. DBSCAN算法　　D. k-means++算法
6. 以下关于K-means算法的说法,不正确的是(　　)。
   A. 平均时间复杂度为$O(Nkt)$
   B. 不适合对非凸形状数据进行聚类

C. 对于不同 k 的初始值，聚类结果是相同的

D. 对噪声和离群点敏感

7. 以下关于 EM 算法的说法，不正确的是(　　)。

A. 通过建立对数似然函数求解隐变量 Z 的期望，并寻找最大化期望似然情况下的参数

B. EM 算法不依赖于初始参数值

C. EM 算法是一种迭代优化算法

D. EM 算法需要事先初始化模型参数

8. EM 算法思想可以应用于(　　)。

A. 0-1 二项分布　　B. 高斯混合聚类

C. 隐马尔可夫模型　　D. 学习贝叶斯网络的概率

E. 以上均可

9. 在 GMM 算法中，聚类结果与(　　)有关。

A. k 的取值　　B. 每个 (μ,σ_1^2) 的取值

C. 数据的分布　　D. 以上都有关

二、算法分析题

1. 编写算法，使用 scikit-learn 模块中的 DBSCAN 函数对西瓜数据集进行聚类，并可视化聚类结果。

2. 编写算法，使用 scikit-learn 模块中的 AgglomerativeClustering 函数对鸢尾花数据进行聚类，并可视化聚类结果。

3. 已知一个班里有 50 个男生，50 个女生，且男生站左侧，女生站右侧。假定男生的身高服从正态分布(μ_1,σ_1^2)，女生的身高则服从另一个正态分布(μ,σ_2^2)。此时可以用极大似然法(MLE)，分别通过这 50 个男生和 50 个女生的样本来估计这两个正态分布的参数。如果情况更复杂一点，即 50 个男生和 50 个女生的数据混在一起了，只拥有 100 个人的身高数据，却不知道这 100 个人的具体性别信息。试利用 EM 算法建立求男生和女生身高均值的算法模型。

4. 利用 GMM 算法和 K-means 算法的区别是什么？各自有什么优势？

5. 对于基于二项分布的 EM 算法来说，请说明为什么求解 E 步就可以得到最优解。