# 第七章 聚类分析

## 7.3 K均值聚类法

用层次聚类法聚类时，随着聚类样本对象的增多，计算量会迅速增加，而且聚类结果—谱系图会十分复杂，不便于分析。特别是样品的个数很大（如$n \geq 100$）时，层次聚类法的计算量非常大，将占据大量的计算机内存空间和较多的计算时间。为了改进上述缺点，一个自然的想法是先粗略地分一下类，然后按某种最优原则进行修正，直到将类分得比较合理为止。基于这种思想就产生了动态聚类法，也称逐步聚类法。

动态聚类适用于大型数据。动态聚类法有许多种方法，这里介绍一种比较流行的动态聚类法—$K$均值法，它是一种快速聚类法，该方法得到的结果简单易懂，对计算机的性能要求不高，因而应用广泛。该方法由麦克奎因（Macqueen）于 1967 年提出。

## 7.3 K均值聚类法

### 1. K均值聚类基本思想

算法的思想是假定样本集中的全体样本可分为$C$类，并选定$C$个初始聚类中心，然后，根据最小距离原则将每个样本分配到某一类中，之后不断迭代计算各类的聚类中心，并依据新的聚类中心调整聚类情况，直到迭代收敛或聚类中心不再改变。

$K$均值聚类算法最后将总样本集$G$划分为$C$个子集：$G_1,G_2,\cdots,G_C$，它们满足下面条件：

（1）$G_1 \cup G_2 \cup \cdots \cup G_C = G$；

（2）$G_i \cap G_j = \Phi$ $(1 \le i < j \le C)$；

（3）$G_i \ne \Phi$，$G_i \ne G$ $(1 \le i \le C)$。

设$m_i$（$i=1,\cdots,C$）为$C$个聚类中心，记

$$J_e = \sum_{i=1}^{C} \sum_{\omega \in G_i} \left\| \omega - m_i \right\|^2 ,$$

使$J_e$最小的聚类是误差平方和准则下的最优结果。

## 7.3 K均值聚类法

### 2. K均值聚类算法步骤

$K$均值聚类算法描述如下：

（1）初始化。设总样本集$G=\{\omega_j, j=1,2,\cdots,n\}$是$n$个样品组成的集合，聚类数为$C$（$2\le C\le n$），将样本集$G$任意划分为$C$类，记为$G_1,G_2,\cdots,G_C$，计算对应的$C$个初始聚类中心，记为$m_1,m_2,\cdots,m_C$，并计算$J_e$。

（2）$G_i=\Phi$（$i=1,2,\cdots,C$），按最小距离原则将样品$\omega_j$（$j=1,2,\cdots,n$）进行聚类，即若$d(\omega_j,G_k)=\min\limits_{1\le i\le C} d(\omega_j,m_i)$，则$\omega_j\in G_k$，$G_k=G_k\cup\{\omega_j\}$，$j=1,2,\cdots,n$。

## 7.3 K均值聚类法

### 2. K均值聚类算法步骤

重新计算聚类中心

$$m_i = \frac{1}{n_i} \sum_{\omega_j \in G_i} \omega_j \text{，} \quad i = 1,2,\cdots,C \text{，}$$

式中，$n_i$为当前$G_i$类中的样本数目。并重新计算$J_e$。

（3）若连续两次迭代的$J_e$不变，则算法终止，否则算法转（2）。

实际计算时，可以不计算$J_e$，只要聚类中心不发生变化，算法即可终止。

## 7.3 K均值聚类法

例 7.3 已知聚类的指标变量为 $x_1, x_2$，四个样本点的数据分别为 $\omega_1=(1,3)$，$\omega_2=(1.5,3.2)$，$\omega_3=(1.3,2.8)$，$\omega_4=(3,1)$.

试用 $K$ 均值聚类分析把样本点分成 2 类。

解　现要分为两类$G_1$和$G_2$类，设初始聚类为$G_1=\{\omega_1\}, G_2=\{\omega_2,\omega_3,\omega_4\}$，则初始聚类中心为

$G_1$类：为$\omega_1$值，即$m_1=(1,3)$。

$G_2$类：$m_2=\left(\dfrac{1.5+1.3+3}{3},\dfrac{3.2+2.8+1}{3}\right)=(1.9333,2.3333)$.

## 7.3 K均值聚类法

计算每个样本点到$G_1, G_2$聚类中心的距离

$$d_{11}=\|\omega_1-m_1\|=\sqrt{(1-1)^2+(3-3)^2}=0,\ d_{12}=\|\omega_1-m_2\|=1.1470;$$

$$d_{21}=\|\omega_2-m_1\|=0.5385,\ d_{22}=\|\omega_2-m_2\|=0.9690;$$

$$d_{31}=\|\omega_3-m_1\|=0.3606,\ d_{32}=\|\omega_3-m_2\|=0.7867;$$

$$d_{41}=\|\omega_4-m_1\|=2.8284,\ d_{42}=\|\omega_4-m_2\|=1.7075;$$

得到新的划分为：$G_1=\{\omega_1,\omega_2,\omega_3\}$，$G_2=\{\omega_4\}$，新的聚类中心为

$G_1$类：$m_1=\left(\dfrac{1+1.5+1.3}{3},\dfrac{3+3.2+2.8}{3}\right)=(1.2667,3.0)$。

$G_2$类：为$\omega_4$值，即$m_2=(3,1)$.

## 7.3 K均值聚类法

重新计算每个样本点到$G_1, G_2$聚类中心的距离

$d_{11} = \|\omega_1 - m_1\| = 0.2667$，$d_{12} = \|\omega_1 - m_2\| = 2.8284$；

$d_{21} = \|\omega_2 - m_1\| = 0.3073$，$d_{22} = \|\omega_2 - m_2\| = 2.6627$；

$d_{31} = \|\omega_3 - m_1\| = 0.2028$，$d_{32} = \|\omega_3 - m_2\| = 2.4759$；

$d_{41} = \|\omega_4 - m_1\| = 2.6466$，$d_{42} = \|\omega_4 - m_2\| = 0$；

所以，得新的划分为：$G_1 = \{\omega_1, \omega_2, \omega_3\}$，$G_2 = \{\omega_4\}$。

可见，新的划分与前面的相同，聚类中心没有改变，聚类结束。

## 7.3 K均值聚类法

```
#程序文件 Pex73.py
import numpy as np
from sklearn.cluster import KMeans
a=np.array([[1, 3],[1.5, 3.2],[1.3, 2.8],[3, 1]])
md=KMeans(n_clusters=2) # 构建模型
md.fit(a) # 求解模型
labels=1+md.labels_ # 提取聚类标签
centers=md.cluster_centers_ # 提取聚类中心,每一行是一个聚类中心
print(labels,'\n----------\n',centers)
```

## 7.3 K均值聚类法

### 3. 如何确定K均值聚类的聚类数k值

对于$K$均值聚类来说，如何确定簇数$k$值是一个至关重要的问题，为了解决这个问题，通常会选用探索法，即给定不同的$k$值，对比某些评估指标的变动情况，进而选择一个比较合理的$k$值。本节将介绍非常实用的两种评估方法，即簇内离差平方和拐点法和轮廓系数法。

## 7.3 K均值聚类法

### 3. 如何确定K均值聚类的聚类数k值

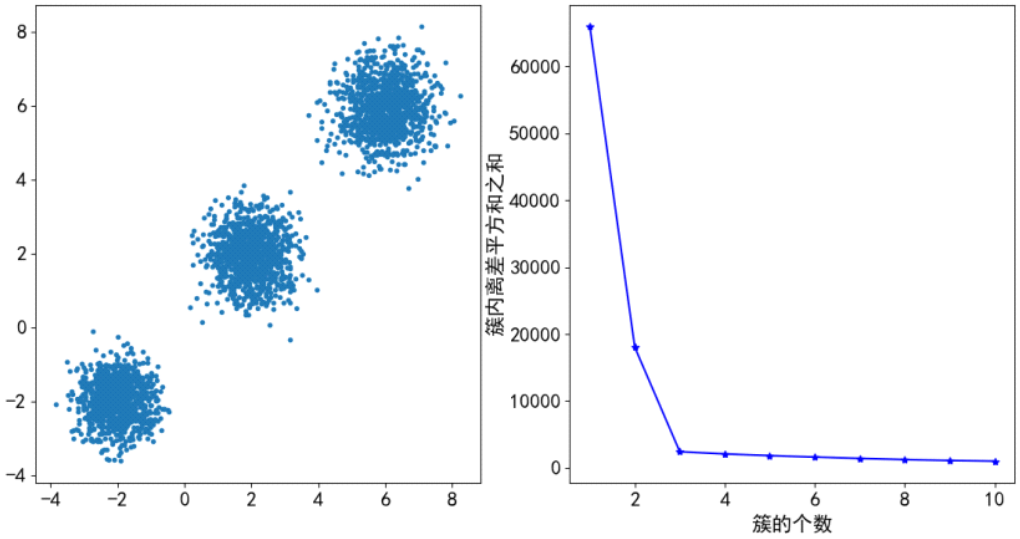
$$J_e = \sum_{i=1}^{C} \sum_{\omega \in G_i} \left\| \omega - m_i \right\|^2$$

**1.拐点法**

簇内离差平方和拐点法的思想很简单，就是在不同的$k$值下计算簇内离差平方和，然后通过可视化的方法找到“拐点”所对应的$k$值。重点关注的是斜率的变化，当斜率由大突然变小时，并且之后的斜率变化缓慢，则认为突然变换的点就是寻找的目标点，因为继续随着簇数$k$的增加，聚类效果不再有大的变化。

为了验证这个方法的直观性，这里随机生成三组二维正态分布数据，首先基于该数据绘制散点图。模拟的数据呈现三个簇，接下来基于这个模拟数据，使用拐点法，绘制簇的个数与总的簇内离差平方和之间的折线图。

## 7.3 K均值聚类法




（A）生成三个簇的样本点　　（B）拐点法选择合理的$k$值

图　模拟数据选择簇数$k$值

当簇的个数为 3 时形成了一个明显的拐点，3 之后的簇对应的簇内离差平方和的变动都很小，合理的$k$值应该为 3，与模拟的三个簇数据是吻合的。

## 7.3 K均值聚类法

```
#程序文件 Pz74
import numpy as np
import matplotlib.pyplot as plt; from sklearn.cluster import KMeans
mean=np.array([[-2, -2],[2, 2], [6,6]])
cov=np.array([[[0.3, 0], [0, 0.3]],[[0.4, 0], [0, 0.4]],[[0.5, 0], [0, 0.5]]])
x0=[]; y0=[];
for i in range(3):
    x,y=np.random.multivariate_normal(mean[i], cov[i],1000).T
    x0=np.hstack([x0,x]); y0=np.hstack([y0,y])
```

## 7.3 K均值聚类法

```
plt.rc('font',size=16); plt.rc('font',family='SimHei')
plt.rc('axes',unicode_minus=False); plt.subplot(121)
plt.scatter(x0,y0,marker='.') # 画模拟数据散点图
X=np.vstack([x0,y0]).T
TSSE=[]; K=10
for k in range(1,K+1):
    SSE = []
    md = KMeans(n_clusters=k); md.fit(X)
    labels = md.labels_; centers = md.cluster_centers_
    for label in set(labels):
        SSE.append(np.sum((X[labels == label,:] -centers[label,:])**2)
    TSSE.append(np.sum(SSE))
plt.subplot(122); plt.style.use('ggplot')
plt.plot(range(1,K+1), TSSE, 'b*-')
plt.xlabel('簇的个数'); plt.ylabel('簇内离差平方和之和'); plt.show()
```

## 7.3 K均值聚类法

### 3. 如何确定K均值聚类的聚类数k值

**2.轮廓系数法**

该方法综合考虑了簇的密集性与分散性两个信息，如果数据集被分割为理想的$k$个簇，那么对应的簇内样本会很密集，而簇间样本会很分散。

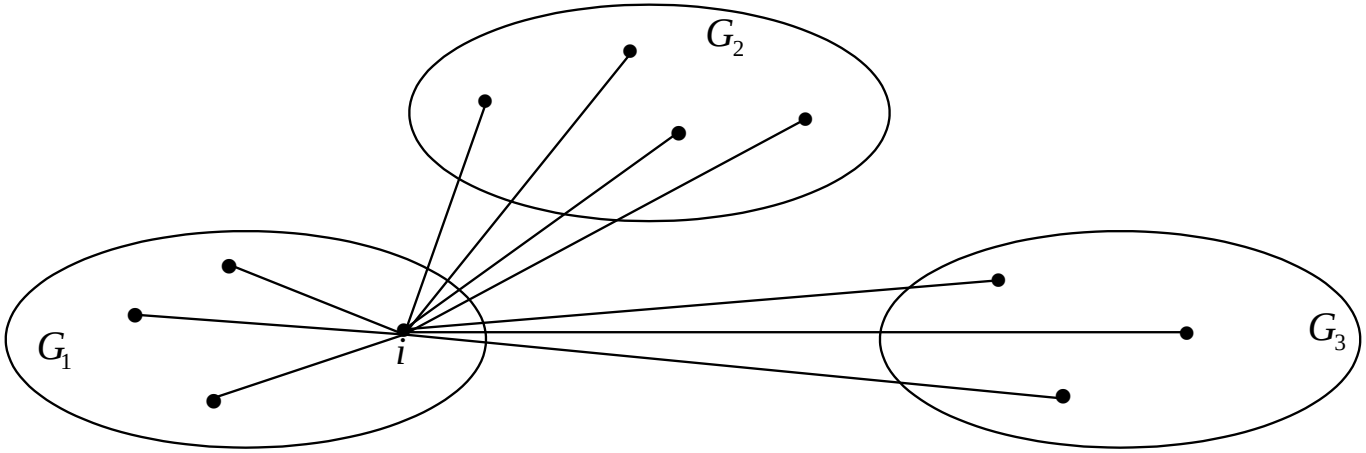



图　轮廓系数计算示意图

## 7.3 K均值聚类法

如图所示，假设数据集被拆分为 3 个簇$G_1,G_2,G_3$，样本点$i$对应的$a_i$值为所有$G_1$中其他样本点与样本点$i$的距离平均值；样本点$i$对应的$b_i$值分两步计算，首先计算该点分别到$G_2$和$G_3$中样本点的平均距离，然后将两个平均值中的最小值作为$b_i$的度量。

定义样本点$i$的轮廓系数

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)},$$

$k$个簇的总轮廓系数定义为所有样本点轮廓系数的平均值。

## 7.3 K均值聚类法

当总轮廓系数小于 0 时，说明聚类效果不佳；当总轮廓系数接近于 1 时，说明簇内样本的平均距离非常小，而簇间的最近距离非常大，进而表示聚类效果非常理想。

上面的计算思想虽然简单，但是计算量是很大的，当样本量比较多时，运行时间会比较长。有关轮廓系数的计算，可以直接调用 sklearn.metrics 中的函数 silhouette_score。需要注意的是，该函数接受的聚类簇数必须大于等于 2。

## 7.3 K均值聚类法

利用上面同样的模拟数据，画出的簇数与轮廓系数对应关系图如图所示，当$k$等于 3 时，轮廓系数最大，且比较接近于 1，说明应该把模拟数据聚为 3 类比较合理，同样与模拟数据的 3 个簇是吻合的。

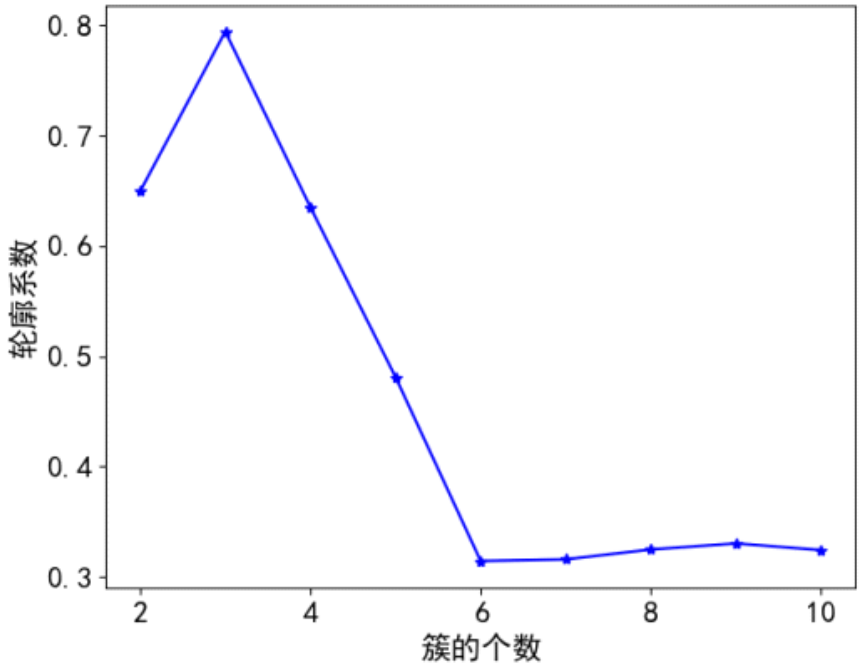



图 轮廓系数法选择合理的$k$值

## 7.3 K均值聚类法

```
#程序文件 Pz75.py
import numpy as np; import matplotlib.pyplot as plt
from sklearn.cluster import KMeans; from sklearn import metrics
X=np.load("data74.npy")
S=[]; K=10
for k in range(2,K+1):
    md = KMeans(k); md.fit(X)
    labels = md.labels_;
    S.append(metrics.silhouette_score(X,labels,metric='euclidean'))
#计算轮廓系数
```

## 7.3 K均值聚类法

```
plt.rc('font',size=16); plt.rc('font',family='SimHei')
plt.plot(range(2,K+1), S, 'b*-')
plt.xlabel('簇的个数'); plt.ylabel('轮廓系数'); plt.show()
```

## 7.3 K均值聚类法

### 4. K均值聚类的应用

在做 KMeans 聚类时需要注意两点:

(1) 聚类前必须指定具体的簇数$k$值，如果$k$值是已知的，可以直接调用 cluster 子模块中的 KMeans 函数，对数据集进行分割；如果$k$值是未知的，可以根据行业经验或前面介绍的两种方法确定合理的$k$值。

(2) 对原始数据集做必要的标准化处理。由于 KMeans 的思想是基于点之间的距离实现“物以聚类”的，所以，如果原始数据集存在量纲上的差异就必须对其进行标准化的预处理。数据集的标准化处理可以借助于 sklearn 子模块 preprocessing 中的 scale 函数或 minmax_scale 函数。

## 7.3 K均值聚类法

### 4. K均值聚类的应用

scale 函数的标准化公式为：

$$x^* = \frac{x - \mu}{\sigma},$$

minmax_scale 函数的标准化公式为

$$x^* = \frac{x - x_{\min}}{x_{\max} - x_{\min}},$$

其中，$\mu$、$\sigma$、$x_{\min}$、$x_{\max}$分别为$x$取值的均值、标准差、最大值和最小值。

iris 数据集是常用的分类实验数据集，下面我们用该数据集来验证 KMeans 聚类的效果。

## 7.3 K均值聚类法

例 7.4　Iris 数据集由 Fisher 于 1936 收集整理。Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个属性，数据格式如表所示。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

## 7.3 K均值聚类法

表　Iris 数据集数据（全部数据见数据文件 iris.csv）

| | Sepal_Length | Sepal_Width | Petal_Length | Petal_Width | Species |
|---|---|---|---|---|---|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3 | 1.4 | 0.2 | setosa |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 149 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 150 | 5.9 | 3 | 5.1 | 1.8 | virginica |

如表所示，数据集的前四个变量分别为花萼的长度、宽度及花瓣的长度、宽度，它们之间没有量纲上的差异，故无须对其做标准化处理，最后一个变量为鸢尾花所属的种类。如果将其聚为 3 类，所得结果为各簇样本量分别为 60，50，38。为了直观验证聚类效果，对比建模后的三类与原始数据三类的差异，绘制花瓣长度与宽度的散点图如图所示。

## 7.3 K均值聚类法

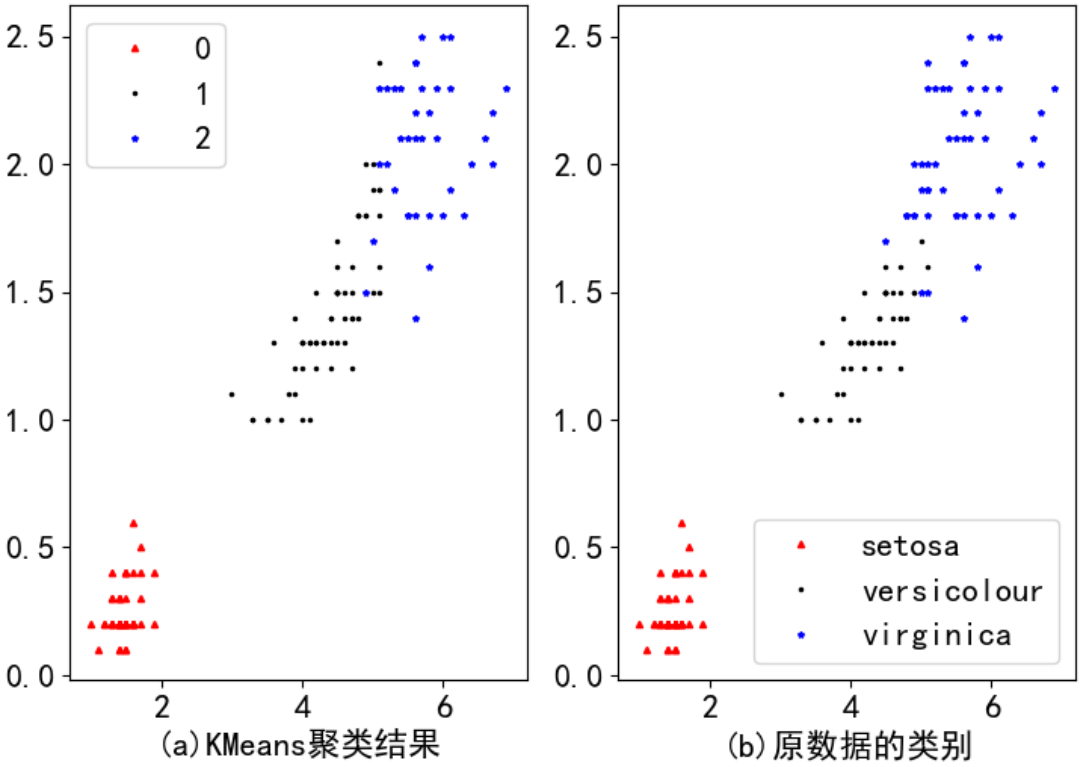



图　KMeans 聚类效果与原始类别的对比

## 7.3 K均值聚类法

```
#程序文件 Pex76.py
import numpy as np; import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
a=pd.read_csv("iris.csv")
b=a.iloc[:,:-1]
md=KMeans(3); md.fit(b)  #  构建模型并求解模型
labels=md.labels_ ;  centers=md.cluster_centers_
b['cluster']=labels  #  数据框 b 添加一个列变量 cluster
```

## 7.3 K均值聚类法

```
c=b.cluster.value_counts() # 各类频数统计
plt.rc('font',family='SimHei'); plt.rc('font',size=16)
str1=['^r','.k','*b']; plt.subplot(121)
for i in range(len(centers)):
    plt.plot(b['Petal_Length'][labels==i],b['Petal_Width']
             [labels==i], str1[i],markersize=3,label=str(i))
    plt.legend(); plt.xlabel("(a)KMeans 聚类结果")
plt.subplot(122); str2=['setosa','versicolour','virginica']
```

## 7.3 K均值聚类法

```
ind=np.hstack([np.zeros(50),np.ones(50),2*np.ones(50)])
for i in range(3):
    plt.plot(b['Petal_Length'][ind==i],b['Petal_Width'][ind==i],
                    str1[i],markersize=3,label=str2[i])
    plt.legend(loc='lower right'); plt.xlabel("(b)原数据的类别")
plt.show()
```

# 第七章 聚类分析

## 7.4 谱聚类法

### 1. 谱聚类基本思想

近年来，谱聚类法日渐流行，并被应用到诸多领域，例如图像分析、数据挖掘以及文本聚类。

谱聚类是从图论中演化出来的算法，后来在聚类中得到了广泛的应用。它的主要思想是把所有的数据看做空间中的点，这些点之间可以用边连接起来。

距离较远的两个点之间的边权重值较低，而距离较近的两个点之间的边权重值较高，通过对所有数据点组成的图进行切图，让切图后不同的子图间边权重和尽可能的低，而子图内的边权重和尽可能的高，从而达到聚类的目的。

# 第七章 聚类分析

## 7.4 谱聚类法

### 1. 谱聚类基本思想

因此，谱聚类的学习需要对图论、代数和矩阵分析都有比较深入的基础。

## 7.4 谱聚类法

### 2. 谱聚类实现流程

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}=\begin{bmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_n^T \end{bmatrix}$$

$$X_i=(x_{i1},x_{i2},\cdots,x_{im})^T,i=1,2,\cdots,n$$

给定一个数据集，由n个m维观测数据组成。目标是把该数据集中的n个观测数据分成k组，使得组内的数据点彼此相似，而组间的点彼此相异。

接下来，计算数据对Xi和Xj之间的关联性或相似度，从而构成关联矩阵 A (相似矩阵、邻接矩阵)。

## 7.4 谱聚类法

### 2. 谱聚类实现流程

**矩阵A的元素由下式给出：**

$$A_{ij} = \exp[-\frac{\| X_i - X_j \|^2}{2\sigma^2}], i \neq j \qquad A_{ii} = 0$$

**其中**

$$X_i = (x_{i1}, x_{i2}, \cdots, x_{im})^T, i = 1, 2, \cdots, n$$

**比例因子 $\sigma$ 决定了关联性随着Xi和Xj之间距离下降的快慢程度。Ng，Jordan和Weiss在2002年提出了一种自动选择比例因子的方法。 高斯核参数，取值对聚类效果是有影响的，1.0**

## 7.4 谱聚类法

### 2. 谱聚类实现流程

$$A=\begin{bmatrix} 0 & A_{12} & \cdots & A_{1n} \\ A_{21} & 0 & \cdots & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{n1} & A_{n2} & \cdots & 0 \end{bmatrix}_{n\times n}$$

构造对角矩阵D（**度矩阵**），其中第ii个元素值是矩阵A的第i行所有元素之和，即

$$D_{ii}=\sum_{j=1}^{n}A_{ij}$$

## 7.4 谱聚类法

### 2. 谱聚类实现流程

$$D=\begin{bmatrix} D_{11} & 0 & \cdots & 0 \\ 0 & D_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & D_{22} \end{bmatrix}_{n\times n}$$

**接着，构造矩阵$L$ （n阶方阵，拉普拉斯矩阵）**

$$L = D^{-1/2} A D^{-1/2}$$

## 7.4 谱聚类法

### 2. 谱聚类实现流程

接下来，要找出矩阵L的特征向量和特征值。令L的前k个最大的特征值分别为

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$$

其对应的特征向量分别记为（均为n维列向量）

$$u_1, u_2, \cdots, u_k$$

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix}$$

矩阵U由上述特征向量构成（矩阵U为n行k列），

$$U = [u_1, u_2, \cdots, u_k]$$

## 7.4 谱聚类法

### 2. 谱聚类实现流程

上式表明，U的列向量对应于L的特征向量。通常，所有的特征向量都是单位长度，并且彼此之间正交。

接着，构造矩阵Y。把U的每一行都进行单位化，使其具有单位长度。Y的元素由下式给出：

$$y_{ij} = \frac{u_{ij}}{\sqrt{\sum_j u_{ij}^2}}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix}$$

$$Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1k} \\ y_{21} & y_{22} & \cdots & y_{2k} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nk} \end{bmatrix}$$

## 7.4 谱聚类法

### 2. 谱聚类实现流程

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}=\begin{bmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_n^T \end{bmatrix}$$

$$Y=\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1k} \\ y_{21} & y_{22} & \cdots & y_{2k} \\ \vdots & \vdots & & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nk} \end{bmatrix}$$

**接下来就可以聚类了。把Y的n行看作是k维空间的n个观测量，然后采用k均值法对n个观测量进行聚类。如果Y的第i行被分配到第j个类，则把Xi分配到第j个类.**

## 7.4 谱聚类法

### 3. 谱聚类具体步骤

上述方法是Ng，Jordan和Weiss（NJW）提出的，NJW算法具体步骤如下：

Step1. 生成关联矩阵A (相似矩阵)；

$$A_{ij} = \exp[-\frac{\| X_i - X_j \|^2}{2\sigma^2}], i \neq j$$

Step2. 利用A中的元素构造矩阵D；

$$D_{ii} = \sum_{j=1}^{n} A_{ij}$$

Step3. 利用公式计算L；

$$L = D^{-1/2} A D^{-1/2}$$

Step4. 计算L的特征向量和特征值；

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \qquad u_1, u_2, \cdots, u_k$$

Step5. 把L的前k个特征值所对应的特征向量存入矩阵U $U = [u_1, u_2, \cdots, u_k]$

## 7.4 谱聚类法

### 3. 谱聚类具体步骤

Step6. 把U的行向量归一化，使其具有单位长度，归一化的矩阵命名为Y；

$$y_{ij} = \frac{u_{ij}}{\sqrt{\sum_j u_{ij}^2}}$$

Step7. 对Y的行向量进行K均值聚类；

Step8. 如果Y的第i行被分配到第j个类，则把Xi分配到第j个类。

## 7.4 谱聚类法

### 3. 谱聚类具体步骤

优点：对于高维数据具有降维作用，比传统聚类算法好

缺点：相似度矩阵的构造方法不同，会有影响；

在数据重叠区域的聚类效果可能不理想。

## 7.4 谱聚类法

### 4. 谱聚类的Python实现

Sklearn.cluster.SpectralClustering 函数提供了一种基于谱聚类算法的无监督学习模型，其作用是将数据集中的数据聚为不同的类别。该算法通过对数据样本间的相似度矩阵进行特征分解，得到样本的降维表示，在低维空间中聚类。

SpectralClustering(n_clusters, n_init, gamma)

n_clusters：int，指定聚类类别数，默认值为 8。

n_init：int，指定聚类中心初始化的次数，默认值为 10。

gamma：float，调节相似度矩阵稠密程度的参数，当参数值较大时矩阵更稠密，特征值和特征向量的估计更加准确。默认值为 1.

## 7.4 谱聚类法

### 4. 谱聚类的Python实现

例 7.4　Iris 数据集由 Fisher 于 1936 收集整理。Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个属性，数据格式如表所示。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

## 7.4 谱聚类法

### 4. 谱聚类的Python实现

```
#程序文件Pex77.py
import numpy as np; import pandas as pd
from sklearn.cluster import SpectralClustering
import matplotlib.pyplot as plt
a=pd.read_csv("iris.csv")
b=a.iloc[:,:-1]
md=SpectralClustering(n_clusters=3, random_state=0, gamma=0.5);
md.fit(b)  #构建模型并求解模型
labels=md.labels_ ;
b['cluster']=labels  #数据框b添加一个列变量cluster
c=b.cluster.value_counts()  #各类频数统计
```

## 7.4 谱聚类法

### 4. 谱聚类的Python实现

```
plt.rc('font',family='SimHei'); plt.rc('font',size=16)
str1=['^r','.k','*b']; plt.subplot(121)
for i in range(3):
    plt.plot(b['Petal_Length'][labels==i],b['Petal_Width']
             [labels==i], str1[i],markersize=3,label=str(i))
    plt.legend(); plt.xlabel("(a)KMeans聚类结果")
plt.subplot(122); str2=['setosa','versicolour','virginica']
ind=np.hstack([np.zeros(50),np.ones(50),2*np.ones(50)])
for i in range(3):
    plt.plot(b['Petal_Length'][ind==i],b['Petal_Width'][ind==i],
             str1[i],markersize=3,label=str2[i])
    plt.legend(loc='lower right'); plt.xlabel("(b)原数据的类别")
plt.show()
```