第 17 章 聚类分析

聚类分析(cluster analysis)是一种常见的非监督学习方法。

假定样本数据为$\{\mathbf{x}_i\}_{i=1}^n$，其中$\mathbf{x}_i=(x_{i1}\cdots x_{ip})'$为$p$维特征向量。

希望将样本数据分为若干“小组”(subgroup) 或“聚类”(cluster)，使得同一聚类的观测值尽可能相似，而不同聚类的观测值尽可能不同。

聚类分析也称为**相似性分组**(similarity grouping)。

应该如何定义“相似性”？一般使用欧氏距离(Euclidean distance)。

例 市场细分(market segmentation)。假设企业拥有大量的消费者数据，包括消费者性别、年龄、收入、职业、购买习惯等。为了投放广告或推销产品，应如何将消费者市场细分成若干聚类？

作为一种非监督学习方法，聚类分析带有一定的主观性，因为我们并不知道观测值真正的类别归属。

聚类分析有不同的方法，可能得到不同的结果。

最流行的两种聚类分析方法为“K均值聚类”(*K*-means clustering)与“分层聚类”(hierarchical clustering)。

17.1 K 均值聚类的思想

K 均值聚类(*K*-means clustering)起源于波兰数学家 Steinhaus(1956)，而成型于美国统计学家 McQueen(1967)。

它是一种自上而下(top-down)的聚类方法，须预先确定样本中的聚类数目，即 K 的具体取值，比如根据经验或试错。

假设数据可分为 K 类(K 已知)。

将观测值下标$\{1,\cdots,n\}$划分(partition)为K个不相交的集合$\{C_1,\cdots,C_K\}$。

其中，$C_k \bigcap C_{k'} = \varnothing \quad (\forall\, k \neq k')$，而且$C_1 \bigcup C_2 \bigcup \cdots \bigcup C_K = \{1,\cdots,n\}$(所有聚类之并集为整个样本)。

每个观测值都有唯一的类别归属。

$i \in C_k$意味着第i个观测值$\mathbf{x}_i$属于第k个聚类。

希望每个聚类的“组内变动”(within-cluster variation)越小越好。

记聚类k的均值或中心位置(cluster centriod)为

$$\mathbf{c}_k \equiv \frac{1}{|C_k|}\sum_{i \in C_k} \mathbf{x}_i \qquad (17.1)$$

其中，$|C_k|$表示聚类k共有几个观测值。

样本中共有K个中心位置(均值)，故此算法称为“K均值聚类”(K-means clustering)。

对于聚类k中的某观测值$\mathbf{x}_i$ $(i \in C_k)$，称其到聚类中心位置的离差$(\mathbf{x}_i - \mathbf{c}_k)$为“误差”(error)。

将聚类k中所有误差的平方和加总，即为聚类k的**误差平方和**(Sum of Squared Errors，简记 SSE)：

$$\text{SSE}_k \equiv \sum_{i \in C_k} \left\| \mathbf{x}_i - \mathbf{c}_k \right\|^2 \qquad (17.2)$$

其中，$\left\| \mathbf{x}_i - \mathbf{c}_k \right\|$为欧氏距离，即各分量平方和的开根号。

将所有聚类的误差平方和加总，即可得到全样本的误差平方和：

$$\text{SSE} \equiv \sum_{k=1}^{K} \sum_{i \in C_k} \left\| \mathbf{x}_i - \mathbf{c}_k \right\|^2 \qquad (17.3)$$

其中，SSE 也称为**组内平方总和**(Total Within Sum of Squares)。

寻找对于样本下标集$\{1, \cdots, n\}$的一个划分$\{C_1, \cdots, C_K\}$，使得全样本的误差平方和最小化：

$$\min_{C_1, \cdots, C_K} \text{SSE} = \sum_{k=1}^{K} \sum_{i \in C_k} \left\| \mathbf{x}_i - \mathbf{c}_k \right\|^2 \qquad (17.4)$$

但此问题很难求解。因为在将n个观测值分到K个聚类时，由于每个观测值都有K个聚类可供选择，故共有$\underbrace{K \times K \times \cdots \times K}_{n个} = K^n$种分法。

例如，对于$K = 5$而$n = 100$的小样本，就有多达$5^{100} \approx 7.89 \times 10^{69}$种的分类方法。

一般很难以找到最小化问题(17.4)的全局最小解(global minimum)。

17.2 K 均值聚类的算法

K 均值聚类使用如下的迭代算法来近似求解。

首先，如果我们知道聚类的中心位置$\mathbf{c}_k$，则可计算聚类的分布，即每个观测值应归属哪一聚类(归入最近的类别)，这称为“分配过程”。

其次，一旦知道聚类的分布，则可更新每个聚类的中心位置$\mathbf{c}_k$，这称为“更新过程”。

将以上两个步骤迭代，即可找到最优化问题的局部最小解(local minimum)。

K 均值算法的具体步骤如下：

1、随机选择每个聚类的中心位置(初始化)

2、循环执行以下两步，直至收敛：

(1) 将每个观测值重新分配到离它最近的聚类(分配)；

(2) 更新每个聚类的中心位置(更新)。

在每次循环迭代时，全样本的误差平方和 SSE 肯定下降；因为在每步迭代时，所有观测值都被分配到更近的聚类。

如果 SSE 不再下降，则达到一个局部最小解。

在以上第 1 步初始化时，也可以先将所有观测值随机分到K个聚类，再进行迭代。

由于K均值算法仅能找到局部最小值(未必是全局最小值)，故其聚类结果依赖于初始的聚类中心位置(或聚类分布）。

如果使用不同的初始聚类中心位置，则可能得到不同的局部最小解。

在实践中，一般会尝试多个不同的初始聚类中心位置(设置不同的随机数种子)，然后选择最佳结果，最小化全样本 SSE。

我们用模拟数据来演示K均值算法的收敛过程。

究竟什么是聚类？

在理论上，不同聚类的数据可视为来自不同的总体。

在下面的模拟中，第 1 类数据为来自标准正态$N(0,1)$的 50 个随机观测值

第 2 类数据为来自正态分布$N(5,1)$的 50 个随机观测值，散点图参见图 17.1：

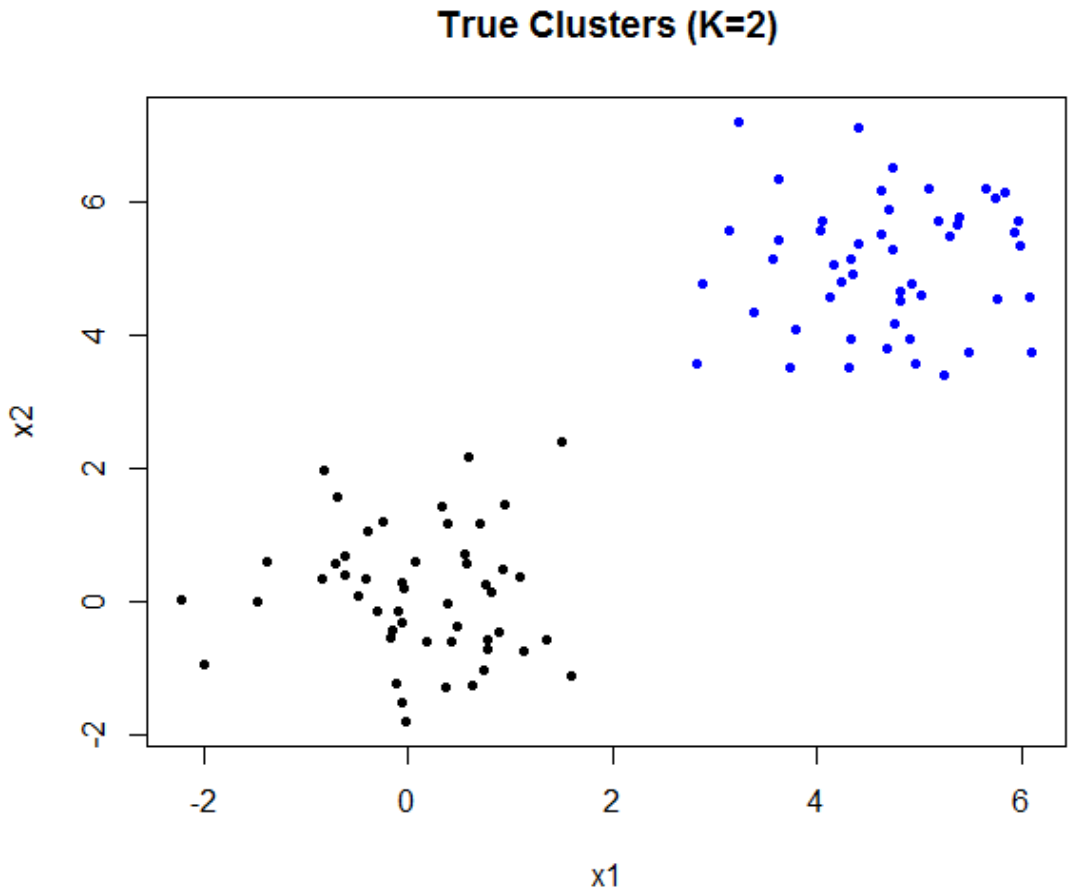

图 17.1 模拟生成的两类数据

在图 17.1 中，黑点表示第 1 类数据，而蓝点表示第 2 类数据，二者分别聚集在二维特征空间(x_1, x_2)的不同区域。

此模拟数据非常简单，用肉眼即可将数据分为两类，这里仅作为演示目的。

现实数据的特征向量一般存在于更高维空间，无法直接画图可视化；而且不同类别之间也通常有交叠。

使用 R 包 `animation` 演示K均值算法的收敛过程，设聚类数目为$K = 2$(即真实的聚类数目)，并以黑色与蓝色区分两类数据，结果参见图 17.2。

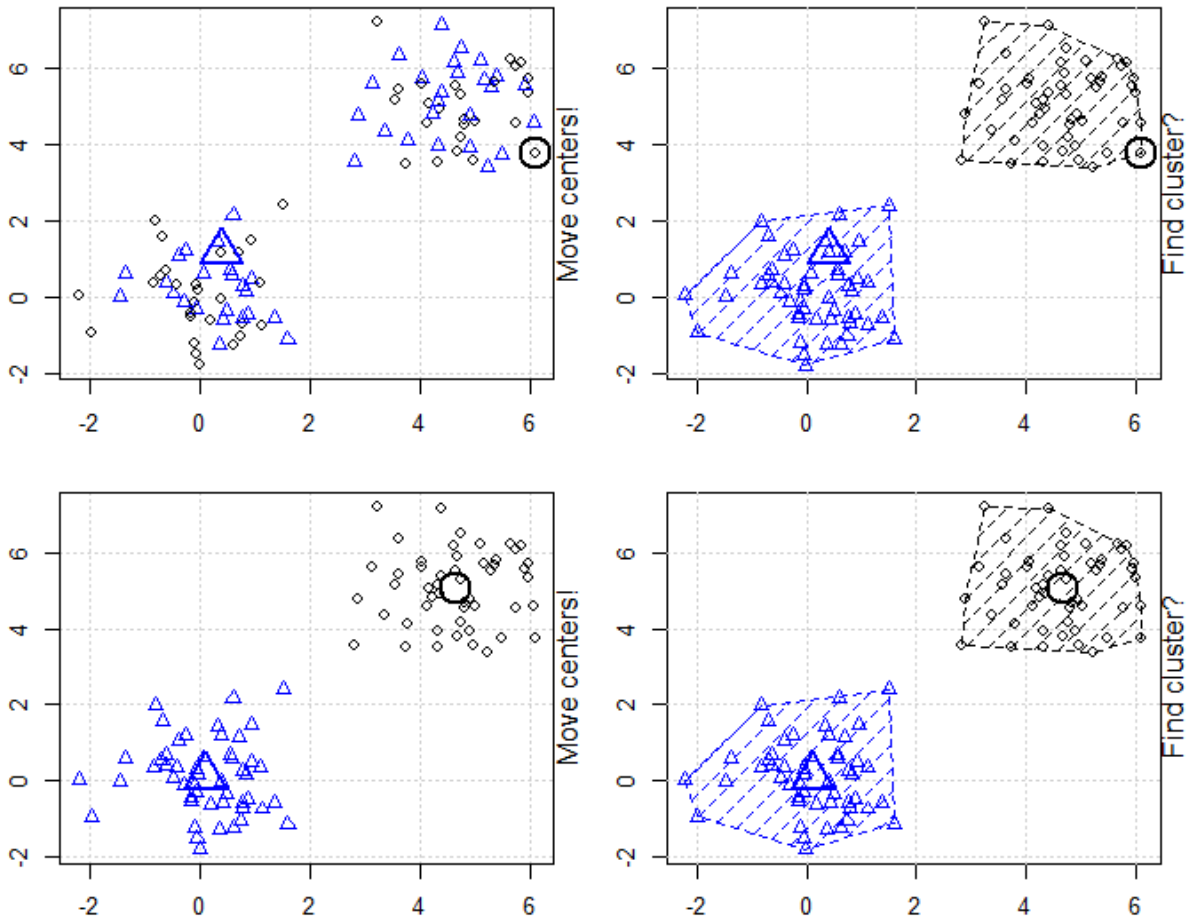

图 17.2 K 均值算法的收敛过程

在图 17.2 中，左上角为K均值算法的第 1 步，即初始化。它随机选择了初始的聚类中心位置(图中“大圆”与“大三角”)；以及随机的初始聚类分布(图中“小圆”与“小三角”)。

右上角为K均值算法的第 2 步，即分配过程(图中纵轴显示 Find cluster?)，根据当前聚类的中心位置，将所有观测值归入最近的聚类中。

左下角为第 3 步，即更新过程(图中纵轴显示 Move centers!)，重新计算聚类的中心位置。

右下角为第 4 步，再次进行分配过程，但发现所有观测值的聚类归属都不再改变，故算法在第 4 步即收敛。

17.3 如何选择 K

使用K均值聚类的一个关键问题是，应如何选择聚类数目K。

一种方法是，根据研究者的专业领域知识(domain knowledge)来选择，但比较主观。

另一种方法是，尝试不同的聚类数目K，考察全样本 SSE 的下降幅度，然后用**手肘法**(elbow method)进行判断，参见图 17.3。

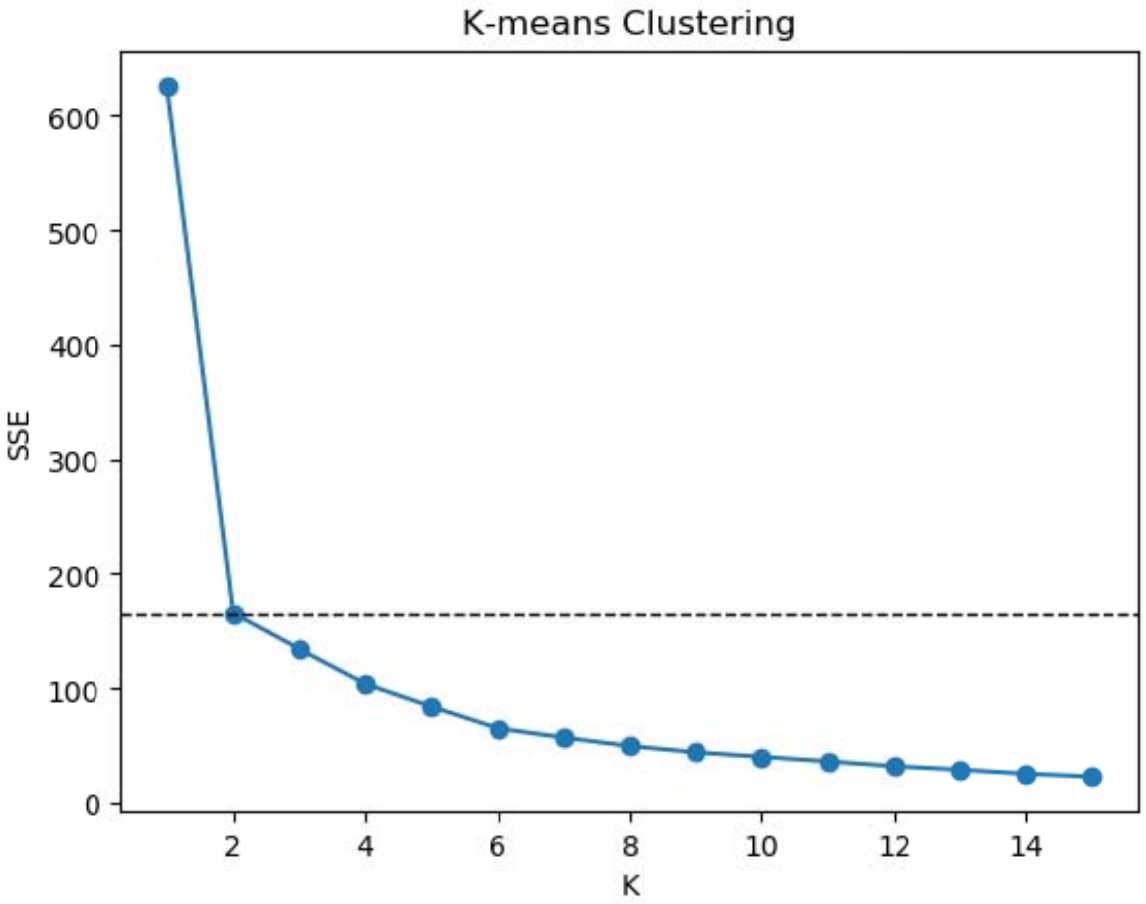

图 17.3 使用手肘法选择K(参见第 17.4 节)

手肘法依然有主观性；比如，如何判断手肘的拐弯处，可能有争议。

另一种相对客观的方法为，使用**信息准则**(information criterion)来选择K，比如 AIC 信息准则：

$$\min_{K} \ \mathrm{AIC}(K)=\mathrm{SSE}(K)+2pK \qquad (17.5)$$

其中，上式右边的第 1 项$\mathrm{SSE}(K)$为全样本误差平方和(作为K的函数)，是对模型拟合优度的奖励。

如果让K变大，增多参数，虽可使得$\mathrm{SSE}(K)$下降，但易导致过拟合。

在极端情形下，$K=n$，则每个观测值都是一个聚类，使得训练误差 $\text{SSE}(K)=0$，但没有意义。

上式右边的第 2 项为对模型复杂度(参数过多)的惩罚；其中，由于$\mathbf{c}_k$为 p维，故参数总数为pK。也可使用对于模型复杂度惩罚更为严厉的 BIC 信息准则：

$$\min_K \ \text{BIC}(K)=\text{SSE}(K)+\ln(n)\cdot pK \qquad (17.6)$$

在上式中，由于通常$\ln(n)>2$，故 BIC 对于模型复杂度的惩罚比 AIC 更为严厉，可能导致选择更小的聚类数目K。

17.4 分层聚类

K均值聚类需要预先指定聚类数目K，但K可能不好确定。

另一种聚类方式为**分层聚类**(hierarchical clustering)。

分层聚类是一种自下而上(bottom-up)的逐次聚类方法(agglomerative clustering)。

它无须预先设定聚类数目，而从个体层面开始寻找最近的邻居，然后层层往上，形成“树状图”(dendrogram)，最后才选择聚类数目。

假设在二维的特征空间中，有五个观测值A, B, C, D, E，参见图17.4。

首先，由于A与C相距最近，故二者首先聚为一类$\{A, C\}$。

其次，由于D与E相距第二近，故二者接着聚为一类$\{D, E\}$。

第三，观测值B与聚类$\{A, C\}$最近，故这三个观测值聚为一类$\{A, B, C\}$。

最后，所有五个观测值都聚为一类$\{A, B, C, D, E\}$。

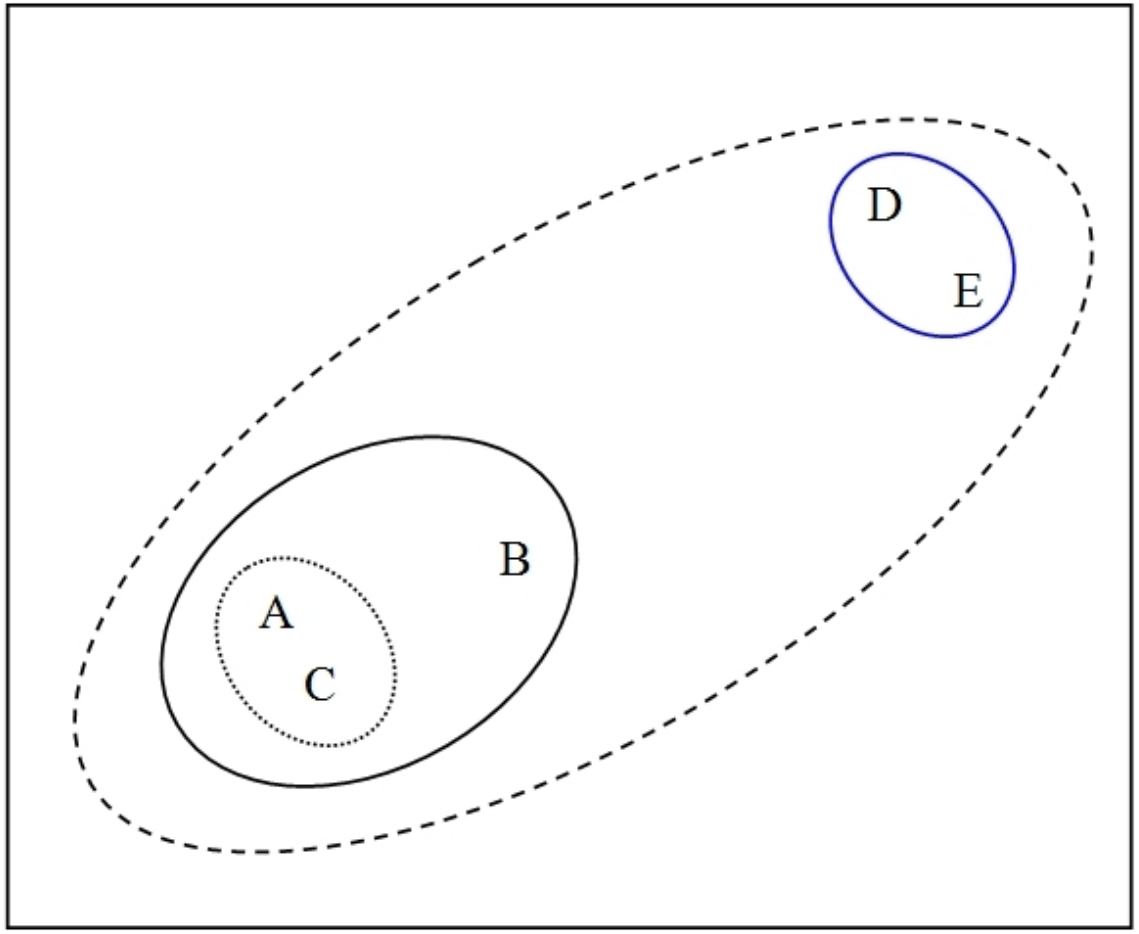

图 17.4 在特征空间进行分层聚类

在高维的特征空间中，一般无法画类似于图 17.4 的分层聚类图。但依然可画如图 17.5 所示的**树状图**(dendrogram)。

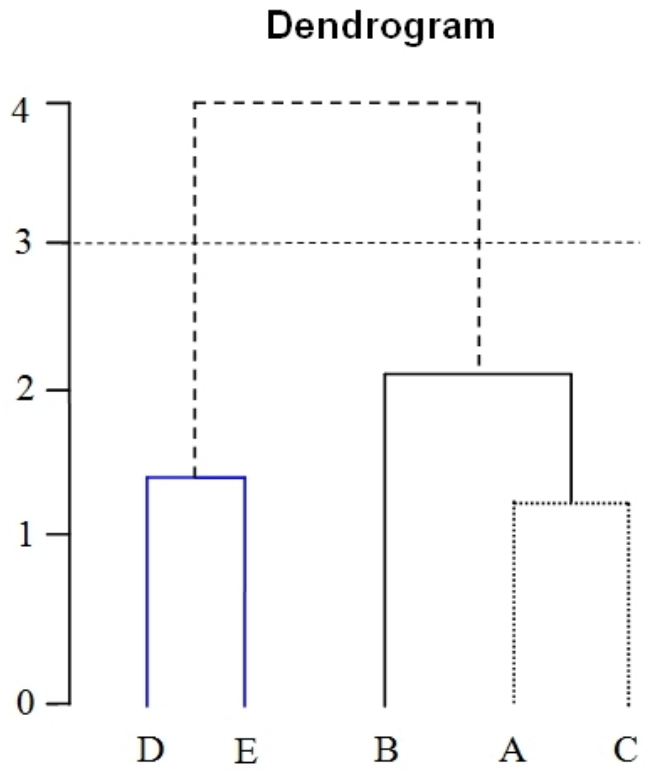

图 17.5 分层聚类的树状图

得到树状图后，在不同的高度“砍树”(cut tree)，即可得到不同的聚类数目K。

例如，在高度为 3 的地方砍树，则会得到两个聚类$\{D, E\}$与$\{A, B, C\}$。

在高度为 2 的地方砍树，则会得到三个聚类$\{D, E\}$、$\{B\}$与$\{A, C\}$。

在高度为 1 的地方砍树，则会得到五个聚类$\{A\}$，$\{B\}$，$\{D\}$，与$\{E\}$(每个观测值构成一个聚类)。

总结起来，分层聚类算法的步骤如下：

(1) 以每个观测值作为一个聚类；

(2) 找到距离最近的两个聚类，然后合并(merge)；

(3) 重复第(2)步，直至所有观测值都在同一聚类中。

如何度量两个聚类之间的距离或**相异度**(dissimilarity)？若两个聚类分别仅含一个观测值，则两个聚类间的距离就是两个观测值之间的欧氏距离。

但当聚类中包含多个观测值时，如何度量两个聚类之间的距离，则有不同的方法，称为**连接**(linkage)。

第一种方法是度量两个聚类中所有观测值之间的最大距离(maximal inter-cluster dissimilarity)，称为**完全连接**(complete linkage)，参见图 17.6。

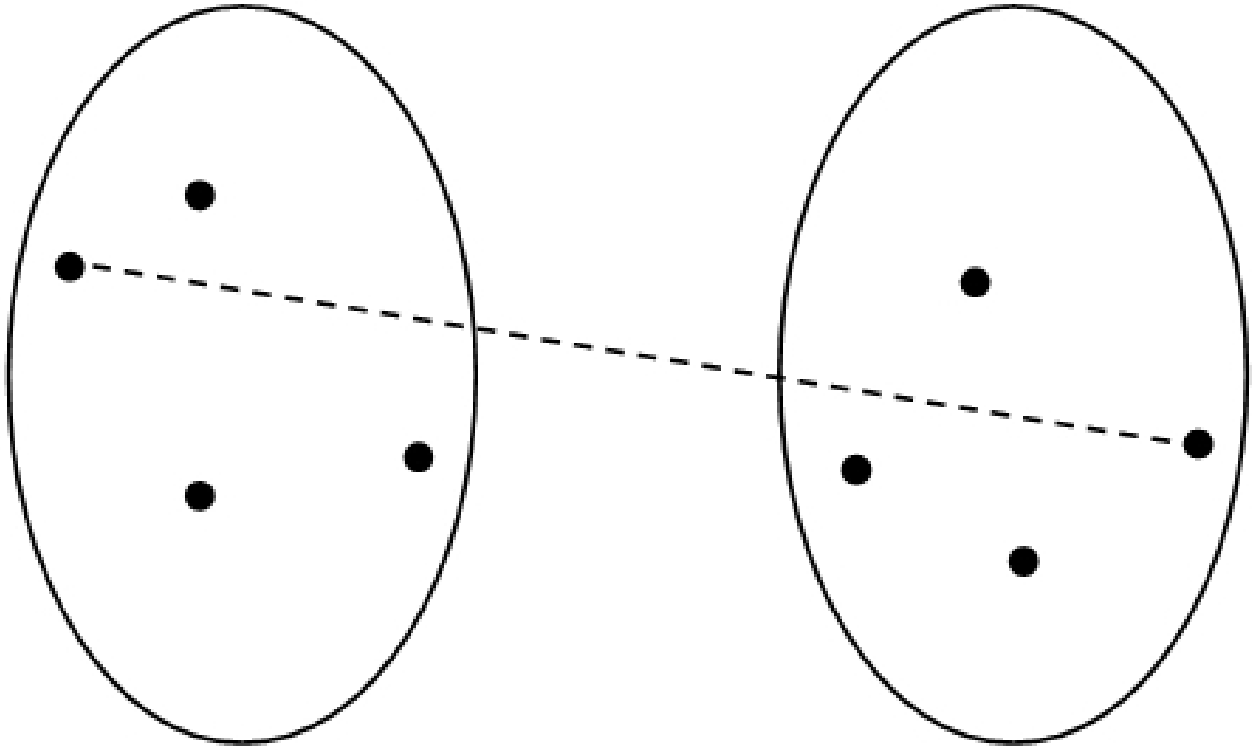

图 17.6 完全连接

第二种方法是度量两个聚类中所有观测值之间的最小距离(minimal inter-cluster dissimilarity)，称为**单一连接**(single linkage)，参见图 17.7。

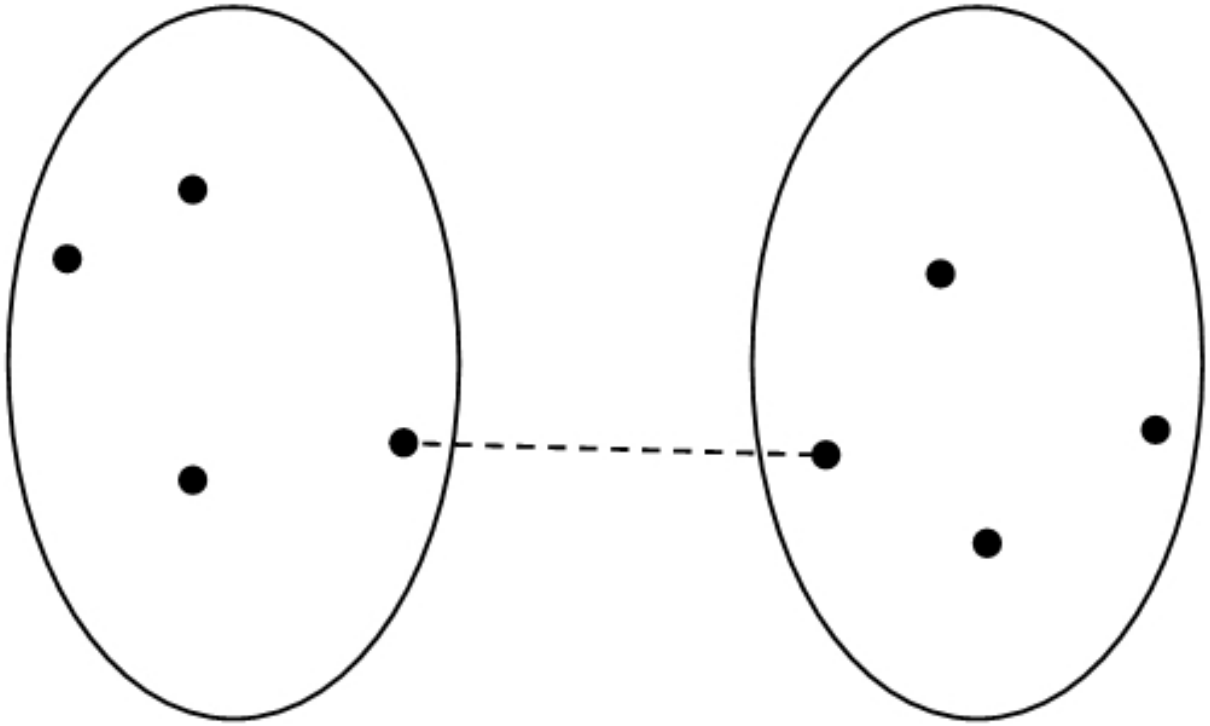

图 17.7 单一连接

第三种方法是度量两个聚类的中心位置(centroid)之间的距离，称为**中心连接**(centriod linkage)，参见图 17.8。

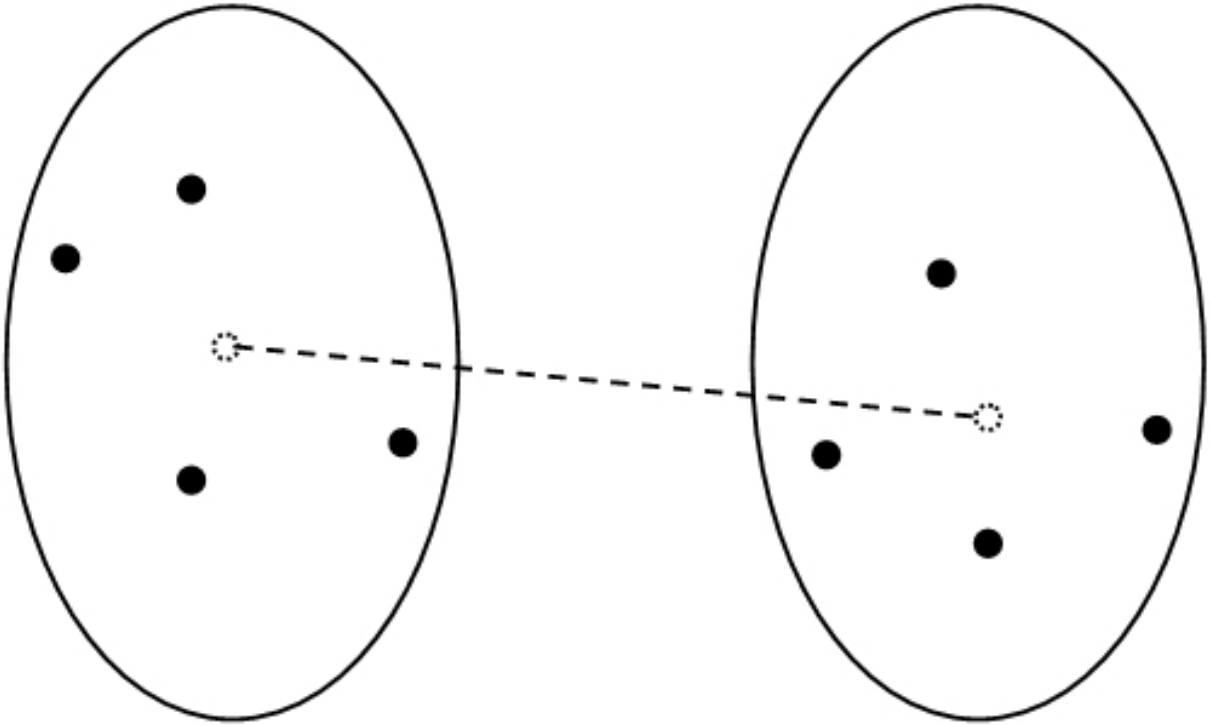

图 17.8 中心连接

第四种方法是度量两个聚类所有观测值之间的平均距离(mean inter-cluster dissimilarity)，称为**平均连接**(average linkage)。

在此例中，每个聚类有 4 个观测值，故共有$4\times4=16$对观测值，计算这 16 对观测值之间的相异度(all pairwise dissimilarity)，再进行平均即为“平均连接”，参见图 17.9。

分层聚类使用的连接函数，对于所得的树状图可能有较大影响，参见下文的案例。

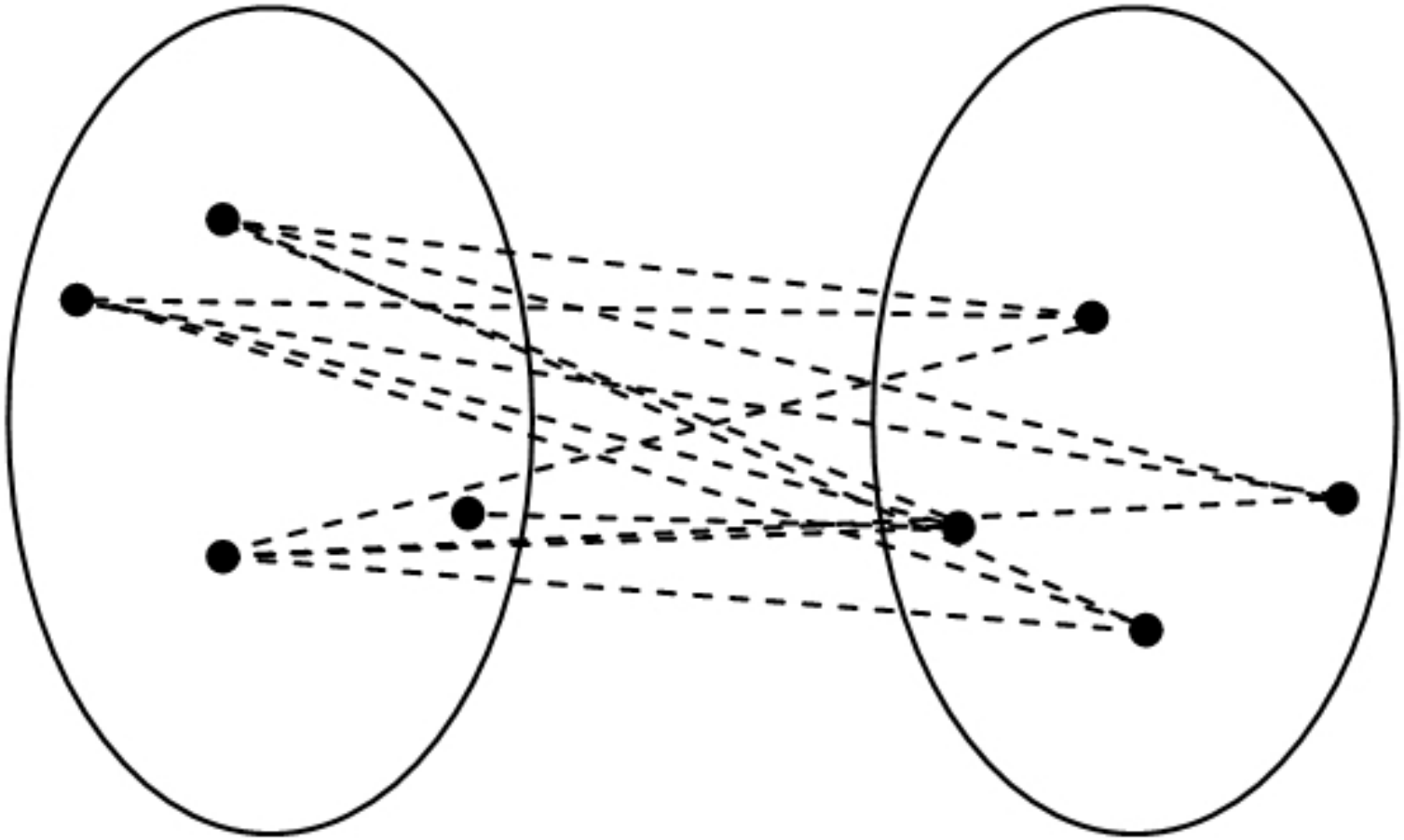

图 17.9 平均连接

在实践中，由于聚类分析的前提为对距离的度量，故一般应先将变量标准化，使得所有变量的均值都为 0，而标准差均为 1。

这样可避免少数变量对于距离的影响太大，导致聚类结果出现扭曲。

另外，对于聚类分析，也可进行“稳健性检验”(robustness check)，即从样本中随机选择“子样本”(subsamples)进行聚类，考察所得聚类结果是否类似。

17.5 基于相关系数的距离指标

聚类分析一般以欧氏距离作为“相异度”(dissimilarity)的度量标准。

但有时更适合使用其他指标，比如“基于相关系数的距离”(correlation-based distance)。

如果使用基于相关系数的距离指标，则两个观测值$\mathbf{x}_i = (x_{i1} \cdots x_{ip})'$与$\mathbf{x}_j = (x_{j1} \cdots x_{jp})'$之间的相关系数越大，其距离就越近：

$$dist(\mathbf{x}_i, \mathbf{x}_j) \equiv 1 - corr(\mathbf{x}_i, \mathbf{x}_j) \qquad (17.7)$$

其中，$corr(\mathbf{x}_i, \mathbf{x}_j)$为观测值$\mathbf{x}_i$与观测值$\mathbf{x}_j$的相关系数。

如果$corr(\mathbf{x}_i, \mathbf{x}_j) = 1$，则二者的距离为 0。

反之，如果$corr(\mathbf{x}_i, \mathbf{x}_j) = -1$，则达到最远距离 2。

即使两个观测值的相关系数很接近于 1，二者的欧氏距离也可能很大。

比如，$\mathbf{x}_j = 10\mathbf{x}_i$，此时二者的相关系数为 1，但在特征空间的欧氏距离却较远。

以 `iris` 数据为例。

如果我们画第 50 个样例与第 99 个样例的“观测值轮廓”(observation profile)，会发现二者的观测值轮廓形状(shape of observation profile)很接近，因为它们都属于“杂色鸢尾花”(versicolor)，参见图 17.10。

假设遇到一株特别大的杂色鸢尾花，它的各方面尺度都比第 99 个观测值大一倍。

若按照相关系数来度量距离，则此超大杂色鸢尾花与第 99 个观测值的距离为 0；但若以欧氏距离度量，则二者距离较远。

更直观地，下面画图展示。

首先，导入所需模块：

```
In [1]: import matplotlib.pyplot as plt
   ...: import seaborn as sns
```

其次，载入 iris 数据，提取第 50 与 99 个样例的特征变量，分别记为 iris50 与 iris99：

```
In [2]: iris = sns.load_dataset('iris')
   ...: iris50 = iris.iloc[50, :-1]
   ...: iris99 = iris.iloc[99, :-1]
```

将第 99 个样例的特征变量都乘以 2，并记为 iris99_2：

```
In [3]: iris99_2 = 2 * iris99
```

同时画这些样例的观测值轮廓图，结果参见图 17.10：

```
In [4]: fig = plt.figure(figsize=(10, 6))
   ...: plt.plot(iris50, 'ko-', label='Observation 50')
   ...: plt.plot(iris99, 'bo-', label='Observation 99')
   ...: plt.plot(iris99_2, 'ko--', label='Hypothetical
                Observation')
   ...: plt.title('Observation Profile')
   ...: plt.legend()
   ...: plt.tight_layout()
```

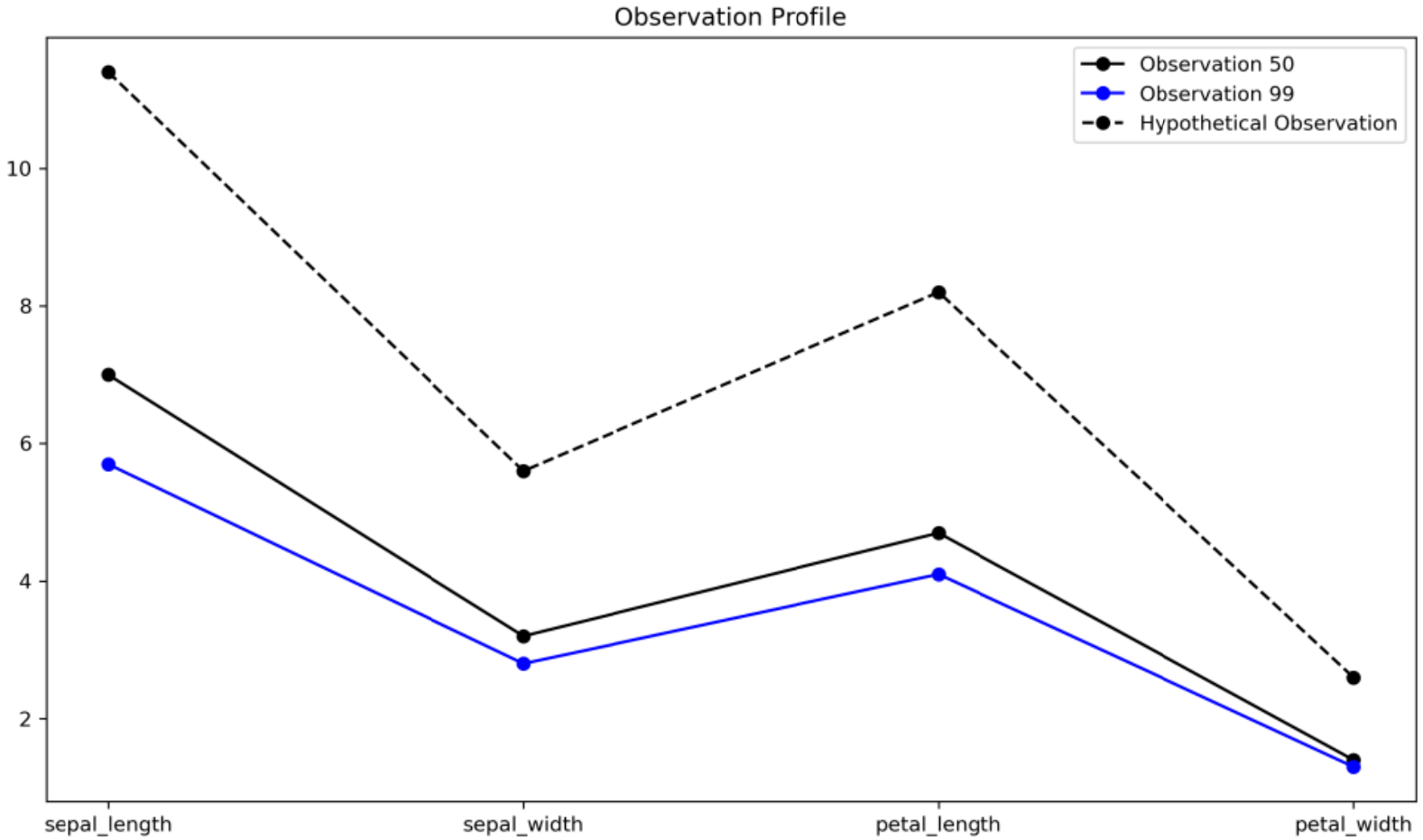

图 17.10 观测值的轮廓图

作为例子，假设电商根据消费者的购买偏好进行聚类分析。

如果以相关系数作为距离指标，则只要两个消费者的购买模式(shape of profile)相近，即可能归入同一类，而无论其绝对购买量。

总之，使用何种指标度量距离，取决于研究或应用的需要。

17.6 K 均值聚类的 Python 案例

我们先以模拟数据演示K 均值聚类的 Python 操作。

使用模拟数据的好处在于，知道数据生成过程(以及真实的聚类)，且可在二维的特征空间进行可视化。

首先，导入本章所需模块：

```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import seaborn as sns
   ...: from sklearn.cluster import KMeans
   ...: from sklearn.cluster import
                AgglomerativeClustering
   ...: from sklearn.preprocessing import
                StandardScaler
   ...: from mpl_toolkits.mplot3d import Axes3D
   ...: from scipy.cluster.hierarchy import linkage,
```

```
                dendrogram, fcluster
   ...: from scipy.spatial.distance import squareform
```

在此案例中，第 1 类数据为来自$N(0,1)$的 50 个随机观测值； 而第 2 类数据为来自$N(3,1)$的 50 个随机观测值。

先生成第 1 类模拟数据：

```
In [2]: np.random.seed(1)
   ...: cluster1 = np.random.normal(0, 1,
                                    100).reshape(-1, 2)
```

```
...: cluster1 = pd.DataFrame(cluster1, columns=['x1',
                                                'x2'])
...: cluster1['y'] = 0
```

其中，第 1 个命令设定随机数种子。 第 2 个命令从$N(0,1)$随机抽取 100 个观测值，并使用 `reshape()`方法，将其形状改为50×2矩阵(其中，`reshape(-1,2)`表示不设定行数，而由 Python 自动确定行数)。

第 3 个命令将此矩阵设为数据框 `cluster1`，并添加变量名 `x1` 与 `x2`。

最后 1 个命令在数据框中添加变量 `y`，取值全部为 0。

展示数据框 cluster1 的前 5 个观测值：

```
In [3]: cluster1.head()
Out[3]:
         x1        x2   y
0  1.624345 -0.611756  0
1 -0.528172 -1.072969  0
2  0.865408 -2.301539  0
3  1.744812 -0.761207  0
4  0.319039 -0.249370  0
```

类似地，生成第 2 类模拟数据：

```
In [4]: np.random.seed(10)
   ...: cluster2 = np.random.normal(3, 1,
                                    100).reshape(-1, 2)
   ...: cluster2 = pd.DataFrame(cluster2, columns=['x1',
                                'x2'])
   ...: cluster2['y'] = 1
```

其中，第 2 个命令从$N(3,1)$随机抽取 100 个观测值，并将其形状改为50×2矩阵。第 3 个命令将此矩阵设为数据框 cluster2，并添加变量名 x1 与 x2。最后 1 个命令在数据框中添加变量 y，取值全部为 1。

展示数据框 cluster2 的前 5 个观测值：

```
In [5]: cluster2.head()
Out[5]:
         x1        x2   y
0  4.331587  3.715279   1
1  1.454600  2.991616   1
2  3.621336  2.279914   1
3  3.265512  3.108549   1
4  3.004291  2.825400   1
```

将 cluster1 与 cluster2 纵向合并为数据框 data，并展示其形状：

```
In [6]: data = pd.concat([cluster1, cluster2])
   ...: data.shape
Out[6]: (100, 3)
```

其中，pd.concat()函数表示将数据框连接或并置(concatenate)，默认以纵轴方向(axis=0)进行合并。

若要进行横向合并，可加入参数“axis=1”(详见第 18 章)。

更直观地，画模拟数据的散点图，并根据数据类别上色，结果参见图 17.11：

```
In [7]: sns.scatterplot(x='x1', y='x2', data=cluster1,
                        color='k', label='y=0')
   ...: sns.scatterplot(x='x1', y='x2', data=cluster2,
                        color='b', label='y=1')
   ...: plt.legend()
   ...: plt.title('True Clusters (K=2)')
```

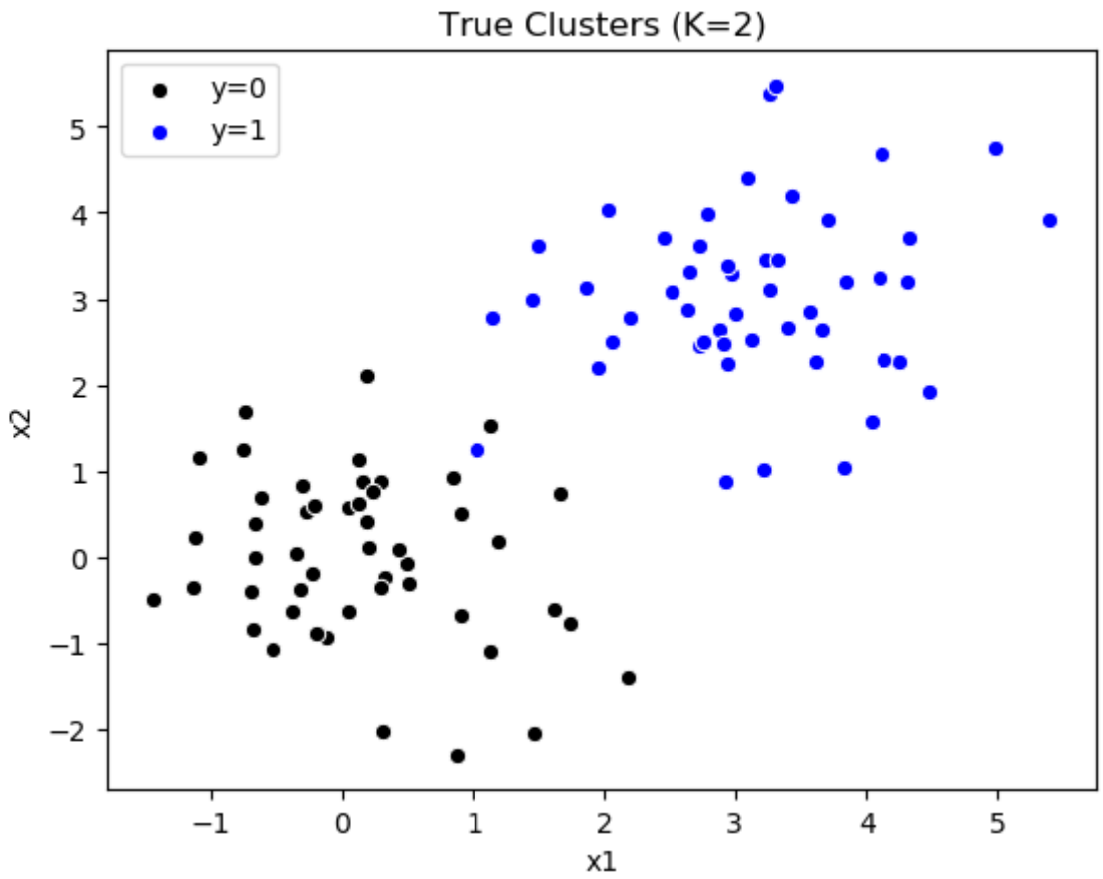

图 17.11 包含两个聚类的模拟数据

从图 17.11 可见，两类数据在二维的特征空间略有交叠。取出数据矩阵 X 与分类变量 y：

```
In [8]: X = data.iloc[:, :-1]
   ...: y = data.iloc[:, -1]
```

下面，使用 sklearn 的 KMeans 类进行K均值聚类分析：

```
In [9]: model = KMeans(n_clusters=2, random_state=123,
n_init=20)
```

其中，参数“n_clusters=2”设定聚类数目$K=2$，而参数“n_init=20”表示进行 20 次K均值聚类(随机设定不同的初始化)，并选择最优结果。

此命令创建了 KMeans 类的一个实例 model。

使用 fit()方法进行估计：

```
In [10]: model.fit(X)
Out[10]:
KMeans(algorithm='auto', copy_x=True, init='k-means++',
       max_iter=300, n_clusters=2, n_init=20,
       n_jobs=None, precompute_distances='auto',
       random_state=123, tol=0.0001, verbose=0)
```

通过 model 的 labels_属性，可得每个样例的聚类归属：

```
In [11]: model.labels_
Out[11]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

通过 model 的 cluster_centers_属性，可得每个聚类的中心位置：

```
In [12]: model.cluster_centers_
Out[12]:
array([[0.15504503, 0.00842918],
       [3.16672098, 3.07129467]])
```

通过 model 的 inertia_属性，可得“组内平方总和”(Total Within Sum of Squares)：

```
In [13]: model.inertia_
Out[13]: 165.69714015317402
```

更直观地，通过“混淆矩阵”将聚类分析的预测结果与真实聚类进行对比：

```
In [14]: pd.crosstab(y, model.labels_,
rownames=['Actual'],
                     colnames=['Predicted'])
Out[14]:
Predicted   0   1
Actual
0          50   0
1           1  49
```

结果显示，在 100 个观测值中，只有一个观测值被错误归类。

将聚类分析的结果画散点图，并以聚类结果上色：

```
In [15]: sns.scatterplot(x='x1', y='x2',
                         data=data[model.labels_==0],
                         color='k', label='y=0')
    ...: sns.scatterplot(x='x1',y='x2',
                         data=data[model.labels_==1],
                         color='b', label='y=1')
    ...: plt.legend()
    ...: plt.title('Estimated Clusters (K=2)')
```

其中，第 1 个命令的参数“data=data[model.labels_==0]”选出满足条件“model.labels_==0”(归入第 0 类) 的观测值，而第 2 个命令的参数“data=data[model.labels_==1]”则选出满足条件“model.labels_==1”(归入第 1 类)的观测值，结果参见图 17.12。

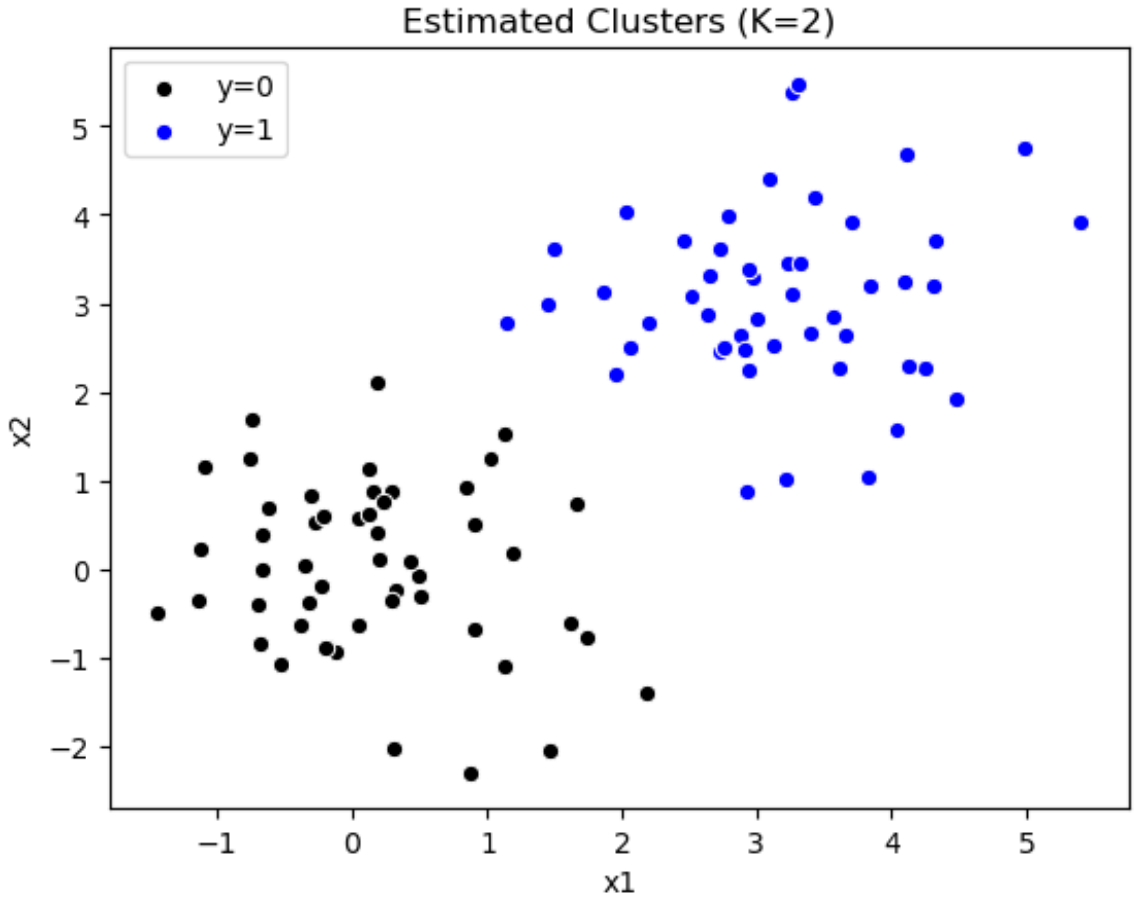

图 17.12 K均值聚类的估计结果

下面，假设聚类数目$K=3$，再次进行K均值聚类分析，并显示聚类结果：

```
In [16]: model = KMeans(n_clusters=3, random_state=2,
n_init=20)
    ...: model.fit(X)
    ...: model.labels_
Out[16]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 2, 0, 0, 2, 2, 2, 0, 2,
       0, 2, 2, 0, 0, 2, 2, 2, 2, 0, 0, 2, 0, 0, 2, 0, 2, 2, 0, 2, 2, 2,
       2, 2, 0, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

更直观地，画图展示所估计的三个聚类，结果参见图 17.13：

```
In [17]: sns.scatterplot(x='x1', y='x2',
                         data=data[model.labels_==0],
                         color='k', label='y=0')
    ...: sns.scatterplot(x='x1', y='x2',
                         data=data[model.labels_==1],
                         color='b', label='y=1')
    ...: sns.scatterplot(x='x1', y='x2',
                         data=data[model.labels_==2],
                   color='cornflowerblue', label='y=2')
    ...: plt.legend()
    ...: plt.title('Estimated Clusters (K=3)')
```

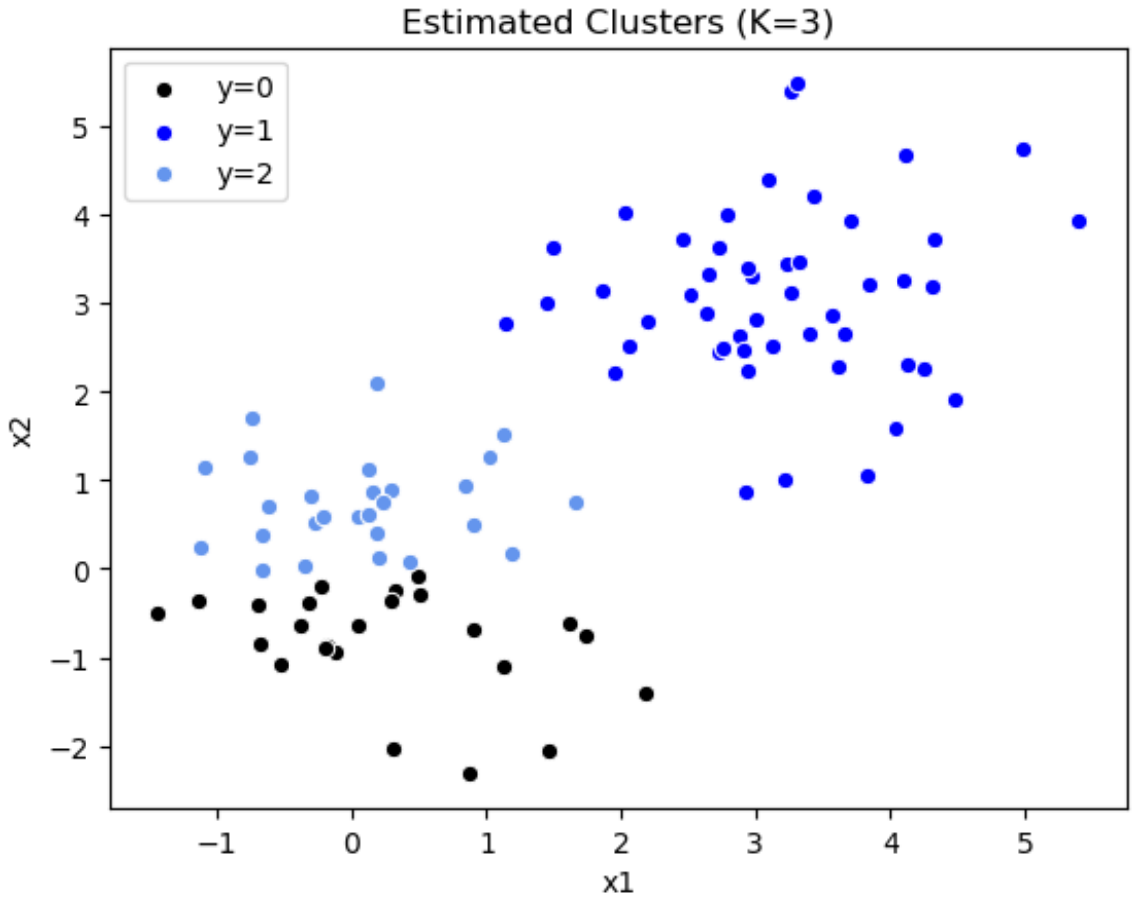

图 17.13 K均值聚类的估计结果($K=3$)

从图 17.13 可见，由于设定$K=3$，故聚类分析得到 3 个不同的聚类。

但这并不意味着，数据中真有 3 个聚类(此模拟数据仅包含 2 个聚类)。

进一步，尝试以 4 个聚类($K=4$)进行估计，并展示聚类结果：

```
In [18]: model = KMeans(n_clusters=4, random_state=3,
n_init=20)
    ...: model.fit(X)
    ...: model.labels_
Out[18]:
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 3, 3, 1, 3, 3, 1, 1, 1, 3, 1,
       3, 1, 1, 3, 3, 1, 1, 1, 1, 3, 3, 1, 3, 3, 1, 3, 1, 1, 3, 1, 1, 1,
```

```
       1, 1, 3, 1, 1, 1, 2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 1, 2, 2, 2, 2, 0,
       0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2,
       0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0])
```

更直观地，画散点图展示所估计的四个聚类，结果参见图 17.14：

```
In [19]: sns.scatterplot(x='x1', y='x2',
                         data=data[model.labels_==0],
                         color='b', label='y=0')
   ...: sns.scatterplot(x='x1', y='x2',
                        data=data[model.labels_==1],
                        color='cornflowerblue',
                        label='y=1')
```

```
...: sns.scatterplot(x='x1', y='x2',
                     data=data[model.labels_==2],
                    color='darkblue', label='y=2')
...: sns.scatterplot(x='x1', y='x2',
                     data=data[model.labels_==3],
                     color='k', label='y=3')
...: plt.legend()
...: plt.title('Estimated Clusters (K=4)')
```

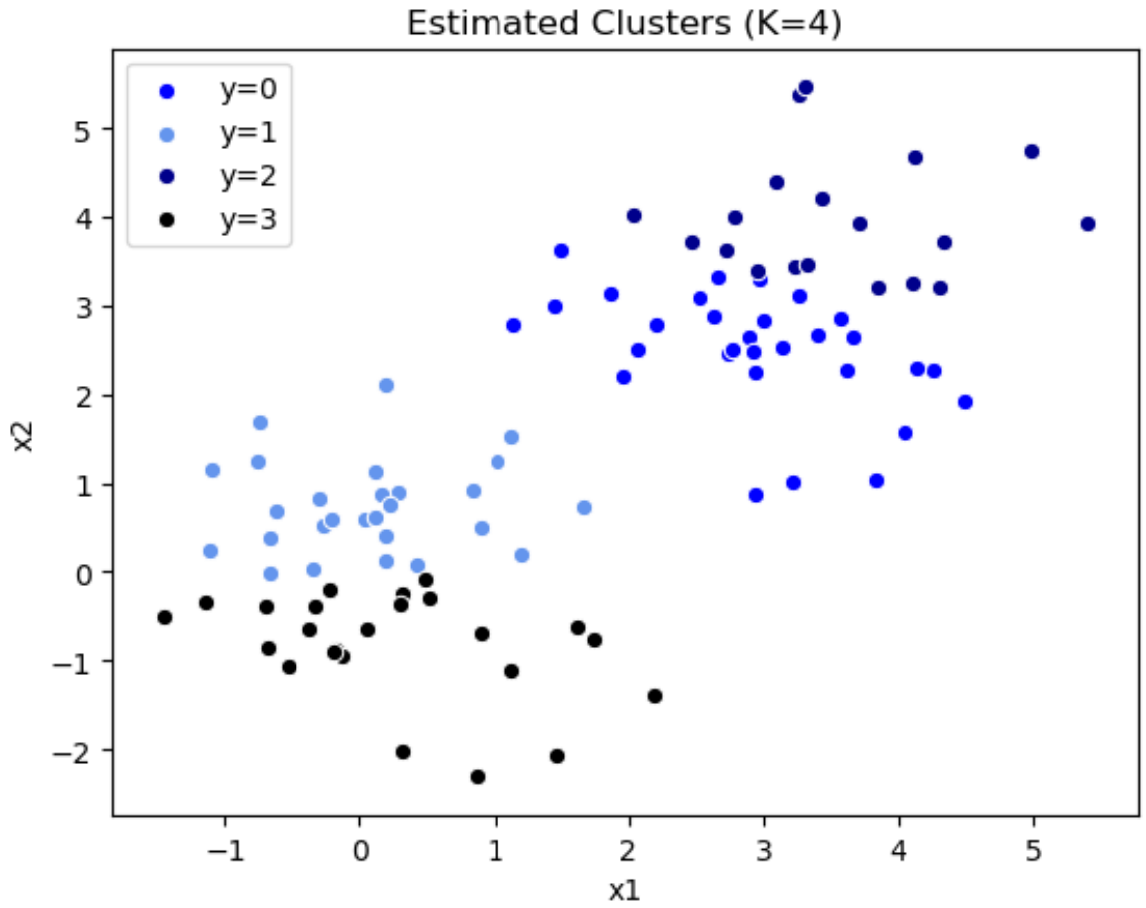

图 17.14 K均值聚类的估计结果($K=4$)

下面，我们考察不同的“n_init”参数(n_init=1 或 30)，对于目标函数(组内平方总和)最小值的影响(仍假设有 4 个聚类)：

```
In [20]: model = KMeans(n_clusters=4, random_state=4,
                        n_init=1)
    ...: model.fit(X)
    ...: model.inertia_
Out[20]: 115.33891221039043
```

```
In [21]: model = KMeans(n_clusters=4, random_state=4,
                        n_init=30)
    ...: model.fit(X)
    ...: model.inertia_
Out[21]: 103.83149722946976
```

结果显示，如果设定参数“n_init=1”(仅做 1 次初始化)，其组内平方总和(SSE)为 115.3389，大于设定参数“n_init=30”(进行 30 次初始化)的组内平方总和 103.8315。

SSE = 115.3389 仅为局部最小值，并非全局最小值。

那么，SSE =103.8315 是否为全局最小值呢？

由于此样本较小($n=100$)，不妨设定参数“n_init=1000”：

```
In [22]: model = KMeans(n_clusters=4, random_state=4,
                        n_init=1000)
    ...: model.fit(X)
    ...: model.inertia_
Out[22]: 103.83149722946976
```

尽管使用了 1000 次不同的初始化，SSE 的最小值依然为 103.8315，故这很可能就是全局最小值。

在此案例中，究竟聚类数目K应取何值？

尽管真实的聚类数目$K=2$，但希望由数据来告诉我们。

首先，使用手肘法。

为此，针对$K=1,\cdots,15$，使用 for 循环，计算相应的误差平方和 SSE：

```
In [23]: sse = []
    ...: for k in range(1,16):
    ...:     model = KMeans(n_clusters=k, random_state=1,
                            n_init=20)
    ...:     model.fit(X)
    ...:     sse.append(model.inertia_)
```

其中，第 1 个命令初始化误差平方和 sse 为一个空的列表，依次进行 K 均值聚类($K=1,\cdots,15$)，并将所得误差平方和添入列表 sse。

展示所得误差平方和 sse：

```
In [24]: sse
Out[24]:
[626.7960530195322,
 165.69714015317402,
 134.29161574012807,
 103.87529585375258,
 83.9333877973445,
 65.13561308660752,
```

```
56.98826806584698,
49.54135363043872,
44.080487929834405,
40.06178737509699,
36.060325231908465,
31.855863623276058,
28.793206786003438,
25.254202357796395,
22.73767108394651]
```

更直观地，画图展示聚类个数与误差平方和的关系，结果参见上文的图 17.3：

```
In [25]: plt.plot(range(1, 16), sse, 'o-')
    ...: plt.axhline(sse[1], color='k', linestyle='--',
                     linewidth=1)
    ...: plt.xlabel('K')
    ...: plt.ylabel('SSE')
    ...: plt.title('K-means Clustering')
```

从图 17.3 可知，应选择$K=2$，因为 SSE 的“手肘”在$K=2$处拐弯。

下面使用 AIC 信息准则选择聚类数目K。计算 AIC 信息准则，并展示其结果：

```
In [26]: aic = sse + 2 * 2 * np.arange(1, 16)
    ...: aic
Out[26]:
array([630.79605302, 173.69714015, 146.29161574, 119.87529585,
       103.9333878 ,  89.13561309,  84.98826807,  81.54135363,
        80.08048793,  80.06178738,  80.06032523,  79.85586362,
        80.79320679,  81.25420236,  82.73767108])
```

计算 AIC 信息准则的最小值，以及相应的位置索引：

```
In [27]: min(aic)
Out[27]: 79.85586362327606

In [28]: np.argmin(aic)
Out[28]: 11
```

更直观地，画图展示聚类数目与 AIC 的关系，结果参见图 17.15：

```
In [29]: plt.plot(range(1, 16), aic,'o-')
    ...: plt.axvline(np.argmin(aic) + 1, color='k',
                     linestyle='--', linewidth=1)
    ...: plt.xlabel('K')
    ...: plt.ylabel('AIC')
    ...: plt.title('K-means Clustering')
```

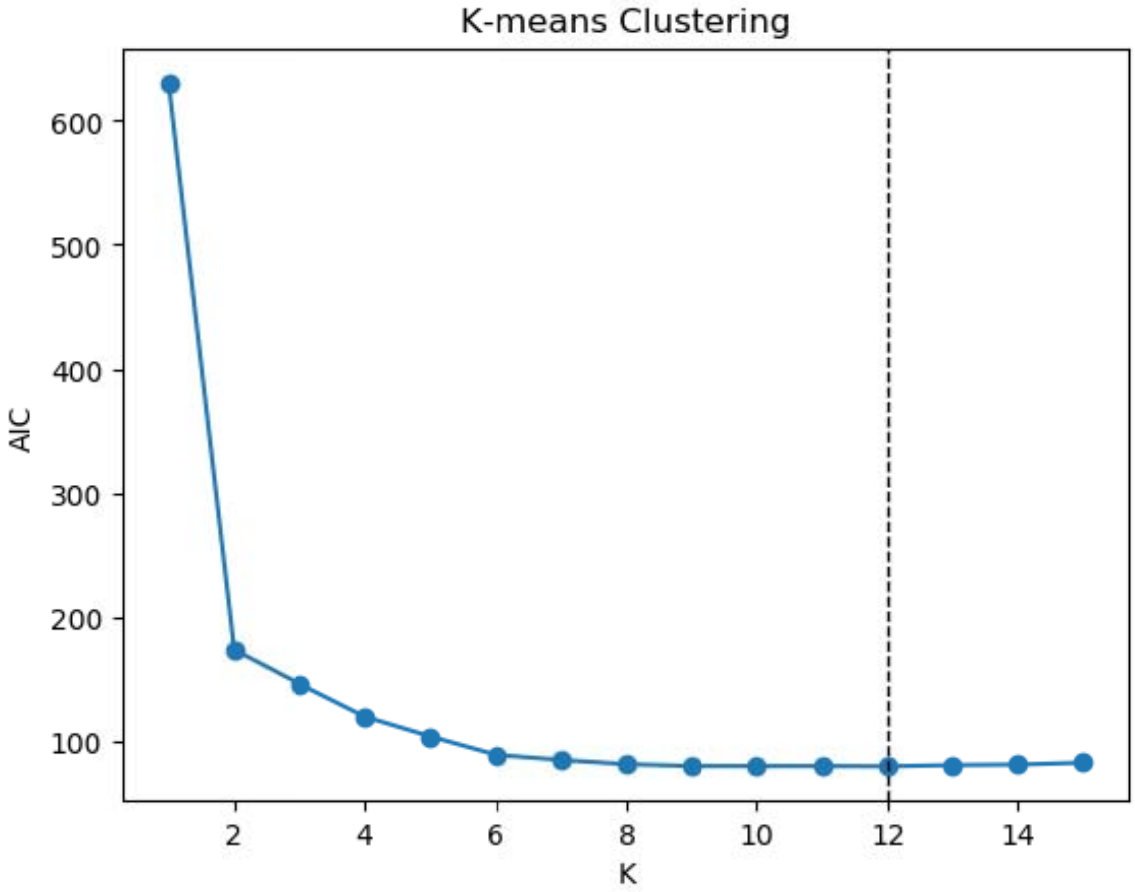

图 17.15 使用 AIC 准则选择K

结果显示，AIC 准则选择$K=12$，显然聚类数目太多，导致过拟合(因为真实$K=2$)。这是因为 AIC 准则对于模型过于复杂的惩罚不够严厉。

下面计算 BIC 信息准则，并展示结果:

```
In [30]: bic = sse + 2 * np.log(100) * np.arange(1, 16)
    ...: bic
Out[30]:
array([636.00639339, 184.1178209 , 161.92263686, 140.71665734,
       129.98508966, 120.39765532, 121.46065067, 123.22407661,
       126.97355128, 132.16519109, 137.37406932, 142.37994809,
       148.52763162, 154.19896757, 160.89277666])
```

计算 BIC 信息准则的最小值，以及相应的位置索引：

```
In [31]: min(bic)
Out[31]: 120.39765531846461
In [32]: np.argmin(bic)
Out[32]: 5
```

更直观地，画图展示聚类数目与 BIC 的关系，结果参见图 17.16：

```
In [33]: plt.plot(range(1, 16), bic, 'o-')
    ...: plt.axvline(np.argmin(bic) + 1, color='k',
                     linestyle='--', linewidth=1)
    ...: plt.xlabel('K')
    ...: plt.ylabel('BIC')
    ...: plt.title('K-means Clustering')
```

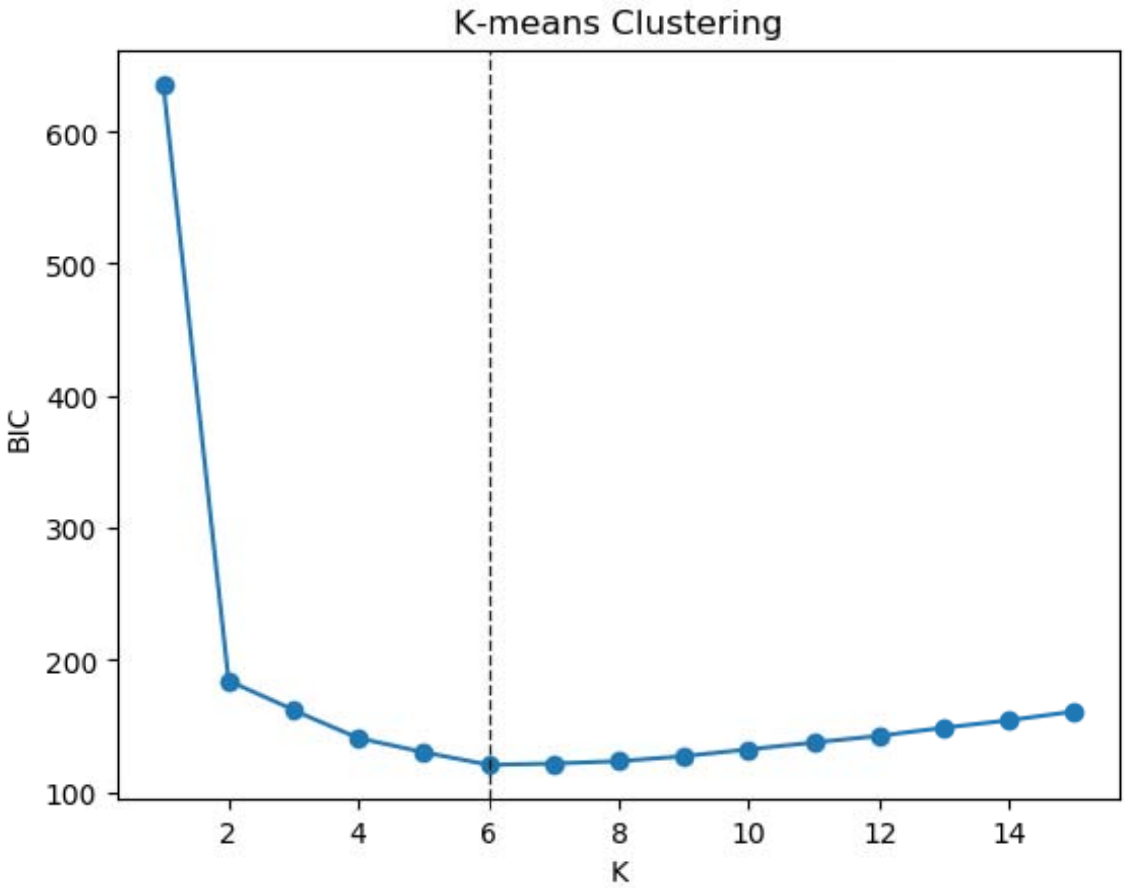

图 17.16 使用 BIC 准则选择K

结果显示，BIC 准则选择$K=6$，依然大于真实的聚类数目。

由此可见，使用传统的信息准则选择K均值聚类的数目，可能并不可靠。

下面，我们以真实数据 `iris` 为例，演示K均值聚类的 Python 操作。

为便于在三维空间进行可视化，仅使用前 3 个特征变量(即花萼长度、花萼宽度与花瓣长度)进行聚类分析。

首先，从 seaborn 模块载入 iris 数据，并展示前 5 个观测值：

```
In [34]: iris = sns.load_dataset('iris')
    ...: iris.head()
Out[34]:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

为方便画图，下面使用 map()方法，将变量 species 的赋值从“setosa，versicolor，virginica”变为相应的“0，1，2”：

```
In [35]: Dict = {'setosa': 0, 'versicolor': 1,
                 'virginica': 2}
    ...: iris.species = iris.species.map(Dict)
```

其中，先将映射定义为字典 Dict，再用 map()方法完成变换。

然后，画前 3 个特征变量的三维散点图，并根据鸢尾花品种上色，结果参见图 17.17：

```
In [36]: fig = plt.figure()
    ...: ax = fig.add_subplot(111, projection='3d')
    ...: ax.scatter(iris['sepal_length'],
                    iris['sepal_width'], iris['petal_length'],
                    c=iris['species'], cmap='rainbow')
    ...: ax.set_xlabel('sepal_length')
    ...: ax.set_ylabel('sepal_width')
    ...: ax.set_zlabel('petal_length')
```

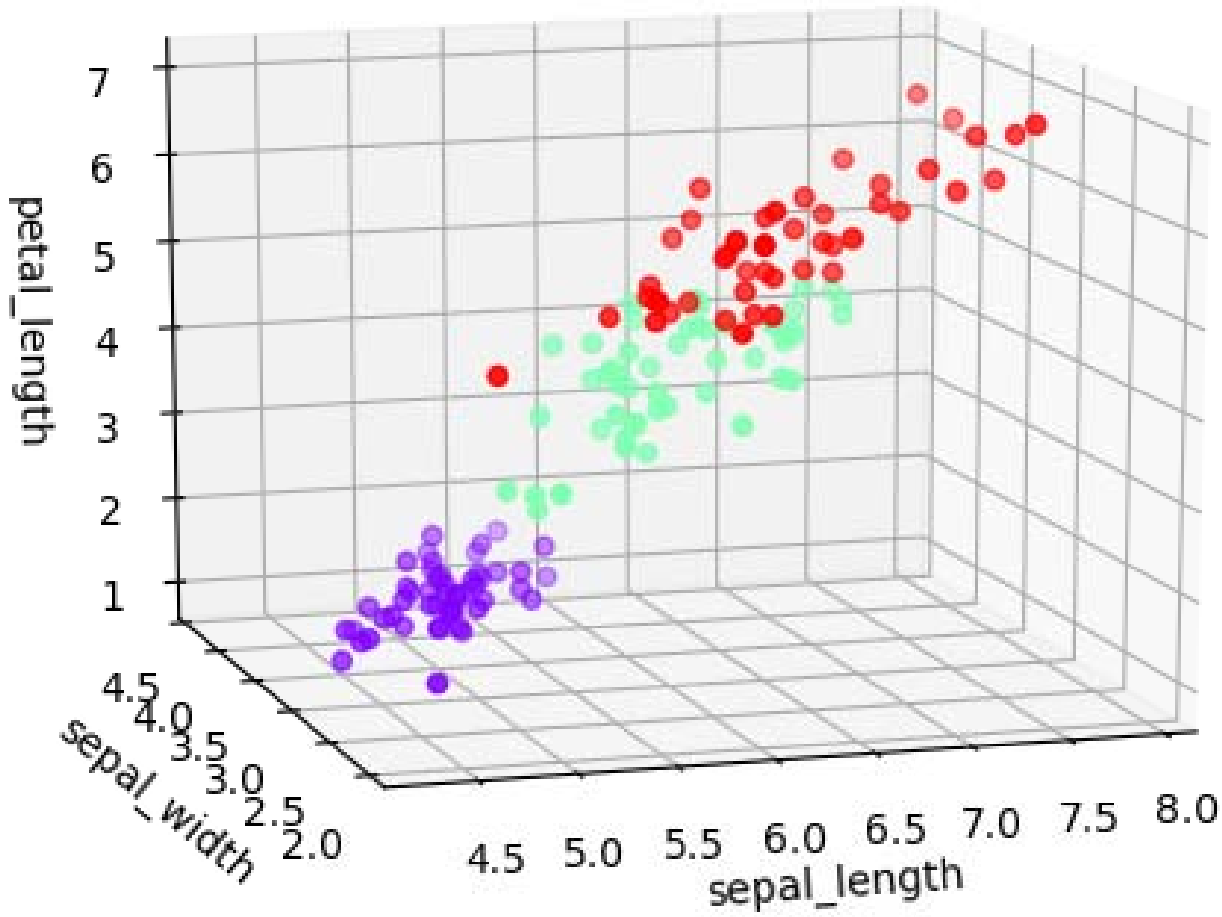

图 17.17 Iris 数据的三维散点图

从图 17.17 可见，三类不同的鸢尾花品种，聚集在三维特征空间(sepal_length，sepal_width，petal_length)的不同区域。

下面，假设$K=3$，使用 iris 数据的前 3 个特征变量进行K均值聚类分析：

```
In [37]: X3 = iris.iloc[:, :3]
    ...: model = KMeans(n_clusters=3, random_state=1,
n_init=20)
```

其中，第 1 个命令取出 iris 数据的前 3 个特征变量，并记为 X3。

使用 fit()方法进行估计，并展示聚类结果：

```
In [38]: model.fit(X3)
    ...: model.labels_
Out[38]:
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

目测可知，K均值聚类的结果基本正确，但所用类别标签不同。

为此，将聚类结果 model.labels_设为数据框，并以 map()方法更新标签：

```
In [39]: labels = pd.DataFrame(model.labels_,
columns=['label'])
    ...: d = {0: 2, 1: 0, 2: 1}
    ...: pred = labels.label.map(d)
```

下面，展示“混淆矩阵”，并计算“分类准确率”：

```
In [40]: table = pd.crosstab(iris.species, pred,
                             rownames=['Actual'],
                             colnames=['Predicted'])

   ...: table
Out[40]:
Predicted   0   1   2
Actual
0          50   0   0
1           0  45   5
2           0  13  37
```

```
In [41]: accuracy = np.trace(table) / len(iris)
    ...: accuracy
Out[41]: 0.88
```

其中，“np.trace(table)”计算 table 矩阵的“迹”(trace)，即主对角线元素之和，而“len(iris)”为数据框 iris 的长度，即样本容量。

尽管只使用了 3 个特征变量，聚类分析的准确率已达到 88%。

如果使用全部 4 个特征变量，可得到更高的准确率(参见习题)。

更直观地，画三维散点图，并根据聚类归属(pred)上色，结果参见图 17.18：

```
In [42]: fig = plt.figure()
    ...: ax = fig.add_subplot(111, projection='3d')
    ...: ax.scatter(iris['sepal_length'],
                    iris['sepal_width'], iris['petal_length'],
                    c=pred, cmap='rainbow')
    ...: ax.set_xlabel('sepal_length')
    ...: ax.set_ylabel('sepal_width')
    ...: ax.set_zlabel('petal_length')
```

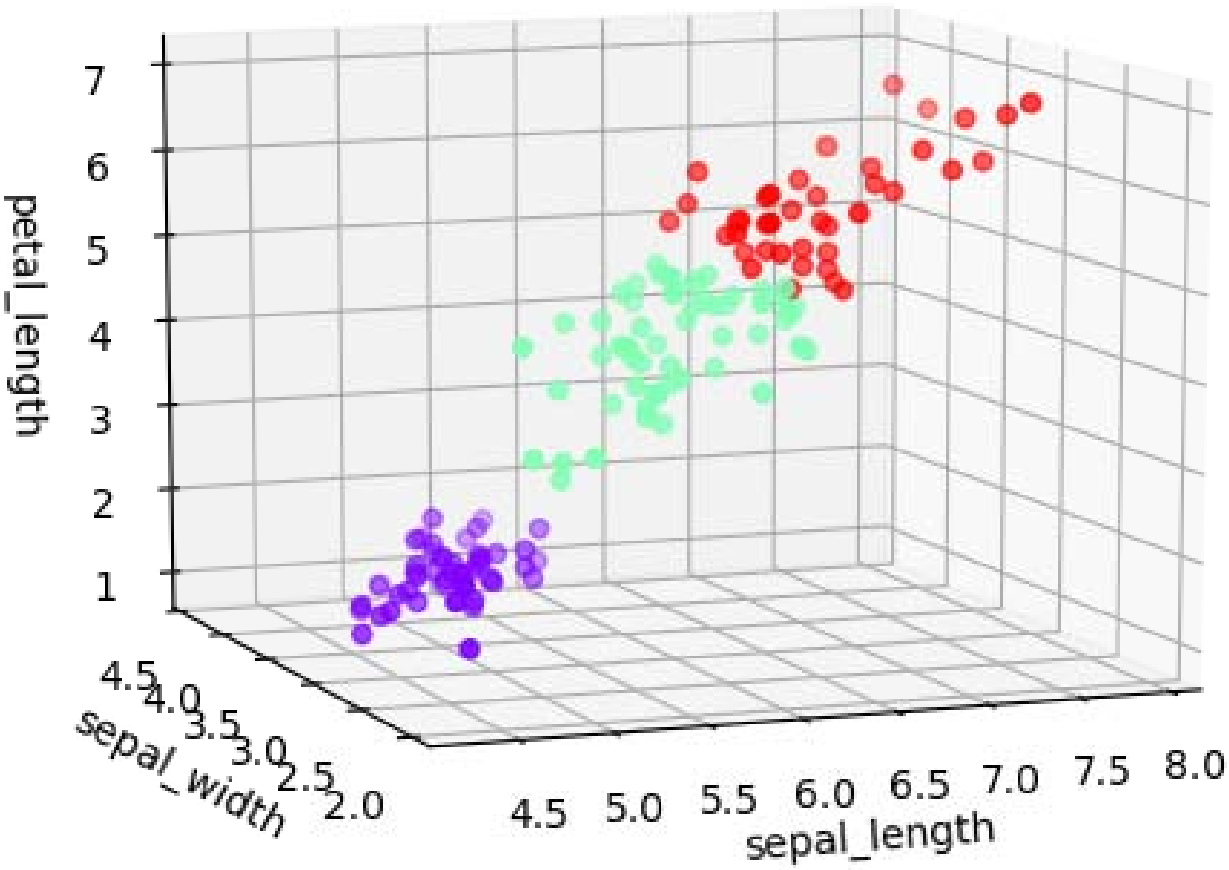

图 17.18 Iris 数据的三维散点图(根据聚类上色)

对比图 17.18 与 17.17(以及混淆矩阵)可知，K 均值聚类的预测结果与真实类别很接近。

其中，左下角的山鸢尾(setosa)的分类完全正确。

另外两个品种的鸢尾花，由于特征变量存在交叠，故出现分类错误。

17.7 分层聚类的 Python 案例

继续以 `iris` 数据演示在 Python 中进行分层聚类，分别使用欧氏距离，以及基于相关系数的距离指标。

* 详见教材，以及配套 Python 程序（现场演示）。