第5章 分类与预测

分类和预测是两种数据分析形式，可以用于描述重要数据类的模型或预测未来的数据趋势。分类主要是预测分类标号(离散属性)，构造一个分类模型，输入样本的属性值，输出对应的类别，将每个样本映射到预先定义好的类别中。预测主要是建立连续值函数模型，预测给定自变量对应的因变量的值，评估无标号样本类，或评估给定样本可能具有的属性值或值区间。在这种观点下，分类和回归是两类主要预测问题。其中，分类是预测离散或标称值，而回归用于预测连续或有序值。然而，在数据挖掘界，一种被广泛接受的观点是“预测类标号称为分类，预测连续值称为预测”。

分类和预测具有广泛的应用，包括信誉证实、医疗诊断、性能预测和选择购物等。

训练集：数据库中为建立模型而被分析的数据元组形成训练集。训练集中的单个元组称为训练样本，每个训练样本有一个类别标记。一个具体样本的形式可为(v_1，v_2，…，v_n；c)；其中，v_i 表示属性值，c 表示类别。

测试集：用于评估分类模型的准确率。

5.1 分类的基本过程

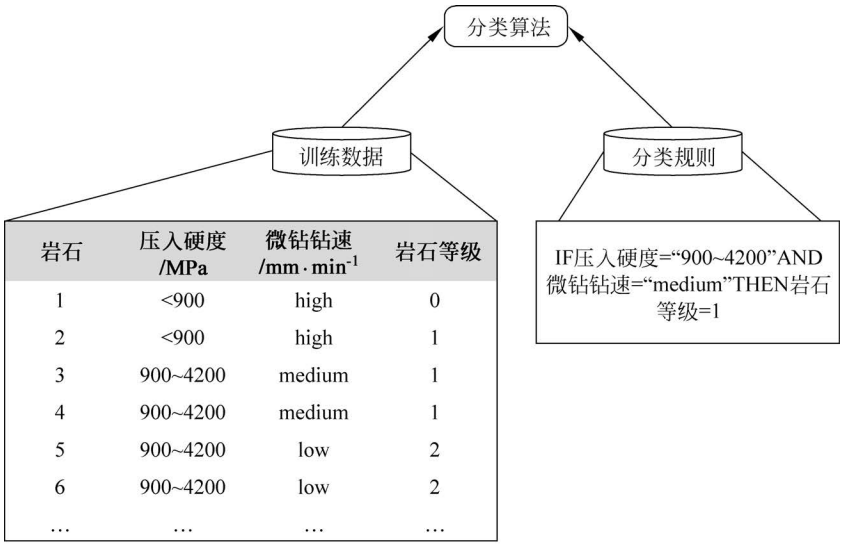
数据分类是一个两步过程。

第一步，建立一个模型，描述预定的数据类或概念集。通过分析由属性描述的数据库元组来构造模型。假定每个元组属于一个预定义的类，由一个称作类标号属性的属性确定。对于分类，数据元组也称作样本、实例或对象。为建立模型而被分析的数据元组形成训练数据集。训练数据集中的单个元组称作训练样本，并随机地由样本群选取。由于提供了每个训练样本的类标号，该步也称作有指导的学习(即模型的学习在被告知每个训练样本属于哪个类的“指导”下进行)。它不同于无指导的学习/聚类，每个训练样本的类标号是未知的，要学习的类集合或数量也可能事先不知道。通常，学习模型以分类规则、判定树或数学公式的形式提供。例如，石油公司为岩石数据建立了一个数据库，可以学习分类规则，根据压入硬度、微钻钻速以及岩石等级来判断岩石的等级(见图 5.1(a))。该规则可以用来为以后的数据样本分类，也能为数据库的内容提供更好的理解。

第二步(见图 5.1(b))，使用模型进行分类。首先评估模型(分类法)的预测准确率。保持(Holdout)是一种使用类标号样本测试集的简单方法。这些样本随机选取，并独立于训练样本。模型在给定测试集上的准确率是被模型正确分类的测试样本的百分比。对于每个测试样本，将已知的类标号与学习模型预测的类标号做比较。注意，如果模型的准确率根据

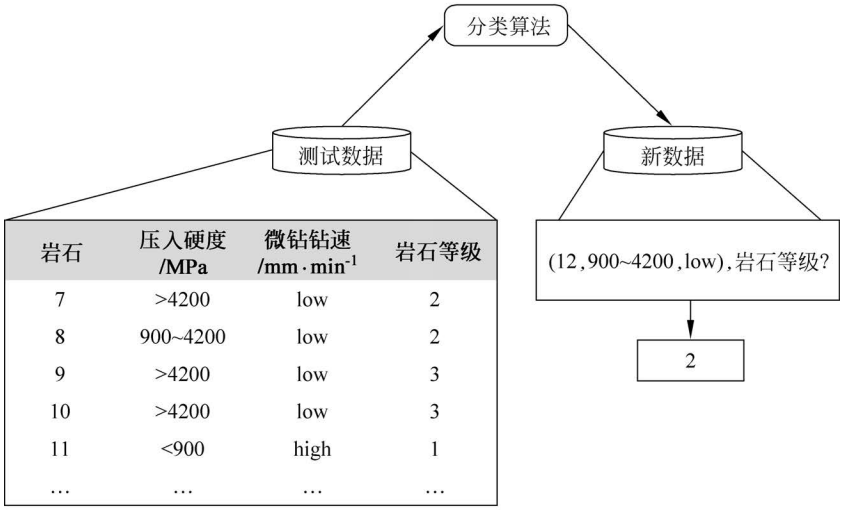
训练数据集评估，评估可能是乐观的，因为学习模型倾向于过度贴合训练数据的特征（即某些只存在于训练数据中的异常特征也会被学习到）。因此，需要使用测试集对模型进行评估。

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元组或对象进行分类（这种数据在机器学习中也称为“未知的”或“先前未见到的”数据）。例如，在图 5.1(a)通过分析岩石的等级学习到的分类规则可以用来预测新的岩石等级。



岩石	压入硬度 /MPa	微钻钻速 /mm·min^{-1}	岩石等级
1	<900	high	0
2	<900	high	1
3	900~4200	medium	1
4	900~4200	medium	1
5	900~4200	low	2
6	900~4200	low	2
…	…	…	…

(a)



岩石	压入硬度 /MPa	微钻钻速 /mm·min^{-1}	岩石等级
7	>4200	low	2
8	900~4200	low	2
9	>4200	low	3
10	>4200	low	3
11	<900	high	1
…	…	…	…

(b)

图 5.1　数据分类过程

(a) 学习：用分类算法分析训练数据。这里，类标号属性是岩石等级，学习模型或分类法以分类规则形式提供。(b) 分类：测试数据用于评估分类规则的准确率。如果准确率是可以接受的，则规则用于新的数据元组分类

5.2 分类模型的构造方法

5.2.1 数据准备

对数据使用下面的预处理，以便提高分类和预测过程的准确性、有效性和可规模性。

(1) 数据清理：旨在消除或减少数据噪声(如使用平滑技术)和处理遗漏值(如使用该属性最常出现的值，或根据统计，用最可能的值替换遗漏值)的数据预处理。尽管大部分分类算法都有处理噪声和遗漏值的机制，但该步骤有助于减少数据学习时的混乱。

(2) 相关性分析：数据中许多属性可能与分类和预测任务不相关。例如，记录银行贷款的签署时间可能与是否成功申请到贷款不相关。此外，其他属性可能是冗余的。因此，可以进行相关分析，删除学习过程中不相关或冗余的属性。在机器学习中，这一过程称为特征选择。包含这些属性将减慢和误导学习步骤。理想地，用在相关分析上的时间，加上从“压缩的”结果子集上学习的时间，应当少于由原来的数据集合上学习所花的时间。因此，这种分析可以帮助提高分类的有效性和可规模性。

(3) 数据变换：数据可以泛化到较高层概念。对于连续值属性，这一步非常有用。例如，属性“微钻钻速”的数值可以泛化为离散的区间，如 low、medium 和 high。类似地，标称值，如 street，可以泛化到高层概念，如 city。由于泛化压缩了原来的训练数据，学习时的输入/输出操作将减少，数据也可以规范化。

(4) 规范化：将属性的所有值按比例缩放，使得它们落入较小的指定区间，如－1.0～1.0 或 0.0～1.0。例如，在使用距离度量的方法中，可以防止具有较大初始域的属性(如 income)相对于具有较小初始域的属性(如二进位属性)赋予的权重过大。

5.2.2 分类方法

(1) 机器学习方法。

① 决策树法(如 ID3，C4.5，SLIQ)。

② 规则归纳(如 Aprior，CBA(Classification Based on Association)算法，LB(Large Bayes)算法是综合概率统计和关联规则的知识而提出的分类算法)。

③ 基于支持向量机(Support Vector Machine，SVM)分类。

(2) 统计方法：知识表示是判别函数和原型事例。

① 贝叶斯法。

② 非参数法(近邻学习或基于事例的学习)。

(3) 神经网络方法。

BP 算法(error BackPropagation，误差逆传播)，模型表示是前馈神经网络模型。

(4) 粗糙集：知识表示是产生式规则。

5.2.3 方法评估标准

(1) 预测准确率。

预测准确率描述模型正确预测未知对象类别的能力。例如,通过 10 折交叉验证方法评估。

(2) 计算复杂度。

计算复杂度包括时间复杂度和空间复杂度。涉及生成和使用模型的计算成本。

(3) 模型描述的简洁度(可理解性)。

模型描述的简洁度涉及学习模型提供的理解和洞察的层次。

(4) 鲁棒性。

鲁棒性描述模型正确预测给定噪声数据或具有空缺值的数据的能力。

(5) 可伸缩性。

可伸缩性涉及给定大量数据,有效地构造模型的能力。

5.3 基于决策树(判定树)的分类

从机器学习中引出的决策树(Decision Tree)算法是一种较为通用并被深入研究的函数逼近方法,目前已形成了多种决策树算法,如 CLS、ID3、CHAID、CART、C4.5 等。

决策树的起源是于 1962 年提出的 CLS(Concept Learning System),CLS 是由 Hunt、Marin 和 Stone 为了研究人类概念模型而得来的,该模型为很多决策树算法的发展奠定了很好的基础; 1984 年,L. Breiman 等人提出了 CART(Classification and Regression Tree)算法,采用基尼指数作为属性分裂的标准; 1979—1986 年,J. R. Quinlan 提出了 ID3 算法,但 ID3 算法无法处理连续型数据,且未考虑缺失值的处理; 1993 年,J. R. Quinlan 又提出了 C4.5 算法,克服了 ID3 算法的一些不足; 1996 年,M. Mehta 和 R. Agrawal 等人提出了一种高速可伸缩的有监督的寻找学习分类算法 SLIQ(Supervised Learning In Quest),破除了部分主存的限制; 1996 年,J. Shafer 和 R. Agrawal 等人提出了可伸缩并行归纳决策树分类方法 SPRINT(Scalable PaRallelizable Induction of Decision Trees),真正意义上破除主存限制; 1998 年,R. Rastogi 等人提出一种将建树和修剪相结合的分类算法 PUBLIC(A Decision Tree that Integrates Building and Pruning),是典型的预剪枝算法,也采用基尼指数作为属性分裂标准,可以说是 CART 的一种改进。此后还陆续涌现了许多改进决策树,如 C5.0、C&RT(Classification and Regression Tree)、QUEST(Quick Unbiased Efficient Statistical Tree)等。

决策树是一个类似流程图的树结构,其中树的每个内部结点代表对一个属性的测试,即形式为($a_i = v_i$)的逻辑判断,其中,a_i 是属性,v_i 是该属性的某个属性值,其分支就代表测试的每个结果,也就是每一种可能的值和一条边一一对应,叶子结点指定一个类别,代表类或类分布,树的最顶层结点是根结点。决策树分类方法采用自顶向下的递归方式。接 5.1 节中的例子,一棵典型的决策树如图 5.2 所示,它表示概念“岩石可钻性”,预测油气开采中岩石的钻探难易程度,在这里暂且将“微钻钻速”大于 135mm · min^{-1} 的称为高速,记为

high；21～134mm·min^{-1} 的称为中速，记为 medium；小于 20mm·min^{-1} 的称为低速，记为 low；“岩石可钻性”大于 0.6m·h^{-1} 的称为高可钻性(高)，小于 0.6m·h^{-1} 的称为低可钻性(低)。内部结点用矩形表示，而树叶用椭圆表示。

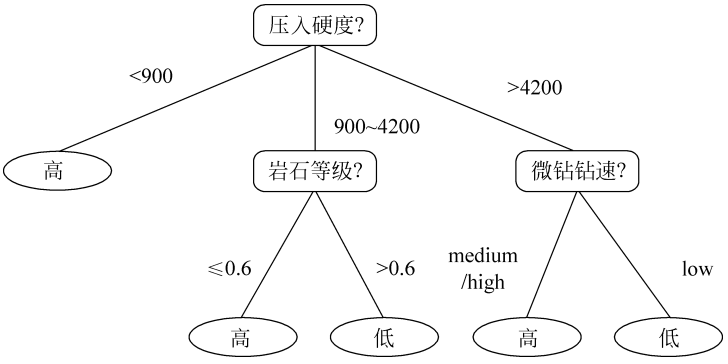


图 5.2 “岩石可钻性”的判定树

5.3.1 决策树分类步骤

使用决策树进行分类一般有两个主要步骤：①建立决策树，利用训练样本生成决策树模型；②修剪决策树。

建立决策树采用自上而下分而治之的方法。开始时，所有的数据都在根结点，所有记录用所选属性递归地进行分割，属性的选择基于一个启发式规则或者一个统计的度量。当满足以下两个条件之一时停止数据分割：①一个结点上的数据都是属于同一个类别；②不存在更多属性进行数据分割。

修剪决策树主要通过消除统计噪声或数据波动的影响来净化树。决策树的修剪就是使用一个叶子结点来替代一棵子树。若检测到某规则下的子树中的误分类率较高，则进行替换。

通常，建树阶段的耗时要比修剪阶段的大，因为建树阶段要对数据进行多次扫描，而修剪阶段只需访问生成的决策树。

5.3.2 决策树 ID3 算法

本节以 ID3 算法为例对决策树分类进行解释。ID3 与 C4.5 是 Quinlan 为了从数据中归纳分类模型而构造的算法。有以下特点：给定一个记录的集合，每个记录都有相同的结构，类别属性所取的值只限于{true,false}，{success,failure}，{yes,no}等离散值；决策树中每一个非叶子结点对应着一个非类别属性，树枝代表这个属性的值；一个叶子结点代表从树根到叶子结点之间的路径对应的记录所属的类别属性值；每个非叶子结点与具有最大信息量的非类别属性相关联。采用信息增益作为标准来选择能最有效分类样本的属性。

基本过程如下。

算法：**Generate_decision_tree**。由给定的训练数据产生一棵判定树。
输入：训练样本 samples，由离散属性值表示；候选属性的集合 attribute_list。
输出：一棵判定树。

步骤：

(1) 创建结点 N；
(2) **if** samples 都在同一个类 C **then**
(3) return N 作为叶子结点，以类 C 标记；
(4) **if** attribute _list 为空 **then**
(5) return N 作为叶子结点，标记为 samples 中最普通的类；
(6) 选择 attribute_list 中具有最高信息增益的属性 test_attribute；
(7) 标记结点 N 为 test_attribute；
(8) **for each** test_attribute 中的未知值 a_i
(9) 由结点 N 长出一个条件为 test_attribute = a_i 的分支；
(10) 设 s_i 是 samples 中 test_attribute = a_i 的样本的集合；
(11) **if** s_i 为空 **then**
(12) 加上一个树叶，标记为 samples 中最普通的类；
(13) **else** 加上一个由 Generate_decision_tree(s_i, attribute_list-test_attribute)返回的结点。

基本策略如下。

- 树以代表训练样本的单个结点开始(步骤 1)。
- 如果样本都在同一个类，则该结点成为树叶，并用该类标号(步骤 2 和 3)。
- 否则，算法使用称为信息增益的基于熵的度量作为启发信息，选择能够最好地将样本分类的属性(步骤 6)。该属性成为该结点的“测试”或“判定”属性(步骤 7)。在算法的该版本中，所有的属性都是分类的，即离散值。连续属性必须离散化。
- 对测试属性的每个已知的值，创建一个分支，并据此划分样本(步骤 8～10)。
- 算法使用同样的过程，递归地形成每个划分上的样本判定树。一旦一个属性出现在一个结点上，就不必在该结点的任何后代上考虑它(步骤 13)。
- 递归划分步骤仅在以下任一条件成立时停止。

a. 给定结点的所有样本属于同一类(步骤 2 和 3)。

b. 没有剩余属性可以用来进一步划分样本(步骤 4)。在此情况下，使用多数表决(步骤 5)。这涉及将给定的结点转换成树叶，并用样本中的多数所在的类标记它。

c. 分支 test_attribute=a_i 没有样本(步骤 11)。在这种情况下，以 samples 中的多数类创建一个树叶(步骤 12)。

5.3.3 属性选择方法

在决策树生成方法中，通常使用信息增益来帮助确定生成每个结点时所应采用的属性。选择具有最高信息增益的属性作为当前结点的测试属性。这种度量称作属性选择度量或分裂的优劣度量。该属性使得对结果划分中的样本分类所需的信息量最小，并反映划分的最小随机性或“不纯性”。这种信息理论方法使得对一个对象分类所需的期望测试数目最小，并确保找到一棵简单的(但不必是最简单的)树。

设 S 是 s 个数据样本的集合。假定类标号属性具有 m 个不同值，定义 m 个不同类 C_i ($i = 1,2,\cdots,m$)。设 s_i 是类 C_i 中的样本数。对一个给定的样本分类所需的期望信息由式(5.1)给出。

$$I(S_1,S_2,\cdots,S_m)=-\sum_{i=1}^{m}p_i\log_2(p_i) \tag{5.1}$$

其中，p_i 是任意样本属于 C_i 的概率，并用 $\frac{S_i}{S}$ 估计。注意，对数函数以 2 为底，因为信息用二进制编码。

设属性 A 具有 v 个不同值 $\{a_1,a_2,\cdots,a_v\}$。可以用属性 A 将 S 划分为 v 个子集 $\{S_1,S_2,\cdots,S_v\}$；其中，S_j 包含 S 中这样一些样本，它们在 A 上具有值 a_j。如果 A 选作测试属性（即最好的划分属性），则这些子集对应由包含集合 S 的结点生长出来的分支。设 s_{ij} 是子集 S_j 中类 C_i 的样本数。根据 A 划分子集的熵或期望信息由式(5.2)给出。

$$E(A)=\sum_{j=1}^{v}\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}I(s_{1j},s_{2j},\cdots,s_{mj}) \tag{5.2}$$

项 $\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}$ 充当第 j 个子集的权，并且等于子集（即 A 值为 a_j）中的样本个数除以 S 中的样本总数。熵值越小，子集划分的纯度越高。注意，对于给定的子集 S_j，有

$$I(s_{1j},s_{2j},\cdots,s_{mj})=-\sum_{i=1}^{m}P_{ij}\log_2(P_{ij}) \tag{5.3}$$

其中，$P_{ij}=\frac{S_{ij}}{|S_j|}$，是 S_j 中的样本属于 C_i 的概率。

在 A 上分支将获得的编码信息是

$$\text{Gain}(A)=I(S_1,S_2,\cdots,S_m)-E(A) \tag{5.4}$$

换言之，Gain(A)是由于知道属性 A 的值而导致的熵的期望压缩。

算法计算每个属性的信息增益。具有最高信息增益的属性选作给定集合 S 的测试属性。创建一个结点，并以该属性标记，对属性的每个值创建分支，并据此划分样本。

例 5.1 表 5.1 给出了取自百度百科的岩石可钻性数据元组。类标号属性“岩石可钻性”有两个不同值，即{高，低}，因此有两个不同的类（$m=2$）。设类 C_1 对应“高”，而类 C_2 对应“低”。“高”类有 9 个样本，“低”类有 5 个样本。为计算每个属性的信息增益，首先使用式(5.1)，计算对给定样本分类所需的期望信息：

$$I(s_1,s_2)=I(9,5)=-\frac{9}{14}\log_2\frac{9}{14}-\frac{5}{14}\log_2\frac{5}{14}=0.940$$

表 5.1 岩石可钻性数据元组

RID	压入硬度	微钻钻速	岩　石	岩 石 等 级	Class：岩石可钻性
1	＜900	high	no	0	yes
2	＜900	high	no	0	yes
3	900～4200	high	no	1	no
4	＞4200	medium	yes	2	yes
5	＞4200	low	yes	2	yes
6	＞4200	low	yes	3	no
7	900～4200	low	yes	1	no
8	＜900	high	no	0	yes

续表

RID	压入硬度	微钻钻速	岩 石	岩 石 等 级	Class：岩石可钻性
9	<900	high	yes	1	yes
10	>4200	medium	yes	2	yes
11	<900	high	yes	0	yes
12	900～4200	medium	no	2	no
13	900～4200	high	yes	2	yes
14	>4200	medium	yes	3	no

在实现时先确定程序所需环境和工程包(以 Python 实现为例)：

```
01 #-*- coding: UTF-8 -*-
02 from math import log
03 import operator
```

人工对数据集进行属性定义。

- 压入硬度：0 代表<900,1 代表 900～4200,2 代表>4200。
- 微钻钻速：0 代表 low,1 代表 medium,2 代表 high。
- 岩石：0 代表否(包括土、冰、煤等),1 代表是。
- 岩石等级：0 代表软岩石,1 代表中硬岩石,2 代表硬岩石,3 代表坚硬岩石。
- 类别(岩石可钻性)：yes 代表高,no 代表低。

```
01 """
02 函数说明:创建数据集
03 """
04 def createDataSet():
05     dataSet = [[0, 2, 0, 0, 'yes'], [0, 2, 0, 0, 'yes'],[1, 2, 0, 1, 'no'], [2, 1, 1, 2, 'yes'],
[2, 0, 1, 2, 'yes'], [2, 0, 1, 3, 'no'], [1, 0, 1, 1, 'no'], [0, 2, 0, 0, 'yes'],[0, 2, 1, 1, 'yes'],
[2, 1, 1, 2, 'yes'], [0, 2, 1, 0, 'yes'], [1, 1, 0, 2, 'no'], [1, 2, 1, 2, 'yes'], [2, 1, 1, 3, 'no']]
06     labels = ['压入硬度', '微钻钻速', '是岩石', '岩石等级']    #分类属性
07     return dataSet, labels                                  #返回数据集和分类属性
```

下一步需要计算每个属性的熵。从属性“压入硬度”开始。需要观察“压入硬度”的每个样本值的“高”和“低”分布。对每个分布计算期望信息,如表 5.2 所示。

表 5.2 根据压入硬度计算每个分布的期望信息

压 入 硬 度	可钻性高(yes)数量	可钻性低(no)数量	期望信息得分
<900	$s_{11}=5$	$s_{21}=0$	$I(s_{11},s_{21})=0$
900～4200	$s_{12}=1$	$s_{22}=3$	$I(s_{12},s_{22})=0.811$
>4200	$s_{13}=3$	$s_{23}=2$	$I(s_{13},s_{23})=0.971$

使用式(5.2),如果样本按“压入硬度”划分,对一个给定的样本分类所需的期望信息为

$$E(\text{压入硬度})=\frac{5}{14}I(s_{11},s_{21})+\frac{4}{14}I(s_{12},s_{22})+\frac{5}{14}I(s_{13},s_{23})=0.579$$

```
01 """
02 函数说明:计算给定数据集的经验熵(香农熵)
03 Parameters:
04     dataSet -数据集
```

```
    Returns:
        shannonEnt -经验熵(香农熵)
    """
    def calcShannonEnt(dataSet):
        numEntries= len(dataSet)                          #返回数据集的行数
        labelCounts = {}                                  #保存每个标签(Label)出现次数的字典
        for featVec in dataSet:                           #对每组特征向量进行统计
            currentLabel = featVec[-1]                    #提取标签(Label)信息
            if currentLabel not in labelCounts.keys():    #如果标签(Label)没有放入统计次数的
                                                          #字典,添加进去
                labelCounts[currentLabel] = 0
            labelCounts[currentLabel] += 1                #Label 计数
        shannonEnt = 0.0                                  #经验熵(香农熵)
        for key in labelCounts:                           #计算香农熵
            prob = float(labelCounts[key]) / numEntires   #选择该标签(Label)的概率
            shannonEnt -= prob * log(prob, 2)
        return shannonEnt                                 #返回经验熵(香农熵)
```

因此,这种划分的信息增益为

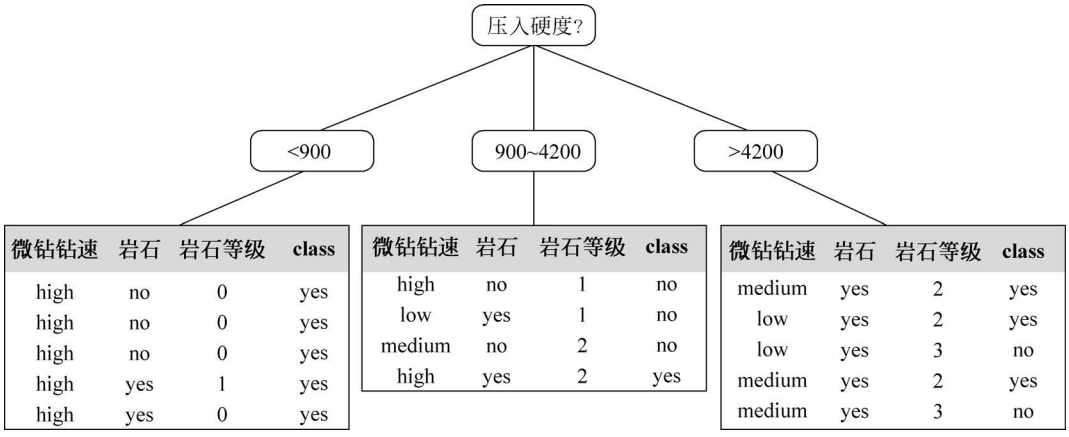
$$\text{Gain}(\text{压入硬度}) = I(S_1, S_2) - E(\text{压入硬度}) = 0.361$$

类似地,可以计算 Gain(微钻钻速)、Gain(岩石)和 Gain(岩石等级)。

```
    """
    函数说明:按照给定特征划分数据集
    Parameters:
        dataSet - 待划分的数据集
        axis - 划分数据集的特征
        value - 需要返回的特征的值
    """
    def splitDataSet(dataSet, axis, value):
        retDataSet = []                                   #创建返回的数据集列表
        for featVec in dataSet:                           #遍历数据集
            if featVec[axis] == value:
                reducedFeatVec = featVec[:axis]           #去掉 axis 特征
                reducedFeatVec.extend(featVec[axis+1:])   #将符合条件的添加到返回的数据集
                retDataSet.append(reducedFeatVec)
        return retDataSet                                 #返回划分后的数据集
    """
    函数说明:选择最优特征
    Parameters:
        dataSet -数据集
    Returns:
        bestFeature -信息增益最大的(最优)特征的索引值
    """
    def chooseBestFeatureToSplit(dataSet):
        numFeatures = len(dataSet[0]) - 1                 #特征数量
        baseEntropy = calcShannonEnt(dataSet)             #计算数据集的香农熵
        bestInfoGain = 0.0                                #信息增益
        bestFeature = -1                                  #最优特征的索引值
        for i in range(numFeatures):                      #遍历所有特征
            featList = [example[i] for example in dataSet]#获取 dataSet 的第 i 个特征
            uniqueVals = set(featList)                    #创建 set 集合{},元素不可重复
            newEntropy = 0.0                              #经验条件熵
```

```
        for value in uniqueVals:                                    #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)            #subDataSet 划分后的子集
            prob = len(subDataSet) / float(len(dataSet))            #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)         #根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy                         #信息增益
        print("第%d 个特征的增益为%.3f" % (i, infoGain))             #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                               #计算信息增益
            bestInfoGain = infoGain                                 #更新信息增益,找到最大
                                                                    #的信息增益
            bestFeature = i                                         #记录信息增益最大的特征
                                                                    #的索引值
    return bestFeature                                              #返回信息增益最大的特征
                                                                    #的索引值
```

由于“压入硬度”在属性中具有最高信息增益,它被选作测试属性。创建一个结点,用“压入硬度”标记,并对于每个属性值引出一个分支。样本据此划分,如图 5.3 所示。注意,落在分区压入硬度“<900”的样本都属于同一类。由于它们都属于同一类“高”,因此要在该分支的端点创建一个树叶,并用“高”标记。算法返回的最终判定树如图 5.3 所示。



微钻钻速	岩石	岩石等级	class
high	no	0	yes
high	no	0	yes
high	no	0	yes
high	yes	1	yes
high	yes	0	yes

微钻钻速	岩石	岩石等级	class
high	no	1	no
low	yes	1	no
medium	no	2	no
high	yes	2	yes

微钻钻速	岩石	岩石等级	class
medium	yes	2	yes
low	yes	2	yes
low	yes	3	no
medium	yes	2	yes
medium	yes	3	no

图 5.3 判定树

```
"""
函数说明:统计 classList 中出现次数最多的元素(类标签)
Parameters:
    classList -类标签列表
Returns:
    sortedClassCount[0][0] -出现次数最多的元素(类标签)
"""
def majorityCnt(classList):
    classCount = {}
    for vote in classList:                                  #统计 classList 中每个元素出现的次数
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)
                                                            #根据字典的值降序排序
    return sortedClassCount[0][0]                           #返回 classList 中出现次数最多的元素
"""
```

```
函数说明:递归构建决策树
Parameters:
    dataSet -训练数据集
    labels -分类属性标签
    featLabels -存储选择的最优特征标签
Returns:
    myTree -决策树
"""
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet]          #取分类标签(可钻性高低:yes
                                                              #or no)
    if classList.count(classList[0]) == len(classList):       #如果类别完全相同则停止划分
        return classList[0]
    if len(dataSet[0]) == 1:                                  #遍历完所有特征时返回出现次数
                                                              #最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)              #选择最优特征
    bestFeatLabel = labels[bestFeat]                          #最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                               #根据最优特征的标签生成树
    del(labels[bestFeat])                                     #删除已经使用的特征标签
    featValues = [example[bestFeat] for example in dataSet]   #得到最优特征的属性值
    uniqueVals = set(featValues)                              #去掉重复的属性值
    for value in uniqueVals:
        subLabels=labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),
subLabels, featLabels)   #递归调用函数 createTree(),遍历特征,创建决策树
    return myTree
```

构建决策树的目的是实现分类功能。

```
"""
函数说明:使用决策树执行分类
Parameters:
    inputTree -已经生成的决策树
    featLabels -存储选择的最优特征标签
    testVec -测试数据列表,顺序对应最优特征标签
Returns:
    classLabel -分类结果
"""
def classify(inputTree, featLabels, testVec):
    firstStr = next(iter(inputTree))                          #获取决策树结点
    secondDict = inputTree[firstStr]                          #下一个字典
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key], featLabels, testVec)
            else:
                classLabel = secondDict[key]
    return classLabel
```

当决策树功能模块搭建好后,即可使用主函数调用完成分类。

```
if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
    testVec = [0, 1]                                    #测试数据
    result = classify(myTree, featLabels, testVec)
    if result == 'yes':
        print('岩石可钻性高')
    if result == 'no':
        print('岩石可钻性低')
```

例 5.2 表 5.3 给出了不同天气情况下是否适合打高尔夫球的数据。类标号属性“适合打高尔夫”有两个不同值，即{yes，no}。类 yes 有 9 个样本，类 no 有 5 个样本。

表 5.3 高尔夫运动环境训练数据元组

天 气	温 度	湿 度	风 况	适合打高尔夫
晴	85	85	无	no
晴	80	90	有	no
多云	83	78	无	yes
雨	70	96	无	yes
雨	68	80	无	yes
雨	65	70	有	no
多云	64	65	有	yes
晴	72	95	无	no
晴	69	70	无	yes
雨	75	80	无	yes
晴	75	70	有	yes
多云	72	90	有	yes
多云	81	75	无	yes
雨	71	80	有	no

从属性“天气”开始，观察每个样本值的 yes 和 no 分布，对每个分布计算期望信息，如表 5.4 所示。

表 5.4 根据天气计算每个分布的期望

天 气	适合打高尔夫天数	不适合打高尔夫天数	期望信息得分
晴	$s_{11}=2$	$s_{21}=3$	$I(s_{11}, s_{21})=0.971$
多云	$s_{12}=4$	$s_{22}=0$	$I(s_{12}, s_{22})=0$
雨	$s_{13}=3$	$s_{23}=2$	$I(s_{13}, s_{23})=0.971$

接下来请读者自行计算。

5.3.4 基本决策树方法的改进

针对 ID3 算法的特点，可以从以下 4 方面进行改进：①连续属性的处理；②测试属性的改进；③属性的选择；④遗失数据的处理。

1. 连续属性的处理

ID3 要求所有的属性都必须是符号量或是无序的离散值。因此该算法首先需要改进以便容许可取连续值的属性。

可以从连续型数据入手，使用属性离散化技术使 ID3 能够利用连续型数据。最简单的策略是采用二分法对连续属性进行处理，这正是 C4.5 决策树算法中采用的机制。假设 c_i 有连续的属性值，先将其值排序为 $A_1, A_2, \cdots, A_m$，对每个值 A_j（$j=1,2,\cdots,m$），将所有记录划分成两部分（$\leqslant A_j$ 和 $>A_j$），针对每个划分分别计算信息增益，选择取最大增益值的值进行划分。

2. 测试属性的改进

基本的决策树归纳方法对一个测试属性的每个取值均产生一个相应分支，且划分相应的数据样本集。这样的划分会导致产生许多小的子集。随着子集被划分得越来越小，划分过程将会由于子集规模过小所造成的统计特征不充分而停止（如按 key 分类）。

可以将一个（取离散值）属性的若干值组合在一起，这样在测试该属性时，将是对属性的一组取值进行测试。

3. 属性的选择

信息增益选择方法有一个很大的缺陷，它总是会倾向于选择属性值多的属性，如果在表 5.3 的数据记录中加一个“姓名”属性，假设 14 条记录中的每个人姓名不同，那么信息增益就会选择“姓名”作为最佳属性，因为按“姓名”分裂后，每个组只包含一条记录，而每条记录只属于一类（要么购买计算机要么不购买），因此纯度最高，以“姓名”作为测试分裂的结点下面有 14 个分支。但是这样的分类没有意义，它没有任何泛化能力。

针对这一问题，人们也提出了许多方法，如 C4.5 决策树采用增益-比率（GainRatio）方法，将每个属性取值的概率考虑在内，衡量属性分裂数据的广度和均匀性。增益-比率首先将数据进行分割：

$$\mathrm{SplitInfo}(S,A)=-\sum_{i=1}^{v}\frac{|S_i|}{|S|}\log_2\frac{|S_i|}{|S|} \tag{5.5}$$

其中，$S_1 \sim S_v$ 是 v 个值的属性 A 分割 S 而形成的 v 个样本子集。SplitInfo 的值代表由训练集 S 划分成对于属性 A 测试的 v 个输出的 v 个分区产生的信息。v 越大，则 SplitInfo（S，A）的取值通常会越大。在高尔夫的例子里 SplitInfo（S，天气）为

$$-\frac{5}{14}\log_2\frac{5}{14}-\frac{4}{14}\log_2\frac{4}{14}-\frac{5}{14}\log_2\frac{5}{14}=1.58$$

通过分割信息重新计算信息增益：

$$\mathrm{GainRatio}(S,A)=\frac{\mathrm{Gain}(S,A)}{\mathrm{SplitInfo}(S,A)} \tag{5.6}$$

增益率准则对可取值数目较少的属性有所偏好，因此增益-比率使用了一个启发式思想：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

```
"""
函数说明:定义统计等值元素
"""
equalNums = lambda x,y: 0 if x is None else x[x==y].size
"""
函数说明:定义计算输入序列信息熵
Parameters:
    equalNums -统计等值元素
Returns:
    entropy -信息熵
"""
def singleEntropy(x):                                    #定义计算信息熵的函数
    x = np.asarray(x)                                    #转换为 NumPy 矩阵
    xValues = set(x)                                     #取所有不同值
    entropy = 0                                          #计算熵值
    for xValue in xValues:
        p = equalNums(x, xValue) / x.size
        entropy -= p * math.log(p, 2)
    return entropy
"""
函数说明:定义计算某特征 feature 条件下 y 的条件信息熵
Parameters:
    feature -数据集中的特征
Returns:
    entropy -条件信息熵
"""
def conditionnalEntropy(feature, y):
    feature = np.asarray(feature)                        #转换为 NumPy
    y = np.asarray(y)
    featureValues = set(feature)                         #取特征的不同值
    entropy = 0                                          #计算熵值
    for feat in featureValues:
    #y[feature == feat]是取 y 中 feature 元素值等于 feat 的元素索引的 y 的元素的子集
        p = equalNums(feature, feat) / feature.size
        entropy += p * singleEntropy(y[feature == feat])
    return entropy
"""
函数说明:定义信息增益
Parameters:
    feature -数据集中的特征
Returns:
    Gain -某特征的信息增益
"""
def infoGain(feature, y):
    return singleEntropy(y) -conditionnalEntropy(feature, y)
"""
函数说明:定义信息增益率
Parameters:
    feature -数据集中的特征
Returns:
    GainRatio -某特征的信息增益率
"""
def infoGainRatio(feature, y):
    return 0 if singleEntropy(feature) == 0 else infoGain(feature, y) / singleEntropy(feature)
```

CART 决策树使用 Gini Index(基尼指数)来选择划分属性。如果数据集 T 包含来自 n

个类的样本，则 Gini 指数定义为

$$\mathrm{Gini}(T)=1-\sum_{j=1}^{n}p_j^2 \tag{5.7}$$

其中，p_j 表示类 j 出现的频率。Gini(T)反映了从数据集 T 中随机抽取两个样本，其类别不一致的概率。因此，Gini(T)越小，则数据集 T 的纯度越高。如果一个划分将数据集 T 分成两个子集 S_1 和 S_2，则分割后的 $\mathrm{Gini}_{\mathrm{split}}$ 是

$$\mathrm{Gini}_{\mathrm{split}}(T)=\frac{n_1}{n}\mathrm{Gini}(S_1)+\frac{n_2}{n}\mathrm{Gini}(S_2) \tag{5.8}$$

其中，n_1 和 n_2 表示子集 S_1 和 S_2 的数据量。最小 $\mathrm{Gini}_{\mathrm{split}}$ 就被选择作为数据分割的标准。

以例 5.1 中岩石可钻性数据为例，先用基尼指数估算一下整个样本 D 的不纯度：

$$\mathrm{Gini}(D)=1-\left(\frac{9}{14}\right)^2-\left(\frac{5}{14}\right)^2=0.459$$

为了找出 D 中元组的分裂准则，需要计算每个属性的基尼指数。从属性微钻钻速开始，微钻钻速有 3 个元素，子集有 8 个，考虑其真子集(6 个)，存在 6 种形成数据集 D 的两个分区的可能方法。考虑微钻钻速的二元划分有{low ，medium}和{high}，{low， high}和{medium}，{medium， high}和{low}。

对第一种划分有：属于{low ，medium}的子集 S_1 中有 7 个样本，属于{high}的子集 S_2 中有 7 个样本，则：

$$\mathrm{Gini}(S_1)=1-\left(\frac{3}{7}\right)^2-\left(\frac{4}{7}\right)^2=0.490$$

$$\mathrm{Gini}(S_2)=1-\left(\frac{6}{7}\right)^2-\left(\frac{1}{7}\right)^2=0.245$$

$$\mathrm{Gini}_{\{\mathrm{low,medium}\}和\{\mathrm{high}\}}(\text{微钻钻速})=\frac{7}{14}\mathrm{Gini}(S_1)+\frac{7}{14}\mathrm{Gini}(S_2)=0.367$$

同理可算出第二种划分的基尼指数为 0.443，第三种划分的基尼指数为 0.407。因此，微钻钻速属性的最好二元划分为第一种，即{low ，medium}和{high}。

然后再考虑压入硬度，压入硬度根据其取值可分为{low，medium，high}。同理可以找出最好的分割点是{medium，high}和{low}，基尼指数为 0.317。同理可求得属性岩石和岩石等级。

若只考虑属性“微钻钻速”和“压入硬度”，最终，属性压入硬度和分裂子集产生最小的基尼指数，不纯度降低为 0.459－0.317＝0.142；二元划分{medium ，high}和{low}导致 D 中元组的不纯度降低最大，返回作为分裂准则，用结点 N 做标记，由它长出两个分支。

读者可以试试考虑所有 4 个属性的情况。

4. 遗失数据的处理

可以利用属性 A 中最常见的值来替代一个遗失或未知属性 A 的值。

5.3.5 树剪枝

当判定树创建时，由于数据中的噪声和局外者，许多分支反映的是训练数据中的异常。剪枝方法可用于处理这种过分适应数据问题。同时在模型学习中，为了尽可能正确地分类训练样本，结点划分过程将不断重复，有时会造成决策分支过多，这时就可能因训练样本学

得“太好”了，以至于把训练集自身的一些特点当作所有数据都具有的一般性质而导致过拟合。

树剪枝目标是选择尽可能小的决策树，通常使用统计度量，剪去最不可靠的分支，能够消除决策树过拟合，提高分类速度，提高树独立于测试数据正确分类的可靠性。

有以下两种常用的剪枝方法。

1. 事前修剪

事前修剪(Pre-pruning)又称为先剪枝、预剪枝，通过提前停止分支生成过程，即通过在当前结点上就判断是否需要继续划分该结点所含训练样本集来实现。一旦停止分支，当前结点就成为树叶。该树叶可能包含多个不同类别的训练样本。Public 算法中采用的就是事前修剪方法。

在构造树时，可以利用统计意义下的 x^2、信息增益等度量，用于评估分裂的优良性。先计算在一个结点划分对系统性能的增益，如果这个增益小于某个指定阈值，则划分将停止。

然而，选取一个适当的阈值是困难的。较高的阈值可能导致过分简化的树，而较低的阈值可能导致多余树枝无法修剪。在阈值选取方面有以下思路：决策树的产生过程是一个递归过程，不断递归分解相应的数据集合将会导致所分析的数据对象(数据子集)变得越来越小，因而从统计角度来看没有任何意义。有关的统计方法可以帮助决定最大的无意义数据子集的规模。可以引入意外阈值这一概念，即若一个给定子集的样本数小于这一阈值，就停止分解这一子集，产生一个叶子结点并标记为其中类别个数最多的类别；由于大规模数据库中数据的变化程度和规模都较大，因此假设一个叶子结点所含数据子集中的样本均属同一个类别是不太合理的，这时可以引入分类阈值来帮助解决这一问题。即若一个结点所含数据集中属于某一类别的样本数大于这一阈值，就可以停止分解这一子集。

2. 事后修剪

事后修剪(Post-pruning)又称为后剪枝，由“完全生长”的树剪去分支，通过删除结点的分支，剪掉树结点。在剪枝过程中，将一些子树删除而用叶子结点来代替，这个叶子结点所属的类用这棵子树中大多数实例所属的类代替，并且在相应叶子结点上标记出所属这个类的训练实例所占的比例。Sprint 算法中采用的就是事后修剪方法。

一般情况下，后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。后剪枝过程是在生成完全决策树之后进行的，并且要自底向上地对树中的所有非叶子结点进行逐一考查，因此，其训练时间开销比未剪枝决策树和预剪枝决策树要大得多。后剪枝算法很多，如 CART 算法中采用的代价复杂性剪枝(Cost Complexity Pruning，CCP)、C4.5 中采用的悲观错误剪枝(Pessimistic Error Pruning，PEP)、SLIQ 算法中采用的最小描述长度(Minimal Description Length，MDL)剪枝。后剪枝一般依据两种剪枝标准：期望错误率最小原则，即对树中的内部结点计算其剪枝/不剪枝可能出现的期望错误率，比较后加以取舍，通过选择期望错误率最小的子树剪枝；最小描述长度原则，即依据“最简单地解释最期望的”思想，对决策树进行二进制位编码，编码所需二进制位最少的树即为“最佳剪枝树”。

1) 代价复杂性剪枝

该方法把树的复杂度看作树中叶子结点的个数和树误分类的元组所占百分比，即错误率的函数。

对于树中每个非树叶子结点，算法计算出该结点被修剪后可能出现的期望错误率。同时，根据每个分支的分类错误率，以及每个分支权重（样本分布），计算对该结点剪枝的期望错误率。如果剪去该结点导致较高的期望错误率，则保留该子树；否则剪去该子树。在产生一组经过修剪的候选决策树之后，使用一个独立的测试集（剪枝集）评估每棵树的准确率，就能得到具有最小期望错误率的判定树。

2）悲观剪枝

悲观剪枝（Pessimistic Pruning，PEP）是 Quinlan 在 1987 年提出的，也使用错误率评估，但不需要像 REP（错误率降低修剪）一样，需要用部分样本作为测试数据，而是完全使用训练数据来生成决策树，又用这些训练数据来完成剪枝。决策树生成和剪枝都使用训练集，所以会产生错误分类。

把一棵具有多个叶子结点的子树的分类用一个叶子结点来替代的话，误判率肯定是上升的。于是需要把子树的误判计算加上一个经验性的惩罚因子。对于一个叶子结点，它覆盖了 N 个样本，其中有 E 个错误，那么该叶子结点的错误率为 $\frac{E+0.5}{N}$。这个 0.5 就是惩罚因子，一棵子树，它有 L 个叶子结点，那么该子树的误判率估计为

$$e=\frac{\left(\sum_{i=1}^{L}E_i+0.5L\right)}{N_i} \tag{5.9}$$

可以看到一棵子树虽然具有多个叶子结点，但由于加上了惩罚因子，所以子树的误判率计算未必有显著的优势。剪枝后内部结点变成了叶子结点，其误判个数 J 也需要加上一个惩罚因子，变成 $J+0.5$。那么子树是否可以被剪枝就取决于剪枝后 $J+0.5$ 是否在标准误差内。简而言之，首先计算“剪枝前错误率 e”，然后计算“剪枝前误判次数均值 $E=Ne$”“剪枝前误判次数标准差 $\mathrm{var}=\sqrt{Ne(1-e)}$”和“剪枝后的错误率 $e_{后}$”“剪枝后误判次数均值 $E_{后}=Ne_{后}$”；当 $(E-\mathrm{var})>E_{后}$ 时表示剪枝成功。

3）最小描述长度（MDL）剪枝

根据编码（如对树编码、对树的异常编码）所需要的二进制位位数对树进行剪枝。“最佳”剪枝树是最小化编码二进制位位数的树。

这一原则遵循的理念是：最简单的就是最好的。

5.3.6 由决策树（判定树）提取分类规则

1. 表示形式

决策树所表示的分类知识可用 IF-THEN 形式表示。对从根到树叶的每条路径创建一个规则。沿着给定路径上的每个属性-值对形成规则前件（IF 部分）的一个合取项。叶子结点包含类预测，形成规则后件（THEN 部分）。特别是当给定的树很大时，IF-THEN 规则易于理解。

例 5.3 还是以概念岩石可钻性，预测油气开采中岩石的钻探难易程度的决策树为例，沿着由根结点到叶子结点的路径，图 5.2 的判定树可以转换成 IF-THEN 分类规则，如表 5.5 所示。

表 5.5 IF-THEN 分类规则

IF 压入硬度 = “<900”	THEN 岩石可钻性 = “yes”
IF 压入硬度 = “900～4200” AND 岩石等级 = “≤1”	THEN 岩石可钻性 = “yes”
IF 压入硬度 = “900～4200” AND 岩石等级 = “>1”	THEN 岩石可钻性 = “no”
IF 压入硬度 = “>4200” AND 微钻钻速 = “medium/high”	THEN 岩石可钻性 = “yes”
IF 压入硬度 = “>4200” AND 微钻钻速 = “low”	THEN 岩石可钻性 = “no”

所提取的每个规则之间蕴含着析取(逻辑 OR)关系,不可能存在规则冲突,且每种可能的属性-值组合都存在一个规则。

2. 规则准确率和覆盖率

对于给定的元组,如果规则前件中的条件都成立,则说规则前件被满足,并且规则覆盖了该元组。

规则 R 可用它的覆盖率和准确率来评估,给定类标记的数据集中的一个元组 X,设 n_{cover} 为规则 R 覆盖的元组数,n_{correct} 为 R 正确分类的元组数,$|D|$ 是 D 中的元组数,则准确率 accuracy(R)和覆盖率 coverage(R)分别为

$$\text{accuracy}(R)=\frac{n_{\text{cover}}}{|D|} \tag{5.10}$$

$$\text{coverage}(R)=\frac{n_{\text{correct}}}{|D|} \tag{5.11}$$

5.3.7 决策树归纳的可扩展性

现有决策树算法,包括 ID3 和 C4.5 算法,其有效性已通过在多个小型数据集上的学习归纳得到验证。但当应用这些算法对大规模现实世界数据库进行数据挖掘时,算法的有效性和可扩展性就成为应用的关键。大多数决策树算法都局限于在计算机内存中处理整个数据集;而在数据挖掘应用领域,数据集通常都包含数以百万计的记录,现有决策树算法构造相应决策树时,会不断地进行内存与外存之间的数据交换,从而使数据挖掘性能变得很差,使得这类算法的可扩展性受到较大限制。

构建树过程所需要的内存大小,直接取决于学习样本集的大小。因此在执行构建过程之前,需对样本集进行容量估计。若所需内存小于可用内存,则直接将学习样本集调入内存进行树的构建;否则,需要采用优化算法(如 SLIQ、SPRINT 等)进行优化或借助数据库管理实现“内存扩充”来解决内存不足的问题。

SLIQ 使用若干驻留磁盘的属性表和单个驻留主存的类表。对于表 5.6 中的样本数据,SLIQ 产生的属性表和类表如图 5.4 所示。每一个属性具有一个属性表,在 RID(记录标识)建立索引。每个元组由一个从每个属性表的一个表目到类表的一个表目(存放给定元组的类标号)的链接表示。而类表表目链接到它在判定树中对应的叶子结点。类表驻留在主存,因为判定树在构造和剪枝时,经常访问它。类表的大小随训练集中元组数目成比例增长。当类表不能放在主存时,SLIQ 的性能下降。

表 5.6 类岩石可钻性元组的样本数据

RID	微钻钻速	压入硬度	岩石可钻性
1	high	880	yes
2	high	700	yes
3	medium	2600	no
4	low	5300	no

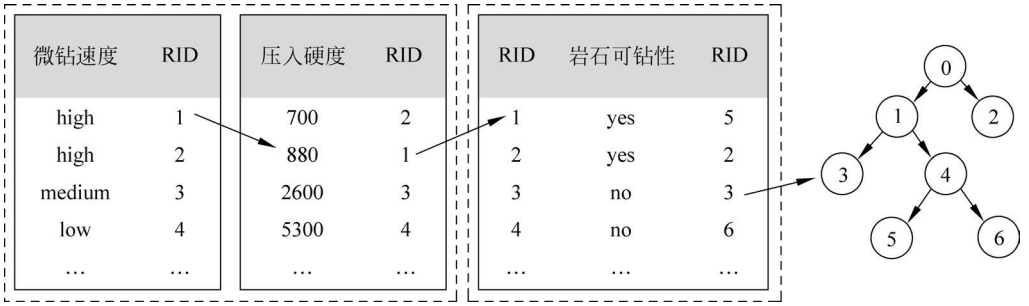


图 5.4 SLIQ 使用的属性表和类表

5.4 其他分类方法

根据分类思想可以将分类方法大致分为两类：基于距离的分类策略和基于统计的分类策略。其中，基于距离的分类策略的相似性用距离来表征，距离越近，相似性越大，距离越远，相似性越小。距离的计算方法有多种，最常用的是通过计算每个类的中心来完成，典型的有 KNN 等。

5.4.1 K-最邻近(近邻)分类

K-最邻近(K-Nearest Neighbor，KNN)分类算法，或者说邻近算法是数据挖掘分类技术中最简单的方法之一。所谓 K 最邻近，就是 K 个最近的邻居的意思，说的是每个样本都可以用它最接近的 K 个邻近值来代表。邻近算法就是将数据集合中每一个记录进行分类的方法。

基于类比学习，训练样本用 n 维数值属性描述。每个样本代表 n 维空间中的一个点。这样，所有的训练样本都存放在 n 维模式空间中。给定一个未知样本，K-最邻近分类法搜索模式空间，找出最接近未知样本的 K 个训练样本。这 K 个训练样本是未知样本的 K 个“近邻”。“邻近性”用欧几里得距离定义。其中，两个点 $X=(x_1,x_2,\cdots,x_n)$ 和 $Y=(y_1,y_2,\cdots,y_n)$ 的欧几里得距离是

$$d(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \tag{5.12}$$

未知样本被分配到 K 个最邻近者中最公共的类。当 $K=1$ 时，未知样本被指定到模式空间中与之最邻近的训练样本的类。

算法：KNN 分类算法。
输入：训练数据 T；近邻数目 K；待分类的元组 t。
输出：类别 c。
步骤：

(1) $N=\varnothing$；
(2) **FOR** each $d\in T$ **DO BEGIN**
(3)　　**IF** $|N|\leqslant K$ **THEN**
(4)　　$N=N\cup\{d\}$；
(5)　　**ELSE**
(6)　　**IF** $u\in N$ such that $\mathrm{sim}(t,u)<\mathrm{sim}(t,d)$ **THEN BEGIN**
(7)　　$N=N-\{u\}$；
(8)　　$N=N\cup\{d\}$；
(9)　　**END**
(10)**END**
(11)$c=$class to which the most $u\in N$。

KNN 算法简单，易于理解，易于实现，无须估计参数，无须训练，适合对稀有事件进行分类(例如，当流失率很低时，如低于 0.5%，构造流失预测模型)，特别适合于多分类问题(对象具有多个类别标签)。例如，根据基因特征来判断其功能分类，KNN 比 SVM 的表现要好，但采用的是懒惰学习法，对测试样本分类时的计算量大，内存开销大，评分慢；同时可解释性较差，无法给出决策树那样的规则。

例 5.4　根据表 5.7 中两种类型共 10 个点的空间位置，判断点(6.653，10.849)属于哪一类别。

表 5.7　空间点坐标元组的样本数据

序　　号	*X* 轴坐标	*Y* 轴坐标	类　　别
1	1.2	2.8	1
2	1.9	3.7	1
3	2.5	3.8	1
4	4.8	7.9	1
5	9.7	2.6	0
6	5.6	7.8	1
7	10.8	2.7	0
8	13.7	22.7	0
9	5.48	14.82	0
10	11.23	17.16	0

根据 KNN 算法，需要将新加入的点(6.653，10.849)与数据中包含的所有的点进行距离比较，然后选取前 K 个距离最近的点统计其所属的类别，从而推断点(6.653，10.849)所属的类别。

在实现时先确定程序所需环境和工程包(以 Python 实现为例)：

```
01 import numpy as np
02 import matplotlib.pyplot as plt
03 from collections import Counter
```

构建空间点数据集：

```
#定义训练集
X_train = np.array([[1.2,2.8], [1.9,3.7], [2.5,3.8], [4.8,7.9], [9.7,2.6],[5.6,7.8],
[10.8,2.7], [13.7,22.7], [5.48,14.82], [11.23,17.16]])
Y_train = np.array([1,1,1,1,0,1,0,0,0,0])
```

构建算法函数：

```
x = np.array([6.653,10.849])          #需判断的样本点
"""
函数说明:排序并选出距离样本点 x 最小的 k 个点,算法关心的不是距离本身的大小,而是这 k 个
最小距离对应的是样本集中的哪 k 个样本,从而判断 k 个样本有多少属于 0 类,有多少属于 1 类
Parameters:
    X_train -点的坐标
"""
distance = []
for x_train in X_train:
    d = np.sqrt(np.sum((x-x_train) ** 2))
    distance.append(d)                #计算样本点与各点间的距离
out = np.argsort(distance)            #对距离列表 distance 元素排序,选出最小的 k 个点
k = 5                                 #定义 k 值
"""
函数说明:判断选出的前 k 个点判断属于哪一类
Parameters:
    Y_train -点的标签(所属的类别)
"""
topK_y = [Y_train[i] for i in out[:k]]
r = Counter(topK_y)
s = r.most_common(1)[0][0]
print(s)
```

因为数据集是涉及空间位置的点坐标,所以完成分类后将结果以图的形式显示出来。

```
"""
函数说明:画图并显示
Parameters:
    X_train -点的坐标
    Y_train -点的标签(所属的类别)
"""
plt.scatter(X_train[Y_train==0,0],X_train[Y_train==0,1],c='g')
plt.scatter(X_train[Y_train==1,0],X_train[Y_train==1,1],c='r')
plt.scatter(x[0],x[1],c='b')
plt.show()
```

5.4.2 基于统计的分类策略

基于统计的分类方法中最经典的就是朴素贝叶斯分类器(Naive Bayes Classifier,NBC),本节以 NBC 为例介绍基于统计的分类策略。NBC 可以预测类隶属关系的概率,如一个给定的元组属于一个特定类的概率。可以与判定树和神经网络分类算法相媲美。当被用于大型数据库时,贝叶斯分类也已表现出高准确率与高速度。假定一个属性值对给定类的影响独立于其他属性的值(类条件独立性,属性条件独立性假设),预测未知样本的类别为

后验概率最大的那个类别。

设 X 是类标号未知的数据样本，设 H 为某种假定，例如，数据样本 X 属于某特定的类 C。对于分类问题，我们希望确定 $P(H|X)$——给定观测数据样本 X，假定 H 成立的概率。$P(H|X)$是后验概率，即条件 X 发生的情况下 H 发生的后验概率。例如，假定数据样本由水果组成，用它们的颜色和形状描述。假定 X 表示“红色和圆形”，H 表示“X 是苹果”，则 $P(H|X)$反映当我们看到 X 是红色并是圆形时，对 X 是苹果的确信程度。$P(H)$是先验概率，或 H 的先验概率。对于我们的例子，它是任意给定的数据样本为苹果的概率，而不管数据样本看上去如何。与先验概率 $P(H)$相比，后验概率 $P(H|X)$基于更多的信息（如背景知识）。类似地，$P(X|H)$是在假设 H 成立（即“X 是苹果”）的条件下，观察到的 X（红色且圆形）的概率。$P(X)$是 X 的先验概率。

贝叶斯定理提供了一种由 $P(X)$、$P(H)$和 $P(X|H)$计算后验概率 $P(H|X)$的方法：

$$P(H \mid X)=\frac{P(X \mid H)P(H)}{P(X)} \tag{5.13}$$

其中，$P(X)$表示 X 的先验概率，$P(H)$表示 H 的先验概率，$P(X|H)$表示条件 H 下 X 的后验概率，$P(H|X)$表示条件 X 下 H 的后验概率。

用贝叶斯定理来估计后验概率 $P(H|X)$的困难在于 $P(X|H)$是所有属性上的联合概率，难以从有限的训练样本直接估计而得。为避开这个障碍，朴素贝叶斯分类器采用了属性条件独立性假设，即每个属性独立地对分类结果发生影响。则式(5.13)可重写为

$$P(H \mid X)=\frac{P(X \mid H)P(H)}{P(X)}=\frac{P(H)}{P(X)}\prod_{i=1}^{d}P(x_i \mid H) \tag{5.14}$$

其中，d 为属性数目，x_i 为 X 在第 i 个属性上的取值。

基本工作步骤如下。

(1) 每个数据样本用一个 n 维特征向量 $\boldsymbol{X}=\{x_1,x_2,\cdots,x_n\}$表示，描述由属性 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，给定一个未知的没有类标号的数据样本 X，分类法将预测 X 属于具有最高后验概率（条件 X 下）的类。即朴素贝叶斯分类将未知的样本分配给类 C_i，当且仅当

$$P(C_i \mid X)>P(C_j \mid X),\quad 1\leqslant j\leqslant m, j\neq i \tag{5.15}$$

最大化 $P(C_i|X)$，其最大的类 C_i 称为最大后验假定。根据贝叶斯定理，即式(5.13)，有

$$P(C_i \mid X)=\frac{P(X \mid C_i)P(C_i)}{P(X)} \tag{5.16}$$

由于 $P(X)$对于所有类为常数，只需要最大化 $P(X|C_i)P(C_i)$即可。如果类的先验概率未知，则通常假定这些类是等概率的，即 $P(C_1)=P(C_2)=\cdots=P(C_m)$。并据此对 $P(C_i|X)$最大化。注意，类的先验概率可以用 $P(C_i)=\frac{s_i}{s}$估算；其中，s_i 是类 C 中的训练样本数，而 s 是训练样本总数。

给定具有许多属性的数据集，计算 $P(X|C_i)$的开销可能非常大。为降低开销，可以做类条件独立的朴素假定。给定样本的类标号，假定属性值条件相互独立，即在属性间不存在

依赖关系。这样：

$$P(X\mid C_i)=\prod_{k=1}^{n}P(x_k\mid C_i) \tag{5.17}$$

概率 $P(x_k|C_i)$可以由训练样本估值。

(3) 为对未知样本 X 分类，对每个类 C_i，计算 $P(X|C_i)P(C_i)$。样本 X 被指派到类 C_i，当且仅当

$$P(X\mid C_i)P(C_i)>P(X\mid C_j)P(C_j),\quad 1\leqslant j\leqslant m,j\neq i \tag{5.18}$$

即 X 被指派到其 $P(X|C_i)P(C_i)$最大的类 C_i。

以伯努利朴素贝叶斯为例，伯努利朴素贝叶斯分类适用于离散数据，且设计用于二进制/布尔特征，即假定样本特征的条件概率分布服从二项分布，即“0-1 分布”。

例 5.5 根据表 5.8 中近 10 天天气数据预测无风、不潮湿、不闷热但是多云的天气会不会下雨。

表 5.8 天气元组的样本数据

时 间	有 风	潮 湿	多 云	闷 热	是否下雨
第 1 天	No	Yes	No	Yes	Yes
第 2 天	Yes	Yes	Yes	Yes	Yes
第 3 天	Yes	Yes	Yes	No	Yes
第 4 天	No	Yes	Yes	No	Yes
第 5 天	No	Yes	No	No	No
第 6 天	No	Yes	No	Yes	Yes
第 7 天	Yes	Yes	No	Yes	No
第 8 天	Yes	No	No	Yes	Yes
第 9 天	Yes	Yes	No	Yes	Yes
第 10 天	No	No	No	No	No

将各种属性和是否下雨中 No 用 0 标识，Yes 用 1 标识，则表 5.8 中 10 天以来下雨的情况就是[1，1，1，1，0，1，0，1，1，0]。首先构建数据集，设 x 表示一天当中的 4 种天气属性，y 表示当天的标签，即是否下雨。

```
01 import numpy as np
02 """
03 函数说明:根据表格构建数据集
04 """
05 x = np.array([[0,1,0,1],[1,1,1,1],[1,1,1,0],[0,1,1,0],[0,1,0,0],[0,1,0,1],
06                [1,1,0,1],[1,0,0,1],[1,1,0,1],[0,0,0,0]])
07 y = np.array([1,1,1,1,0,1,0,1,1,0])
```

构建基础贝叶斯，此处以 Scikit-learn 中的朴素贝叶斯包为例。

```
01 import warnings
02 from abc import ABCMeta, abstractmethod
03 import numpy as np
04 from scipy.special import logsumexp
05 """
06 调包说明:调取 sklearn 的数据处理包
07 """
```

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.preprocessing import binarize
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import label_binarize
from sklearn.utils import deprecated
from sklearn.utils.extmath import safe_sparse_dot
from sklearn.utils.multiclass import _check_partial_fit_first_call
from sklearn.utils.validation import check_is_fitted, check_non_negative
from sklearn.utils.validation import _check_sample_weight
__all__ = [
    "BernoulliNB",
    ]
class _BaseNB(ClassifierMixin, BaseEstimator, metaclass=ABCMeta):
    @abstractmethod
"""
函数说明：计算 X 的非归一化后验对数概率，predict、predict_proba 和 predict_log_proba 将通过_check_X 输入并将其交给_joint_log_likelihood
"""
    def _joint_log_likelihood(self, X):
@abstractmethod
"""
函数说明：在子类中被实际检查覆盖.仅用于 predict* 方法
"""
    def _check_X(self, X):
"""
函数说明：对一组测试向量 X 执行分类
Parameters:
        X -输入样本
    Return:
        C -X 的预测目标值
"""
    def predict(self, X):
        check_is_fitted(self)
        X = self._check_X(X)
        jll = self._joint_log_likelihood(X)
        return self.classes_[np.argmax(jll, axis=1)]
"""
函数说明：返回测试向量 X 的对数概率估计
Parameters:
        X -输入样本
    Return:
        C -返回模型中每个类的样本的对数概率.这些列对应于按排序顺序排列的类
"""
    def predict_log_proba(self, X):
        check_is_fitted(self)
        X = self._check_X(X)
        jll = self._joint_log_likelihood(X)
        #normalize by P(x) = P(f_1, ..., f_n)
        log_prob_x = logsumexp(jll, axis=1)
        return jll - np.atleast_2d(log_prob_x).T
"""
函数说明：返回测试向量 X 的概率估计
Parameters:
        X -输入样本
```

```
    Return:
        C -返回模型中每个类的样本的对数概率.这些列对应于按排序顺序排列的类
"""
    def predict_proba(self, X):
        return np.exp(self.predict_log_proba(X))
```

构建伯努利贝叶斯的类。

```
"""
Parameters:
    alpha -定义是否附加(拉普拉斯/利德斯通)平滑参数.如果为0,则不附加
    binarize -定义样本特征的二值化(映射到布尔值)阈值.如果没有,则假定输入已包含二进制向量
    fit_prior -定义是否学习类别的先验概率.如果否,则使用统一的先验概率
    class_prior -定义类别的先验概率.如果指定,则不根据数据调整先验概率
"""
class BernoulliNB(_BaseDiscreteNB):
    def __init__(self, *, alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None):
        self.alpha = alpha
        self.binarize = binarize
        self.fit_prior = fit_prior
        self.class_prior = class_prior
    def _check_X(self, X):
        """Validate X, used only in predict* methods."""
        X = super()._check_X(X)
        if self.binarize is not None:
            X = binarize(X, threshold=self.binarize)
        return X
    def _check_X_y(self, X, y, reset=True):
        X, y = super()._check_X_y(X, y, reset=reset)
        if self.binarize is not None:
            X = binarize(X, threshold=self.binarize)
        return X, y
    """
    函数说明:计数和平滑特征
    """
    def _count(self, X, Y):
        self.feature_count_ += safe_sparse_dot(Y.T, X)
        self.class_count_ += Y.sum(axis=0)
    """
    函数说明:对原始计数应用平滑并重新计算对数概率
    """
    def _update_feature_log_prob(self, alpha):
        smoothed_fc = self.feature_count_ + alpha
        smoothed_cc = self.class_count_ + alpha * 2
        self.feature_log_prob_ = np.log(smoothed_fc) - np.log(
            smoothed_cc.reshape(-1, 1) )
    """
    函数说明:计算样本 X 的后验对数概率
    """
    def _joint_log_likelihood(self, X):
        n_features = self.feature_log_prob_.shape[1]
        n_features_X = X.shape[1]
```

```
45            if n_features_X != n_features:
46                raise ValueError(
47                    % (n_features, n_features_X))
48            neg_prob = np.log(1 - np.exp(self.feature_log_prob_))
49            jll = safe_sparse_dot(X, (self.feature_log_prob_ - neg_prob).T)
50            jll += self.class_log_prior_ + neg_prob.sum(axis=1)
51            return jll
```

调用伯努利朴素贝叶斯类完成天气预测。

```
01 bnb = BernoulliNB()
02 bnb.fit(x,y)
03 day_pre=[[0,0,1,0]]
04 pre = bnb.predict(day_pre)
05 print("预测结果如下\n:",'*'*50)
06 print('结果为:',pre)
07 print('*'*50)
08 """
09     函数说明:进一步查看概率分布
10 """
11 pre_pro = bnb.predict_proba(day_pre)
12 print("不下雨的概率为: ",pre_pro[0][0],"\n下雨的概率为",pre_pro[0][1])
```