第 6 章 数据预处理及离群点检测

从外部获取的原始数据可能会受到噪声的影响，并伴有缺失值、不一致以及异常数据，这些都会降低数据挖掘和分析的效率，因此需要进行数据预处理以改善数据质量，让数据更好地适宜数据分析的模型或方法。数据预处理的内容主要涉及数据清洗、数据转换和数据规约，这些工作几乎占据了整个数据挖掘和分析一半以上的工作量。离群点检测涉及离群点的概念以及其检测方法，离群点是那些明显偏离一般值的观测对象，离群点检测就是寻找出这些异常对象的行为过程。

6.1 数据清洗

6.1.1 数据清洗的意义及内容

6.1.1.1 数据清洗的意义

随着科学技术的进步与发展，生产计算机硬件的成本在逐渐降低，人们通过各种技术手段从环境获取大量的数据信息存储在设备硬件上，然后对收集来的数据进行挖掘和分析，试图得到有价值的训练模型、隐含信息以及可用知识，用来解决特定问题，指导实际工作。

阿里通过大数据分析，可以获悉国内零售市场的增长情况，也可以了解人们更喜欢哪些商品。通过对这些数据的整理、分析，可以为商家甚至为整个行业提供生产指导，为社会带来巨大的经济价值。百度通过采集用户输入的数据，可以实时反映社会热点，这对舆情监控带来了极大便利。今日头条则收集用户对信息的点击行为，分析用户的偏好，给用户做个性化推荐，这门核心技术使其开发公司——字节跳动的估值达 750 亿美元。

互联网行业是一个充满奇迹的行业。奇迹的诞生往往源于对原始数据的积累和对数据的精准分析。实际上，海量的数据并不是每一条都有用的。例如，从天猫上采集了 10TB 的数据，包含电子产品、男装女装、食品等各类商品类别与销量，对于一家经营男性服装的企业来说，只有男装数据才具有现实意义，该企业可以从数据集中筛选出男装的销售信

息，并根据季节的不同来给自己的产品设计合适的款式并做出恰当的价格定位；对于销售笔记本电脑的门店来说，则需要根据电子产品的行情来进行定价与备货。

对当前的总结，对未来的预测，都建立在精准的分析之上，精准分析的基础是用于准确的数据集。对原始数据进行整理、标注，形成一份“干净”的数据，使其适合特定场景，这个过程就是数据清洗的意义所在。

6.1.1.2 数据清洗的内容

对于已经采集到的数据，如果能够直接用于数据分析当然最好，然而大多数情况下，从原始数据源采集回来的数据集可能存在异常值、空值等。因而数据清理的主要内容就是处理缺失值、异常值，删除原始数据集中的无关数据、重复数据，平滑噪声数据，筛选出与数据分析有关的数据。通常基于不同的原则清洗数据，可能得到的结果差别很大，因此在采集数据的过程中就应提前定义好规则，以降低后期数据清洗的成本。

6.1.2 数据清洗方法与数据集成

数据清洗的主要目标在于将数据格式化，清除异常数据和重复数据，纠正错误数据。

6.1.2.1 异常数据处理

在数据预处理时，发现采集的数据集中个别数值明显偏离其余观测值，那么这个数值就是可能的异常值。例如，小学六年级男生的平均体重在 43kg 左右，身高在 1.52m 上下，然而采集的数据集中某个男生的体重为 42kg，身高却是 15.1m，很明显该数据不合理，远远高于正常数据，因此需要对这样的数据进行分析和处理。常见的异常数据分析方法有：

（1）统计量判别法。计算该组数据的最小值，最大值以及平均值，来检测某个数据是否超出合理范围。

（2）3σ 原则。依据正态分布，距离平均值 3σ 以外的数值出现属于小概率事件，此时，异常值出将现在距离平均值 3σ 以外的地方。

（3）箱线图判断法。箱线图可以直观地查看数据分布情况，如果数据超出箱线图的上下边界就认为是异常值。

当异常值出现时，是否需要进行处理应视具体情况而定。常见的异常值的处理方法有：

（1）删除记录。数据量足够多的情况下直接删除异常值记录，不进行考虑。

（2）视为缺失。将异常值看作缺失值，按缺失值的处理方法进行操作。

（3）平均值修正。使用靠近异常值的几个数据的平均值代替，或使用整个数据集的平均值代替。

（4）不做处理。异常值当成正常数据进行操作。

例题 6.1　中先设置异常值，再分别采用散点图、箱线图以及 3σ 原则进行异常值检测，散点图如图 6.1 所示，箱线图如图 6.2 所示。

例题 6.1 异常值检测。

```
(1) import numpy as np                                          #np 作为 NumPy 库别名
(2) import pandas as pd                                         #pd 作为 Pandas 库别名
(3) import matplotlib. pyplot as plt                            #matplotlib. pyplot 模块别名 plt
(4) plt. rcParams['font. family'] = ['SimHei']                  #用来显示中文符号
(5) DF = pd. DataFrame(np. arange(1, 21), columns = ['x'])      #自变量
(6) DF['y'] = DF['x']* 2+1                                      #因变量
(7) DF. iloc[5, 1] = 256                                        #设置异常值
(8) DF. iloc[15, 1] = 186                                       #设置异常值
(9) plt. figure(1)
(10) plt. scatter(DF['x']. values, DF['y']. values, c = 'red')  #绘制散点图
(11) plt. title('散点图方式进行异常值检测')
(12) plt. show( )
(13) plt. savefig('Fig6_1. jpg')                                #绘制箱线图
(14) plt. figure(2)
(15) plt. boxplot(DF['y']. values, sym = 'red', notch = True)
(16) plt. title('箱线图方式进行异常值检测')
(17) plt. show( )
(18) plt. savefig('Fig6_2. jpg')
(19) M = DF['y']. mean( )                                       #计算均值
(20) S = DF['y']. std( )                                        #计算标准差
(21) Idy = (M- 3 * S>DF['y']) | (M+3 * S<DF['y'])               #使用 3σ 原则筛选异常值
(22) Out = DF[Idy]
(23) print('3σ 原则进行异常值检测: \n')
(24) print(Out)
3σ 原则进行异常值检测:                                          #显示结果
   x    y
5  6  256
```

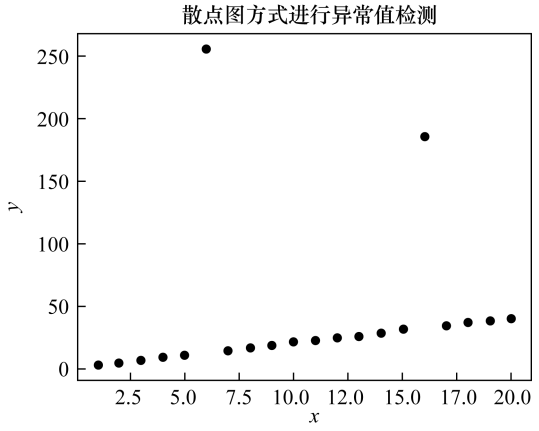


图 6.1 散点图检测异常值

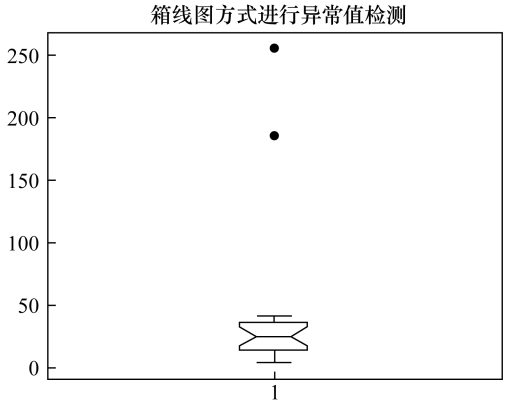


图 6.2 箱线图检测异常值

从案例 6.1 可知，通过绘制散点图和箱线图可以快速有效地找到两个异常值，但使用 3σ 原则却只筛选出一个异常值，这是因为 3σ 原则假设数据集服从正态分布，所以对于不符合该要求的数据，效果不是很好。

6.1.2.2 缺失值处理

在数据采集过程中由于某种原因有个别数据无法获取或者遗漏或者丢失造成数据缺失，这时需要对缺失值进行处理，除了删除记录和不做处理外，可用插补方法对缺失值进行数据填补，常用的插补方法有：

（1）采用数据均值、中位数或众数进行插补；

（2）使用默认值代替缺失值；

（3）采用缺失值附近的数据进行插补；

（4）回归方法即对缺失值属性建立拟合模型，采用模型预测缺失值；

（5）插值法即利用已知数据建立合适的插值函数，计算缺失值的近似值并代替缺失值。

利用 Pandas 可以快速的找到缺失数据，并对数据缺失数据进行处理，案例 6.2 将利用 Pandas 库函数完成缺失值（NaN）的检测、统计、删除以及填充。

例题 6.2 缺失值检测和处理。

```
(1) import numpy as np                                          #np 作为 NumPy 库别名
(2) import pandas as pd                                         #pd 作为 Pandas 库别名
(3) DF = pd. DataFrame(np. arange(1, 16). reshape(3, 5),
    columns = ['x1', 'x2', 'x3', 'x4', 'x5',])
(4) DF. iloc[:, 2] = np. nan                                    #设置第三列全为缺失值
(5) DF. iloc[1, :] = np. nan                                    #设置第三行全为缺失值
(6) print('DF: \n', DF)
(7) print('检测缺失值: \n', DF. isnull( ))                      #isnull( )函数检测缺失值
(8) print('统计缺失值: \n', DF. isnull( ). sum( ))              #isnull( ).sum( )统计缺失值
DF:                                                             #结果显示
      x1     x2     x3     x4     x5
0     1. 0   2. 0   NaN    4. 0   5. 0
1     NaN    NaN    NaN    NaN    NaN
2     11. 0  12. 0  NaN    14. 0  15. 0
检测缺失值:                                                     #结果显示
      x1     x2     x3     x4     x5
0     False  False  True   False  False
1     True   True   True   True   True
2     False  False  True   False  False
统计缺失值:                                                     #结果显示
x1    1
x2    1
x3    3
x4    1
x5    1
dtype: int64
(9) DF. iloc[0, 2] = 3                                          #指定位置赋值
(10) DF. iloc[1, 0] = 6                                         #指定位置赋值
(11) print('DF: \n', DF)
(12) print('删除 NaN 所在的行: \n', DF. dropna( ))              #删除 NaN 所在的行
(13) print('删除 NaN 所在的列: \n', DF. dropna(axis = 1))      #删除 NaN 所在的列
```

```
DF:                                                          #结果显示
      x1      x2      x3      x4      x5
0     1. 0    2. 0    3. 0    4. 0    5. 0
1     6. 0    NaN     NaN     NaN     NaN
2     11. 0   12. 0   NaN     14. 0   15. 0
删除 NaN 所在的行:                                           #结果显示
      x1      x2      x3      x4      x5
0     1. 0    2. 0    3. 0    4. 0    5. 0
删除 NaN 所在的列:                                           #结果显示
      x1
0     1. 0
1     6. 0
2     11. 0
(14) DF. iloc[0, 2] = np. nanP                               #指定位置赋值 NaN
(15) DF. iloc[1, 0] = np. nan                                #指定位置赋值 NaN
(16) print('DF:\n', DF)
(17) print('删除全是 NaN 的行:\n', DF. dropna(how = 'all'))    #删除全是 NaN 的行
DF:                                                          #结果显示
      x1      x2      x3      x4      x5
0     1. 0    2. 0    NaN     4. 0    5. 0
1     NaN     NaN     NaN     NaN     NaN
2     11. 0   12. 0   NaN     14. 0   15. 0
删除全是 NaN 的行:                                           #结果显示
      x1      x2      x3      x4      x5
0     1. 0    2. 0    NaN     4. 0    5. 0                  #固定值填充
2     11. 0   12. 0   NaN     14. 0   15. 0                 #向前填充
(18) print('DF:\n', DF)
(19) print('用 0 填充:\n', DF. fillna(0))                     #利用字典按列用均值进
(20) print('向前填充:\n', DF. fillna(method = 'ffill'))       #行缺失值填充
(21) print('通过字典按列用均值填充: \n ')
(22) print(DF. fillna({'x1':DF['x1']. mean( ), 'x4':DF['x4']. mean( )}))
DF:                                                          #结果显示
      x1      x2      x3      x4      x5
0     1. 0    2. 0    NaN     4. 0    5. 0
1     NaN     NaN     NaN     NaN     NaN
2     11. 0   12. 0   NaN     14. 0   15. 0
```

```
用 0 填充:                                          #结果显示
      x1      x2      x3      x4      x5
0     1.0     2.0     0.0     4.0     5.0
1     0.0     0.0     0.0     0.0     0.0
2    11.0    12.0     0.0    14.0    15.0
向前填充:                                          #结果显示
      x1      x2      x3      x4      x5
0     1.0     2.0     NaN     4.0     5.0
1     1.0     2.0     NaN     4.0     5.0
2    11.0    12.0     NaN    14.0    15.0
通过字典按列用均值填充:                            #结果显示
      x1      x2      x3      x4      x5
0     1.0     2.0     NaN     4.0     5.0
1     6.0     NaN     NaN     9.0     NaN
2    11.0    12.0     NaN    14.0    15.0
```

例题 6.2 分别采用 isnull( ) 函数来检测数据中的缺失值，利用 isnull( ).sum( ) 方法统计数据中的缺失值，使用 dropna( ) 函数删除数据中的缺失值，运用 fillna( ) 函数对数据中的缺失值进行填充，除了利用前面的非缺失值进行填充外(method = ‘ffill’)，也可以使用后面的非缺失值进行填充(method = ‘bfill’)，还可以通过字典对多列进行填充。

6.1.2.3 噪声数据处理

噪声数据是指在测量变量时测量值可能出现的相对于真实值的偏差或错误，这样的数据会影响后续数据分析操作的正确性与效果。噪声数据的一般处理方法包括：

1. 分箱法

将待处理的数据按照一定的规则放进一些箱子(区间) 中，考察每一个箱子(区间) 中的数据，然后采用某种方法分别对各个箱子(区间) 中的数据进行处理。在采用分箱技术时，需要确定的两个主要问题，即如何分箱及如何对每个箱子中的数据进行平滑处理。下面将通过举例说明三种分箱方法。

例如，客户收入属性 income 的数据排序后的值如下(人民币，单位：元)：

800 1000 1200 1500 1500 1800 2000 2300 2500 2800 3000 3500 4000 4500 4800 5000

(1) 等深分箱法。等深分箱法也称为统一权重法，该方法将数据集按记录行数分箱，每箱具有相同的记录数，每箱记录数称为箱子的深度。

等深分箱法：设定权重(箱子深度) 为 4，分箱后的结果如下：

箱 1：800 1000 1200 1500

箱 2：1500 1800 2000 2300

箱 3：2500 2800 3000 3500

箱 4：4000 4500 4800 5000

（2）等宽分箱法。等宽分箱法也称为统一区间法，该方法将数据集在整个属性值的区间上平均分布，即每个箱的区间范围是一个常量，称为箱子宽度。

等宽分箱法：设定区间范围(箱子宽度)为人民币 1000 元，分箱后的结果如下：

箱 1：800 1000 1200 1500 1500 1800

箱 2：2000 2300 2500 2800 3000

箱 3：3500 4000 4500

箱 4：4800 5000

（3）用户自定义区间。用户可以根据需要自定义区间，当用户明确希望观察某些区间范围内的数据分布时，使用这种方法可以方便地帮助用户达到目的。

用户自定义：如将客户收入划分为 1000 元以下、1000～2000 元、2000～3000 元，3000～4000 元和 4000 元以上几组，分箱后的结果如下：

箱 1：800

箱 2：1000 1200 1500 1500 1800 2000

箱 3：2300 2500 2800 3000

箱 4：3500 4000

箱 5：4500 4800 5000

分箱后就需要对每个箱子中的数据进行平滑处理，常见的数据平滑方法如下：

（1）按平均值平滑，对同一箱值中的数据求平均值，用平均值替代该箱子中的所有数据；

（2）按边界值平滑，对箱子中的每一个数据，使用该箱中距离左右两侧边界值较小的边界值代替箱子中的所有数据；

（3）按中值平滑，取箱子的中值，用来替代箱子中的所有数据。

2. 聚类方法

将抽象对象的集合分组为由类似的对象组成的多个类，然后找出并清除那些落在簇之外的值(孤立点)，这些孤立点被视为噪声数据。

3. 回归方法

试图发现两个相关的变量之间的变化模式，通过使数据适合一个函数来平滑数据，即通过建立数学模型来预测数值，所采用的方法一般包括线性回归和非线性回归。

另外，除了以上的噪声处理的方法，近年来小波等技术也被引入了数据降噪，感兴趣的读者，可自行查阅相关资料。最后，数据清洗的一般步骤可以概括为先加载原始数据，再找出噪声数据，然后进行数据清洗，最后保存新的数据。

6.1.2.4 数据集成

实际中的数据种类繁多，有连续的、离散的、定性的、定量的等，往往需要将多文件

或多个数据库中的异构数据进行合并，然后存放在一个统一的数据库中进行存储，这就是数据集成。此时需要考虑以下问题：

1. 实体识别

由于数据来源不同，其中的概念定义不尽相同，主要考虑以下几种情况。

（1）同名异义。来自不同源的数据特征的名称一样，但表示的内容不同。例如，数据源 A 的某个数据特征的名称是 ID，表示学生的学号，而数据源 B 的某个数据特征的名称也是 ID，但是表示的是产品编号。

（2）异名同义。来自不同源的数据特征的名称不一样，但表示的内容一样。例如，数据源 A 的某个数据特征的名称是 ID，表示学号，数据源 B 的某个数据特征的名称 XueHao，记录的也是学生的学号。

（3）单位不统一。来自不同源的数据记录的单位不一样，如统计身高，一个数据源使用米作单位，而另一个数据源使用厘米作单位。

2. 冗余性识别

冗余性识别是指数据中存在冗余，一般分以下两种情况。第一种，同一属性多次出现，如两个数据源都记录了每天的最高温度和最低温度，当数据集成时就出现两次。第二种，同一属性命名不一致而引起数据重复，如两个数据源分别要求测量每天的温度，但是特征的名称不一样，这样数据集成时也存在数据的重复。

3. 数据不一致

编码使用的不一致问题和数据表示的不一致问题。例如身份证号码，旧的身份证号码是 15 位，新的身份证号码是 18 位；再如，日期“2022/08/25”和“25/08/2022”表示的是同一天。

仔细检测和整合不同源数据是数据集成的主要任务，这将是提高数据分析速度和质量的保障。

6.2 数据转换

数据集中不同的特征或属性通常都有不同的量纲，由此造成的数值间的差异很大，为了消除量纲和取值范围的差异可能产生的影响，需要对数据依场景进行适当转换以适应问题求解，为此需要对数据进行适当转换。

6.2.1 数据标准化

6.2.1.1 离差标准化数据

离差标准化又称最小-最大规范化，本质上是对原始数据进行线性变换，将原始数据映射到[0，1]之间，所用公式为：

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{6.1}$$

其中，x 代表原始数据，$x_{\min}$代表原始数据的最小值，$x_{\max}$代表原始数据的最大值，x'代表规范化后的数据。离差标准化保留了原来数据中保存的关系，是消除量纲和数据取值范围影响的最简单方法。

这种方法的缺点有：

（1）若数值集中且某个数值特别大，则规范化后多数值集中在 0 附近；

（2）若遇到不在$[x_{\min}, x_{\max}]$范围内的数据时，会引起系统出错。

6.2.1.2 标准差标准化数据

标准差标准化又称为零均值标准化或 Z-score 标准化，是应用比较多的数据标准化方法，经过该方法处理后，数据的均值为 0，标准差为 1，所用公式为：

$$x' = \frac{x - \bar{x}}{\sigma} \tag{6.2}$$

其中，x 代表原始数据，$\bar{x}$ 代表原始数据的均值，σ 代表原始数据的标准差，x'代表规范化后的数据。该方法要求原始数据的分布尽可能的接近正态分布(高斯分布)，否则，规范化后的效果可能会变的很糟糕。

6.2.1.3 小数定标规范化

小数定标规范化通过移动原始数据的小数位数，将原始数据映射到[-1，1]之间，移动的小数位数取决于原始数据中绝对值的最大值，所有公式为：

$$x' = \frac{x}{10^k} \tag{6.3}$$

为了帮助读者理解数据标准化，例题 6.3 演示对数据分别进行离差标准化、标准差标准化和小数定标规范化。

例题 6.3 数据标准化。

```
(1) import numpy as np                                  #np 作为 NumPy 库别名
(2) import pandas as pd                                 #pd 作为 Pandas 库别名
(3) np. random. seed(1)
(4) A = np. random. randint(1, 1000, size = (5, 4))
(5) DF = pd. DataFrame(A)
(6) DF1 = (DF- DF. min( ))/(DF. max( )- DF. min( ))     #离差标准化
(7) DF2 = (DF- DF. mean( ))/DF. std( )                  #标准差标准化
(8) DF3 = DF/10 * * np. ceil(np. log10(DF. abs( ). max( )))   #小数定标规范化
(9) ff = lambda x: '%. 3f '%x                           #显示 3 位小数
```

```
(10) DF11 = DF1. applymap(ff)
(11) DF22 = DF2. applymap(ff)
(12) DF33 = DF3. applymap(ff)
(13) print('DF: \n', DF)
(14) print('离差标准化结果: \n', DF11)
(15) print('标准差标准化: \n', DF22)
(16) print('小数定标规范化: \n', DF33)
```

DF: #结果显示

	0	1	2	3
0	38	236	909	73
1	768	906	716	646
2	848	961	145	130
3	973	584	750	509
4	391	282	179	277

离差标准化结果: #结果显示

	0	1	2	3
0	0. 000	0. 000	1. 000	0. 000
1	0. 781	0. 924	0. 747	1. 000
2	0. 866	1. 000	0. 000	0. 099
3	1. 000	0. 480	0. 792	0. 761
4	0. 378	0. 063	0. 045	0. 356

标准差标准化: #结果显示

	0	1	2	3
0	- 1. 475	- 1. 058	1. 047	- 1. 035
1	0. 429	0. 923	0. 500	1. 300
2	0. 637	1. 086	- 1. 119	- 0. 803
3	0. 963	- 0. 029	0. 596	0. 742
4	- 0. 554	- 0. 922	- 1. 023	- 0. 204

小数定标规范化: #结果显示

	0	1	2	3
0	0. 038	0. 236	0. 909	0. 073
1	0. 768	0. 906	0. 716	0. 646
2	0. 848	0. 961	0. 145	0. 130
3	0. 973	0. 584	0. 750	0. 509
4	0. 391	0. 282	0. 179	0. 277

6.2.2 数据变换

数据变换也是对原始数据进行规范化处理，此处主要是指采用简单函数对原始数据进行变换操作，常见的函数包括平方、开平方、取对数以及差分运算等，分别用到的公式如下：

$$x' = x^2 \tag{6.4}$$

$$x' = \sqrt{x} \tag{6.5}$$

$$x' = \log(x) \tag{6.6}$$

$$\nabla f(x_k) = f(x_{k+1}) - f(x_k) \tag{6.7}$$

上述函数的使用依数据和场景而定，例如，当原始数据取值较大时，可以采用开方或取对数的方法将数据值变小；当数据值较小时可以采用平方运算扩大数据值；在时间序列分析中，经常使用对数变换或差分运算将非平稳序列转换为平稳序列；

个人年收入从1万元到1亿元，这是一个很大的区间，使用对数变换对其进行压缩是一种常用的变换处理方法。

6.2.3 数据离散化

一些用于数据分析的分类算法，如ID3算法、Apriori算法等，需要有分类属性形式的数据，因而需要将连续属性变换成分类属性。连续属性离散化本质上就是将连续的属性空间划分为若干个区间，最后用不同的符号或整数值代表每个子区间中的数据。离散化设计两个任务：确定分类数以及如何将连续属性值映射到这些分类值。

常用的离散化方法有：

(1) 等宽法，将属性的值域分成具有相同宽度的区间，每个区间用一个数据值表示，区间的个数由数据本身的特点决定或者用户指定，类似于制作频率分布表。

(2) 等频法，将相同数量的记录放进每个区间，区间的个数用户指定。和等宽法不同的是，每个区间的数据个数是相等的。

(3) 基于聚类分析的方法，一维聚类方法包括两个步骤：首先将连续属性的值用聚类算法(如K-Means算法)进行聚类，然后再将聚类得到的簇进行处理，合并到一个簇的连续属性值做同一标记。聚类分析的离散化方法也需要用户指定簇的个数，从而决定产生的区间数。

等宽法和等频法简单、易操作，但都需要指定区间个数；等宽法对离群点比较敏感，倾向于不均匀地把属性值分布到各个区间中，可能会造成有些区间数据较多，有些区间数据较少，这对决策模型是不利的；等频法虽可以避免上述问题，却有可能将相同的数据值分配到不同的区间内，以满足每个区间上都要有相同记录个数的要求。

近年来，基于熵的离散化方法，各种基于小波变换的特征提取方法，以及自上而下的卡方分裂算法等也被不同的学者在数据分析和机器学习中采用。需要注意的是，前面介绍了多种数据转化方法，具体使用哪种方法对数据进行预处理更为适合，需要经过实验进行

验证。

6.3 数据归约

现实中，收集的数据量可能会很大，如果直接进行数据挖掘和分析可能会耗费大量的时间。数据归约是指在尽可能保持数据原貌的前提下，最大限度地精简数据量。原数据可以通过数据集归约后表示，归约后的数据集接近保持原数据的完整性，但数据量比原数据小得多，与非归约数据相比，在归约的数据上进行挖掘，所需的时间和内存资源更少，挖掘将更有效，并产生相同或几乎相同的分析结果。

为此，数据规约的目标在于寻求最小的属性子集并确保新数据子集的概率分布尽可能接近原始数据集的概率分布，而其意义在于：

（1）减少数据挖掘时间；

（2）降低存储数据成本；

（3）减少无效和错误数据对模型的影响，提供模型的准确率。数据规约的主要内容涉及属性规约(维规约)和数量规约。

6.3.1 属性规约

属性归约也称为特征规约，是指通过减少属性(合并属性或删除属性)的方式压缩数据量，从而提高模型的计算效率，降低运算和存储成本。

属性归约常用方法有以下五种：

（1）合并属性，将一些旧的属性合并为新的属性。

（2）逐步向前选择，从当前空的属性集开始，每次从原来属性集合中选择一个当前最优的属性添加到当前属性子集中，直到无法选出最优属性或满足一定阈值约束为止。

（3）逐步向后删除，从当前全属性集开始，每次从当前属性子集中选择一个当前最差的属性并将其从当前属性子集中消去，直到无法选出最差属性为止或满足一定阈值约束为止。

（4）决策树归纳，利用决策树的归纳方法对初始数据进行分类归纳学习，获得一个初始决策树，所有没有出现在这个决策树上的属性均可认为是无关属性，因此将这些属性从初始集合中删除，剩下属性就可以构成一个较优的属性子集。

（5）主成分分析，用较少的变量去解释原始数据中的大部分变量，即将许多相关性很高的变量转化成彼此相互独立或不相关的变量。

以上方法中，逐步向前选择、逐步向后删除以及决策树归纳都是直接删除不相关的属性，主成分分析通常构造出比原始变量个数少、能解释大部分数据的新变量，即主成分，来代替原始变量进行建模。下面将通过例题 6.4 展示主成分分析法施行属性规约。

例题 6.4　表 6.1 给出了 14 个记录的 8 个属性值，通过主成分分析实现数据降维。

表 6.1 统计数据表

	x1	x2	x3	x4	x5	x6	x7	x8
PR_1	1131.2	6.9	40.3	683.2	185.9	39.0	6.8	560.0
PR_2	700.0	3.6	62.7	1232.0	289.0	90.5	9.9	254.8
PR_3	369.6	0.9	21.8	481.6	98.6	24.6	1.6	100.8
PR_4	624.4	1.9	31.4	414.4	134.4	33.2	0.5	204.4
PR_5	960.4	3.3	39.8	795.2	179.2	39.9	4.8	770.0
PR_6	996.8	3.5	91.8	1870.4	510.7	131.3	8.4	744.8
PR_7	616.0	2.2	55.4	1142.4	282.2	78.8	2.4	296.8
PR_8	1355.2	3.8	61.0	1108.8	244.2	62.3	5.0	498.4
PR_9	1136.8	5.3	110.9	2128.0	665.3	177.4	6.9	1002.4
PR_10	694.4	2.2	54.9	996.8	266.6	72.6	2.2	383.6
PR_11	350.0	2.7	23.5	470.4	103.0	29.1	2.4	109.2
PR_12	50.4	0.2	3.9	78.4	17.9	4.9	0.2	28.0
PR_13	904.4	3.9	52.6	929.6	219.5	59.6	6.0	478.8
PR_14	1078.0	2.5	63.3	1064.0	273.3	73.5	3.7	324.8

表 6.1 的数据保存在当前路径下的 PCA_data1.xlsx 文件中，可以通过读取数据将其导入，经过主成分分析法降维后的数据保存在当前路径下的 PCA_data2.xlsx 文件中，下面两部分代码展示了主成分分析法的基本操作过程。

```
(1) import numpy as np                                  #np 作为 NumPy 库别名
(2) import pandas as pd                                 #pd 作为 Pandas 库别名
(3) from sklearn.decomposition import PCA               #导入 PCA( )函数
(4) inputfile = '…/CS20220803/PCA_data1.xlsx'           #读取原始数据路径
(5) outputfile = '…/CS20220803/PCA_data2.xlsx'          #保存降维后数据路径
(6) A = pd.read_excel(inputfile,header = None)          #读入数据
(7) DF = pd.DataFrame(A)
(8) data = (DF- DF.mean( ))/DF.std( )                   #数据标准差标准化
(9) pca = PCA( )                                        #生成模型
(10) pca.fit(data)                                      #进行数据拟合
(11) L1 = pca.components_                               #返回模型各个特征向量
(12) print('L1:\n', np.round(L1, 3))
(13) Z1 = pca.explained_variance_ratio_                 #返回各个成分各自的方差
                                                        百分比(又称贡献率)
(14) print('Z1:\n', np.round(Z1, 3))
```

```
L1:                                                    #8 个单位特征向量
 [[0. 321     0. 295     0. 389     0. 385     0. 38     0. 371     0. 320     0. 355]
  [0. 415     0. 598     - 0. 23    - 0. 279   - 0. 316  - 0. 372   0. 278     0. 157]
 [- 0. 451    0. 103     - 0. 04    0. 054     - 0. 037  0. 075     0. 771     - 0. 425]
  [0. 668     - 0. 363   0. 226     0. 111     - 0. 149  - 0. 069   0. 135     - 0. 559]
  [0. 038     - 0. 624   - 0. 123   0. 037     - 0. 159  - 0. 211   0. 43      0. 581]
 [- 0. 102    0. 136     - 0. 158   0. 862     - 0. 252  - 0. 345   - 0. 139   - 0. 027]
  [- 0. 16    0. 061     0. 54      - 0. 047   - 0. 761  0. 278     - 0. 062   0. 131]
 [- 0. 193    0. 032     0. 642     - 0. 11    0. 254    - 0. 688   0. 006     0. 005]]
Z1:
[0. 767   0. 13   0. 054   0. 028   0. 019   0. 001   0. 0   0. 0]
```

L1 返回了 8 个单位特征向量，Z1 返回了各个成分的方差百分比(又称贡献率)，方差百分比越大说明向量权重越大。可以看出，当选取前 3 个主成分时，累计的贡献率可以达到 95%以上，这说明采用前 3 个主成分进行计算时结果就很好了。因此重建 PCA 模型，选取前 3 个主成分对原始数据进行降维处理。

L2 返回了 3 个单位特征向量，Z2 返回了各个成分的方差百分比(贡献率)，Reduced_data 返回了降维后的数据，原始数据为 8 维(属性)14 个记录，采用主成分分析法后，将其降为 3 维 14 个记录，同时这 3 维 14 个记录的数据拥有原始数据 95%以上的信息。最后可以保存降维后的数据，以备进一步使用。另外，必要时可以对原始数据进行恢复，恢复后的数据不全和原始数据相同，这是因为主成分分析法在降维过程中有一定的信息损失。注意，此处只讲解了主成分分析法的应用，并没有涉及到主成分分析法的理论，感兴趣的读者可以自主查阅相关理论，加深理论知识学习。

```
(15) pca = PCA(3)                                      #选取 3 个主成分
(16) pca. fit(data)
(17) L2 = pca. components_                             #返回模型 3 个特征向量
(18) print('L2: \n', np. round(L2, 3))
(19) Z2 = pca. explained_variance_ratio_               #返回 3 个成分各自的方差
                                                       百分比(贡献率)
(20) print('Z2: \n', np. round(Z2, 3))
(21) Reduced_data = pca. transform(data)               #生成降维后的数据
(22) print('Reduced_data: \n', np. round(Reduced_data, 3))
L2:                                                    #前 3 个单位特征向量
[[ 0. 321    0. 295     0. 389     0. 385     0. 38     0. 371     0. 32     0. 355]
  [0. 415    0. 598     - 0. 23    - 0. 279   - 0. 316  - 0. 372   0. 278    0. 157]
 [- 0. 451   0. 103     - 0. 04    0. 054     - 0. 037  0. 075     0. 771    - 0. 425]]
```

```
Z2:
[0. 767 0. 13 0. 054]
Reduced_data:
 [[ 0. 732      2. 618      0. 180]
  [ 1. 065     - 0. 008      1. 848]
 [- 2. 824     - 0. 554      0. 107]
 [- 2. 189     - 0. 159     - 0. 612]
  [ 0. 067      1. 042     - 0. 643]
  [ 3. 458     - 0. 868      0. 402]
 [- 0. 277     - 1. 051     - 0. 157]
  [ 1. 011      0. 873     - 0. 644]
  [ 5. 224     - 1. 022     - 0. 417]
 [- 0. 302     - 0. 751     - 0. 443]
   [- 2. 37      0. 080      0. 444]
 [- 4. 358     - 0. 696      0. 161]
  [ 0. 399      0. 716      0. 205]
  [ 0. 364      - 0. 22     - 0. 43 ]]
pd. DataFrame(Reduced_data). to_excel(outputfile)                    #保存降维数据
Recovered_data = pca. inverse_transform(Reduced_data)                #必要时可以恢复原数据
```

6.3.2 数值规约

数值归约也称为样本规约，样本归约就是从数据集中选出一个有代表性的样本的子集。子集大小的确定要考虑计算成本、存储要求、估计量的精度以及其它一些与算法和数据特性有关的因素。数值规约包括非参数方法和参数方法，其中常见的非参数方法有直方图、聚类以及抽样。

6.3.2.1 直方图

直方图使用分箱思路近似数据分布。用直方图归约数据，就是将直方图中的桶(箱)的个数由观测值的数量 n 减少到 k 个，使数据变成一块一块地呈现。为了压缩数据，通常让一个桶代表给定属性的一个连续值域。桶的划分可以是等宽的，也可以是等频的。

举例说明如何采用直方图进行数值规约，例如某餐馆的菜品单价(单位：元)分别为：12，12，12，15，15，15，15，18，18，18，20，20，20，25，25，25，28，28，28，28，30，30，30，30，32，32，32，32，32，35，35，35，35，35，35，35，40，40，40，40，40，40，45，45，45，50，50，50。使用单桶显示这些数据的直方图如图 6.3 所示，每个单桶代表一个价格。为了压缩数据，可以让每个桶代表指定属性的一个取值区间，价格属

性的等宽直方图如图 6.4 所示，此时每个桶代表一个价格区间。

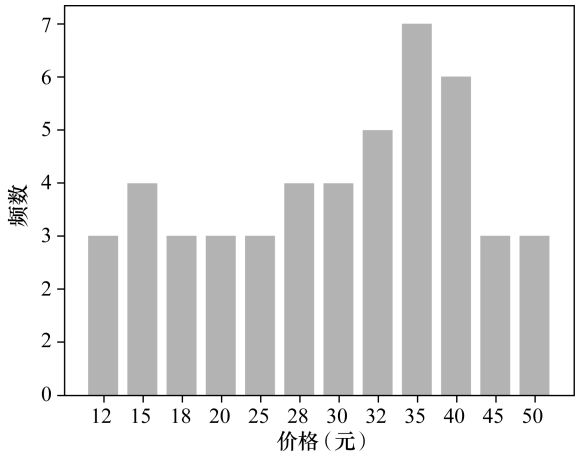


图 6.3　单桶代表一个价格

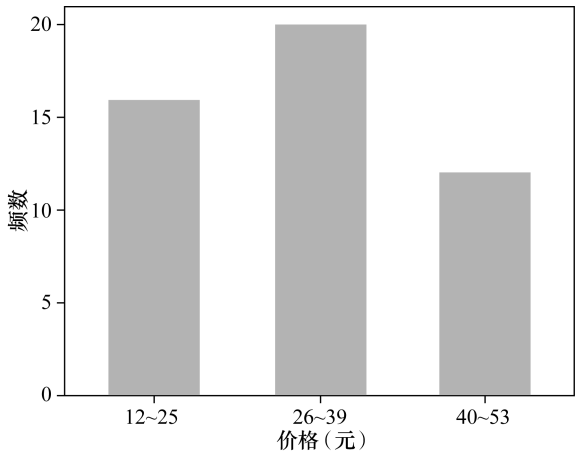


图 6.4　单桶代表一个价格区间

6.3.2.2　聚类

聚类算法是将数据划分为簇，使簇内的数据对象尽可能“相似”，而簇间的数据对象尽可能“相异”。在数据归约中，用每个数据簇中的代表替换实际数据，以达到数据归约的效果。

6.3.2.3　抽样

抽样通过选取随机样本(子集)，实现用小数据代表大数据的过程。抽样的方法包括简单随机抽样、聚类抽样和分层抽样等。

参数方法中使用模型估计数据，就可以用只存放模型参数代替存放实际数据，从而实

现数值规约，常用的参数方法有回归模型和对数线性模型，此处以回归模型为例进行说明。

在线性回归模型中，对数据拟合得到一条直线，利用该直线代替数据。例如，已知点（12，15），（14，19），（16，21），（18，29），（21，32），（23，35），（25，41），（26，43），可以采用线性函数 y=ax+b 对这些已知点进行规约，利用线性回归获得参数 a 和 b，这样就可以通过该线性函数上对应的点来近似这些已知点。多元回归是线性回归的扩展，用两个或多个自变量的线性函数对因变量建模。

6.4 离群点检测

采集到的数据可能会包含一些异常记录，对这些异常记录的检测和解释有很重要的意义。在大部分数据挖掘和分析过程中都会将这种差异记录视为噪声而丢弃，但异常检测在电子商务、网络入侵、工业损毁检测、金融欺诈、股票分析、医疗处理等领域却有着比较好的应用效果。

6.4.1 离群点及其类型

异常检测的实质是寻找观测值和参考值之间有意义的偏差。离群点检测是异常检测中最常用的检测方法之一，目的是检测出那些与正常数据行为或特征属性差别较大的异常数据或行为。常见的离群点的种类及其特点如下。

6.4.1.1 全局离群点

当一个数据对象明显地偏离了数据集中绝大多数对象时，该数据对象就是全局离群点。全局离群点有时也称为点异常，是最简单的一类离群点。全局离群点检测的关键问题是针对所考虑的应用，找到一个合适的偏离度量。度量选择不同，检测方法的划分也不同。在许多应用中，全局离群点的检测都是重要的。例如，在账目审计过程中，不符合常规交易数目的记录很可能被视为全局离群点，应该搁置并等待严格审查。

6.4.1.2 条件离群点

与全局离群点不同，当且仅当在某种特定情境下，一个数据对象显著地偏离数据集中的其他对象时，该数据对象称为条件离群点。例如，今天的气温是 32℃，这个值是否异常取决于地点和时间。条件离群点依赖于选定的情境，又称为情境离群点。在检测条件离群点时，条件必须作为问题定义的一部分加以说明。一般地，在条件离群点检测中所考虑对象的属性分为条件属性和行为属性。条件属性是指数据对象中定义条件的属性。行为属性是指数据对象中定义对象特征的属性。

在温度的例子中，时间、地点是条件属性，温度是行为属性。条件属性的意义会影响离群点检测的质量，因此条件属性作为背景知识的一部分，常由领域专家确定。局部离群

点是条件离群点的一种。如果数据集中一个对象的密度显著地偏离它所在的局部区域的密度，该对象就属于一个局部离群点。

6.4.1.3 集体离群点

当数据集中的一些数据对象显著地偏离整个数据集时，该集合形成集体离群点。集体离群点中的个体数据对象可能不是离群点。如图 6.5 所示，黑色对象形成的集合密度远大于数据集中的其他对象，因此是一个集体离群点，但是每个黑色的数据对象个体对于整个数据集而言不属于离群点。

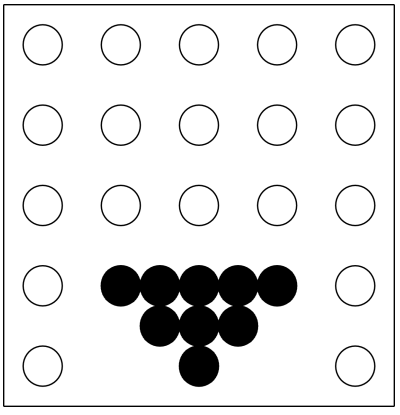


图 6.5 集体离群点

不同于全局或条件离群点，在集体离群点检测中，除了考虑个体对象的行为，还要考虑集体的行为，如在考试时有多位考生频繁地要求去卫生间。因此，为了检测集体离群点，需要一些关于对象之间联系的背景知识，如对象之间的距离或相似性测量方法。

6.4.2 离群点检测方法

离群点是不属于某个总体的数据点，它是一种与其它点的值相差甚远的异常观察值，是一种与其它结构良好的数据值不同的观察值。其常见的检测方法有如下四种。

6.4.2.1 基于统计模型的离群点检测法

大部分基于统计的离群点检测方法是构建一个概率分布模型，并计算对象符合该模型的概率，把具有低概率的对象视为离群点。基于统计模型的离群点检测方法的前提是必须知道数据集服从什么分布；对于高维数据，检验效果可能很差。

6.4.2.2 基于邻近度的离群点检测方法

通常可以在数据对象之间定义邻近性度量，把远离大部分点的对象视为离群点。对于一维、二维或三维的数据可以做散点图观察，但对大数据集不适用，并且对参数选择敏感，具有全局阈值，不能处理具有不同密度区域的数据集。

6.4.2.3 基于密度的离群点检测方法

考虑数据集可能存在不同密度区域这一事实，从基于密度的观点分析，离群点是在低密度区域中的对象，一个对象的离群点得分是该对象周围密度的逆。该方法给出了对象是离群点的定量度量，并且即使数据具有不同的区域也能够很好地处理，但对于大数据集不适用，参数选择是困难的。

6.4.2.4 基于聚类的离群点检测方法

一种利用聚类检测离群点的方法是丢弃远离其他簇的小簇；另一种更系统的方法是首先聚类所有对象，然后评估对象属于簇的程度(离群点得分)。基于聚类技术来发现离群点可能是高度有效的；聚类算法产生的簇的质量对该算法产生的离群点的质量影响非常大。

基于统计模型的离群点检测方法需要满足统计学原理，如果分布已知，检验可能非常有效。基于邻近度的离群点检测方法比统计学方法更容易使用，因为确定数据集有意义的邻近度量比确定它的统计分布更容易。基于密度的离群点检测与基于邻近度的离群点检测密切相关，因为密度常用邻近度定义：一种是定义密度为到 k 个最邻近的平均距离的倒数，如果该距离小，则密度高；另一种是使用 DBSCAN 聚类算法，一个对象周围的密度等于该对象指定距离 d 内对象的个数。

6.5 数据离散化案例

数据离散化可以采用等宽法、等频法以及基于聚类分析的方法，如何通过程序实现呢？Pandas 提供了 cut( )函数，可以进行连续数据的等宽离散化，该函数虽然不能直接实现等频离散化，但可以通过定义函数将相同数量的记录放进每个区间。一维聚类可采用 K-Means 算法先进行聚类，然后再对数据做离散化。cut( )函数的使用方法以及三种离散化方法的应用参见例题 6.5，三种方法的结果分别如图 6.6、图 6.7 和图 6.8 所示。

例题 6.5 数据离散化。

```
(1) import numpy as np                                  #np 作为 NumPy 库别名
(2) import pandas as pd                                 #pd 作为 Pandas 库别名
(3) np.random.seed(50)
(4) A=np.random.randint(1,100,size=50)                  #生成数据
(5) K=5                                                 #区间个数
(6) bins=[0, 20, 40, 60, 80, 100]                       #分类区间
(7) C1=pd.cut(A,bins, labels=range(K))                  #等宽离散化,labels 命名
                                                        分类为 0,1,2,3,4
(8) print('原始数据 5 行 10 列显示:\n', A.reshape(5, 10))   #显示等宽分组后各个类的
(9) print('等宽分组:\n',C1.value_counts( ))                中的数据个数
```

```
原始数据 5 行 10 列显示:
[[49   97   12   34   95   5   71   71   23   6]                    #结果显示
[  3   96   72   69   79   36   93   92   27   91]
[  7   21   44   32   50   86   42   65   7   20]
[  3   80   31   36   27   61   67   77   97   68]
[  3   11   52   1   94   94   95   1   12   31]]
等宽分组:
0      13                                                           #结果显示
1      10
2      5
3      11
4      11
dtype:int 64
(10) def SRcut(data, k):                                            #定义函数实现等频法
        x0 = data. quantile(np. arange(0, 1+1/k, 1/k))              #x0 为 Series 对象
        x0[0] = x0[0]* 0. 99                                        #第一个区间的左端点取数
        x1 = pd. cut(data, x0, labels = range(k))                   据中最小值的 99% ,目的在
        return x1                                                   于将最小值统计在内,默认
                                                                    左开右闭区间将不统计
                                                                    最小值
(11) C2 = SRcut(pd. Series(A),K)
(12) print('等频分组:\n',C2. value_counts( ))
等频分组:                                                           #结果显示
4      10
3      10
2      10
1      10
0      10
dtype: int64
(13) from sklearn. cluster import KMeans                            #导入 KMeans
(14) kmodel = KMeans(n_clusters = K)                                #建立分类模型
(15) kmodel. fit(A. reshape(len(A), 1))                             #训练模型
(16) x1 = pd. DataFrame(kmodel. cluster_centers_). sort_values(0)   #输出聚类中心,并进行排序
(17) x2 = x1. rolling(2). mean( )                                   (默认随机排序)
(18) x3 = x2. dropna( )                                             #相邻两项求中点,作为边界点
(19) w = [A. min( )*  0. 99]+list(x3[0])+[A. max( )]                #去掉 NaN
```

```
(20) C3 = pd. cut(A, w, labels = range(K))
(21) print('聚类分析方法分组:\n', C3. value_counts( ))                    #结果显示
聚类分析方法分组:
0      12
1      11
2      5
3      11
4      11
dtype: int64
(22) def Cplot(data, C, k):
        import matplotlib. pyplot as plt
        plt. rcParams['font. family'] = ['SimHei']
        plt. rcParams['axes. unicode_minus'] = False
        plt. figure( )
        for j in range(0, k):
          plt. plot(data[C = = j], [j for i in C[C = = j]],'ro')
        plt. ylim(- 1, k)
        return plt                                                        #调用函数,绘制结果
(23) plt1 = Cplot(A, C1, K)
(24) plt1. title('等宽法离散化')
(25) plt2 = Cplot(A, C2, K)
(26) plt2. title('等频法离散化')
(27) plt3 = Cplot(A, C3, K)
(28) plt3. title('聚类方法离散化)
```

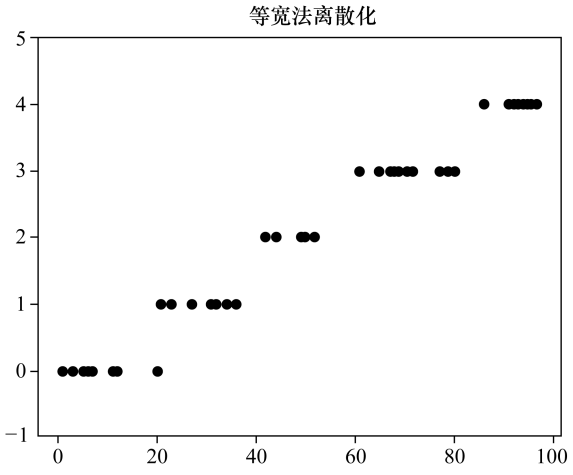


图 6.6 等宽法离散化结果

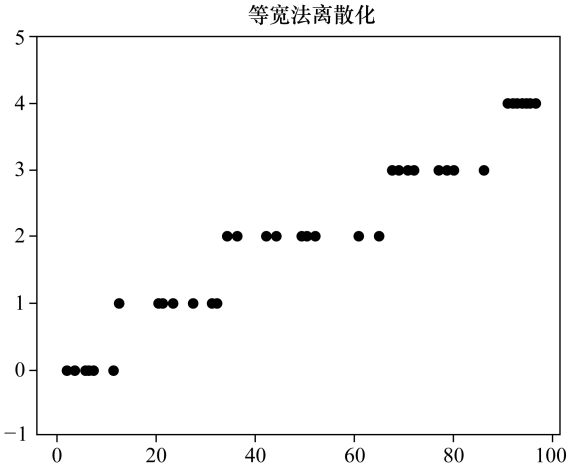


图 6.7　等频法离散化结果

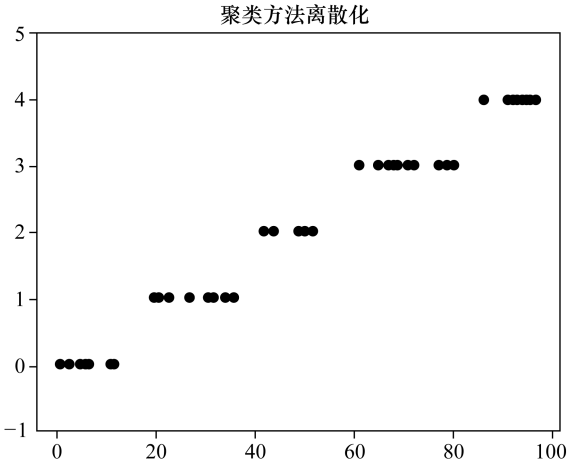


图 6.8　聚类方法离散化结果

例题 6.5 采用等宽法、等频法以及一维聚类方法(K-Means 聚类)对数据进行了离散化，将数据分为 5 类，同一类采用了相同的标识，此例中等宽法与聚类方法的结果比较相近。

6.6 本章小结

先介绍了数据清洗的意义和内容，再介绍了数据清洗的方法，内容涉及异常数据处理、缺失值处理和噪声数据处理，最后介绍数据集成，对不同源数据进行整合。数据转换可用以改善数据格式，内容涉及采用离差标准化、标准差标准化和小数定标规范化对数据进行标准化，采用常用函数进行数据变换以及采用等宽法、等频法和聚类方法对连续属性进行离散化。数据归约内容涉及属性规约和数值规约，在尽可能保持数据原貌的前提下，

最大限度地精简数据量，以提高数据挖掘和分析的性能和效率。离群点检测内容涉及离群点类型以及离群点检测的常用方法，用以发现与大部分其它对象显著不同的对象。

6.7 习题

1. 在数据清洗中，异常数据如何处理？
2. 数据转换中，有哪些方法可以进行数据标准化？
3. 属性规约常用的方法有哪些？
4. 常见的离群点检测方法有哪些？
5. 某班有 25 名同学，采集到他们的高等数学成绩分别为 66，68，72，54，83，95，61，75，75，85，77，45，78，78，78，81，83，79，79，81，89，91，87，93，88。求该班成绩的最小值、最大值、平均值、中位数以及标准差，并画出该组成绩的箱线图。

第 7 章 数据分类算法

数据分类(Data Classification)是一种重要的数据分析形式，也是数据挖掘相关应用中最有价值的技术之一。分类分析用预测方法预测给定数据对象的标签，它被广泛地应用到手写数字识别、医疗诊断、人脸识别等众多领域。本章主要介绍 4 种常用的分类方法：k-近邻算法、决策树、朴素贝叶斯以及支持向量机。

7.1 数据分类概述

7.1.1 分类问题的描述

分类是在一群已知类别标号的样本中，学会一个**分类器**(分类函数或模型)，让它能够把待分类的数据映射到给定的类别中。显然，当给定一个数据集 $X=\{x_1, x_2, \cdots, x_n\}$ 和一组类标签 $C=\{C_1, C_2, \cdots, C_m\}$，分类问题就是去确定一个映

$f: X \rightarrow C$，使得每个样本 x_i 被分配到一个类 C_j 中。一个类 C_j 包含映射到该类中的所有样本，即 $C_j=\{x_i \mid f(x_i)=C_j, 1 \leqslant i \leqslant n$，且 $x_i \in X\}$。一般地，这些类是被预先定义的、非交叠的，所以分类算法归属于监督学习。

比如在考察学生成绩时，若百分制分数划分为优秀、良好、一般、及格、不及格五个类别，即为一个分类问题，其中 X 是包含百分制分数在内的学生信息，类别 $C=\{$优秀、良好、一般、及格、不及格$\}$。

7.1.2 分类的一般过程

分类的目的是利用学习所得的分类器对新的数据集进行分类标号预测，故其过程分为学习和分类两个阶段，如图 7.1 所示。

学习阶段(建立模型)，建立一个分类模型(分类器)，用于描述预定的数据类集或概念集。该阶段，分类算法通过对训练集的分析或“学习”来构造分类器。其中训练集是为建立模型而被分析的数据库元组以及与之相关联的类别号构成。样本一般用 n 维向量$(x_1, x_2, \cdots, x_n)$表示，$x_i(i=1, 2, \cdots, n)$是样本在 n 个数据库属性/特征描述 $A_i(i=1, 2, \cdots, n)$上的度量。类标号 $C=\{C_1, C_2, \cdots, C_m\}$ 的每个值 C_i 充当一个类别或者类，其属

性是标称的，是离散的和无序的。

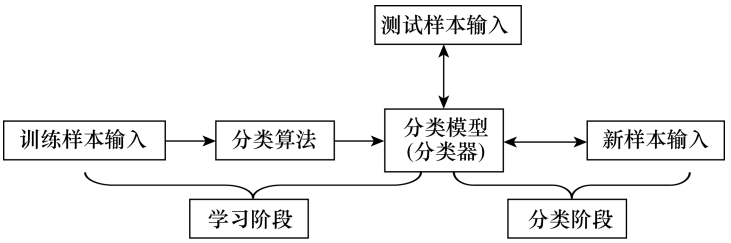


图 7.1　数据分类过程示意图

分类阶段(应用模型)。该阶段是使用模型预测给定数据的类标号。首先需要对模型(分类器)的预测准确率进行评估。一般情况下，利用除训练集之外的样本随机抽取后进行测试。保持法和交叉验证是两种基于给定数据随机选样划分训练集和测试集的常用方法。

经检测，如果认为模型的准确率可以接受，就可以用它对类标号未知的数据样本进行分类。

7.2 k-最近邻分类算法

k-最近邻(k-Nearest Neighbor，kNN)分类算法是数据挖掘分类技术中最简单常用的方法之一。它由 Cover 和 Hart 在 1968 年提出，是一种基于距离度量的分类算法，其基本思想是找出与给定测试样本某种距离度量最靠近的 k 个训练样本，然后基于这 k 个“邻居”的信息来进行预测。

7.1.1　kNN 算法概述

kNN 算法的工作原理是：对于个样本数集 S，假设其中每个数据都已有类别标号，即样本集每一数据与所属分类的对应关系已知。当对无标签数据 Z_u 进行分类预测时，kNN 算法遍历搜素样本集 S，找出与它距离最近的 k 个已知数据，并将这 k 个已知数据的大多数的分类标号分配给 Z_u，如图 7.2 所示，对待分类样本“ * ”进行分类，寻找与该样本距离最近的 7 个样本，其中 5 个为黑色，那么“ * ”就归属于黑色这一类。kNN 算法的实现主要包括 k 值的确定和“邻近性”的度量两个关键部分。

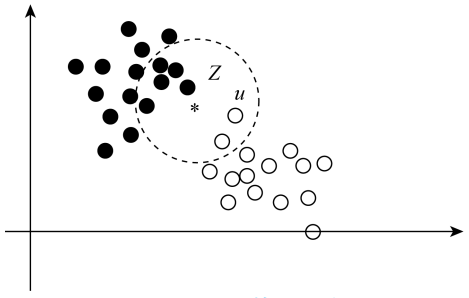


图 7.2　kNN 算法示意图

1. k 值的确定

在 kNN 算法中，k 的合理取值至关重要，过大或者过小都会直接影响到分类效果。通常用验证法来确定：从较小值(比如取 k = 1)开始，利用测试样本估计分类器的错误率。逐次增加 k 值，重复检测过程，直到选出错误率最小的 k。

2. “邻近性”的度量

kNN 算法用距离来度量样本数据的“邻近性”，如欧氏距离、曼哈顿距离。设两元组设为 $X_1=(x_{11}, x_{12}, \cdots, x_{1n})$，$X_2=(x_{21}, x_{22}, \cdots, x_{2n})$，它们的欧氏距离为：

$$d(X_1, X_2)=\sqrt{\sum_{i=1}^{n}(x_{1i}-x_{2i})^2} \tag{7.1}$$

曼哈顿距离为：

$$d(X_1, X_2)=\sum_{i=1}^{n}|x_{1i}-x_{2i}| \tag{7.2}$$

kNN 算法在计算距离之前，通常把所有属性值进行标准化处理，目的是防止出现因某些属性过大或过小的计量单位导致偏差，如用最小-最大规范化把数值属性 A 的值 v 变换到[0, 1]区间：

$$v'=\frac{v-\min_A}{\max_A-\min_A} \tag{7.3}$$

其中，$\min_A$，$\max_A$ 分别是属性 A 的最小值、最大值。

显然，上述度量“邻近性”的方法适合数值属性数据。对于标称属性数据，可通过简单地比较元组 X_1 和 X_2 对应的属性值。如果二者相同，则二者之差为 0；如果二者不同，则二者之差取 1。当然也可以采取其它手段将非数值型数据量化为数值型数据。例如样本属性为颜色，可将颜色转换为灰度值来实现距离的计算。

同时，在 kNN 分类算法中，若样本在某给定属性上的值缺失，也需要进行处理。比如元组 X_1 和/或 X_2 对应属性 A 的值缺失，我们假定取最大的可能差：在假定每个属性都已经映射到[0, 1]之间的前提下，对于标称属性，若 A 的一个或两个对应值缺失，则取差值为 1。对于数值属性，若 A 的两个对应值都缺失，则取差值为 1；若 A 的一个对应值缺失，另一值存在为 v'，则取差值为 $|1-v'|$ 和 $|0-v'|$ 中的最大者。

7.2.2 kNN 算法实现

kNN 算法没有明显的训练过程，它在训练阶段只是把数据保存下来，训练时间开销为 0，等收到测试样本后直接进行处理。下面我们以一个具体事例，说明 kNN 分类算法的实现过程。

例题 7.1　设某班 20 位同学的成绩及相应类别如下表。现新转入一位同学张佳，其成绩为 83，令 $k=5$，试采用 kNN 分类算法判断张佳的成绩属哪个类别。

学生 id	姓名	成绩	类别	学生 id	姓名	成绩	类别
X_1	秦好	96	A	X_{11}	张峰	56	E
X_2	张杰	85	B	X_{12}	王波	76	C
X_3	魏鑫	90	A	X_{13}	刘涛	69	D
X_4	梁瑞	67	D	X_{14}	石阡	78	C
X_5	李涵	49	E	X_{15}	朱莉	83	B
X_6	厉韩	66	D	X_{16}	张骞	33	E
X_7	施杨	75	C	X_{17}	马燕	90	A
X_8	于敏	89	B	X_{18}	曹刈	79	C
X_9	张弛	97	A	X_{19}	张韧	82	B
X_{10}	黄莉	86	B	X_{20}	付涛	65	D

解： 因为只有成绩是与类别相关的属性，用 X_i 表示第 i 位同学的成绩，用曼哈顿距离(7.2)式度量 X_i 与张佳的成绩 Z_1 之间的距离 D_{ij}。

学生 id	姓名	成绩	类别	D_{ij}	学生 id	姓名	成绩	类别	D_{ij}
X_1	秦好	96	A	13	X_{11}	张峰	56	E	27
X_2	张杰	85	B	2	X_{12}	王波	76	*C*	7
X_3	魏鑫	90	A	7	X_{13}	刘涛	69	*D*	14
X_4	梁瑞	67	D	16	X_{14}	石阡	78	C	5
X_5	李涵	49	E	34	X_{15}	朱莉	83	B	0
X_6	厉韩	66	D	17	X_{16}	张骞	33	E	50
X_7	施杨	75	C	8	X_{17}	马燕	90	A	7
X_8	于敏	89	B	6	X_{18}	曹刈	79	C	4
X_9	张弛	97	*A*	14	X_{19}	张韧	82	B	1
X_{10}	黄莉	86	*B*	3	X_{20}	付涛	65	D	18

根据距离发现与 Z_1 最邻近的 5 个数据是 X_{15}、X_{19}、X_2、X_{10}、X_{18}、X_{14}。

学生 id	姓名	成绩	类别	D_{ij}
X_{15}	朱莉	83	B	0
X_{19}	张韧	82	B	1
X_2	张杰	85	B	2
X_{10}	黄莉	86	B	3
X_{18}	曹刈	79	C	4

其中，B 类别 4 个，C 类别 1 个，判定张佳的成绩为类别 B。

综上，对未知类别属性的数据集中的每个点依次执行以下操作，即可实现 kNN 分类算法：

Step1. 计算已知类别数据集中的点与当前点之间的距离；

Step2. 按照距离递增次序排序；

Step3. 选取与当前点距离最小的 k 个点；

Step4. 确定前 k 个点所在类别的出现频率；

Step5. 返回前 k 个点出现频率最高的类别作为当前点的预测分类。

下面是例题 7.1 的 Python 代码实现：

```
import numpy as np
import math
def createDataSet( ):                                    #给出训练数据以及对应的类别
    data = np. array([[96], [85], [90], [67], [49], [66], [75], [89] ,[97], [86], [56], [76],
                      [69], [78], [83], [33], [90], [79], [82], [65]])
    labels = ['A', 'B', 'A', 'D', 'E', 'D', 'C', 'B', 'A', 'B', 'E', 'C', 'D', 'C', 'B', 'E', 'A', 'C', 'B', 'D']
    return data, labels
def classify(input, dataSet, label, k):                  #通过 KNN 进行分类
dataSize = len(dataSet)
diff = tile(input, dataSize) -  dataSet
sqdiff = diff * * 2
squareDist = sum(sqdiff, axis = 1)
dist = squareDist* * 0. 5
sortedDistIndex = argsort(dist)
classCount = { }
for i in range(k):
    voteLabel = label[sortedDistIndex[i]]
    classCount[voteLabel] = classCount. get(voteLabel, 0) + 1
    maxCount = 0
for key, value in classCount. items( ):
    if value > maxCount:
        maxCount = value
        classes = key
    return classes
dataSet, labels = createDataSet( )                       #运行分类
input = np. array([83])
```

```
K = 5
output = classify(input, dataSet, labels, K)
print(output)
```

kNN 算法是思想简单、易于实现的分类方法。但因为它需要计算待分类样本与全部训练集样本的距离，所以不适用于数据量过大、属性较多的数据集分类。并且 kNN 算法可理解性差，无法给出像决策树那样的规则。

7.3 决策树分类方法

决策树是一种归纳分类算法，即从一组无次序、无规则，但有类别标号的样本集中推理出的树形表示的分类规则。与决策树相关的重要分类算法有很多，本节主要介绍 ID3 和 C4.5 算法。

7.3.1 决策树分类算法概述

决策树又称为分类树，是一种类似于流程图的树结构，其中每个内部结点对应一个属性上的测试，所包含的样本集根据属性测试的结果被划分到下一级节点中；每个叶节点存放一个类标号(决策结果)，树的最顶层节点是根节点，从根节点到每个叶子节点的路径对应一条决策规则，具体结构如图 7.3 所示。

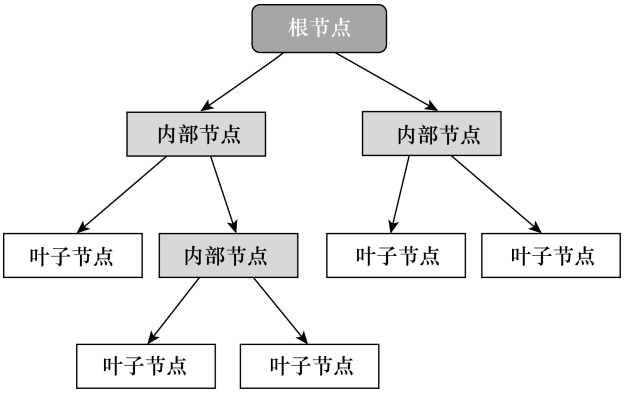


图 7.3 决策树模型

显然，决策树分类算法遵循自顶而下、分而治之的策略，即给定一个类标号未知的数据样本 X，在决策树的内部节点测试该数据的属性值，并根据不同的属性值判断从该节点向下的分支，直到叶节点得到结论。跟踪一条由根节点到叶节点的路径可得一条合取规则。决策树算法的最大优势在于它的学习过程不需要任何领域知识或参数设置，只需要训练数据集，能够用“属性-结论”式表示即可。决策树的建模过程通常分为决策树的生成和剪枝两部分。

7.3.1.1 决策树的生成

假设节点 N 对应的样本集为 S_N，$C=\{C_1, C_2, \cdots, C_k\}$ 是其类别属性，决策树归纳算法的基本思路为：

（1）如果 S_N 中所有的样本点都属于一个类别 C_N，则 N 为叶节点，并用分类号 C_N 标记该节点；

（2）如果 S_N 中样本点归属于多个类别，选择一个“好”的属性 A，以属性 A 命名该节点，并将 N 作为内部节点。然后按属性 A 的取值将集合 S_N 划分为较小子集，针对每个子集创建 A 的子节点；

（3）以每个子节点作为 N，重复调用(1)和(2)，直至能将所有训练数据进行完美分类(如图 7.4)。

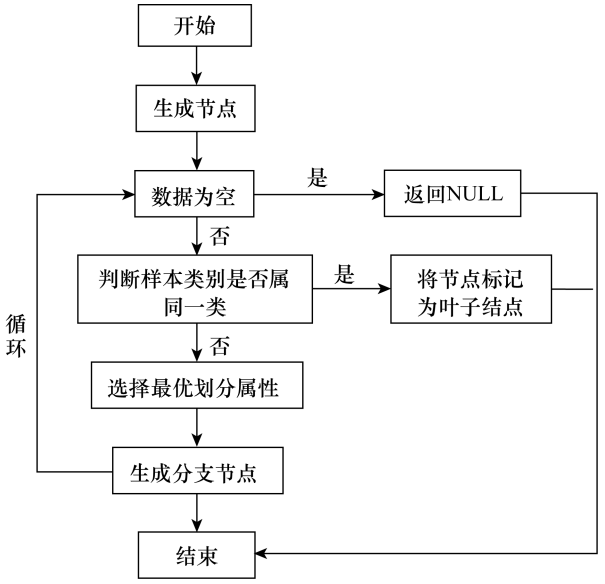


图 7.4 决策树归纳算法流程图

从上述递归过程不难看出，生成决策树必须解决两个关键问题：第一，最优属性选择。即树生长的每次递归都必须要选择一个属性作为测试条件，将测试集划分为更纯的子集。该属性的选择顺序不仅影响决策树的结构，还会影响决策树的准确率。常用的属性选择度量方法有信息增益、信息增益比、基尼指数、距离度量等等，而不同的度量方法将会产生不同的决策树算法；第二，停止分裂准则。为了不让树过分成长，通常需要提前停止树的生长过程，一般有以下三种情况则停止生长：

（1）节点包含的所有样本都属于同一类，或者所有的样本都具有相同的属性值；

（2）子节点为空，即不存在与这该节点条件相关联的样本点，此时将该节点设为叶节点，但其类别标号采用其父节点上多数样本的类别标号；

（3）训练集 S_N 的候选属性集合为空，但类别标号却不同。此时，将该节点设为叶节

点，其类别标号采用该节点多数样本的类别标号。

决策树生成的基本算法如图 7.5 所示。

算法：Generate_decision_tree(samples， attribute_list)

输入：训练样本集 samples，由离散值属性表示；候选属性的集合 attribute_list。

输出：一棵决策树。

(1) 创建结点 ；

(2) if samples 都在同一个类 中，then 返回 作为叶结点，以类 标记；

(3) if attribute_list 为空，then 返回 作为叶结点，标记为 samples 中最普通的类；

(4) 选择 attribute_list 中具有最高信息增益属性 test_attribute；

(5) 标记结点 为 test_attribute；

(6) for each test_attribute 中的值 ，由节点 长出一个条件为 test_attribute =的分支；

(7) 设 是 samples 中 test_attribute =ai 的样本的集合；

(8) if 为空，then 加一个叶节点，标记为 samples 中最普通的类；

(9) 否则加上一个由 Generate_decision_tree(， attribute_list- test_attribute)返回的结点。

图 7.5 决策树生成基本算法

7.3.1.2 决策树的剪枝

在决策树创建时，其生长方式是“完全的”，但由于实际训练数据样本中存在的噪声、离群点等，会使得决策树的许多分枝反映的是训练数据中的异常。剪枝则是克服数据异常的主要技术，即通过剪掉庞杂的、不可靠的分枝提高决策树模型泛化能力，让决策树得到简化、易于理解。常用的剪枝方法有**预剪枝**和**后剪枝**。

预剪枝是在决策树生成的同时决定是否结束树的生长，一旦停止则将该结点当作叶子结点，不再予以划分。当然，该方法会导致树的生长不充分，出现“欠拟合”现象。

后剪枝是在决策树生长停止之后进行的剪枝。一般从树的叶子节点开始逐步向根节点进行修剪，剪枝通过删除节点的分枝并用叶子替换它而剪掉给定点上的子树。子树中最多数的类标号即为该叶子节点的类标号。理论上，后剪枝优于预剪枝，但其计算复杂度高。

决策树模型形式直观，易于理解。算法所需的数据准备工作相对较少，不需要使用者了解更多的背景知识，适合探索式的知识发现，并且学习和分类步骤简单快速，具有很好的准确率。决策树算法之间的差别在于创建树时的属性选择和剪枝机制的不同，下面我们以 ID3 和 C4.5 算法为例进行介绍。

7.3.2 ID3 算法

在 20 世纪 70 年后期到 80 年代初期，J. Ross Quinlan 开发了决策树算法——迭代二分器(Iterative Dichotomier，ID3)。该算法扩展了概念学习系统，是一种贪心学习方法，它将信息增益作为属性选择度量。

7.3.2.1 信息增益

信息增益是指划分数据集之前之后信息发生的变化。现假设按某属性 N 划分 D 中的样本，属性 A 根据训练数据的观测具有 m 个不同值(a_1，a_2，…，a_m)。如果 A 是离散值的，则这些值直接对应于 A 上测试的 m 个输出。可以用属性 A 将 D 划分为 m 个分区或子集$\{D_1, D_2, \cdots, D_m\}$，这里 D_j 包含 D 中的样本，它们的 A 值为 a_j。这些分区对应于从结点 N 生长出来的分枝。理想情况下，我们希望该划分产生样本的准确分类。即我们希望每个分区都是纯的。然而，这些分区多半是不纯的，则由属性 A 划分成子集的熵可表示为：

$$H_A(D) = \sum_{j=1}^{m} \frac{|D_j|}{|D|} \times H(D_j) \tag{7.4}$$

其中$\frac{|D_j|}{|D|}$充当第 j 个分区的权重，$H(D)$是 D 的熵。从而由 A 进行分支将获得的信息增益：

$$\text{Gain}(A) = H(D) - H_A(D) \tag{7.5}$$

显然，Gain(A)是原信息需求(仅基于类比例)与新信息需求(对 A 划分后)之间的差。ID3 算法选择具有最高信息增益的属性 A 作为节点 N 的分裂属性。这等价于在“能做最佳分类”的属性 A 上划分，使得完成样本分类还需要的信息最小(即最小化 $H_A(D)$)。

7.3.2.2 ID3 算法思想

ID3 算法的核心思想是用信息增益作为在各级节点属性选择的标准，以期在非叶结点处测试时，获得被测试记录最大的类别信息。即在树的创建时，计算每个属性的信息增益，将具有最高信息增益的属性作为集合 S 的划分标准，创建一个结点，并以该属性标记。对属性的每个不同取值创建分枝，然后递归调用上述过程，直至所有属性的信息增益均很小或没有属性可供选择为止。具体实现步骤如下：

Step1. 创建一个节点。若其包含的样本均为同一类，则算法停止，该节点改成叶节点，并用该类别标记；

Step2. 否则，采用信息增益作为度量标准进行分枝，具有最高信息增益的属性将成为该节点的测试属性；

Step3. 对测试属性的每个值创建分枝，并据此划分样本；

Step4. 自顶而下递归调用上述过程，直到给定节点的所有样本属性都属于同一类，或者子节点为空，或者可选属性集合为空，则停止递归。

需要注意的是决策树算法从根节点开始，根节点包括所有的样本。下面我们以一个实例来展示 ID3 算法的具体实现过程。

例题 7.2　以下表数据为例，利用信息增益作为属性选择度量生成决策树。

序号	家庭收入	年级	性别	电脑
1	中等	低	1	0
2	中等	低	0	1
3	低	低	1	0
4	高	高	1	1
5	中等	高	1	1
6	中等	高	0	1
7	高	高	1	1
8	中等	低	1	0
9	高	低	1	0
10	低	高	0	1

解：记所给样本数据集为 D，显然 4 个属性都是离散的，类别属性有 2 个值 0 和 1，即 $m=2$。设类 C_1 对应 1(有电脑)，类 C_2 对应 0(无电脑)，类 C_1 有 6 个样本，类 C_2 有 4 个样本。

为 D 创建(根)节点 N。为了找出这些样本的划分准则，需要计算每个属性的信息增益。首先利用式(5.30)计算 D 中样本分类所需要的期望信息：

$$H(D)=-\frac{6}{10}\log_2\frac{6}{10}-\frac{4}{10}\log_2\frac{4}{10}=0.972$$

下面依据(7.4)式计算一个属性的期望信息需求：

$$H_{\text{家庭收入}}(D)$$

$$=\frac{3}{10}\times\left(-\frac{2}{3}\log_2\frac{2}{3}-\frac{1}{3}\log_2\frac{1}{3}\right)+\frac{5}{10}\times\left(-\frac{3}{5}\log_2\frac{3}{5}-\frac{2}{5}\log_2\frac{2}{5}\right)+\frac{2}{10}\times$$

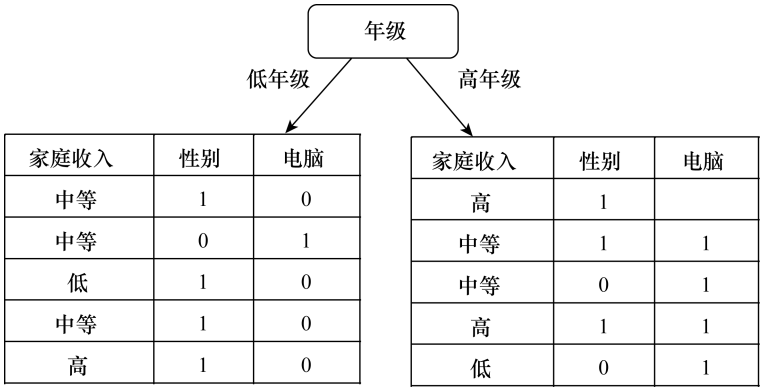
$$\left(-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}\right)$$

$=0.961$ 位。

该种划分的信息增益为：

$$\text{Gain(家庭收入)}=H(D)-H_{\text{家庭收入}}(D)=0.972-0.961=0.011\text{ 位}$$

类似求得 Gain(年级)=0.612 位，Gain(性别)=0.282 位。由于年级在属性中具有最高的信息增益，所以它被选作划分属性。节点 N 标记为年级，并且每个属性值生长出一个分支。然后将该样本集进行划分，如图 7.6 所示



家庭收入	性别	电脑
中等	1	0
中等	0	1
低	1	0
中等	1	0
高	1	0

家庭收入	性别	电脑
高	1	
中等	1	1
中等	0	1
高	1	1
低	0	1

图 7.6　基于“年级”属性对根节点的划分

紧接着我们划分子节点，先看右分支生长时发现对于高年级的所有样本，其类别标记都是 1，它是一个叶子节点。再看左分支，该分支需要计算其它两个属性的信息增益：Gain(家庭收入) = 0.171 位，Gain(性别) = 0.721 位，所以以性别作为划分标准，得最后的决策树如图 7.7 所示。

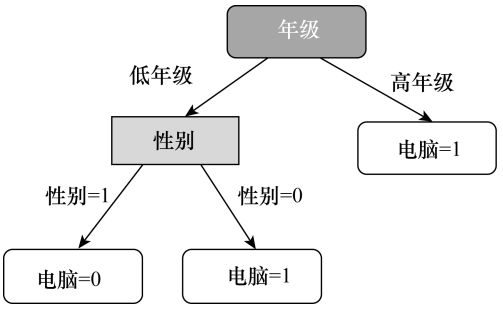


图 7.7　基于信息增益生成的决策树

下面是例题 7.2 的 Python 代码实现：

```
from math import log
def createDataSet( ):                                    #创建数据集
    dataSet = [['中等', '低', '1', '0'], ['中等', '低', '0', '1'], ['低', '低', '1', '0'], ['高', '高', '1', '1'],
        ['中等', '高', '1', '1'], ['中等', '高', '0', '1'], ['高', '高', '1', '1'], ['中等', '低', '1', '0'],
        ['高', '低', '1', '0'], ['低', '高', '0', '1']]
    featureName = ['家庭收入', '年级', '性别']
    return dataSet, featureName
def splitDataSet(dataSet, axis, value):                  #分割数据集
    retDataSet = [ ]
    for featVec in dataSet:
```

```
            if featVec[axis] = = value:
                deletFeatVec = featVec[:axis]
                deletFeatVec. extend(featVec[axis+1:])
                retDataSet. append(deletFeatVec)
    return retDataSet
def ShannonEntropy(dataSet):                                    #计算信息熵
    numEnters = len(dataSet)
    labelCounts = { }
    for featVec in dataSet:
        currentLabel = featVec[- 1]
        if currentLabel not in labelCounts. keys( ):
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] + = 1
    Entropy = 0. 0
        for key in labelCounts:
        prob = float(labelCounts[key]) / numEnters
        Entropy - = prob * log(prob, 2)
    return Entropy
def ConditionalEntropy(dataSet, i, featList, uniqueVals):       #计算条件熵
    ce = 0. 0
    for value in uniqueVals:
        subDataSet = splitDataSet(dataSet, i, value)
        prob = len(subDataSet) / float(len(dataSet))
        ce + = prob * ShannonEntropy(subDataSet) #∑pH(Y|X = xi)
    return ce
def InformationGain(dataSet, dataEntropy, i):                   #计算信息增益
    featList = [example[i] for example in dataSet]
    uniqueVals = set(featList)
    newEntropy = ConditionalEntropy(dataSet, i, featList, uniqueVals)
    infoGain = dataEntropy - newEntropy
    return infoGain
def BestFeatureToSplitByID3(dataSet):                           #选择最优划分
    numFeatures = len(dataSet[0]) - 1
    dataEntropy = ShannonEntropy(dataSet)
    bestInfoGain = 0. 0
    bestFeature = - 1
```

```
    for i in range(numFeatures):
        infoGain = InformationGain(dataSet, dataEntropy, i)
        if(infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature
def createTree(dataSet, featureName, BestFeatureToSplitFunc = BestFeatureToSplitByID3):
                                                    #创建决策树
    classList = [example[- 1] for example in dataSet]
    if classList. count(classList[0]) = = len(classList):
        return classList[0]
    if len(dataSet[0]) = = 1:
    return majorityCnt(classList)
    bestFeat = BestFeatureToSplitFunc(dataSet)
    bestFeatLabel = featureName[bestFeat]
    myTree = {bestFeatLabel:{ }}
    del(featureName[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = featureName[:]
         myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),
subLabels)
    return myTree
dataSet, featureName = createDataSet( )                    #测试决策树
myTree = createTree(dataSet, featureName)
print(myTree)
```

虽然 ID3 算法的理论清晰，方法简单，分类速度较快。但该算法只能处理离散属性的数据，对于连续情况没有考虑，并且它不能处理有缺失的数据。而且 ID3 算法采用信息增益作为属性选择的度量标准，倾向于选择取值较多的属性，这并不一定是最优属性，会导致提供的有价值的信息不多。另外，ID3 算法对过拟合情况也未曾考虑，基于这些不足，Quinlan 在 1993 年提出了改进算法——C4. 5 算法。

7. 3. 3　C4. 5 算法

在 ID3 算法的基础上，C4. 5 算法增加了对连续属性、空缺值的处理，并以信息增益率作为选择属性，克服了利用信息增益选择属性时偏向选择取值多且均匀的属性的不足。

7.3.3.1 信息增益率

信息增益率是在信息增益的基础上发展而来，一个属性(变量)的信息增益比，其定义为：

$$\text{Gainrate}(A)=\frac{\text{Gain}(A)}{Split_A(D)} \tag{7.6}$$

其中

$$Split_A(D)=-\sum_{j=1}^{n}\frac{|D_j|}{|D|}\times\log_2\left(\frac{|D_j|}{|D|}\right) \tag{7.7}$$

称为属性 A 的分裂信息。代表以属性(特征)A 的值对样本集 D 进行划分的信息熵。属性 A 的可能取值数目越多(n 越大)，则 $Split_A(D)$ 的值通常越大，从而抵消了属性取值数目所带来的影响。C4.5 算法在构造决策树时，利用信息增益率最大的属性作为当前节点的分裂属性。当划分信息逐渐减小时，选择相对比较大的信息增益率的属性作为分裂属性，但至少与考察的所有测试的平均增益一样大。

7.3.3.2 连续型属性离散化

假设属性 A 是连续型属性，如长度、年龄的原始值，则必须对数据进行离散化处理。C4.5 算法针对连续属性采用二分法进行离散化处理：

(1) 将样本集在属性 A 上的 n 个属性值由小到大进行排序：v_1，v_2，…，v_n。

(2) 依次将每相邻的两个属性值的平均值看做可能的分裂点：

$$\bar{v}_i=\frac{v_i+v_{i+1}}{2}$$

则共有 $n-1$ 个分裂点，每个分裂点都将样本划分为 2 个子集，分别对应 $A\leqslant\bar{v}_i$ 和 $A>\bar{v}_i$；

(3) 计算每个分裂点 $\bar{v}_i(i=1,2,\cdots,n-1)$ 划分样本集的信息增益，选择具有最大信息增益的分裂点 $\bar{v}$，即为属性 A 二分阈值，它将数据集分割为 $A\leqslant\bar{v}$ 和 $A>\bar{v}$ 两个子集。

另外，C4.5 算法还可以通过删除训练集中空值的样本以保证训练集中的样本属性值全部不为空，或者以某种方式填补空值，比如用属性值的均值、频率最高的属性值填充空缺。

7.3.3.3 剪枝

C4.5 算法采用一种称为悲观剪枝的后剪枝方法，它由 Quinlan 提出，自上而下根据剪枝前后的错误率来判断是否进行子树修剪，因此不需要单独的剪枝数据集。具体做法为：

(1) 计算剪枝前错误率

对于一个叶子节点，它覆盖了 n 个样本，其中有 m 个错误，那么该叶子节点的错误率为 $\frac{m+\varepsilon}{n}$，其中惩罚因子 ε 一般取 0.5。而对一棵子树而言，若其包含了 l 个叶子节点，那

么其错误率即为

$$e_{tree}=\frac{\sum m_i+0.5l}{\sum n_i} \tag{7.8}$$

其中，e_i 表示子树第 i 个叶子节点错误分类的样本个数，n_i 表示子树第 i 个叶子节点中样本总数。

（2）计算剪枝前误判次数均值

对于样本误差率，可以根据经验进行估计。如果将错误分类一个样本记为值 1，正确分类记为值 0，则该树错误分类率 e 服从伯努利分布，则可以估计该树的误判次数均值和标准差：

$$E_{tree}=Ne_{tree},\ Var_{tree}=\sqrt{N\cdot e_{tree}\cdot(1-e_{tree})}$$

其中，$N=\sum n_i$ 为样本总数。

（3）计算剪枝后误判次数均值

当把子树替换为叶子节点后，该叶子节点的误判次数也是一个伯努利分布，其误判概率为 $e_{leaf}=\frac{m+\varepsilon}{n}$，则该叶子节点的误判次数均值为

$$E_{leaf}=Ne_{leaf}$$

（4）判断剪枝条件

虽然子树的错误率一般低于叶节点的错误率，但加上惩罚因子后就未必了。若将内部节点变成叶子节点，子树的误判个数大于对应叶子节点的误判个数一个标准差时，则可剪枝。即当

$$E_{tree}-Var_{tree}\geqslant E_{leaf}$$

成立时，可进行剪枝操作。

C4.5 算法继承了 ID3 优点，并对其进行了多方面的改进，但它在树的构造时，需要对数据集进行多次的顺序扫描和排序，算法效率较低。

7.4 朴素贝叶斯分类

贝叶斯分类方法是统计学分类方法，它以样本可能属于某类的概率作为分类依据。本节以其中最简单，但却可与决策树、神经网络相媲美的分类方法——朴素贝叶斯方法为主介绍贝叶斯分类方法。

7.4.1 相关概念

我们先回忆几个概念。

定义 7.1 （后验概率）设 A，B 是样本空间 Ω 中任意两事件，则称

$$P(A\mid B)=\frac{P(AB)}{P(B)},\ (P(B)>0) \tag{7.9}$$

为事件 B 已发生的条件下，事件 A 发生的**条件概率**。此时，也称 $P(A\mid B)$ 是**后验概率**，或在条件 B 下，A 的后验概率。而称 $P(A)$ 是**先验概率**，或 A 的先验概率。显然，先验概率泛指一类事件发生的概率，通常根据历史资料或主观判断，未经实验证实所确定的概率。而后验概率涉及的是某个特定条件下一个具体的事件发生的概率。

定理 7.1 （贝叶斯定理）设 Ω 是实验的样本空间，A 为其一事件，B_1，B_2，…，B_m 为样本空间 Ω 的一个划分，且 $P(A)>0$，$P(B_i)>0$，$(i=1,2,\cdots,m)$ 则有

$$P(B_i\mid A)=\frac{P(A/B_i)P(B_i)}{\sum_{j=1}^{n}P(A\mid B_j)P(B_j)},\quad i=1,2,\cdots,m \tag{7.10}$$

式(7.10)称为贝叶斯公式。显然，若假设导致事件 A 发生的“原因”有 $B_i(i=1,2,\cdots,m)$ 个，它们互不相容。那么，贝叶斯公式给出的是已知事件 A 确定已经发生了，导致它发生的“原因”B_i 的概率，即可从结果分析原因的过程。

定义 7.2 （贝叶斯分类算法）假设 X 是类的标号未知的数据样本。设 H 为某种假定（如数据样本 X 属于某特定的类 C），称如下公式给出的计算 $P(H|X)$ 的方法为贝叶斯分类算法：

$$P(H\mid X)=\frac{P(X\mid H)P(H)}{P(X)} \tag{7.11}$$

其中 $P(H)$ 是先验概率，$P(X|H)$ 代表假设 H 成立的情况下，观察到 X 的概率。$P(H|X)$ 是后验概率。

贝叶斯分类对完全独立的数据和函数依赖的数据具有较好的分类效果，我们主要介绍对类条件独立的朴素贝叶斯分类方法。

7.4.2 朴素贝叶斯分类

朴素贝叶斯是贝叶斯分类方法的一个特例，它假定样本集的所有属性相互独立。这一假定有效简化了计算，因此在实际应用中，我们希望将复杂的贝叶斯分类拆分成多个朴素贝叶斯来表达。

假设 D 为训练样本集和相关联的类标号集合。其中训练样本的属性集为 $X=\{x_1,x_2,\cdots,x_n\}$，类标号为 $C=\{C_1,C_2,\cdots,C_m\}$，朴素贝叶斯的基本思想就是在此样本出现的条件下，找出各类别 $C_i(i=1,2,\cdots,m)$ 出现的概率 $P(C_i|X)$ 最大值，则该类别即为待分类样本的类别。具体过程如下：

1. 每个数据样本用一个 n 维属性向量 $X=\{x_1,x_2,\cdots,x_n\}$ 表示，描述由 n 个属性 A_1，A_2，…，A_n 对样本的 n 个度量。

2. 假定有 m 个类 C_1，C_2，…，C_m。对于给定的数据样本 X，预测它属于具有最高后验概率的类。即将给定的未知样本分配给类 $C_i(1\leqslant i\leqslant m)$，当且仅当

$$P(C_i\mid X)>P(C_j\mid X)\quad(1\leqslant j\leqslant m,\ j\neq i)$$

我们称 $P(C_i|X)$ 的类 C_i 为最大后验假设。那么由贝叶斯公式可得

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)} \tag{7.12}$$

3. 由于对所有类 $P(X)$ 均为常数，所以只需要 $P(X \mid C_i)P(C_i)$ 最大即可。如果 C_i 类的先验概率未知，则可假定这些类是等概率的，据此将(7.12)的最大化转化为 $P(X \mid C_i)$ 的最大化。否则，最大化 $P(X \mid C_i)P(C_i)$。不过类的先验概率可通过训练样本求得：$P(C_i) = |C_{i,D}|/|D|$，其中 $|C_{i,D}|$ 是 D 中 C_i 类的训练样本数。

4. 给定具有许多属性的数据集，计算 $P(X \mid C_i)$ 的开销可能非常大。为降低此项开销，可以做**类条件独立**的朴素假定。即一个属性在给定类上的影响独立于其它属性值。如此只用考虑分子：

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) \tag{7.13}$$

其中概率 $P(x_1 \mid C_i)$，…，$P(x_n \mid C_i)$ 可由训练样本进行估计。这里 x_k 表示样本 X 在属性 A_k 的取值，对于每个属性，分以下两种情况进行计算：

(1) A_k 为离散属性，则 $P(x_k \mid C_i) = |C_{i,k,D}|/|C_{i,D}|$，其中 $|C_{i,k,D}|$ 是 D 中属性 A_k 的值为 x_k 的 C_i 类的训练样本数。

(2) A_k 为连续值属性，则通常假定该属性服从高斯分布 $N(\mu, \sigma^2)$，从而有

$$P(x_k \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} \frac{(x_k - \mu_{C_i})^2}{2\sigma_{C_i}^2} \tag{7.14}$$

其中，μ_{C_i}，σ_{C_i} 分别是 C_i 类训练样本属性 A_k 的均值和标准差。

5. 对待分类样本 X，对每个类 C_i，计算 $P(X \mid C_i)P(C_i)$。样本 X 被标号类 C_i，当且仅当

$$P(X \mid C_i)P(C_i) > P(X \mid C_j)P(C_j) \qquad (1 \leqslant j \leqslant m,\ j \neq i) \tag{7.15}$$

也就是说，X 被指派到其 $P(X \mid C_i)P(C_i)$ 最大的类。

例题 7.3 利用下表数据，对待测样本 X = (家庭收入 = 中等，年级 = 低，性别 = 0) 进行是否有电脑的分类。

序号	家庭收入	年级	性别	电脑
1	中等	低	1	0
2	中等	低	0	1
3	低	低	1	0
4	高	高	1	1
5	中等	高	1	1
6	中等	高	0	1
7	高	高	1	1
8	中等	低	1	0

续表

序号	家庭收入	年级	性别	电脑
9	高	低	0	0
10	低	高	0	1
11	低	高	1	0

解： 首先估计类先验概率 $P(C)$。记类 C_1 为“电脑=1”，类 C_2 为“电脑=0”，则

$P(C_1)=\frac{6}{11}\approx 0.545$；$P(C_2)=\frac{5}{11}\approx 0.455$

然后为每个属性估计条件概率 $P(x_k \mid C_i)$：

$P(\text{家庭收入}=\text{高} \mid C_1)=\frac{2}{6}\approx 0.333$；　$P(\text{家庭收入}=\text{高} \mid C_2)=\frac{1}{5}=0.2$；

$P(\text{家庭收入}=\text{中等} \mid C_1)=\frac{3}{6}=0.5$；　$P(\text{家庭收入}=\text{中等} \mid C_2)=\frac{2}{5}=0.4$；

$P(\text{家庭收入}=\text{低} \mid C_1)=\frac{1}{6}\approx 0.167$；　$P(\text{家庭收入}=\text{低} \mid C_2)=\frac{2}{5}=0.4$；

$P(\text{年级}=\text{高} \mid C_1)=\frac{5}{6}\approx 0.833$；　$P(\text{年级}=\text{高} \mid C_2)=\frac{1}{5}=0.2$；

$P(\text{年级}=\text{低} \mid C_1)=\frac{1}{6}\approx 0.167$；　$P(\text{年级}=\text{低} \mid C_2)=\frac{4}{5}=0.8$；

$P(\text{性别}=1 \mid C_1)=\frac{3}{6}=0.5$；　$P(\text{性别}=1 \mid C_2)=\frac{4}{5}=0.8$；

$P(\text{性别}=0 \mid C_1)=\frac{3}{6}=0.5$；　$P(\text{性别}=0 \mid C_2)=\frac{1}{5}=0.2$。

于是

$$P(C_1 \mid X)=0.545\times 0.333\times 0.5\times 0.167\times 0.833\times 0.167\times 0.5\times 0.5=5.3\times 10^{-4}$$

$$P(C_2 \mid X)=0.455\times 0.2\times 0.4\times 0.4\times 0.2\times 0.8\times 0.8\times 0.2=3.7\times 10^{-4}$$

由于 $5.3\times10^{-4}>3.7\times10^{-4}$，所以朴素贝叶斯分类方法将样本 X 划分为 C_1 类，即有电脑。下面是例题 7.3 的 Python 代码实现：

```
import numpy as np
import copy as cp
def loadDataSet( ):                                          #加载训练样本
    postingList=[['中等', '低', '1'], ['中等', '低', '0'], ['低', '低', '1'], ['高', '高', '1'],
                 ['中等', '高', '1'], ['中等', '高', '0'], ['高', '高', '1'], ['中等', '低', '1'],
                 ['高', '低', '0'], ['低', '高', '0'], ['低', '高', '1']]
    classVec=[0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0]
```

```
    return postingList, classVec
def createVocabList(dataSet):                                    #求特征向量
    vocabSet = set([ ])
    for document in dataSet:
        vocabSet = vocabSet | set(document)
    return list(vocabSet)
def word2toVec(vocabList, inputWord):                            #向量化
    returnVec = [0] * len(vocabList)
    for word in inputWord:
        if word in vocabList:
            returnVec[vocabList. index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return returnVec
def train(trainMatrix, trainLabels):                             #计算概率
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = (sum(trainLabels)+1. 0)/(float(numTrainDocs)+2. 0* 1. 0)
    p0Num = np. ones(numWords)
    p1Num = np. ones(numWords)
    p0Denom = 2. 0 + len(trainLabels)- sum(trainLabels)
    p1Denom = 2. 0 + sum(trainLabels)
    for i in range(numTrainDocs):
        if trainLabels[i] = = 1:
            p1Num + = trainMatrix[i]
        else:
            p0Num + = trainMatrix[i]
    p0Vect = np. log(p0Num/p0Denom)
    p1Vect = np. log(p1Num/p1Denom)
    return p0Vect, p1Vect, pAbusive
def classify(vec2Clssify, p0Vect, p1Vect, pClass1):              #分类
    p1 = sum(vec2Clssify * p1Vect) + np. log(pClass1)
    p0 = sum(vec2Clssify * p0Vect) + np. log(1- pClass1)
    if p1>p0:
        return 1
```

```
    else:
        return 0
def testing( ):                                    #预测
    listOPosts, listClasses = loadDataSet( )
    wordList = createVocabList(listOPosts)
    dataMatrix = [ ]
    for item in listOPosts:
        dataMatrix. append(word2toVec(wordList, item))
    p0, p1, pAB = train(dataMatrix, listClasses)
    testEntry = ['中等', '低', '0']
    thisClass = np. array(word2toVec(wordList, testEntry))
    print(testEntry, 'classified as: ', classify(thisClass, p0, p1, pAB))
testing()
```

值得注意是，在该例中如果没有第十一个样本，则 $P(\text{年级}=\text{高}\mid C_2)$ 的概率值为 0，将其带入(7.14)式后，$P(C_2\mid X)=0$，它消除了乘积中涉及的所有其它非 0 后验概率的影响，显然是不合理的。面对这种情况，一般采用拉普拉斯修正。具体修改表达式为：

$$\hat{P}(C_i)=\frac{|C_{i,D}|+1}{|D|+N} \tag{7.16}$$

$$\hat{P}(x_k\mid C_i)=\frac{|C_{i,k,D}|+1}{|C_{i,D}|+N_i} \tag{7.17}$$

其中，N 表示训练集 N 中个能的类别数，N_i 表示第 i 个属性可能取值数。如上假设，可估计条件概率为：

$$\hat{P}(C_1)=\frac{6+1}{10+2}=\frac{7}{12};\qquad \hat{P}(C_2)=\frac{4+1}{10+2}=\frac{5}{12};$$

$$\hat{P}(\text{家庭收入}=\text{高}\mid C_1)=\frac{2+1}{6+3}=\frac{1}{3};\qquad \hat{P}(\text{家庭收入}=\text{高}\mid C_2)=\frac{1+1}{4+3}=\frac{2}{7};$$

$$\hat{P}(\text{年级}=\text{高}\mid C_1)=\frac{5+1}{6+2}=\frac{3}{4};\qquad \hat{P}(\text{年级}=\text{高}\mid C_2)=\frac{0+1}{4+2}=\frac{1}{6};$$

$$\hat{P}(\text{性别}=1\mid C_1)=\frac{3+1}{6+2}=\frac{1}{2};\qquad \hat{P}(\text{性别}=1\mid C_2)=\frac{3+1}{4+2}=\frac{2}{3}。$$

显然，拉普拉斯修正值与未校正的估计值很接近，虽有可能改变结果，但避免了零概率值，并且随着训练样本集的增大，修正过程引入的先验影响也会逐渐变得可忽略不计。

朴素贝叶斯分类算法逻辑简单、易于实现，并且分类过程开销小，对小规模的数据表现很好。但朴素贝叶斯算法的核心是假设所有属性都彼此独立，这在现实应用中往往是不成立的。为了放松这一假定条件，人们进行了大量的工作，比如考虑部分属性之间依赖关系的半朴素贝叶斯算法、考虑属性之间联系的贝叶斯网络等。

7.5 支持向量机

支持向量机(Support Vector Machine，SVM)方法建立在统计学习理论基础之上，由Vladimir Vapnik 等于 20 世纪 60 年代在解决模式识别问题时提出。经过发展，SVM 算法已成为机器学习中应用非常广泛的分类算法。

7.5.1 基本模型

支持向量机最初由二分类问题引起，其基本思想是通过不断地拟合找出合适的边界，将相似的点划分到一边。设样本集 $D=\{(X_1, y_1), \cdots, (X_n, y_n)\}$，$y_i \in \{-1, +1\}$ 为类别标号，我们要解决的问题就是在样本空间中找到一个划分超平面。如图 7.8 所示，在同一个数据集可以建立多种不同的超平面，那么究竟哪个是最佳的超平面呢?

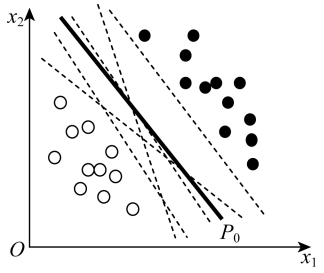


图 7.8 线性可分样本集示意图

由于训练样本的局限性或者噪声因素，我们希望找到的超平面对训练样本的局部扰动有很好的容忍性，所以应为两类训练样本正中间的最好，如图黑色标记的超平面 P_0，它将对新数据有较好的泛化能力。这样的超平面可以用如下线性方程来描述：

$$w^T X + b = 0, \tag{7.18}$$

其中，$w=(w_1; w_2; \cdots; w_d)$ 为法向量，表示平面的方向；b 为位移项，表示平面与原点之间的距离，则样本空间中任意点 x 到超平面 $w^T X+b=0$ 的距离可写为：

$$d = \frac{|w^T X + b|}{\| w \|} \tag{7.19}$$

若超平面 $w^T X+b=0$ 能将训练样本正确分类，则对于 $(X_i, y_i) \in D$，若 $y_i=+1$，则 $w^T X+b>0$；若 $y_i=-1$，则 $w^T X+b<0$，即

$$\begin{cases} w^T X_i + b \geqslant +1, & y_i = +1; \\ w^T X_i + b \leqslant -1, & y_i = -1. \end{cases} \tag{7.20}$$

如图 7.9 所示，距离超平面最近的这几个训练样本点使上式的等号成立，它们被称为**支持向量**，两个异类支持向量到超平面的距离之和为

$$\gamma = \frac{2}{\| w \|} \tag{7.21}$$

称该距离为**间隔**。

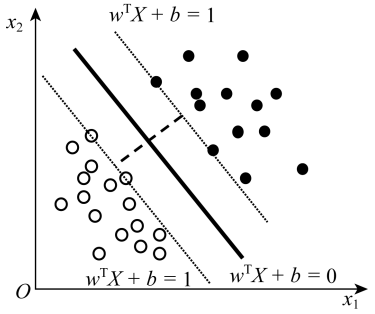


图 7.9　具有最大间隔的超平面示意图

找具有“最大间隔”的划分超平面，也就是在(7.21)约束下，让(7.21)达到最大，即

$$\max_{w,\ b} \frac{2}{\|w\|} \tag{7.22}$$
$$\text{s.t. } y_i(w^TX_i+b)\geqslant 1,\ i=1,\ 2,\ \cdots,\ n.$$

将 $\|w\|^{-1}$ 最大，转化为 $\|w\|^2$ 最小，可得支持向量机的基本模型：

$$\min_{w,\ b} \frac{1}{2}\|w\|^2 \tag{7.23}$$
$$\text{s.t. } y_i(w^TX_i+b)\geqslant 1,\ i=1,\ 2,\ \cdots,\ n.$$

7.5.2　模型求解

对于模型(7.23)，可以用拉格朗日乘子得其拉格朗日函数：

$$L(w,\ b,\ \alpha)=\frac{1}{2}\|w\|^2+\sum_{i=1}^{n}\alpha_i(1-y_i(w^TX_i+b)) \tag{7.24}$$

其中 $\alpha=(\alpha_1;\ \alpha_2;\ \cdots;\ \alpha_n)$，$\alpha_i\geqslant 0$，$(i=1,\ 2,\ \cdots,\ n)$。令 $L(w,\ b,\ \alpha)$ 关于 w 和 b 的偏导为零，解得

$$\begin{cases} w=\sum_{i=1}^{n}\alpha_i y_i X_i, \\ 0=\sum_{i=1}^{n}\alpha_i y_i. \end{cases} \tag{7.25}$$

将(7.25)带入，消去(7.24)中的 w 和 b，从而得到(7.23)的对偶问题

$$\begin{cases} \max\limits_{\alpha}\sum\limits_{i=1}^{n}\alpha_i-\frac{1}{2}\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}\alpha_i\alpha_j y_i y_j X_i^T X_j, \\ \text{s.t. } \sum\limits_{i=1}^{n}\alpha_i y_i=0, \\ \alpha_i\geqslant 0,\ (i=1,\ 2,\ \cdots,\ n). \end{cases} \tag{7.26}$$

通过(7.26)式解出 α 后，继而可得 w 和 b，从而解得间隔最大超平面模型：

$$f(X) = w^T X + b = \sum_{i=1}^{n} \alpha_i y_i X_i^T X + b \tag{7.27}$$

由(7.26)式解出的 α_i 是式(7.24)式的拉格朗日乘子，对应着训练样本(X_i，y_i)，所以，上述求解过程满足 KKT(Karush-Kuhn-Tucker)条件：

$$\begin{cases} \alpha_i \geqslant 0; \\ y_i f(x_i) - 1 \geqslant 0; \\ \alpha_i (y_i f(x_i) - 1) = 0. \end{cases} \tag{7.28}$$

从而，对任意训练样本(X_i，y_i)，总有 $\alpha_i = 0$ 或 $y_i f(X_i) - 1 = 0$。若 $\alpha_i = 0$，则该样本在(7.27)中的求和中不起作用，对 $f(X)$ 没有影响；若 $\alpha_i > 0$，则必有 $y_i f(X_i) = 1$，即该样本点位于最大间隔边界，是支持向量。这告诉我们支持向量机训练完后，大部分的训练样本不需要保留，模型仅与支持向量有关。

对于(7.26)式这个二次规划问题的求解有多种方法，这里不再展开。一旦我们得到训练后的支持向量机，那么将待测样本 X^T 带入(7.27)式，检查结果的符号，若大于 0，则待测样本归属 $y_i = +1$ 的类，反之，待测样本归属 $y_i = -1$ 的类。

7.5.3 核函数

之前讨论的都是假设训练样本为线性可分的。然而现实中，在原始样本空间中能正确划分两类样本的超平面并不总是存在，如图 7.10(a)所示，实心点和空心点是不能用线性分类器算法分开的。

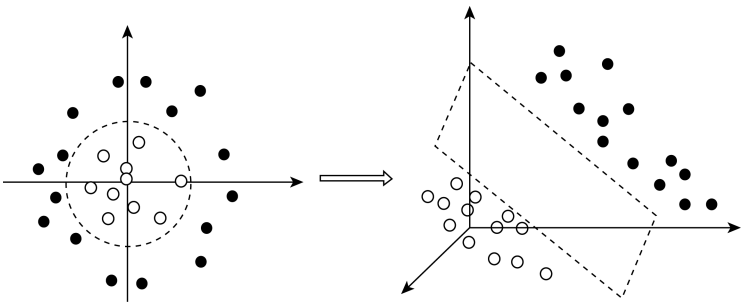


图 7.10　非线性可分样本集示例

但如图 7.10(b)所示，原始样本数据在二维空间里无法线性分割，经过某种非线性映射到高维空间则可进行二类划分。并且，如果原始数据是有限维的，则一定存在一个高维属性空间使样本可进行线性划分。该模型可写为：

$$f(X) = w^T \varphi(X) + b \tag{7.29}$$

其中 $\varphi(X)$ 表示将 X 映射到属性空间。此时，与线性模型(7.23)相对应的模型为：

$$\begin{cases} \min\limits_{w,\ b} \dfrac{1}{2} \| w \|^2 \\ \text{s. t. } y_i (w^T \varphi(X_i) + b) \geqslant 1,\ i = 1,\ 2,\ \cdots,\ n. \end{cases} \tag{7.30}$$

及其对偶问题为：

$$\begin{cases} \max\limits_{\alpha} \sum\limits_{i=1}^{n} \alpha_i - \dfrac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i \alpha_j y_i y_j \varphi(X_i)^T \varphi(X_j), \\ \text{s.t.} \sum\limits_{i=1}^{n} \alpha_i y_i = 0, \\ \alpha_i \geqslant 0, \ (i = 1, 2, \cdots, n) \end{cases} \tag{7.31}$$

为了避免在求解式(7.31)时计算 $\varphi(X_i)^T \varphi(X_j)$，我们利用一个所谓的**核函数**将特征空间中的计算转化到原始样本空间中进行：

$$k(X_i, X_j) = <\varphi(X_i), \varphi(X_j)> = \varphi(X_i)^T \varphi(X_j) \tag{7.32}$$

显然，若这样的核函数 k 存在，则(7.31)就转化为

$$\begin{cases} \max\limits_{\alpha} \sum\limits_{i=1}^{n} \alpha_i - \dfrac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i \alpha_j y_i y_j \varphi k(X_i, X_j), \\ \text{s.t.} \sum\limits_{i=1}^{n} \alpha_i y_i = 0, \\ \alpha_i \geqslant 0, \ (i = 1, 2, \cdots, n). \end{cases} \tag{7.33}$$

从而特征空间超平面模型为：

$$f(X) = w^T \varphi(X^T) + b = \sum_{i=1}^{n} \alpha_i y_i \varphi(X_i^T) \varphi(X) + b = \sum_{i=1}^{n} \alpha_i y_i k(X_i^T, X) + b \tag{7.34}$$

利用核函数避免了高维特征空间的计算，这样的核函数已被广泛研究，常用的有以下几种：

多项式核：$k(X_i, X_j) = (X_i^T X_j)^d$，$d$ 为多项式的幂；

高斯核：$k(X_i, X_j) = \exp\left(-\dfrac{\| X_i - X_j \|^2}{2\sigma^2}\right)$，$\sigma > 0$ 为高斯核的带宽；

Sigmoid 核：$k(X_i, X_j) = \tanh(\beta X_i^T X_j + \theta)$，$\tanh$ 为双曲正切函数，$\beta > 0$，$\theta < 0$。

我们也可以通过核函数的组合得到新的核函数。

理论上，对于任意数据样本都存在一个合适的映射，能够将在低维空间不能划分的样本映射到高维空间中之后能够线性可分。即只要找到合适的核函数，就可将线性不可分问题转化为线性可分问题，然后再采用线性可分 SVM 进行分类。为给定数据确定最佳核函数，这里不再介绍。

7.6 本章小结

本章介绍了数据分析的分类方法，首先给出了相关概念和基本框架，然后具体介绍了 kNN、决策树、贝叶斯以及 SVM 四种常用分类方法。kNN 算法是基于距离度量的分类方法，它的核心思想与“物以群分”异曲同工，寻找与待分类样本最接近的已知样本，通过投票原理预测待分类样的类别。其常用的距离有欧式距离和绝对值距离；决策树方法是比较

易于理解和解释的一种分类方法，它采用自上而下的递归方式建立模型，建模的关键是属性选择度量和剪枝策略。其代表算法，ID3 算法以信息增益作为属性选择度量，C4.5 算法以信息增益率为属性选择度量，悲观剪枝为剪枝策略；朴素贝叶斯分类方法是基于后验概率的贝叶斯定理的，它是在假定样本类是条件独立情况下进行的；支持向量机是一种用于线性和非线性数据的分类算法，它将训练样本数据映射到较高维的特征空间，使用支持向量的样本，寻找能够分离数据的超平面。

7.7 习题

1. 给定最近邻数 k 和描述每个训练样本的属性数 n，写出一个 kNN 分类算法。

2. 简述决策树分类的主要步骤。

3. 给定训练集 D，属性数 n 和训练元组数 $|D|$，分析决策树算法的计算法复杂度。

4. 简述朴素贝叶斯分类的基本思想，并以例 7.2 的数据为训练集，编程实现拉普拉斯修正的朴素贝叶斯分类，对样本 $X=($家庭收入$=$低，年级$=$高，性别$=0)$进行判别。

5. 选择一个数据集，利用高斯核训练一个 SVM，并与 C4.5 算法进行比较。