第 4 章　机器学习

机器学习是研究如何使用机器来模拟人类学习活动的一门科学和技术。本章对机器学习的研究方法进行探讨，并介绍几种典型的机器学习方法。

机器学习

4.1 机器学习基本概念

4.1.1 机器学习含义

1.机器学习的定义

学习是人类具有的一种重要智能行为，但究竟是什么学习，长期以来却众说纷纭，社会学家、逻辑学家和心理学家都有各自不同的看法。

人工智能大师西蒙(Simon)对学习给出比较准确的定义：

定义 1.1 学习表示系统中的自适应变化，该变化能使系统比上一次更有效地完成同一群体所执行的同样任务。

米切尔(Mitchell)给学习下了个比较宽广的定义，使其包括任何计算机程序通过经验来提高某个任务处理性能的行为：

定义 1.2 对于某类任务 T 和性能度量 P，如果计算机程序在 T 上以 P 衡量的性能随着经验 E 而自我完善，那么就称这个计算机程序从经验 E 中学习。

那什么又叫做机器学习(machine learning)？至今，依然没有统一的定义，而且也很难给出一个公认的和准确的定义。但为了便于进行讨论和估计学科的进展，有必要对机器学习给出定义，即使这种定义是不完全和不充分的。

定义 1.3 顾名思义，机器学习是研究如何使用机器来模拟人类学习活动的一门学科。

稍微严格的提法是：

定义 1.4 机器学习是一门研究机器获取新知识和新技能，并识别现有知识的学问。

综合上述两定义，可给出如下定义：

定义 1.5 机器学习是研究机器模拟人类的学习活动、获取知识和技能的理论和方法，以改善系统性能的学科。

这里所说的“机器”，指的就是计算机；现在是电子计算机，以后还可能是中子计算机、量子计算机、光子计算机或神经计算机等。

2.机器学习的主要策略

学习是一项复杂的智能活动，学习过程与推理过程是紧密相连的，按照学习中使用推理的多少，机器学习所采用的策略大体上可分为 4 种——机械学习、示教学习、类比学习和示例学习。学习中所用到的推理越多，系统的能力越强。

机械学习就是记忆，是最简单的学习策略。这种学习策略不需要任何推理过程，外界输入知识的表示方法与系统内部表示方式完全一致，不需要任何处理与转换。这种机械学习在方法上看起来很简单，但由于计算机的存储容量相当大，检索速度又相当快，而且记忆精确、无丝毫误差，所以也能产生人们难以预

料的效果。

示教学习策略相比于机械学习更复杂。对于使用示教学习策略的系统来说，外界输入知识的表达方式与内部表达方式不完全一致，系统在接受外部知识时需要一点推理、翻译和转化工作。MYCIN、DENDRAL等专家系统在获取知识上都采用这种学习策略。

类比学习系统只能得到完成类似任务的有关知识，因此，学习系统必须能够发现当前任务与已知任务的相似之点，由此制定出完成当前任务的方案，因此，比上述两种学习策略需要更多的推理。

采用示例学习策略的计算机系统，事先完全没有完成任务的任何规律性的信息，所得到的只是一些具体的工作例子及工作经验。系统需要对这些例子及经验进行分析、总结和推广，得到完成任务的一般性规律，并在进一步的工作中验证或修改这些规律，因此需要的推理是最多的。此外，还有基于解释的学习、决策树学习、增强学习和基于神经网络的学习等。

3.机器学习系统的基本结构

下面以西蒙的学习定义为出发点，建立起简单的学习模型，然后通过对这个简单模型的讨论，总结出设计学习系统应当注意的某些总的原则。

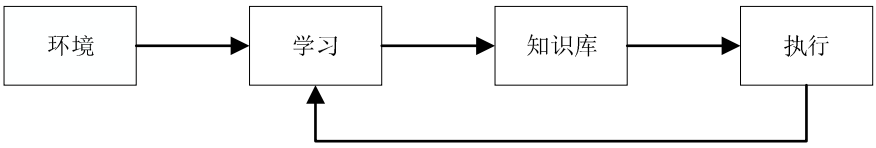

图 4.1 学习系统的基本结构

图 4.1 表示学习系统的基本结构。环境向系统的学习部分提供某些信息，学习部分利用这些信息修改知识库，以增进系统执行部分完成任务的效能，执行部分根据知识库完成任务，同时把获得的信息反馈给学习部分。在具体的应用中，环境、知识库和执行部分决定了具体的工作内容，学习部分所需要解决的问题完全由上述 3 部分确定。下面分别叙述这 3 部分对设计学习系统的影响。

影响学习系统设计的最重要的因素是环境向系统提供的信息。或者更具体地说是信息的质量。知识库里存放的是指导执行部分动作的一般原则，但环境向学习系统提供的信息却是各种各样的。如果信息的质量比较高，与一般原则的差别比较小，则学习部分比较容易处理。如果向学习系统提供的是杂乱无章的指导执行具体动作的具体信息，则学习系统需要在获得足够数据之后，删除不必要的细节，进行总结推广，形成指导动作的一般原则，放入知识库。这样学习部分的任务就比较繁重，设计起来也较为困难

因为学习系统获得的信息往往是不完全的，所以学习系统所进行的推理并不完全是可靠的，它总结出来的规则可能正确，也可能不正确，这要通过执行效果加以检验。正确的规则能使系统的效能提高，应予保留；不正确规则应予修改或从数据库中删除。

知识库是影响学习系统设计的第二个因素。知识的表示有多种形式，比如特征向量、一阶逻辑语句、产生式规则、语义网络和框架等。这些表示方式各有其特点，在选择表示方式时要兼顾以下 4 个方面：

(1)表达能力强。人工智能系统研究的一个重要问题是所选择的表示方式能很容易地表达有关的知识。

(2)易于推理。在具有较强表达能力的基础上，为了使学习系统的计算代价比较低，希望知识表示方式能使推理较为容易。

(3)容易修改知识库。学习系统的本质要求它不断修改自己的知识库，当推广得出一般执行规则后，要加到知识库中。当发现某些规则不适用时要将其删除。因此学习系统的知识表示一般都采用明确、统一的方式，如特征向量、产生式规则等，以利于知识库的修改。

(4)知识表示易于扩展。随着系统学习能力的提高，单一的知识表示已经不能满足需要；一个系统有时

同时使用几种知识表示方式。不但如此，有时还要求系统自己能构造出新的表示方式，以适应外界信息不断变化的需要。因此要求系统包含如何构造表示方式的元级描述。现在，人们把这种元级知识也看成是知识库的一部分。这种元级知识使学习系统的能力得到极大提高，使其能够学会更加复杂的东西，不断地扩大它的知识领域和执行能力。

对于知识库，最后需要说明的一个问题是学习系统不能在全然没有任何知识的情况下凭空获取知识，每一个学习系统都要求具有某些知识理解环境提供的信息，分析比较，做出假设，检验并修改这些假设。因此，更确切地说，学习系统是对现有知识的扩展和改进。

4. 机器学习、深度学习、人工智能的关系

在过去几年中，人工智能出现了爆炸式的发展，尤其是 2015 年之后。大部分原因，要归功于图形处理器（GPU）的广泛应用，使得并行处理更快、更便宜、更强大。另外，人工智能的发展还得益于几乎无限的存储空间和海量数据的出现（大数据运动）：图像、文本、交易数据、地图数据，应有尽有。图 4.2 概括了三者的包含关系。

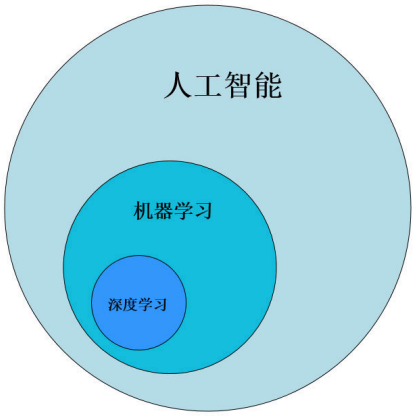

图 4.2 机器学习、深度学习、人工智能关系图

人工智能是一门极富挑战性的科学，从事这项工作的人必须懂得计算机知识，心理学和哲学。人工智能是包括十分广泛的科学，它由不同的领域组成，如机器学习，计算机视觉等等，总的说来，人工智能研究的一个主要目标是使机器能够胜任一些通常需要人类智能才能完成的复杂工作。但不同的时代、不同的人对这种“复杂工作”的理解是不同的。2017 年 12 月，人工智能入选“2017 年度中国媒体十大流行语”。

机器学习是实现人工智能的一种方法。机器学习的概念来自早期的人工智能研究者，已经研究出的算法包括决策树学习、归纳逻辑编程、增强学习和贝叶斯网络等。简单来说，机器学习就是使用算法分析数据，从中学习并做出推断或预测。与传统的使用特定指令集手写软件不同，我们使用大量数据和算法来“训练”机器，由此带来机器学习如何完成任务。

深度学习是实现机器学习的一种技术。早期机器学习研究者中还开发了一种叫人工神经网络的算法，但是发明之后数十年都默默无闻。神经网络是受人类大脑的启发而来的：神经元之间的相互连接关系。但是，人类大脑中的神经元可以与特定范围内的任意神经元连接，而人工神经网络中数据传播要经历不同的层，传播方向也不同。

4.1.2 数据集

要进行机器学习，先要有数据。假定我们收集了一批关于西瓜的数据，例如(色泽=青绿；根蒂=蜷缩；敲声=浊响)，(色泽=乌黑；根蒂:稍蜷；敲声=沉闷)，(色泽=浅白；根蒂=硬挺；敲声=清脆)，……，每对括

号内是一条记录，“=”意思是“取值为” 。

这组记录的集合称为一个“数据集”(data set) ，其中每条记录是关于一个事件或对象(这里是一个西瓜)的描述，称为一个“示例”(instance) 或“样本”(samp1e)。 反映事件或对象在某方面的表现或性质的事项，例如“色泽”、“根蒂”、“敲声”，称为“属性”(attribute) 或“特征”(feature)；属性上的取值，例如“青绿”、“乌黑”，称为“属性值”(attribute va1ue)。属性张成的空间称为“属性空间”(attribute space) 、“样本空间”(samp1e space) 或“输入空间”。例如我们把“色泽”、“根蒂”、“敲声”作为三个坐标轴，则它们张成一个用于描述西瓜的三维空间，每个西瓜都可在这个空间中找到自己的坐标位置，由于空间中的每个点对应一个坐标向量，因此我们也把一个示例称为一个“特征向量”(feature vector)。

一般地，令D $= \{x_1, x_2, \cdots, x_m\}$表示包含 m 个示例的数据集，每个示例由 d 个属性描述(例如上面的西瓜数据使用了 3 个属性)，则每个示例$x_i = (x_{i1}; x_{i2}; \cdots; x_{id})$是 d 维样本空间 X 中的一个向量，$x_i \in X$，其中$x_{ij}$是$x_i$在第 j 个属性上的取值(例如上述第 3 个西瓜在第 2 个属性上的值是“硬挺”)，d 称为样本x_i的“维数”(dimensionality)。

通常我们将数据集分成训练集(training set)和测试集(test set)。从数据中学得模型的过程称为“学习”(learning) 或“训练”(training)，这个过程通过执行某个学习算法来完成。训练过程中使用的数据称为“训练数据”(training data)，其中每个样本称为一个“训练样本”(training samp1e)，训练样本组成的集合称为“训练集”(training set)。学得模型对应了关于数据的某种潜在的规律，因此亦称"假设" (hypothesis)；这种潜在规律自身，则称为“真相”或“真实”(ground-truth)，学习过程就是为了找出或逼近真相。学得模型后，使用其进行预测的过程称为“测试”(test)，其中使用到的数据被称为“测试数据”(test data)，被预测的样本称为“测试样本”(test sample)，测试样本组成的集合称为“测试集”(test set)。

模型应用于新的测量数据之前，我们需要知道模型是否有效，也就是说，我们是否应该相信它的预测结果。不幸的是，我们不能将用于构建模型的数据用于评估模型。因为我们的模型会一直记住整个训练集，所以对于训练集中的任何数据点总会预测正确的标签。这种“记忆”无法告诉我们模型的泛化(generalize)能力如何(换句话说，在新数据上能否正确预测)。

我们要用新数据来评估模型的性能。新数据是指模型之前没有见过的数据，而我们有这些新数据的标签。通常的做法是将收集好的带标签数据分成两部分。一部分作为训练数据构建机器学习模型；其余的作为测试数据评估模型对前所未见的新数据的泛化能力。

代码部分：

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
iris_dataset = load_iris()X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
'''
运行以上程序可以得到输出:
X_train shape: (112, 4)
y_train shape: (112,)
X_test shape: (38, 4)
y_test shape: (38,)
```

4.1.3 模型

1.模型的概念

在监督学习过程中，模型就是所要学习的条件概率分布或决策函数。模型的假设空间(hypothesis space)包含所有可能的条件概率分布或决策函数。例如，假设决策函数是输入变量的线性函数，那么模型的假设空间就是所有这些线性函数构成的函数集合。假设空间中的模型一般有无穷多个。

假设空间用$\mathcal{F}$表示。假设空间可以定义为决策函数的集合

$$\mathcal{F} = \{f|Y = f(X)\} \tag{4.1}$$

其中，X和Y是定义在输入空间$\mathcal{X}$和输出空间$\mathcal{Y}$上的变量。这时$\mathcal{F}$通常是由一个参数向量决定的函数族：

$$\mathcal{F} = \{f|Y = f_\theta(X), \theta \epsilon R^n\} \tag{4.2}$$

参数向量θ取值于 n 维欧氏空间R^n，称为参数空间(parameter space)。

假设空间也可以定义为条件概率的集合

$$\mathcal{F} = \{P|P(Y|X)\} \tag{4.3}$$

其中，X和Y是定义在输入空间$\mathcal{X}$和输出空间$\mathcal{Y}$上的随机变量。这时$\mathcal{F}$通常是由一个参数向量决定的条件概率分布族：

$$\mathcal{F} = \{P|P_\theta(Y|X)，\ \theta \epsilon R^n\} \tag{4.4}$$

参数向量θ取值于 n 维欧氏空间R^n，也称为参数空间。

本书中称由决策函数表示的模型为非概率模型，由条件概率表示的模型为概率模型。为了简便起见，当论及模型时，有时只用其中一种模型。

模型作为机器学习的三要素之一，机器学习研究的是从数据中通过选取合适的算法，自动的归纳逻辑或规则，并根据这个归纳的结果(模型)与新数据来进行预测。其实很大程度上来说机器学习与人的学习有很多共通之处，由图 4.3 我们可以看到人是如何学习来类比机器学习。

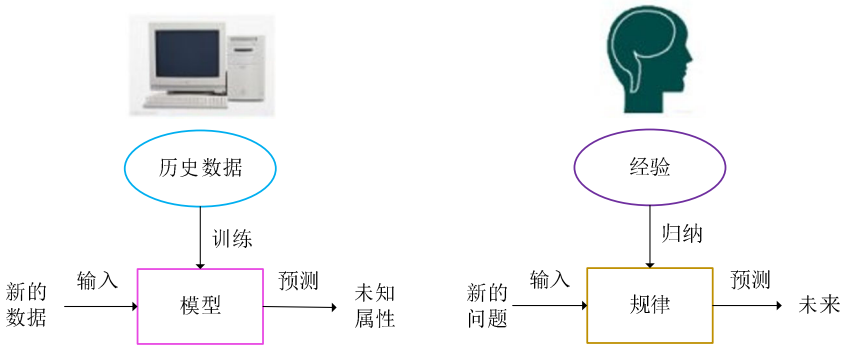

图 4.3 人的学习与机器学习的关系

假设有一对情侣，你是主人公(女友)，2 个月前，朋友给你介绍了一个男友，他是工作狂。为了互相了解，你们每周末都会一起约会吃饭；已经约会有 8 周了，每周吃饭男友都会比约定时间晚到 10 分钟-30 分钟，所以你每次约会也会比约定时间晚 10-30 分钟，并且你总结了一个规律(如果约会前打电话他说在公司，那么基本都是晚到 30 分钟左右，如果他说在家里，那么基本会晚到 10 分钟)，不过男友后来迟到的时间从 10、30 分钟变成了 15、45 分钟，你也自己调整了约定时间后到达的时间。类比: 机器学习方法是计算机利用已有的数据(8 次约会的经验)，得出了某种模型(迟到的规律)，并利用此模型预测未来(是否迟到)的一种方法。

人的学习有两个基本方法，一个是演绎法，一个是归纳法，这两种方法分别对应人工智能中的两种系统：专家系统和机器学习系统。所谓演绎法，是从已知的规则和事实出发，推导新的规则、新的事实，这对应于专家系统。专家系统也是早期的人工智能系统，它也称为规则系统，找一组某个领域的专家，如医学领域的专家，他们会将自己的知识或经验总结成某一条条规则、事实，例如某个人体温超过 37 度、流鼻涕、流眼泪，那么他就是感冒，这是一条规则。当这些专家将自己的知识、经验输入到系统中，这个系统便开始运行，每遇到一些新情况，会将之变为一条条事实。当将事实输入到专家系统时，专家会根据规则或事实进行推导、梳理，并得到最终结论，这便是专家系统。而归纳法是从现有样本数据中不断地观察、归纳、总结出规律和事实，对应机器学习系统或统计学习系统，其侧重于统计学习，从大量的样本中统计、挖掘、发现潜在的规律和事实。

2.如何构建模型：k 近邻算法

现在我们可以开始构建真实的机器学习模型了。scikit-learn 中有许多可用的分类算法。这里我们用的是 k 近邻分类器，这是一个很容易理解的算法。k 近邻算法中 k 的含义是，我们可以考虑训练集中与新数据点最近的任意 *k* 个邻居（比如说，距离最近的 3 个或 5 个邻居），而不是只考虑最近的那一个。然后，我们可以用这些邻居中数量最多的类别做出预测。现在我们只考虑一个邻居的情况。scikit-learn 中所有的机器学习模型都在各自的类中实现，这些类被称为 Estimator 类。k 近邻分类算法是在 neighbors 模块的 KNeighborsClassifier 类中实现的。我们需要将这个类实例化为一个对象，然后才能使用这个模型。这时我们需要设置模型的参数。KNeighborsClassifier 最重要的参数就是邻居的数目，这里我们设为 1：

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit (X_train, y_train)
‘’’
运行以上程序可以得到输出:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=1, p=2,
weights='uniform')
```

knn 对象对算法进行了封装，既包括用训练数据构建模型的算法，也包括对新数据点进行预测的算法。它还包括算法从训练数据中提取的信息。对于 KNeighborsClassifier 来说，里面只保存了训练集。想要基于训练集来构建模型，需要调用 knn 对象的 fit 方法，输入参数为 X_train 和 y_train，二者都是 NumPy 数组，前者包含训练数据，后者包含相应的训练标签。你也会注意到 n_neighbors=1，这是我们传入的参数。scikit-learn 中的大多数模型都有很多参数，但多用于速度优化或非常特殊的用途。

3.模型预测

现在我们可以用这个模型对新数据进行预测了，我们可能并不知道这些新数据的正确标签。想象一下，我们在野外发现了一朵鸢尾花，花萼长 5cm 宽 2.9cm，花瓣长 1cm 宽 0.2cm。这朵鸢尾花属于哪个品种？我们可以将这些数据放在一个 NumPy 数组中，再次计算形状，数组形状为样本数(1)乘以特征数(4)：

```
import numpy as np
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
‘’’
运行以上程序可以得到输出:
```

注意，我们将这朵花的测量数据转换为二维 NumPy 数组的一行，这是因为 scikit-learn 的输入数据必须是二维数组。接下来我们调用 knn 对象的 predict 方法来进行预测：

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(
iris_dataset['target_names'][prediction]))
‘’’
运行以上程序可以得到输出:
Prediction: [0]
Predicted target name: ['setosa']
```

根据我们模型的预测，这朵新的鸢尾花属于类别 0，也就是说它属于 setosa 品种。但我们怎么知道能否相信这个模型呢？我们并不知道这个样本的实际品种，这也是我们构建模型的重点，因此需要做第四节的内容。

4.模型评估

这里需要用到之前创建的测试集。这些数据没有用于构建模型，但我们知道测试集中每朵鸢尾花的实际品种。因此，我们可以对测试数据中的每朵鸢尾花进行预测，并将预测结果与标签（已知的品种）进行对比。我们可以通过计算精度（accuracy）来衡量模型的优劣，精度就是品种预测正确的花所占的比例：

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
'''
运行以上程序可以得到输出:
Test set predictions:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
Test set score: 0.97
```

我们还可以使用 knn 对象的 score 方法来计算测试集的精度：

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
'''
运行以上程序可以得到输出:
Test set score: 0.97
```

对于这个模型来说，测试集的精度约为 0.97，也就是说，对于测试集中的鸢尾花，我们的预测有 97% 是正确的。根据一些数学假设，对于新的鸢尾花，可以认为我们的模型预测结果有 97% 都是正确的。对于我们的植物学爱好者应用程序来说，高精度意味着模型足够可信，可以使用。在后续章节中，我们将讨论提高性能的方法，以及模型调参时的注意事项。

4.1.4 损失函数、风险函数

监督学习问题是在假设空间 F 中选取模型 f 作为决策函数，对于给定的输入 X，由 $f(X)$ 给出相应的输出 Y，这个输出的预测值 $f(X)$ 与真实值 Y 可能一致也可能不一致，用一个损失函数(loss function)或代价函数(cost function)来度量预测错误的程度，损失函数是 $f(X)$ 和 Y 的非负实值函数，记作 $L\left(Y，f(X)\right)$。

统计学习常用的损失函数有以下几种：

（1）0-1 损失函数(0-1 loss function)

$$L\left(Y，f(X)\right)=\begin{cases}1, & Y\neq f(X)\\ 0, & Y=f(X)\end{cases} \tag{4.5}$$

（2）平方损失函数(quadratic loss function)

$$L\left(Y，f(X)\right)=\left(Y-f(X)\right)^2 \tag{4.6}$$

（3）绝对损失函数(absolute loss function)

$$L\left(Y，f(X)\right)=|Y-f(X)| \tag{4.7}$$

（4）对数损失函数(logarithmic loss function)或对数似然损失函数(log-likelihood loss function)

$$L\left(Y，P(Y|X)\right)=-logP(Y|X) \tag{4.8}$$

损失函数值越小，模型就越好。由于模型的输入、输出$(X，Y)$是随机变量，遵循联合分布$P(X，Y)$，所以损失函数的期望是：

$$R_{exp}(f)=E_P\left[L\left(Y，f(X)\right)\right]=\int_{x\times y}L\left(y，f(x)\right)P(x,y)dxdy \tag{4.9}$$

这是理论上模型$f(X)$关于联合分布$P(X，Y)$的平均意义下的损失，称为风险函数(risk function)或期望损失(expected loss)。

学习的目标就是选择期望风险最小的模型。由于联合分布$P(X，Y)$是未知的，$R_{exp}(f)$不能直接计算。实际上，如果知道联合分布$P(X，Y)$，可以从联合分布直接求出条件概率分布$P(Y|X)$，也就不需要学习了。正因为不知道联合概率分布，所以才需要进行学习。这样一来，一方面根据期望风险最小学习模型要用到联合分布，另一方面联合分布又是未知的，所以监督学习就成为一个病态问题 (ill-formed problem)。

4.1.5 性能评估

在现实任务中，我们往往有多种学习算法可供选择，甚至对同一个学习算法，当使用不同的参数配置时，也会产生不同的模。.那么，我们该选用哪一个学习算法、使用哪一种参数配置呢？这就是机器学习中的“模型选择”(model selection)问题，理想的解决方案当然是对候选模型的泛化误差进行评估，然后选择泛化误差最小的那个模型。然而如上面所讨论的，我们无法直接获得泛化误差，而训练误差又由于过拟合现象的存在而不适合作为标准，那么，在现实中如何进行模型性能评估呢？

通常，我们可通过实验测试来对学习器的泛化误差进行评估并进而做出选择。为此，需使用一个“测试集”(testing set)来测试学习器对新样本的判别能力，然后以测试集上的“测试误差”(testing error)作为泛化误差的近似。通常我们假设测试样本也是从样本真实分布中独立同分布采样而得。但需注意的是，测试集应该尽可能与训练集互斥，即测试样本尽量不在训练集中出现、未在训练过程中使用过。

测试样本为什么要尽可能不出现在训练集中呢？为理解这一点，不妨考虑这样一个场景：老师出了 10 道习题供同学们练习，考试时老师又用同样的这 10 道题作为试题，这个考试成绩能否有效反映出同学们学得好不好呢？答案是否定的，可能有的同学只会做这 10 道题却能得高分。回到我们的问题上来，我们希望得到泛化性能强的模型，好比是希望同学们对课程学得很好、获得了对所学知识“举一反三”的能力；训练样本相当于给同学们练习的习题，测试过程则相当于考试。显然，若测试样本被用作训练了，则得到的将是过于“乐观”的估计结果。

可是，我们只有一个包含 m 个样例的数据集$D=\{(x_1，y_1)，(x_2,y_2)，\cdots，(x_m,y_m)\}$，既要训练，又要测试，怎样才能做到呢？答案是：通过对D进行适当的处理，从中产生出训练集 S 和测试集 T。下面介绍几种常见的做法：

1. 留出法

“留出法”(hold-out)直接将数据集 D 划分为两个互斥的集合，其中一个集合作为训练集 S，另一个作为测试集 T，即$D=S\cup T$，$S\cap T=\emptyset$。在 S 上训练出模型后，用 T 来评估其测试误差，作为对泛化误差的估计。以二分类任务为例，假定 D 包含 1000 个样本，将其划分为 S 包含 700 个样本，T 包含 300 个样本，用 S 进行训练后，如果模型在 T 上有 90 个样本分类错误，那么其错误率为(90/300)×100%=30%，相应的，精度为 1-30%=70%。需注意的是，训练/测试集的划分要尽可能保持数据分布的一致性，避免因数据划分过程引入额外的偏差而对最终结果产生影响，例如在分类任务中至少要保持样本的类别比例相似。如果从采样(sampling)的角度来看待数据集的划分过程则保留类别比例的采样方式通常称为“分层采样”(stratified sampling)。例如通过对 D 进行分层采样而获得含 70%样本的训练集 S 和含 30%样本的测试集 T，若 D 包

含 500 个正例、500 个反例，则分层采样得到的 S 应包含 350 个正例、350 个反例，而 T 则包含 150 个正例和 150 个反例，若 S、T 中样本类别比例差别很大，则误差估计将由于训练/测试数据分布的差异而产生偏差。

另一个需注意的问题是，即便在给定训练/测试集的样本比例后，仍存在多种划分方式对初始数据集 D 进行分割，例如在上面的例子中，可以把 D 中的样本排序，然后把前 350 个正例放到训练集中，也可以把最后 350 个正例放到练集中，……这些不同的划分将导致不同的训练/测试集，相应的，模型评估的结果也会有差别。因此，单次使用留出法得到的估计结果往往不够稳定可靠，在使用留出法时，一般要采用若干次随机划分、重复进行实验评估后取平均值作为留出法的评估结果。例如进行 100 次随机划分，每次产生一个训练/测试集用于实验评估，100 次后就得到 100 个结果，而留出法返回的则是这 100 个结果的平均。

此外，我们希望评估的是用 D 训练出的模型的性能，但留出法需划分训练/测试集，这就会导致一个窘境：若令训练集 S 包含绝大多数样本，则训练出的模型可能更接近于用 D 训练出的模型，但由于 T 比较小，评估结果可能不够稳定准确，若令测试集 T 多包含一些样本，则训练集 S 与 D 差别更大了，被评估的模型与用 D 训练出的模型相比可能有较大差别，从而降低了评估结果的保真性(fidelity)。这个问题没有完美的解决方案，常见做法是将大约 2/3～4/5 的样本用于训练，剩余样本用于测试。

2. 交叉验证法

“交叉验证法”(cross validation)先将数据集 D 划分为 k 个大小相似的互斥子集，即$D = D_1 \cup D_2 \cup \cdots \cup D_k, D_i \cap D_j = \emptyset (i \neq j)$。每个子集$D_i$都尽可能保持数据分布的一致性，即从 D 中通过分层采样得到。然后，每次用 k-1 个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得 k 组训练/测试集，从而可进行 k 次训练和测试，最终返回的是这 k 个测试结果的均值。显然，交叉验证法评估结果的稳定性和保真性在很大程度上取决 k 的取值，为强调这一点，通常把交叉验证法称为“k 折交叉验证”(k- fold cross validation)。k 最常用的取值是 10，此时称为 10 折交叉验证；其他常用的 k 值有 5、20 等。

与留出法相似，将数据集 D 划分为 k 个子集同样存在多种划分方式。为减小因样本划分不同而引入的差别，k 折交叉验证通常要随机使用不同的划分重复 p 次，最终的评估结果是这 p 次 k 折交叉验证结果的均值，例如常见的有“10 次 10 折交叉验证”。

假定数据集 D 中包含 m 个样本，若令 k=m，则得到了交叉验证法的一个特例：留一法(Leave-One-Out，简称 LOO)。显然，留一法不受随机样本划分方式的影响，因为 m 个样本只有唯一的方式划分为 m 个子集——每个子集包含一个样本；留一法使用的训练集与初始数据集相比只少了一个样本，这就使得在绝大多数情况下，留一法中被实际评估的模型与期望评估的用 D 训练出的模型很相似。因此，留一法的评估结果往往被认为比较准确。然而，留一法也有其缺陷：在数据集比较大时，训练 m 个模型的计算开销可能是难以忍受的(例如数据集包含 1 百万个样本，则需训练 1 百万个模型)，而这还是在未考虑算法调参的情况下。另外，留一法的估计结果也未必永远比其他评估方法准确；“没有免费的午餐”定理对实验评估方法同样适用。

3. 自助法

我们希望评估的是用 D 训练出的模型。但在留出法和交叉验证法中，由于保留了一部分样本用于测试，因此实际评估的模型所使用的训练集比 D 小，这必然会引入一些因训练样本规模不同而导致的估计偏差。留一法受训练样本规模变化的影响较小，但计算复杂度又太高了，有没有什么办法可以减少训练样本规模不同造成的影响，同时还能比较高效地进行实验估计呢？

“自助法”(bootstrapping)是一个比较好的解决方案，它直接以自助采样法(bootstrap sampling)为基础。给定包含 m 个样本的数据集 D，我们对它进行采样产生数据集D′。每次随机从 D 中挑选一个样本，将其拷贝放入D′，然后再将该样本放回初始数据集 D 中，使得该样本在下次采样时仍有可能被采到；这个过程

重复执行 m 次后，我们就得到了包含 m 个样本的数据集D′，这就是自助采样的结果。显然，D 中有一部分样本会在D′中多次出现，而另一部分样本不出现，可以做一个简单的估计，样本在 m 次采样中始终不被采到的概率是$(1-\frac{1}{m})^m$，取极限得到

$$\lim_{m\to\infty}(1-\frac{1}{m})^m \to \frac{1}{e} \approx 0.368 \tag{4.10}$$

即通过自助采样，初始数据集 D 中约有 36.8%的样本未出现在采样数据集D′中，于是我们可将D′用作训练集，D\D′用作测试集，这样，实际评估的模型与期望评估的模型都使用 m 个训练样本，而我们仍有数据总量约 1/3 的、没在训练集中出现的样本用于测试，这样的测试结果，亦称“包外估计”(out- of-bag estimate)。

自助法在数据集较小、难以有效划分训练/测试集时很有用；此外，自助法能从初始数据集中产生多个不同的训练集，这对集成学习等方法有很大的好处。然而，自助法产生的数据集改变了初始数据集的分布，这会引入估计偏差。因此，在初始数据量足够时，留出法和交叉验证法更常用一些。

4. 调参与最终模型

大多数学习算法都有些参数(parameter)需要设定，参数配量不同，学得模型的性能往往有显著差别。因此，在进行模型性能评估时，除了要对适用学习算法进行选择，还需对算法参数进行设定，这就是通常所说的“参数调节”或简称“调参“(parameter tuning)。

读者可能马上想到，调参和算法选择没什么本质区别：对每种参数配置都训练出模型，然后把对应最好模型的参数作为结果。这样的考虑基本是正确的，但有一点需注意：学习算法的很多参数是在实数范围内取值，因此，对每种参数配置都训练出模型来是不可行的。现实中常用的做法是对每个参数选定一个范围和变化步长，例如在[0 ，0.2]范围内以 0.05 为步长，则实际要评估的候选参数值有 5 个，最终是从这 5 个候选值中产生选定值。显然，这样选定的参数值往往不是“最佳”值，但这是在计算开销和性能估计之间进行折中的结果，通过这个折中，学习过程才变得可行。事实上，即便在进行这样的折中后，调参往往仍很困难。可以简单估算一下：假定算法有 3 个参数，每个参数仅考虑 5 个候选值，这样对每一组训练/测试集就有$5^3 = 125$个模型需考察；;很多强大的学习算法有大量参数需设定，这将导致极大的调参工程量，以至于在不少应用任务中参数调得好不好往往对最终模型性能有关键性影响。

给定包含 m 个样本的数据集 D，在模型评估与选择过程中由于需要留出一部分数据进行评估测试，事实上我们只使用了一部分数据训练模型。因此，在模型选择完成后，学习算法和参数配置已选定，此时应该用数据集 D 重新训练模型。这个模型在训练过程中使用了所有 m 个样本，这才是我们最终提交给用户的模型。

另外，需注意的是，我们通常把学得模型在实际使用中遇到的数据称为测试数据，为了加以区分，模型评估与选择中用于评估测试的数据集常称为“验证集 " (validation set)。例如，在研究对比不同算法的泛化性能时，我们用测试集上的判别效果来估计模型在实际使用时的泛化能力，而把训练数据另外划为训练集和验证集，基于验证集上的性能来进行模型选择和调参。

4.1.6欠拟合、过拟合

在监督学习中，我们想要在训练数据上构建模型，然后能够对没见过的新数据(这些新数据与训练集具有相同的特性)做出准确预测。如果一个模型能够对没见过的数据做出准确预测，我们就说它能够从训练集泛化((generalize)到测试集。我们想要构建一个泛化精度尽可能高的模型。

通常来说，我们构建模型，使其在训练集上能够做出准确预测。如果训练集和测试集足够相似，我们预计模型在测试集上也能做出准确预测。不过在某些情况下这一点并不成立。例如，如果我们可以构建非常复杂的模型，那么在训集上的精度可以想多高就多高。

为了说明这一点，我们来看一个虚构的例子。比如有一个新手数据科学家，已知之前船的买家记录和

对买船不感兴趣的顾客记录，想要预测某个顾客是否会买船。目标是向可能购买的人发送促销电子邮件，而不去打扰那些不感兴趣的顾客。

假设我们有顾客记录，如表 4.1 所示。

表 4.1 顾客数据示例

年龄	拥有的小汽车数量	是否有房子	子女数量	婚姻状况	是否养狗	是否买过船
66	1	是	2	丧偶	否	是
52	2	是	3	已婚	否	是
22	0	否	0	已婚	是	否
25	1	否	1	单身	否	否
44	0	否	2	离异	是	否
39	1	是	2	已婚	是	否
26	1	否	2	单身	否	否
40	3	是	1	已婚	是	否
53	2	是	2	离异	否	是
64	2	是	3	离异	否	否
58	2	是	2	已婚	是	是
33	1	否	1	单身	否	否

对数据观察一段时间之后，我们的新手数据科学家发现了以下规律：“如果顾客年龄大于 45 岁，并且子女少于 3 个或没有离婚，那么他就想要买船。”如果你问他这个规律的效果如何，我们的数据科学家会回答：“100%准确!”的确，对于表中的数据，这条规律完全正确。我们还可以发现好多规律，都可以完美解释这个数据集中的某人是否想要买船。数据中的年龄都没有重复，因此我们可以这样说：66、52、53 和 58 岁的人想要买船，而其他年龄的人都不想买。虽然我们可以编出许多条适用于这个数据集的规律，但要记住，我们感兴趣的并不是对这个数据集进行预测，我们已经知道这些顾客的答案。我们想知道新顾客是否可能会买船。因此，我们想要找到一条适用于新顾客的规律，而在训练集上实现 100%的精度对此并没有帮助。我们可能认为数据科学家发现的规律无法适用于新顾客。它看起来过于复杂，而且只有很少的数据支持。例如，规律里“或没有离婚”这一条对应的只有一名顾客。

判断一个算法在新数据上表现好坏的唯一度量，就是在测试集上的评估。然而从直觉上看，我们认为简单的模型对新数据的泛化能力更好。如果规律是“年龄大于 50 岁的人想要买船”，并且这可以解释所有顾客的行为，那么我们将更相信这条规律，而不是与年龄、子女和婚姻状况都有关系的那条规律。因此，我们总想找到最简单的模型。构建一个对现有信息量来说过于复杂的模型，正如我们的新手数据科学家做的那样，这被称为过拟合(overfitting)。如果你在拟合模型时过分关注训练集的细节，得到了一个在训练集上表现很好、但不能泛化到新数据上的模型，那么就存在过拟合。与之相反，如果你的模型过于简单一一比如说，“有房子的人都买船”一一那么你可能无法抓住数据的全部内容以及数据中的变化，你的模型甚至在训练集上的表现就很差。选择过于简单的模型波称为欠拟合(underfitting)。

我们的模型越复杂，在训练数据上的预测结果就越好。但是，如果我们的模型过于复杂，我们开始过多关注训练集中每个单独的数据点，模型就不能很好地泛化到新数据上。二者之间存在一个最佳位置，可以得到最好的泛化性能。这就是我们想要的模型。图 4.4 给出了过拟合与欠拟合之间的权衡。

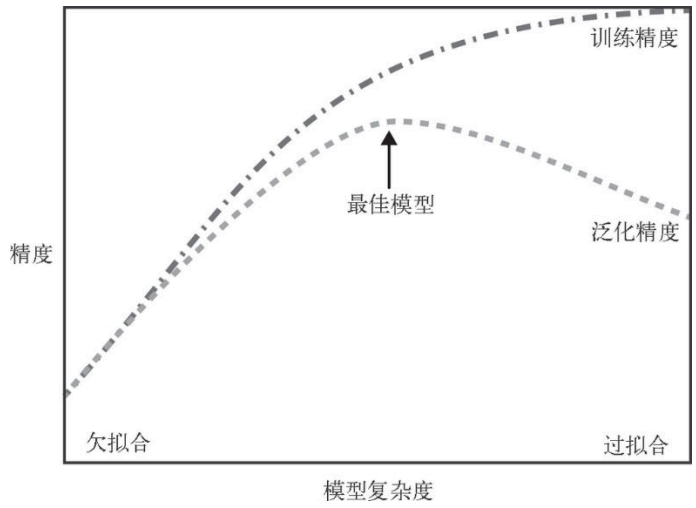

图 4.4 模型复杂度与训练精度和测试精度之间的权衡

需要注意，模型复杂度与训练数据集中输入的变化密切相关：数据集中包含的数据点的变化范围越大，在不发生过拟合的前提下你可以使用的模型就越复杂。通常来说，收集更多的数据点可以有更大的变化范围，所以更大的数据集可以用来构建更复杂的模型。但是，仅复制相同的数据点或收集非常相似的数据是无济于事的。

回到前面卖船的例子，如果我们查看了 10000 多行的顾客数据，并且所有数据都符合这条规律："如果顾客年龄大于 45 岁，并且子女少于 3 个或没有离婚，那么他就想要买船"，那么我们就更有可能相信这是一条有效的规律，比从表 1.1 中仅 12 行数据得出来的更为可信。

收集更多数据，适当构建更复杂的模型，对监督学习任务往往特别有用。在现实世界中，你往往能够决定收集多少数据，这可能比模型调参更为有效。永远不要低估更多数据的力量！

4.1.7 正则化

模型选择的典型方法是正则化(regularization)。正则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项(regularizer)或罚项(penalty term)。正则化项一般是模型复杂度的单调递增函数，模型越复杂，正则化值就越大。比如，正则化项可以是模型参数向量的范数。正则化一般具有如下形式：

$$\min_{f\epsilon F} \frac{1}{N}\sum_{i=1}^{N} L\big(y_i, f(x_i)\big) + \lambda J(f) \tag{4.11}$$

其中，第 1 项是经验风险，第 2 项是正则化项，$\lambda \geq 0$ 为调整两者之间关系的系数。

正则化项可以取不同的形式，例如，回归问题中，损失函数是平方损失，正则化项可以是参数向量的 L_2范数：

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i; w) - y_i)^2 + \frac{\lambda}{2}||w||^2 \tag{4.12}$$

这里，$||w||$表示参数向量w的L_2范数。

正则化项也可以是参数向量的L_1范数：

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i; w) - y_i)^2 + \lambda||w||_1 \tag{4.13}$$

这里，$||w||_1$表示参数向量w的L_1范数。

第 1 项的经验风险较小的模型可能较复杂(有多个非零参数)，这时第 2 项的模型复杂度会较大，正则化的作用是选择经验风险与模型复杂度同时较小的模型。

正则化符合奥卡姆剃刀(Occam's razor)原理，奥卡姆剃刀原理应用于模型选择时变为以下想法：在所有可能选择的模型中，能够很好地解释已知数据并且十分简单才是最好的模型，也就是应该选择的模型。从贝叶斯估计的角度来看，正则化项对应于模型的先验概率，可以假设复杂的模型有较大的先验概率，简单的模型有较小的先验概率。

4.1.8 梯度下降

在微积分中，对多元函数的参数求偏导数，求得参数的偏导数以向量的形式表达就是梯度。如图 4.5 所示，θ_0 到 θ_1 的距离为 θ_0 在函数 $L(\theta)$ 上的梯度，记作 $\partial L/\partial\theta$。那这个梯度向量。$\partial L/\partial\theta$ 有什么意义呢？数学上，梯度越大， 则函数的变化越大。对于函数 $L(\theta)$ 在点 θ_0 处，梯度向量。$\partial L/\partial\theta$ 的方向就是函数 $L(\theta)$ 增加最快的方向。也就是说，沿着梯度向量的方向易于找到函数的最大值。反之亦然，沿着与梯度向量相反的方向，梯度减少最快，易于找到函数的最小值。

假设函数 $L(\theta)$ 为损失函数，为了找到损失函数的最小值，需要沿着与梯度向量相反的方向 $-\partial L/\partial\theta$ 更新变量 θ，这样可以使得梯度减少最快，直至损失收敛至最小值。该算法称为梯度下降算法，其基本公式为：

$$\theta \leftarrow \theta - \eta\frac{\partial L}{\partial\theta} \tag{4.14}$$

其中，$\eta \in \mathcal{R}$为学习率，用于控制梯度下降的幅度(快慢)。在神经网络中，式(4.13)中的变量 θ ，为由神经网络的参数所组成的向量$\theta = (w_1, w_2, \ldots w_n, b_1, b_2, \ldots b_n)$。梯度下降的目标就是找到参数 θ^* 使得神经网络总损失 L 最小，从而确定神经网络中的权重向量 W 和偏置 b。

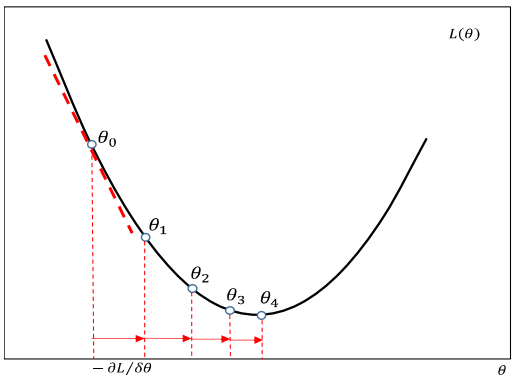

图 4.5 模型复杂度与训练精度和测试精度之间的权衡

梯度下降算法的特点是：越接近目标值，变化越小，梯度下降速度越慢。实际上，梯度下降算法的变种有很多，下面对常用的梯度下降算法进行介绍。

1. 批量梯度下降算法（Batch Gradient Descent, BGD)

批量梯度下降算法是梯度下降算法中最常用的形式，所有样本都参与参数 θ 的更新。假设有 m 个样

本， m 个样本都参与调整参数 θ，因此得到的是一个标准的梯度。
• 优点：易于得到全局最优解：总体迭代次数不多。
• 缺点：当样本数目很多时，训练时间过长，收敛速度变慢。

2. 随机梯度下降算法（Stochastic Gradient Descent, SGD)

随机梯度下降算法原理与批量梯度下降算法类似，区别在于随机梯度是从 m 个样本中随机抽取 n 个样本求解其梯度。
• 优点：训练速度快，每次迭代计算量少。
• 缺点：准确度下降，得到的不一定是全局最优；总体法代次数比较多。

3. 小批量随机梯度下降算法（Min-batch SGD)

批量梯度下降算法与随机梯度下降算法在样本选取上不同，优缺点明显。
• 在训练速度上，随机梯度算法每次采用一个样本进行法代，因此训练速度快；而批量梯度下降算法则因为样本量大，训练速度不能让人满意。
• 在收敛速度上，随机梯度下降算法一次选代的样本量有限，导致梯度变化很大，不能快速收敛到局部最优解。
• 在准确率上，随机梯度下降算法使用部分样本来决定梯度方向，导致得到的可能不是全局最优解，而是局部最优解。

小批量随机梯度下降算法是对批量梯度下降算法和随机梯度下降算法的折衷方法：每次随机从 m 个样本中抽取 k 个进行迭代求梯度，每一次迭代的抽取方式都是随机的，因此部分样本会重复。这样做的好处是计算梯度时让数据与数据之间产生关联，避免数据最终只能收敛到局部最优解。

4.1.9机器学习的类型 (监督、无监督、半监督、弱监督、强化学习)

1. 监督式学习(Supervised learning)，是一个机器学习中的方法。能够由训练资料中学到或建立一个模式(learning model)。并依此模式猜测新的实例。训练资料是由输入物件(一般是向量)和预期输出所组成。函数的输出能够是一个连续的值(称为回归分析)。或是预测一个分类标签(称作分类)。一个监督式学习者的任务在观察完一些训练范例（输入和预期输出）后，去预测这个函数对不论什么可能出现的输入的值的输出。

2. 无监督式学习(Unsupervised Learning)是人工智能网络的一种算法(algorithm)。其目的是去对原始资料进行分类，以便了解资料内部结构。有别于监督式学习网络，无监督式学习网络在学习时并不知道其分类结果是否正确，亦即没有受到监督式增强(告诉它何种学习是正确的)。其特点是仅对此种网络提供输入范例。而它会自己主动从这些范例中找出其潜在类别规则。当学习完成并经测试后，也能够将之应用到新的案例上。无监督学习最典型的样例就是聚类，聚类的目的在于把相似的东西聚在一起，而我们并不关心这一类是什么。因此，一个聚类算法通常仅仅须要知道怎样计算相似度就能够开始工作了。

3. 半监督学习的基本思想是利用数据分布上的模型如果, 建立学习器对未标签样本进行标签。形式化描写叙述为：给定一个来自某未知分布的样本集$S = L \cup U$, 当中 L 是已标签样本集 $L = \{(x1, y1), (x2, y2), \dots , (x\ |L|, y|L|)\}$，$U$是一个未标签样本集$U = \{x^1, x^2, \dots, x^{|U|}\}$，希望得到函数 $f: X \rightarrow Y$ 能够准确地对样本 x 预测其标签 y，这个函数可能是参数的，如最大似然法；可能是

非参数的，如最邻近法、神经网络法、支持向量机法等；也可能是非数值的，如决策树分类。其中 x 与 x' 均为 d 维向量， $y_i \in Y$ 为样本x_i 的标签，$|L|$ 和 $|U|$ 分别为 L 和 U 的大小， 即所包含的样本数。半监督学习就是在样本集 S 上寻找最优的学习器。怎样综合利用已标签样本和未标签样本，是半监督学习须要解决的问题。

4. 弱监督学习发展到现在，主要可以分成三个方向：

   - 不完全监督（incomplete supervison）
   - 不确切监督（inexact supervision）
   - 不准确监督（inaccurate supervision）

不完全监督，就是我们所说的半监督学习，一个训练数据集，有一部分数据是有标签的，也就是我们所说的 (x，y) 是对应的，还有一部分数据，就只有 x 而没有 y， 这种形式的学习，称为半监督学习。

不确切监督，指的是标签的粒度与我们想要解决的问题不匹配，比如对于分类来说，可能只要类别标签，我们就足够了，但是对于检测，或者分割来说，只有类别标签，而没有框，或者 mask 的信息，这种标签就比较 “弱"，这也可以看成是一种弱监督学习。不确切监督，基本研究的都是 bag learning 或者 multiple instance learning 这些问题。

不准确监督，指的是标签的准确性，这个在大数据集里面，经常会出现，数据一般都是人来标的，很难保证每个数据的标签都是完全正确的，所以会有错误标签的情况出现，这些错误标签，可以看成是一种 “噪声”，不准确监督，研究的就是含有噪声标签时的学习问题。

5. 强化学习是一类算法，是让计算机实现从一开始完全随机的进行操作，通过不断地尝试，从错误中学习，最后找到规律，学会了达到目的的方法。这就是一个完整的强化学习过程。让计算机在不断的尝试中更新自己的行为，从而一步步学习如何操自己的行为得到高分。它主要包含 Agent、环境状态、行动、奖励四个元素，强化学习的目标就是获得最多的累计奖励。

4.2 无监督学习

4.2.1 无监督学习的概念

根据训练数据是否拥有标记信息，学习任务可大致划分为两大类“监督学习”(supervised learning) 和“无监督学习”(unsupervised learning)，分类回归是前者的代表，而聚类则是后者的代表。

样本的类别标记未知称为未标记样本。事实上，未标记样本虽未直接包含标记信息，但若它们与有标记样本是从同样的数据源独立同分布来样而来，则它们所包含的关于数据分布的信息对建立模型将大有裨益。图 4.6 给出了一个直观的例示。若仅基于图中的一个正例和一个反例，则由于待判别样本恰位于两者正中间，大体上只能随机猜测；若能观察到图中的未标记样本，则将很有把握地判别为正例。

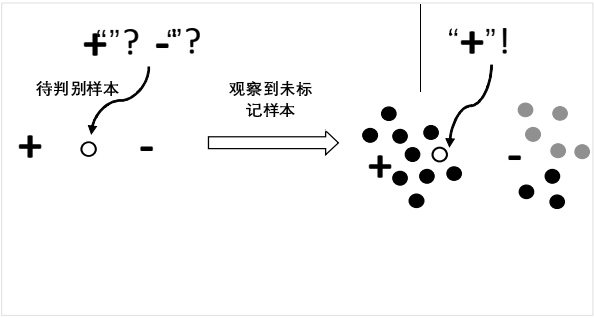

图 4.6 未标记样本效用的例示，右边的灰色点表示未标记样本

要利用未标记样本，必然要做一些将未标记样本所揭示的数据分布信息与类别标记相联系的假设。最常见的是“聚类假设”(cluster assumption)，即假设数据存在簇结构，同一个簇的样本属于同一个类别。图4.1 就是基于聚类假设来利用未标记样本，由于待预测样本与正例样本通过未标记样本的“撮合”聚在一起，与相对分离的反例样本相比，待判别样本更可能属于正类。

无监督学习包括没有已知输出、没有老师指导学习算法的各种机器学习。在无监督学习中，学习算法只有输入数据，并需要从这些数据中提取知识。本章将研究两种类型的无监督学习：数据集变换与聚类。

数据集的无监督变换(unsupervised transformation)是创建数据新的表示的算法，与数据的原始表示相比，新的表示可能更容易被人或其他机器学习算法所理解。无监督变换的一个常见应用是降维(dimensionality reduction)，它接受包含许多特征的数据的高维表示，并找到表示该数据的一种新方法，用较少的特征就可以概括其重要特性。降维的一个常见应用是为了可视化将数据降为二维。无监督变换的另一个应用是找到“构成”数据的各个组成部分。这方面的一个例子就是对文本文档集合进行主题提取。这里的任务是找到每个文档中讨论的未知主题，并学习每个文档中出现了哪些主题。这可以用于追踪社交媒体上的话题讨论，比如选举、枪支管制或流行歌手等话题。

与之相反，聚类算法(clustering algorithm)将数据划分成不同的组，每组包含相似的物项。思考向社交媒体网站上传照片的例子。为了方便你整理照片，网站可能想要将同一个人的照片分在一组。但网站并不知道每张照片是谁，也不知道你的照片集中出现了多少个人。明智的做法是提取所有的人脸，并将看起来相似的人脸分在一组。但愿这些人脸对应同一个人，这样图片的分组也就完成了。

无监督学习的一个主要挑战就是评估算法是否学到了有用的东西。无监督学习算法一般用于不包含任何标签信息的数据，所以我们不知道正确的输出应该是什么。因此很难判断个模型是否“表现很好”。例如，假设我们的聚类算法已经将所有的侧脸照片和所有的正面照片进行分组。这肯定是人脸照片集合的一种可能的划分方法，但并不是我们想要的那种方法。然而，我们没有办法“告诉”算法我们要的是什么，通常来说，评估无监督算法结果的唯一方法就是人工检查。

因此，如果数据科学家想要更好地理解数据，那么无监督算法通常可用于探索性的目的，而不是作为大型自动化系统的一部分。无监督算法的另一个常见应用是作为监督算法的预处理步骤。学习数据的一种新表示，有时可以提高监督算法的精度，或者可以减少内存占用和时间开销。

4.2.2 概率密度函数估计

概率密度函数估计总体上可分为两种方法：参数估计和非参数估计。参数估计是已知概率密度函数的形式，只是其中几个参数未知，目标是根据样本估计这些参数的值，最具代表性的是极大似然估计。二非参数估计不对概率密度函数的形式进行假设，直接用样本进行估计，最具代表性的是朴素贝叶斯分类器。本节也将详细阐述极大似然估计和朴素贝叶斯分类器两种方法。

1. 极大似然估计

估计类条件概率的一种常用策略是先假定其具有某种确定的概率分布形式，再基于训练样本对概率分布的参数进行估计。具体地，记关于类别 c 的类条件概率为$P(x|c)$，假设$P(x|c)$具有确定的形式并且被参数向量θ_c唯一确定，则我们的任务就是利用训练集 D 估计参数θ_c。为明确起见，我们将$P(x|c)$记为$P(x|\theta_c)$。

事实上，概率模型的训练过程就是参数估计(parameter estimation)过程。对于参数估计，统计学界的两个学派分别提供了不同的解决方案：频率主义学派(Frequentist)认为参数虽然未知，但却是客观存在的固定值，因此，可通过优化似然函数等准则来确定参数值；贝叶斯学派(Bayesian)则认为参数是未观察到的随机变量，其本身也可有分布，因此，可假定参数服从一个先验分布，然后基于观测到的数据来计算参数的后验分布。本节介绍源自频率主义学派的极大似然估计(Maximum Likelihood Estimation，简称 MLE)，这是根据数据采样来估计概率分布参数的经典方法。

令D_c表示训练集 D 中第 c 类样本组成的集合，假设这些样本是独立同分布的，则参数θ_c对于数据集D_c的似然是

$$P(D_c|\theta_c) = \prod_{x\epsilon D_c} P(x|\theta_c) \tag{4.14}$$

对θ_c进行极大似然估计，就是去寻找能最大化似然$P(D_c|\theta_c)$的参数值$\hat{\theta}_c$。直观上看，极大似然估计是试图在θ_c所有可能的取值中，找到一个能使数据出现的“可能性”最大的值。

式(4.14)中的连乘操作易造成下溢，通常使用对数似然(log- likelihood)

$$\begin{aligned} LL(\theta_c) &= \log P(D_c|\theta_c) \\ &= \sum_{x\in D_c} log P(x|\theta_c) \end{aligned} \tag{4.15}$$

此时参数θ_c的极大似然估计$\hat{\theta}_c$为

$$\hat{\theta}_c = \arg\max_{\theta_c} LL(\theta_c) \tag{4.16}$$

例如，在连续属性情形下，假设概率密度函数$P(x|c)\sim\aleph(\mu_c, \sigma_c^2)$，则参数$\mu_c$和$\sigma_c^2$的极大似然估计为

$$\hat{\mu}_c = \frac{1}{|D_c|}\sum_{x\in D_c} x \tag{4.17}$$

$$\hat{\sigma}_c^2 = \frac{1}{|D_c|}\sum_{x\in D_c}(x-\hat{\mu}_c)\,(x-\hat{\mu}_c)^T \tag{4.18}$$

也就是说，通过极大似然法得到的正态分布均值就是样本均值，方差就是$(x-\hat{\mu}_c)(x-\hat{\mu}_c)^T$的均值，这显然是一个符合直觉的结果。在离散属性情形下，也可通过类似的方式估计类条件概率。

需注意的是，这种参数化的方法虽能使类条件概率估计变得相对简单，但估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实数据分布。在现实应用中，欲做出能较好地接近潜在真实分布的假设，往往需在一定程度上利用关于应用任务本身的经验知识，否则若仅凭“猜测”来假设概率分布形式，很可能产生误导性的结果。

2. 朴素贝叶斯分类器

基于贝叶斯定理，$P(c|x)$可写成

$$\mathrm{P}(c|x)=\frac{\mathrm{P}(c)\mathrm{P}(x|c)}{\mathrm{P}(x)} \tag{4.19}$$

基于贝叶斯公式来估计后验概率 $\mathrm{P}(c|x)$ 的主要困难在于：类条件概率$\mathrm{P}(x|c)$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(naïve Bayes classifier)采用了“属性条件独立性假设”(attribute conditional independence assumption)：对已知类别，假设所有属性相互独立。换言之，假设每个属性独立地对分类结果发生影响。

基于属性条件独立性假设，式(4.19) 可重写为

$$\mathrm{P}(c|x)=\frac{\mathrm{P}(c)\mathrm{P}(x|c)}{\mathrm{P}(x)}=\frac{\mathrm{P}(c)}{\mathrm{P}(x)}\prod_{i=1}^{d}\mathrm{P}(x_i|c) \tag{4.20}$$

其中 d 为属性数目，x_i为x在第i个属性上的取值。

由于对所有类别来说$\mathrm{P}(x)$相同，因此基于贝叶斯判定准则有

$$h_{nb}(x)=arg\max_{c\in y}\mathrm{P}(c)\prod_{i=1}^{d}\mathrm{P}(x_i|c) \tag{4.21}$$

这就是朴素贝叶斯分类器的表达式。

显然，朴素贝叶斯分类器的训练过程就是基于训练集 D 来估计类先验概率$\mathrm{P}(c)$，并为每个属性估计条件概率$\mathrm{P}(x_i|c)$。

令D_c表示训练集 D 中第 c 类样本组成的集合，若有充足的独立同分布样本，则可容易地估计出类先验概率

$$\mathrm{P}(c)=\frac{|D_c|}{|\mathrm{D}|} \tag{4.22}$$

对离散属性而言，令$D_{c_i x_i}$表示D_c中在第i个属性上取值为x_i的样本组成的集合，则条件概率$\mathrm{P}(x_i|c)$可估计为

$$\mathrm{P}(x_i|c)=\frac{|D_{c_i x_i}|}{|D_c|} \tag{4.23}$$

对连续属性可考虑概率密度函数，假定$\mathrm{P}(x_i|c)\sim\aleph(\mu_{c,i},\sigma_{c,i}^2)$，其中$\mu_{c,i}$和$\sigma_{c,i}^2$分别是第 c 类样本在第$i$个属性上取值的均值和方差，则有

$$\mathrm{P}(x_i|c)=\frac{1}{\sqrt{2\pi}\sigma_{c,i}}exp\left(-\frac{(x_i-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \tag{4.24}$$

下面，我们用西瓜数据集（表 4.2）训练一个朴素贝叶斯分类器，对测试例 “测试 1”进行分类：

表 4.2 西瓜数据集

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖度	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是

2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	稍晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

编号	**色泽**	**根蒂**	**敲声**	**纹理**	**脐部**	**触感**	**密度**	**含糖度**	**好瓜**
测 1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	?

首先估计类先验概率$\mathrm{P}(c)$，显然有

$$\mathrm{P}(\text{好瓜} = \text{是}) = \frac{8}{17} \approx 0.471$$

$$\mathrm{P}(\text{好瓜} = \text{否}) = \frac{9}{17} \approx 0.529$$

然后，为每个属性估计条件概率$\mathrm{P}(x_i|c)$：

$$\mathrm{P}_{\text{青绿}|\text{是}} = \mathrm{P}(\text{色泽} = \text{青绿}|\text{好瓜} = \text{是}) = \frac{3}{8} = 0.375$$

$$\mathrm{P}_{\text{青绿}|\text{否}} = \mathrm{P}(\text{色泽} = \text{青绿}|\text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333$$

$$\mathrm{P}_{\text{蜷缩}|\text{是}} = \mathrm{P}(\text{根蒂} = \text{蜷缩}|\text{好瓜} = \text{是}) = \frac{5}{8} = 0.625$$

$$\mathrm{P}_{\text{蜷缩}|\text{否}} = \mathrm{P}(\text{根蒂} = \text{蜷缩}|\text{好瓜} = \text{否}) = \frac{3}{9} = 0.333$$

$$\mathrm{P}_{\text{浊响}|\text{是}} = \mathrm{P}(\text{敲声} = \text{浊响}|\text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$\mathrm{P}_{\text{浊响}|\text{否}} = \mathrm{P}(\text{敲声} = \text{浊响}|\text{好瓜} = \text{否}) = \frac{4}{9} \approx 0.444$$

$$\mathrm{P}_{\text{清晰}|\text{是}} = \mathrm{P}(\text{纹理} = \text{清晰}|\text{好瓜} = \text{是}) = \frac{7}{8} = 0.875$$

$$\mathrm{P}_{\text{清晰}|\text{否}} = \mathrm{P}(\text{纹理} = \text{清晰}|\text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222$$

$$P_{凹陷|是} = P(脐部 = 凹陷 | 好瓜 = 是) = \frac{6}{8} = 0.750$$

$$P_{凹陷|否} = P(脐部 = 凹陷 | 好瓜 = 否) = \frac{2}{9} \approx 0.222$$

$$P_{硬滑|是} = P(触感 = 硬滑 | 好瓜 = 是) = \frac{6}{8} = 0.750$$

$$P_{硬滑|否} = P(触感 = 硬滑 | 好瓜 = 否) = \frac{6}{9} = 0.667$$

$$P_{密度：0.697|是} = P(密度 = 0.697 | 好瓜 = 是)$$
$$= \frac{1}{\sqrt{2\pi} \times 0.129} exp\left(-\frac{(0.697 - 0.574)^2}{2 \times 0.129^2}\right) \approx 1.959$$

$$P_{密度：0.697|否} = P(密度 = 0.697 | 好瓜 = 否)$$
$$= \frac{1}{\sqrt{2\pi} \times 0.195} exp\left(-\frac{(0.697 - 0.496)^2}{2 \times 0.195^2}\right) \approx 1.203$$

$$P_{含糖：0.460|是} = P(含糖 = 0.460 | 好瓜 = 是)$$
$$= \frac{1}{\sqrt{2\pi} \times 0.101} exp\left(-\frac{(0.460 - 0.279)^2}{2 \times 0.101^2}\right) \approx 0.788$$

$$P_{含糖：0.460|否} = P(含糖 = 0.460 | 好瓜 = 否)$$
$$= \frac{1}{\sqrt{2\pi} \times 0.108} exp\left(-\frac{(0.460 - 0.154)^2}{2 \times 0.108^2}\right) \approx 0.066$$

于是，有

$$P(好瓜 = 是) \times P_{青绿|是} \times P_{蜷缩|是} \times P_{浊响|是} \times P_{清晰|是} \times P_{凹陷|是}$$
$$\times P_{硬滑|是} \times P_{密度：0.697|是} \times P_{含糖：0.460|是} \approx 0.038$$
$$P(好瓜 = 否) \times P_{青绿|否} \times P_{蜷缩|否} \times P_{浊响|否} \times P_{清晰|否} \times P_{凹陷|否}$$
$$\times P_{硬滑|否} \times P_{密度：0.697|否} \times P_{含糖：0.460|否} \approx 6.80 \times 10^{-5}$$

由于 $0.038 > 6.80 \times 10^{-5}$，因此，朴素贝叶斯分类器将测试样本“测试 1”判别为“好瓜”。

需注意，若某个属性值在训练集中没有与某个类同时出现过，则直接基于式(4.23)进行概率估计，再根据式(4.21) 进行判别将出现问题。例如，在使用西瓜数据集训练朴素贝叶斯分类器时，对一个“敲声=清脆”的测试例，有

$$P_{清脆|是} = P(敲声 = 清脆 | 好瓜 = 是) = \frac{0}{8} = 0$$

由于式(4.21)的连乘式计算出的概率值为零，因此，无论该样本的其他属性是什么，哪怕在其他属性上明显像好瓜，分类的结果都将是“好瓜=否”，这显然不太合理。

为了避免其他属性携带的信息被训练集中未出现的属性值“抹去”，在估计概率值时通常要进行“平滑”(smoothing)，常用“拉普拉斯修正”(Laplacian correction)。具体来说，令 N 表示训练集 D 中可能的类别数，N_i表示第 i 个属性可能的取值数，则式(4.22)和(4.23)分别修正为

$$\hat{P}(c) = \frac{|D_c|+1}{|D|+N} \tag{4.25}$$

$$\hat{P}(x_i|c) = \frac{|D_{c,x_i}|+1}{|D_c|+N_i} \tag{4.26}$$

例如，在本节的例子中，类先验概率可估计为

$$\hat{P}(\text{好瓜} = \text{是}) = \frac{8+1}{17+2} = 0.474$$

$$\hat{P}(\text{好瓜} = \text{否}) = \frac{9+1}{17+2} = 0.526$$

类似地，$P_{\text{青绿|是}}$和$P_{\text{青绿|否}}$可估计为

$$\hat{P}_{\text{青绿|是}} = \hat{P}(\text{色泽} = \text{青绿}|\text{好瓜} = \text{是}) = \frac{3+1}{8+3} \approx 0.364$$

$$\hat{P}_{\text{青绿|否}} = \hat{P}(\text{色泽} = \text{青绿}|\text{好瓜} = \text{否}) = \frac{3+1}{9+3} \approx 0.333$$

同时，上文提到的概率$P_{\text{清脆|是}}$可估计为

$$\hat{P}_{\text{清脆|是}} = \hat{P}(\text{敲声} = \text{清脆}|\text{好瓜} = \text{是}) = \frac{0+1}{8+3} \approx 0.091$$

显然，拉普拉斯修正避免了因训练集样本不充分而导致概率估值为零的问题，并且在训练集变大时，修正过程所引入的先验(prior)的影响也会逐渐变得可忽略，使得估值渐趋向于实际概率值。

在现实任务中朴素贝叶斯分类器有多种使用方式。例如，若任务对预测速度要求较高，则对给定训练集，可将朴素贝叶斯分类器涉及的所有概率估值事先计算好存储起来，这样在进行预测时只需“查表”即可进行判别；若任务数据更替频繁，则可采用 “懒惰学习”(lazy learning)方式，先不进行任何训练，待收到预测请求时再根据当前数据集进行概率估值；若数据不断增加，则可在现有估值基础上，仅对新增样本的属性值所涉及的概率估值进行计数修正即可实现增量学习。

4.2.3 聚类

1. 聚类度量

在"无监督学习" (unsupervised learning) 中，训练样本的标记信息是未知的，目标是通过对无标记训练样本的学习来揭示数据的内在性质及规律，为进一步的数据分析提供基础。此类学习任务中研究最多、应用最广的是"聚类" (clustering)。聚类试图将数据集中的样本划分为若干个通常是不相交的子集，每个子集称为一个"簇" (cluster)。 通过这样的划分，每个簇可能对应于一些潜在的概念(类别) ，如"浅色瓜" "深色瓜"，"有籽瓜" "无籽瓜"，甚至"本地瓜""外地瓜"等;需说明的是，这些概念对聚类算法而言事先是未知的，聚类过程仅能自动形成簇结构，簇所对应的概念语义需由使用者来把握和命名。

形式化地说，假定样本集 $D = \{x_1, x_2, \ldots, x_m\}$ 包含 m 个无标记样本，每个样本 $x_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$ 是一个 n 维特征向量，则聚类算法将样本集 D 划分为 k 个不相交的簇 $\{C_l | l = 1, 2, \ldots, k\}$ 时，其中 $C_{l'} \cap_{l' \neq l} C_l = \emptyset$且$D = \cup_{l=1}^{k} C_l$ 。相应地，我们用 $\lambda_j \in \{1,2,\ldots,k\}$ 表示样本的"簇标记" (cluster label)，即 $x_j \in C_{\lambda_j}$。于是，聚类的结果可用包含 m 个元素的簇标记向量 $\lambda = \{\lambda_1, \lambda_2,$

…，λ_m } 表示。

基于不同的学习策略，人们设计出多种类型的聚类算法。本章后半部分将对不同类型的代表性算法进行介绍，但在此之前我们先讨论聚类算法涉及的两个基本问题，分别是性能度量和距离计算。

(1) 性能度量

聚类性能度量亦称聚类"有效性指标" (validity index). 与监督学习中的性能度量作用相似，对聚类结果，我们需通过某种性能度量来评估其好坏;另一方面，若明确了最终将要使用的性能度量，则可直接将其作为聚类过程的优化目标，从而更好地得到符合要求的聚类结果。常见的聚类性能度量外部指标有：

- Jaccard 系数(Jaccard Coefficient，简称 JC)

$$\mathrm{JC} = \frac{a}{a+b+c} \tag{4.27}$$

- FM 指数(Fowlkes and Mallows Index，简称 FMI)

$$\mathrm{FMI} = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}} \tag{4.28}$$

- Rand 指数(Rand Index，简称 RI)

$$\mathrm{RI} = \frac{2(a+d)}{m(m-1)} \tag{4.29}$$

显然，上述性能度量的结果值均在[0,1]区间，值越大越好。

(2) 距离计算

对函数 dist (，)，若它是一个"距离度量" (distance measure) ，则需满足一些基本性质:

- 非负性: $dist\ (x_i, x_j) \geq 0$；
- 同一性: $dist\ (x_i, x_j) = 0$ 当且仅当 $x_i = x_j$；
- 对称性：$dist\ (x_i, x_j) = dist\ (x_j, x_i)$；
- 直递性：$dist\ (x_i, x_j) \leq dist\ (x_i, x_k) + dist\ (x_k, x_j)$

给定样本 $x_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$ 与 $x_j = \{x_{j1}, x_{j2}, \ldots, x_{jm}\}$，最常用的是"闵可夫斯基距离" (Minkowski distance)。

当 $p = 2$ 时，闵可夫斯基距离即欧氏距离(Euclidean distance)：

$$dist_{ed}(x_i, x_j) = \|x_i - x_j\|_2 = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2} \tag{4.30}$$

当 $p = 1$ 时，闵可夫斯基距离即曼哈顿距离(Euclidean distance)：

$$dist_{man}(x_i, x_j) = \|x_i - x_j\|_1 = \sum_{u=1}^{n} |x_{iu} - x_{ju}| \tag{4.31}$$

2. 原型聚类

(1) K均值算法

给定样本集 $D = \{x_1, x_2, \ldots, x_m\}$，“k 均值”($k$-means)算法针对聚类所得簇划分 $C =$

$\{C_1, C_2, \ldots, C_k\}$ 最小化平方误差：

$$E = \sum_{i=1}^{k} \sum_{x \epsilon C_i} \|x - \mu_i\|_2^2 \tag{4.32}$$

其中 $\mu_i = \frac{1}{|C_i|} \sum_{x \epsilon C_i} x$ 是簇 C_i 的均值向量。直观来看，式(4.32)在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度， E 值越小则簇内样本相似度越高。最小化式(4.32)并不容易，找到它的最优解需考察样本集 D 所有可能的簇划分，这是一个 NP 难问题[Aloise et al., 2009]。 因此， k 均值算法采用了贪心策略，通过迭代优化来近似求解式(4.32)。

下面以表 4.2 的西瓜数据集 4.0 为例来演示 k 均值算法的学习过程。为方便叙述，我们将编号为 i 的样本称为 x_i ，这是一个包含"密度"与"含糖率"两个属性值的二维向量。

表 4.2 西瓜数据集 4.0

编号	密度	含糖率	编号	密度	含糖率	编号	密度	含糖率
1	0.697	0.460	11	0.245	0.0557	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

假定聚类簇数 k= 3 ，算法开始时随机选取三个样本 x_6, x_{12}, x_{24} 作为初始均值向量，即

$$\mu_1 = (0.403, 0.237), \quad \mu_2 = (0.343; 0.099), \quad \mu_3 = (0,478; 0.437)$$

考察样本 x_1 =(0.697; 0 .460)，它与当前均值向量μ_1，μ_2，μ_3 的距离分别为 0.369, 0.506, 0.220 ，因此 x_1 将被划入簇 C_3 中。类似的，对数据集中的所有样本考察一遍后，可得当前簇划分为：

$$C_1 = \{x_3, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{13}, x_{14}, x_{17}, x_{18}, x_{19}, x_{20}, x_{23}\}$$
$$C_2 = \{x_{11}, x_{12}, x_{16}\}$$
$$C_3 = \{x_1, x_2, x_4, x_{15}, x_{21}, x_{22}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29}, x_{30}\}$$

于是，可从 C_1 、C_2 、C_3 分别求出新的均值向量，

$$\mu_1' = (0.493; 0.207), \quad \mu_2' = (0.394; 0.066), \quad \mu_3' = (0.602; 0.396)$$

更新当前均值向量后，不断重复上述过程，如图 4.7 所示，第五轮迭代产生的结果与第四轮迭代相同，于是算法停止，得到最终的簇划分。

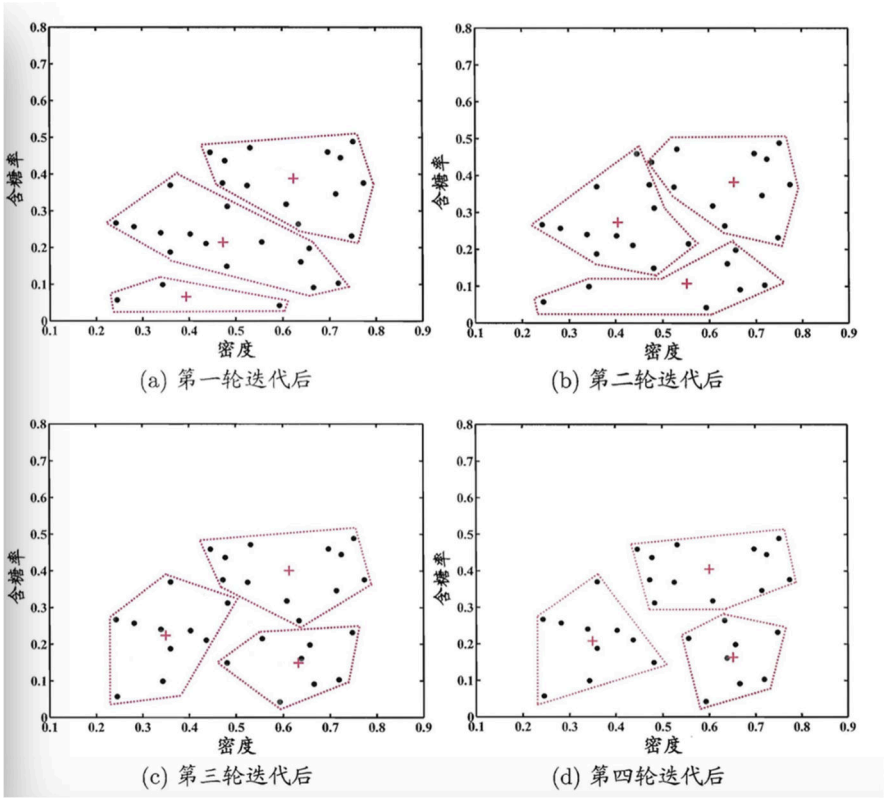

图 4.7 西瓜数据集 4.0 上 k 均值算法($k=3$)在各轮迭代后的结果. 样本点与均值向量分别用 "•" 与 "+" 表示，虚线显示出簇的划分.

(2) 学习向量量化

学习矢量量化是一种结构简单、功能强大的有监督式神经网络分类方法。作为一种最近邻原型分类器，LVQ 在训练过程中通过对神经元权向量（原型向量）的不断更新，对其学习率的不断调整，能够使不同类别权向量之间的边界逐步收敛至贝叶斯分类边界。算法中，对获胜神经元（最近邻权向量）的选取是通过计算输入样本和权向量之间的距离的大小来判断的。与矢量量化（VQ）相比，LVQ 最突出的特点就是其具有自适应学习性。

与普通聚类算法类似(K 均值)，学习向量量化（Learning Vector Quantization，简称 LVQ）也是试图找到一组原型向量来刻画聚类结构，但与一般聚类算法不同的是，LVQ 假设数据样本带有类别标记，学习过程中利用这些标签信息来辅助聚类。

LVQ1：给定样本集 $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_{\mathrm{m}},y_{\mathrm{m}})\}$，每个样本 x_j 是由 n 个属性描述的特征向量 $(x_{j1},x_{j2},\ldots,x_{j\mathrm{n}})$，$y_{\mathrm{i}}\in Y$是样本 x_j 的类别标记。LVQ 的目标是学得一组 n 维原型向量$\{p_1,p_2,\ldots,p_{\mathrm{q}}\}$，每个原型向量代表一个聚类簇，各原型向量预设的类别标记 $\{t_1,t_2,\ldots,t_q\}$，簇标记 $t_i\in Y$。LVQ 算法描述如下所示。

输入：训练集$D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$；
原型向量个数 q，各原型向量预设的类别标记$\{t_1,t_2,\cdots,t_q\}$
学习率$\eta\in(0,1)$

过程：

1：初始化一组原型向量$\{p_1,p_2,\cdots,p_q\}$

2：repeat

3：从样本集D中随机选取样本(x_i, y_i)

4：计算样本x_j与$p_i(1 \le 1 \le q)$的距离：$d_{ji} = ||x_j - p_i||^2$

5：找出与x_j距离最近的原型向量p_i^*，$i^* = argmin_{i\in\{1,2,\cdots,q\}} d_{ji}$

6：if $y_j = t_i^*$ then

7：　　$p' = p_i^* + \eta \cdot (x_j - p_i^*)$

8：else

9：　　$p' = p_i^* - \eta \cdot (x_j - p_i^*)$

10：end if

11：将原型向量p_i^*更新为p'

12：until 满足停止条件

输出：原型向量$\{p_1, p_2, \cdots, p_q\}$

算法第 2~12 行对原型向量(迁移优化中的个体)进行迭代优化。在每一轮迭代中，算法随机选取一个有标记的训练样本，找出与其距离最近的原型向量，并根据两者的类别标记是否一致来对原型向量进行相应的更新。在第 12 行中，若算法的停止条件已满足（已经达到最大迭代次数或原型向量更新很小甚至不再更新），则将当前原型向量作为最终结果返回。

LVQ 的关键是第 6~10 行，即如何更新原型向量。直观上看，对样本 x_j，若最近的原型向量量 p_i*与 x_j 的类别标记相同，则令 p_i*向 x_j 的方向靠拢；此时新原型向量为：

$$p' = p_i* + \eta \cdot (x_j - p_i*)$$

p'与 x_j之间的距离为:

$$||p' - x_j||2 = ||p_i* + \eta \cdot (x_j - p_i*) - x_j||^2 = (1-\eta) \cdot ||p_i* - x_j||^2 \tag{4.33}$$

令学习率$\eta \in (0,1)$，则原型向量 p_i*在更新为 p'之后将更接近 x_j。类似的，若p_i*与 x_j的类别标记不同，则更新之后的原型向量与 x_j 之间的距离将增大为$(1+\eta) \cdot ||p_i* - x_j||^2$，从而更远离 x_j。

在学得一组原型向量$\{p_1, p_2, \ldots, p_q\}$后，即可实现对样本空间 χ 的簇划分，对任意样本 x，它将被划入与其距离最近的原型向量所代表的簇中。换言之，每个原型向量 p_i 定义了与之相关的一个区域 R_i，该区域中每个样本与 p_i 的距离不大于它与其他原型向量 $p'(i' \neq i)$ 的距离，即：

$$R_i = \{x \in \chi \mid ||x - p_i||^2 \le ||x - p_i'||^2, i' \neq i\} \tag{4.34}$$

由此形成了对样本空间 χ 的簇划分$\{R_1, R_2, \ldots, R_q\}$,称为 Voronoi 剖分(Voronoi tessellation)。

LVQ2：每一次迭代过程只能对一个权向量进行更新可能是 LVQ1 算法的一个局限。因此，在 LVQ2 中，提出同时更新两个权向量的方法。寻找输入样本最近邻的两个权向量，若这两个权向量一个与输入样本同类(p')，而另一个与输入样本异类(p'')，则可以同时更新两个权向量：

$$p' = p_i* + \eta \cdot (x_j - p_i*) \tag{4.35}$$

$$p'' = p_i* - \eta \cdot (x_j - p_i*) \tag{4.36}$$

(3) 高斯混合聚类

高斯混合模型（GMM）做聚类首先假设数据点是呈高斯分布的，相对应 K-Means 假设数据点是圆形的，高斯分布的（椭圆形）给出了更多的可能性。此时有两个参数来描述簇的形状：均值和标准差。因为

在 x，y 方向上都有标准差。因此，高斯混合模型每个高斯分布被分配给单个簇，这些簇可以采取任何形状的椭圆形。

给定一批规模为 N 数据集 S_N ,假设有一个特殊的高斯混合模型 $\mathcal{M}_k$ 的分量个数正好等同于数据集 S_N 的规模大小，则模型 $\mathcal{M}_k$ 的参数集合 $\Theta_k = \{\Theta_{i|k},\ i = 1, \dots, N\}$，其中$\Theta_{i|k}$表示 $\mathcal{M}_k$ 第 i 个分量的参数(即密度函数的均值和协方差矩阵)。因此，与集群簇 y_k 相关的 GMM 概率函数的输出是 N 个分量密度的加权和为：

$$p(x|y_k) = \sum_i^N P(\Theta_{i|k}|y_k)\, p\left(x\middle|y_k, \Theta_{i|k}\right) \tag{4.37}$$

其中$P(\theta_{i|k}|y_k)$是给定簇 y_k 属于第 i 个高斯分量的先验概率 μ_{ik}，且$P\left(x\middle|y_k, \Theta_{i|k}\right) = \mathcal{N}_k\left(x; \Theta_{i|k}\right)$是模型第 i 个高斯分量的正态密度函数。为了保证 $P(x|y_k)$ 是密度函数，文章假设混合高斯函数的参数 μ_{ik} 满足 $\sum_i \mu_{ik} = 1$。此时 GMMs 的训练可以表述为一个极大似然问题，参数 Θ_k 和混合参数 μ_{ik} 可使用 EM(期望最大化)方法进行估计。

EM 过程：给定参数先验，根据反馈计算后验，将其作为下一次预测的先验，然后再根据反馈计算后验，如此迭代下去。

(E-step): 由于后验分布(离散截断高斯)和先验分布(高斯分布)不同，参数估计时可以求解一个近似的后验概率分布去逼近真实的后验概率分布。假设真实后验概率分布为：$P(Z|X)$，如式 (4.38) 所示，我们希望用一个近似后验概率分布$Q(Z)$去逼近$P(Z|X)$，在第 t 次迭代中，对于每个 i ，令：

$$Q_t^i\left(z^{(i)}\right) = P(z^{(i)}|x^{(i)}; \theta_{t-1}) \tag{4.38}$$

假设我们现在随机初始化了θ_0，进入第一轮迭代，有：

$$Q_1^i\left(z^{(i)}\right) = P\left(z^{(i)}\middle|x^{(i)}; \theta_0\right)$$

$$= \frac{P\left(x^{(i)}\middle|z^{(i)}; \theta_0\right) P(z^{(i)}; \theta_0)}{\sum_{z^{(i)}} P\left(x^{(i)}\middle|z^{(i)}; \theta_0\right) P(z^{(i)}; \theta_0)} \tag{4.39}$$

(M-step)：为了描述近似的后验概率分布$Q(Z)$和真实后验概率分布$P(Z|X)$的逼近程度，文章引入了 KL 散度(Kullback-Leibler Divergence)，如式(4.40)所示，在概率论或信息论中又称为相对熵(relative entropy)来描述两个概率分布P和Q的逼近程度，相对熵越小，分布越相似。

$$KL(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)} \tag{4.40}$$

在第t次迭代中，为了更新迭代的θ值，我们用(M-step)来进行优化来逐渐逼近极大值点，令：

$$\theta_t = arg\, \max_{\theta} \sum_{i=0}^m \sum_{z^{(i)}} Q_t^i\left(z^{(i)}\right) log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_t^i(z^{(i)})} \tag{4.41}$$

假设我们现在随机初始化了θ_0，进入第一轮迭代，有：

$$\theta_1 = arg\, \max_{\theta} \sum_{i=0}^m \sum_{z^{(i)}} Q_1^i\left(z^{(i)}\right) log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_1^i(z^{(i)})} \tag{4.42}$$

考虑到 $Q_1^i(z^{(i)})$ 是一组常数，我们可以舍弃常数项，进一步简化上面这个要极大化的函数，有

$$\theta_1 = arg\max_{\theta} \sum_{i=0}^{m} \sum_{z^{(i)}} Q_1^i\left(z^{(i)}\right) log \frac{P\left(x^{(i)}, z^{(i)}; \theta\right)}{Q_1^i\left(z^{(i)}\right)}$$
$$= arg\max_{\theta} \sum_{i=0}^{m} \sum_{z^{(i)}} Q_1^i\left(z^{(i)}\right) logP\left(x^{(i)}, z^{(i)}; \theta\right) \quad (4.43)$$

总的来说，整个 EM 过程就是首先初始化模型的参数，基于这个参数对每一个隐变量进行分类，有了隐变量的观测值之后，原来含有隐变量的模型变成了不含隐变量的模型。接下来便可以直接使用极大似然估计来更新模型的参数，再基于新的参数开始新一轮的迭代，直到参数收敛。

3. 密度聚类

密度聚类亦称"基于密度的聚类" (density-based clustering) ，此类算法假设聚类结构能通过样本分布的紧密程度确定。通常情形下，密度聚类算法从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇以获得最终的聚类结果。DBSCAN 是一种著名的密度聚类算法，它基于一组"邻域" (neighborhood) 参数 $(\epsilon, MinPts)$ 来刻画样本分布的紧密程度。给定数据集$D = \{x_1, x_2, \ldots, x_m\}$，定义下面这几个概念：

- ϵ-邻域:对 $x_j \in D$，其 ϵ-邻域包含样本集 D 中与 x_j 的距离不大于 ϵ 的样本，即 $N_\epsilon(x_j) = \{ x_i \in D | dist(x_i, x_j) \leq \epsilon \}$；
- 核心对象(core object)： 若 x_j 的 ϵ -邻域至少包含 *MinPts* 个样本，即$\left|N_\epsilon(x_j)\right| \geq MinPts$，则 x_j 是一个核心对象；
- 密度直达(directly density-reachable)：若 x_j 位于 x_i 的 ϵ -邻域中，且 x_i 是核心对象，则称 x_j 由 x_i 密度直达；
- 密度可达(density-reachable)：对 x_i 与 x_j ，若存在样本序列 $p_1, p_2, \ldots, p_n$，酌，其中 $p_1 = x_i$ ，$p_n = x_j$ 且 p_{i+1} 由 p_i 密度直达，则称 x_j 由 x_i 密度可达；
- 密度相连(density-connected)：对 x_i 与 x_j，若存在 x_k 使得 x_i 与 x_j 均由 x_k 密度可达，则称x_i 与 x_j 密度相连。图4.8 给出了上述概念的直观显示。

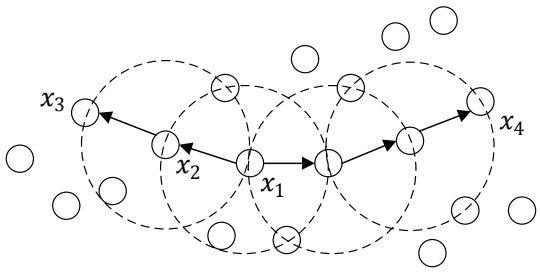

图 4.8 模型复杂度与训练精度和测试精度之间的权衡 DBSCAN 定义的基本概念(MinPts = 3): 虚线显示出ϵ-邻域， x_1 是核心对象，x_2 由 x_1 密度直达， x_3 由 x_1 密度可达，x_3 与 x_4 密度相连。

基于这些概念， DBSCAN 将"簇"定义为：由密度可达关系导出的最大的密度相连样本集合。形式化地说，给定邻域参数(ϵ ，*MinPts*) ， 簇 $C \subseteq D$ 是满足以下性质的非空样本子集：

- 连接性(connectivity)： $x_i \in C$， $x_j \in C \Rightarrow x_i$ 与 x_j 密度相连.

- 最大性(maximality)：$x_i \in C$， x_j 由 x_i 密度可达⇒ $x_j \in C$.

以西瓜数据集 4.0 为例，假定邻域参数 (ϵ ，*MinPts*) 设置为 ε =0.11， *MinPts*=5。 DBSCAN 算法先找出各样本的ϵ-邻域并确定核心对象集合：$\Omega = \{x_3, x_5, x_6, x_8, x_9, x_{13}, x_{14}, x_{18}, x_{19}, x_{24}, x_{25}, x_{28}, x_{29}\}$。 然后，从 Ω 中随机选取一个核心对象作为种子，找出由它密度可达的所有样本，这就构成了第一个聚类簇。假定核心对象 x_8 被选中作为种子，则 DBSCAN 生成的第一个聚类簇为：

$$C_1 = \{x_6, x_7, x_8, x_{10}, x_{12}, x_{18}, x_{19}, x_{20}, x_{23}\}$$

然后，DBSCAN 将 C_1 中包含的核心对象从 Ω 中去除: $\Omega = \Omega \backslash C_1 = \{x_3, x_5, x_9, x_{13}, x_{14}, x_{24}, x_{25}, x_{28}, x_{29}\}$。再从更新后的集合 Ω 中随机选取一个核心对象作为种子来生成下一个聚类簇。上述过程不断重复，直至 Ω 为空。图 4.9 显示出 DBSCAN 先后生成聚类簇的情况。C_1 之后生成的聚类簇为：

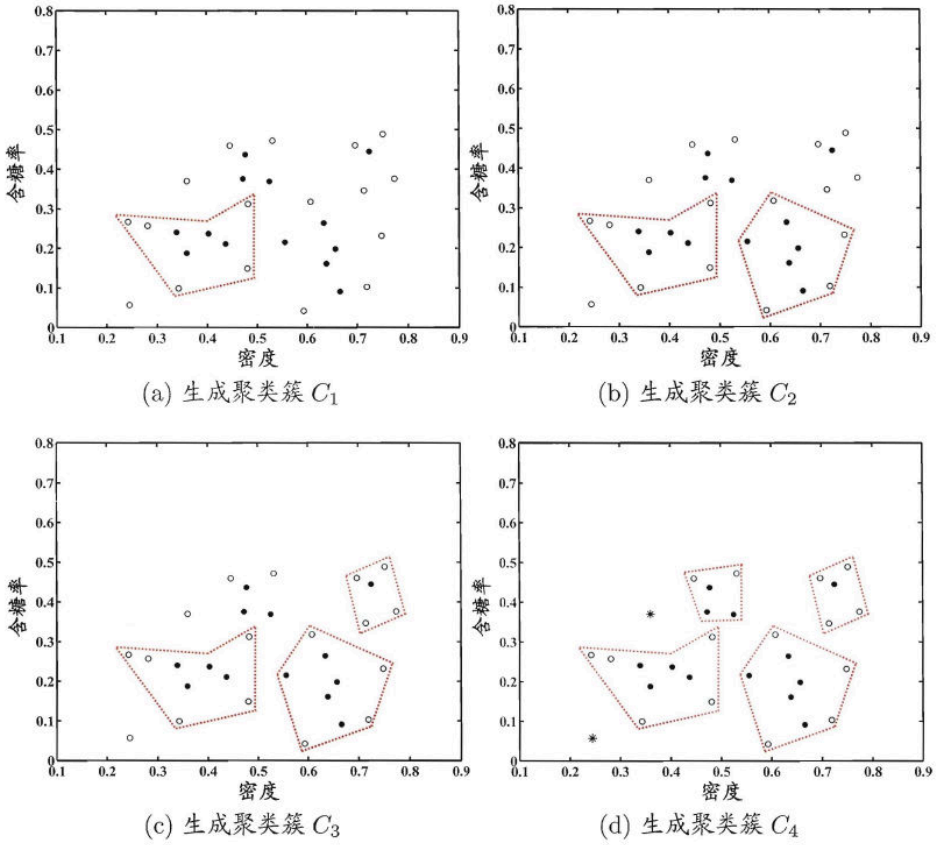

图 4.9 DBSCAN 算法(ϵ= 0.11, *MinPts* = 5)生成聚类簇的先后情况. 核心对象、非核心对象、噪声样本分别用" • " "。" "*"来表示，虚线显示出簇的划分.

$$C_1 = \{x_3, x_4, x_5, x_9, x_{13}, x_{14}, x_{16}, x_{17}, x_{21}\}$$
$$C_1 = \{x_1, x_2, x_{22}, x_{26}, x_{29}\}$$
$$C_1 = \{x_{24}, x_{25}, x_{27}, x_{28}, x_{30}\}$$

4. 层次聚类

层次聚类(hierarchical clustering)试图在不同层次对数据集进行划分，从而形成树形的聚类结构。数据集

的划分可采用"自底向上"的聚合策略，也可采用"自顶向下"的分拆策略。

AGNES 是一种采用自底向上聚合策略的层次聚类算法。它先将数据集中的每个样本看作一个初始聚类簇，然后在算法运行的每一步中找出距离最近的两个聚类簇进行合并，该过程不断重复，直至达到预设的聚类簇个数。这里的关键是如何计算聚类簇之间的距离。实际上，每个簇是一个样本集合，因此，只需采用关于集合的某种距离即可。例如，给定聚类簇 C_i 与 C_j，可通过下面的式子来计算距离:

$$\text{最小距离：} d_{\min}(C_i,C_j) = \min_{x\epsilon C_i, z\epsilon C_j} dist(x,z) \tag{4.44}$$

$$\text{最大距离：} d_{\max}(C_i,C_j) = \min_{x\epsilon C_i, z\epsilon C_j} dist(x,z) \tag{4.45}$$

$$\text{平均距离：} d_{avg}(C_i,C_j) = \frac{1}{|C_i||C_j|}\sum_{x\epsilon C_i}\sum_{z\epsilon C_j} dist(x,z) \tag{4.46}$$

显然，最小距离由两个簇的最近样本决定，最大距离由两个簇的最远样本决定，而平均距离则由两个簇的所有样本共同决定。当聚类簇距离由 d_{min}、d_{max} 或 d_{avg} 计算时，AGNES 算法被相应地称为“单链接”(single-linkage) 、"全链接" (complete-linkage) 或"均链接" (average-linkage) 算法。

以西瓜数据集 4.0 为例，令 AGNES 算法一直执行到所有样本出现在同一个簇中，即 $k=1$，则可得到图 4.10 所示的"树状国" (dendrogram) ，其中每层链接一组聚类簇。

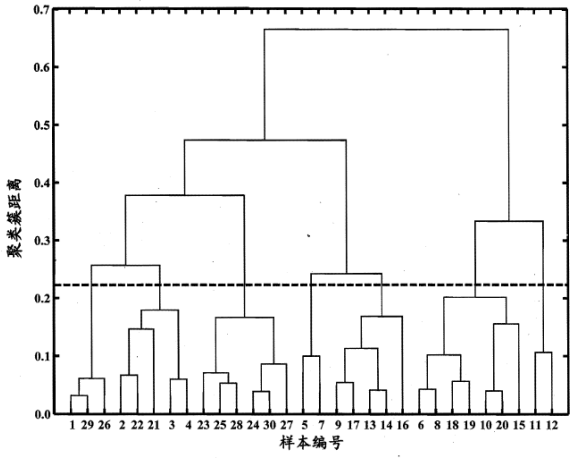

图 4.10 西瓜数据集4.0上AGNES 算法生成的树状图{采用d_{max}). 横轴对应于样本编号 ，纵轴对应于聚类簇距离.

在树状图的特定层次上进行分割，则可得到相应的簇划分结果。例如，以图 4.10 中所示虚线分割树状图, 将得到包含 7 个聚类簇的结果:

$$C_1 = \{x_1, x_{26}, x_{29}\};\ C_2 = \{x_2, x_3, x_4, x_{21}, x_{22}\}$$
$$C_3 = \{x_{23}, x_{24}, x_{25}, x_{27}, x_{28}, x_{30}\};C_4 = \{x_5, x_7\};$$
$$C_5 = \{x_9, x_{13}, x_{14}, x_{16}, x_{17}\};C_6 = \{x_6, x_8, x_{10}, x_{15}, x_{18}, x_{19}, x_{20}\};$$
$$C_7 = \{x_{11}, x_{12}\}$$

将分割层逐步提升，则可得到聚类簇逐渐减少的聚类结果。例如图 4.11 显示出了从图 4.10 中产生 7 至 4 个聚类簇的划分结果。

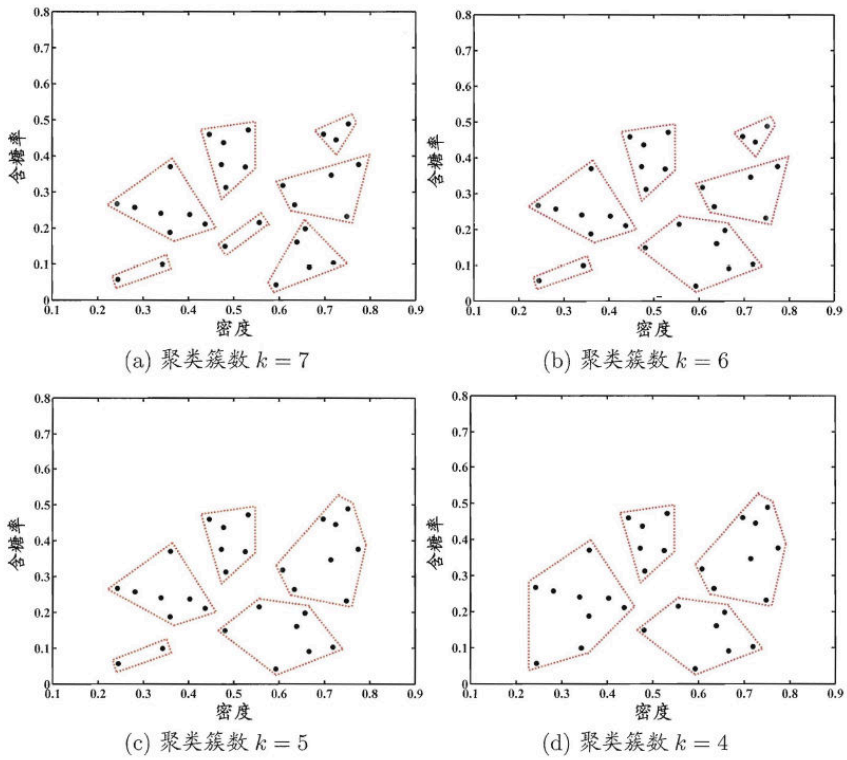

图 4.11 西瓜数据集4.0 上AGNES 算法(采用d_{max})在不同聚类簇数($k=7, 6, 5, 4$)时的簇划分结果.样本点用"•"表示，虚线显示出簇划分.

4.2.4 特征降维

始终贯穿本书的一个难题就是对数据和结果的展示，这是因为这本书只是二维的，而在通常的情况下我们的数据不是如此。有时我们会显示三维图像或者只显示其相关特征，但是数据往往拥有超出显示能力的更多特征。数据显示并非大规模特征下的唯一难题，对数据进行简化还有如下一系列的原因：

1）使得数据集更易使用；

2）降低很多算法的计算开销；

3）去除噪声；

4）使得结果易懂。

在已标注与未标注的数据上都有降维技术。这里我们将主要关注未标注数据上的降维技术，该技术同时也可以应用于已标注的数据。

第一种降维的方法称为主成分分析(Principal Component Analysis，PCA)。在 PCA 中，数据从原来的坐标系转换到了新的坐标系，新坐标系的选择是由数据本身决定的。第一个新坐标轴选择的是原始数据中方差最大的方向，第二个新坐标轴的选择和第一个坐标轴正交且具有最大方差的方向。该过程一直重复、重复次数为原始数据中特征的数目。我们会发现，大部分方差都包含在最前面的几个新坐标轴中。因此，我们可以忽略余下的坐标轴，即对数据进行了降维处理。

另外一种降维技术是因子分析(Factor Analysis)。在因子分析中，我们假设在观察数据的生成中有一些观察不到的隐变量(latent variable)。假设观察数据是这些隐变量和某些噪声的线性组合。那么隐变量的数据可能比观察数据的数目少，也就是说通过找到隐变量就可以实现数据的降维。因子分析已经应用于社会科学、金融和其他领域中了。

还有一种降维技术就是独立成分分析(Independent Component Analysis，ICA)。ICA 假设数据是从 N 个数据源生成的，这一点和因子分析有些类似。假设数据为多个数据源的混合观察结果，这些数据源之间在

统计上是相互独立的，而在 PCA 中只假设数据是不相关的。同因子分析一样，如果数据源的数目少于观察数据的数目，则可以实现降维过程。

在上述 3 种降维技术中，PCA 的应用目前最为广泛，因此本节主要关注 PCA。下面我们将对 PCA 进行介绍，然后再通过一段 Python 代码来运行 PCA。

1. 在 NumPy 中实现 PCA

将数据转换成前 N 个主成分的伪代码大致如下：

1）去除平均值

2）计算协方差矩阵

3）计算协方差矩阵的特征值和特征向量

4）将特征值从大到小排序

5）保留最上面的 N 个特征向量

6）将数据转换到上述 N 个特征向量构建的新空间中

建立一个名为 pca.py（见附件）的文件并将下列代码加人用于计算 PCA。

```
From numpy import *
def loadDataset(fileName,delim = '\t'):
fr = open(fileName)
stringArr = [line.strip().split(delim) for line in fr.readlines()]
datArr=[map(float,line) for line in stringArr]
return mat(datArr)
def pca(dataMat, topNfeat=9999999):
meanVals = mean(dataMat,axis=0)
meanRemoved = dataMat-meanVals
covMat = cov(meanRemoved,rowvar=0)
eigVals,eigVects = linalg.eig(mat(covMat))
eigValInd=argsort(eigVals)
eigValInd= eigValInd[:-(topNfeat+1):-1]
redEigVects=eigVects[:,eigValInd]
lowDDataMat=meanRemoved*redEigVects
reconMat=(lowDDataMat*redEigVects.T)+meanVals
return lowDDataMat, reconMat
```

上述代码包含了通常的 NumPy 导入和 1oadDataSet()函数。这里的 1oadDataSet()函数使用了两个 list comprehension 来构建矩阵。

pca()函数有两个参数：第一个参数是用于进行 PCA 操作的数据集，第二个参数 topNfeat 则是一个可选参数，即应用的 N 个特征。如果不指定 topNfeat 的值，那么函数就会返回前 9999999 个特征，或者原始数据中全部的特征。

首先计算并减去原始数据集的平均值。然后，计算协方差矩阵及其特征值，接着利用 argsort()函数对特征值进行从小到大的排序。根据特征值排序结果的逆序就可以得到 topNfeat 个最大的特征向量。这些特征向量将构成后面对数据进行转换的矩阵，该矩阵则利用 N 个特征将原始数据转换到新空间中。最后，原始数据被重构后返回用于调试，同时降维之后的数据集也被返回了。

一切都看上去不错，是不是？在进入规模更大的例子之前，我们先看看上面代码的运行效果以确保其

结果正确无误。

```
import pca
```

我们在 testSet.txt 文件中加入一个由 1000 个数据点组成的数据集，并通过如下命令将该数据集调入内存：

```
dataMat = pca.loadDataSet(' testSet.txt')
```

于是，我们就可以在该数据集上进行 PCA 操作：

```
lowDMat, reconMat = pca.pca(dataMat, 1)
```

lowDMat 包含了降维之后的矩阵，这里是个一维矩阵，我们通过如下命令进行检查：

```
shape (lowDMat)
 ‘’’
运行以上程序可以得到输出:
(1000, 1)
```

我们可以通过如下命令将降维后的数据和原始数据一起绘制出来：

```
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(dataMat[:0].flatten().A[0],dataMat[:1].flatten().A[0],marker='^',s=90)
ax.scatter(reconMat[:0].flatten().A[0],reconMat[:1].flatten().A[0],marker='o',s=50,c='red')
```

我们应该会看到和图 4.12 类似的结果。使用如下命令来替换原来的 PCA 调用，并重复上述过程：

```
lowDMat, recoMat = pca.pca(dataMat,2)
```

既然没有剔除任何特征，那么重构之后的数据会和原始的数据重合。我们也会看到和图 4.12 类似的结果(不包含图 4.12 中的直线)。

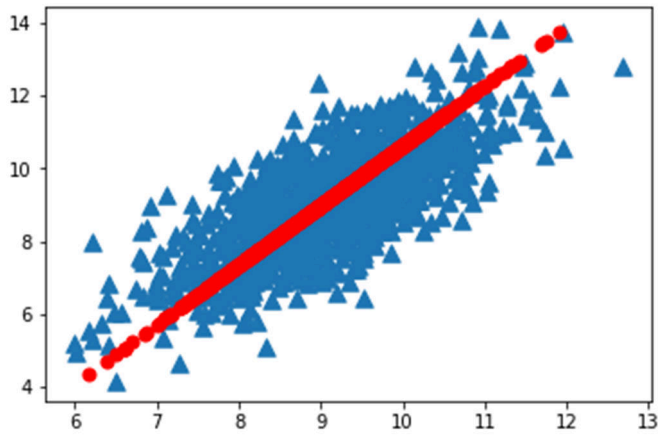

图 4.12 原始数据集（三角形点表示）及第一主成分（圆点表示）

4.3 有监督学习

4.3.1 有监督学习的概念

有监督学习通过已有的训练样本去训练得到一个最优模型，再利用这个模型将所有的输入映射为相应的输出，对输出进行简单的判断从而实现预测和分类的目的，也就具有了对未知数据进行预测和分类的能力。简单来说，就像有标准答案的练习题，然后再去考试，相比没有答案的练习题然后去考试准确率更高。监督学习中的数据中是提前做好了分类信息的，它的训练样本中是同时包含有特征和标签信息的，因此根据这些来得到相应的输出。如何解决分类问题和回归是机器学习其中两个主要任务，分类就是将实例数据划分到合适的分类中，回归主要用于预测数值型数据。分类和回归属于监督学习，这类算法必须知道预测什么，即目标变量的分类信息。

训练数据由一组训练实例组成。在监督学习中，每一个例子都是一对由一个输入对象（通常是一个向量）和一个期望的输出值（也被称为监督信号）。有监督学习算法分析训练数据，并产生一个推断的功能，它可以用于映射新的例子。一个最佳的方案将允许该算法正确地在标签不可见的情况下确定类标签。用已知某种或某些特性的样本作为训练集，以建立一个数学模型(如模式识别中的判别模型，人工神经网络法中的权重模型等)，再用已建立的模型来预测未知样本，故此种方法称为有监督的学习。

4.3.2 回归

1.线性回归

回归是一种技术，用于建模和分析变量之间的关系，并且经常是它们如何贡献的方式，并与一起产生的特定结果相关。线性回归是指完全由线性变量组成的回归模型。从简单情况开始，单变量线性回归是一种用于使用线性模型(即线)来模拟单个输入自变量(特征变量)和输出因变量之间的关系的技术。

更一般的情况是多变量线性回归，其中为多个独立输入变量(特征变量)与输出因变量之间的关系创建模型。该模型保持线性，因为输出是输入变量的线性组合。我们可以对多变量线性回归建模如下：

$$Y = a_1 * X_1 + a_2 * X_2 + a_3 * X_3 + \cdots + a_n * X_n + b \tag{4.47}$$

其中 a_n 是系数，X_n 是变量， b 是偏差。正如我们所看到的，这个函数不包含任何非线性，所以它

只适用于建模线性可分数据。这很容易理解，因为我们只是使用系数权重 a_n 来加权每个特征变量 X_n 的重要性。我们使用随机梯度下降(SGD)来确定这些权重 a_n 和偏差 b 。关于线性回归的几个关键点：

- 建模快速简单，特别适用于要建模的关系不是非常复杂且数据量不大的情况
- 非常直观的理解和解释
- 线性回归对异常值非常敏感

2.多项式回归

当我们要创建适合处理非线性可分数据的模型时，我们需要使用多项式回归。在这种回归技术中，最佳拟合线不是一条直线，这是一条符合数据点的曲线。对于一个多项式回归，一些自变量的功效大于 1。例如，我们可以有这样的东西：

$$Y = a_1 * X_1 + a_2 * X_2^2 + a_2 * X_3^3 + \cdots + a_n * X_n^n + b \tag{4.48}$$

我们可以有一些变量有指数，其他变量没有，还可以为每个变量选择我们想要的确切指数。然而，选择每个变量的确切指数自然需要一些关于数据怎样与输出相关的知识。请参阅下面的图表，以便直观的比较线性回归和多项式回归。

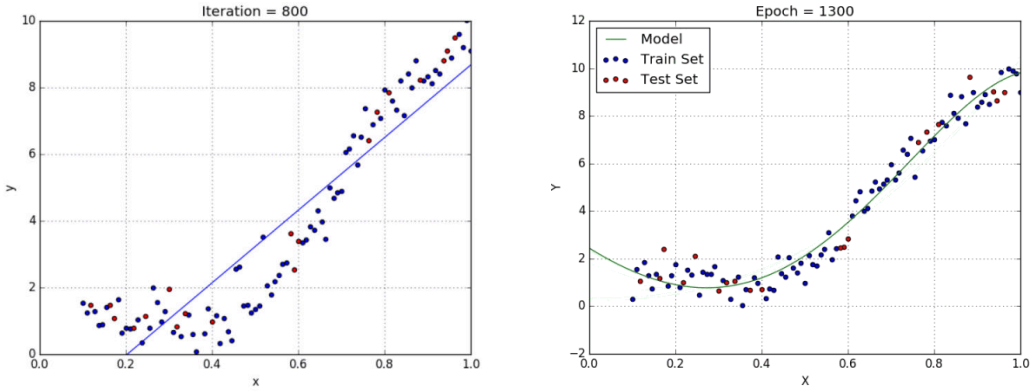

图 4.13 线性回归和多项式回归的比较

线性与多项式回归的数据是非线性可分的。关于多项式回归有几个要点：

- 能够模拟非线性可分的数据，线性回归不能做到这一点。它总体上更灵活，可以模拟一些相当复杂的关系
- 完全控制要素变量的建模(要设置指数)
- 需要仔细的设计。需要一些数据的知识才能选择最佳指数
- 如果指数选择不当，容易过度拟合

3. 岭回归

标准线性或多项式回归在特征变量之间存在高共线性的情况下将失败。共线性是自变量之间存在近似线性关系。高度共线性的存在可以通过几种不同的方式来确定：

- 尽管从理论上讲，该变量应该与 Y 高度相关，但回归系数并不显著。
- 添加或删除 X 特征变量时，回归系数会发生显著变化。

- X 特征变量具有较高的成对相关性(检查相关矩阵)。

我们可以看到标准线性回归的优化函数，以获得有关岭回归如何帮助的一些见解：

$$\min\|X_w - y\|^2 \tag{4.49}$$

其中 X 表示特征变量，w 表示权重，y 表示地面实况。岭回归是一种补救措施，旨在缓解模型中回归预测变量之间的共线性。共线性是一种现象，其中多元回归模型中的一个特征变量可以由其他人以相当程度的准确度线性预测。由于特征变量如此相关，所以最终回归模型在其近似方面受到严格限制，即具有高方差。为了缓解这个问题，岭回归为变量增加了一个小的平方偏差因子：

$$\min\|X_w - y\|^2 + z\|w\|^2 \tag{4.50}$$

这种平方偏差因子将特征变量系数从该刚度中剔除，向模型中引入少量偏差，但大大减少了方差。关于岭回归的几个关键点：

- 这种回归的假设与最小平方回归相同
- 它缩小了系数的值，但没有达到零，这表明没有特征选择功能

4．套索回归

套索回归与岭回归非常相似，因为两种技术都有相同的前提。我们再次在回归优化函数中增加一个偏置项，以减少共线性的影响，从而减少模型方差。但是，不是像岭回归那样使用平方偏差，而是使用绝对值偏差的套索：

$$\min\|X_w - y\|^2 + z\|w\| \tag{4.51}$$

岭回归和套索回归之间存在一些差异，基本上可以归结为 L2 和 L1 正则化的性质差异：

- 内置特征选择：经常被提及为 L1 范数的一个有用属性，而 L2 范数不具有这种特性。这实际上是 L1 范数的结果，其倾向于产生稀疏系数。例如，假设模型有 100 个系数，但其中只有 10 个系数具有非零系数，这实际上是说“其他 90 个预测变量对预测目标值没有用处”。 L2 范数产生非稀疏系数，所以没有这个属性。因此，可以说套索回归做了一种“参数选择”形式，因为未被选中的特征变量将具有总权重 0。
- 稀疏性：指矩阵(或向量)中只有极少数条目非零。 L1 范数具有产生具有零值或具有很小的大系数的非常小值的许多系数的属性。这与套索回归执行一种特征选择的前一点相关。
- 计算效率：L1 范数没有解析解，但 L2 范数有。这使得 L2 范数可以通过计算有效地进行计算。然而，L1 范数解决方案确实具有稀疏性，这使得它可以与稀疏算法一起使用，这使得计算的计算效率更高。

5. ElasticNet 回归

ElasticNet 是套索和岭回归技术的混合体。它既使用了 L1 和 L2 正则化，也使用了两种技术的效果：

$$\min\|X_w - y\|^2 + \mathrm{z}_1\|w\| + \mathrm{z}_2\|w\|^2 \tag{4.52}$$

在套索和岭回归之间进行权衡的一个实际优势是，它允许 Elastic-Net 回归在旋转的情况下继承岭回归

的一些稳定性。关于 ElasticNet 回归的几个关键点：

- 它鼓励在高度相关变量的情况下的群体效应，而不是像套索那样将其中一些置零。
- 对所选变量的数量没有限制。

4.3.3 分类

1、*k*-邻近算法

(1) *k*-邻近算法的概述

k-近邻算法采用测量不同特征值之间的距离方法进行分类。该方法的思路是：如果一个样本在特征空间中的 *k* 个最相似(即特征空间中最邻近)的实例中的大多数属于某一个类别，则该样本也属于这个类别。所谓 *k*-近邻算法，即是给定一个训练数据集，对新的输入样本，在训练数据集中找到与该实例最邻近的 *k* 个实例，这 *k* 个实例的多数属于某个类，就把该输入样本分类到这个类中。选择不同的 *k*，得到的分类结果可能会有不同，当无法判定当前待分类点是从属于已知分类中的哪一类时，我们可以依据统计学的理论看它所处的位置特征，衡量它周围邻居的权重，把它归为到权重更大的那一类。这是 *k*-近邻算法的核心思想。

KNN（*k*-NearestNeighbour）方法虽然从原理上也依赖于极限定理，但在类别决策时，只与极少量的相邻样本有关。由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。

k-近邻算法使用的模型实际上对应于对特征空间的划分。*k* 值的选择，距离度量和分类决策规则是该算法的三个基本要素：

① *k* 值的选择会对算法的结果产生重大影响。*k* 值较小意味着只有与输入样本较近的训练实例才会对预测结果起作用，但容易发生过拟合；如果 *k* 值较大，优点是可以减少学习的估计误差，但缺点是学习的近似误差增大，这时与输入样本较远的训练实例也会对预测起作用，是预测发生错误。在实际应用中，*k* 值一般选择一个较小的数值，通常采用交叉验证的方法来选择最优的 *k* 值。随着训练实例数目趋向于无穷和 *k*=1 时，误差率不会超过贝叶斯误差率的 2 倍，如果 *k* 也趋向于无穷，则误差率趋向于贝叶斯误差率；

② 该算法中的分类决策规则往往是多数表决，即由输入样本的 *k* 个最临近的训练实例中的多数类决定输入样本的类别；

③ 距离度量一般采用 LP 距离，当 *p*=2 时，即为欧氏距离，在度量之前，应该将每个属性的值规范化，这样有助于防止具有较大初始值域的属性比具有较小初始值域的属性的权重过大。

(2) *k*-近邻算法的用途和优缺点

KNN 算法不仅可以用于分类，还可以用于回归。通过找出一个样本的 *k* 个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权重(weight)，如权重与距离成反比。实现 *k*-近邻算法时，主要考虑的问题是如何对训练数据进行快速 *k*-近邻搜索，这在特征空间维数大及训练数据容量大时非常必要。

① 优点：精度高、对异常值不敏感、无数据输入假定。

② 缺点以及部分解决方案：计算复杂度高、空间复杂度高。当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 *k* 个邻居中大容量类的样本占多数，常用的解决方法是利用距离给每个已知的样本赋予权重；因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的 *k* 个最近邻点，所以计算量较大。目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本；该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

③ 适用数据范围：数值型和标称型。

注意：KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

(3) *k*-近邻算法的一般流程

① 收集数据：可以使用任何方法。

② 准备数据：距离计算所需要的数值，最好是结构化的数据格式。

③ 分析数据：可以使用任何方法。

④ 测试算法：计算错误率。

⑤ 使用算法：首先需要输入样本数据和结构化的输出结果，然后运行 k-近邻算法判定输入数据分别属于哪个分类，最后应用对计算出的分类执行后续的处理。

(4) *k*-近邻算法的实现

对未知类别属性的数据集中的每个点依次执行以下操作：

①计算已知类别数据集中的点与当前点之间的距离（*k*-近邻算法常用欧氏距离和马氏距离）；

②按照距离递增次序排序；

③选取与当前点距离最小的 *k* 个点；

④确定前 *k* 个点所在类别的出现频率；

⑤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。

2、决策树

(1) 决策树算法的概述

决策树(Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。Entropy=系统的凌乱程度。决策树是一种树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。一个决策树包含三种类型的节点：

①决策点（□），是对几种可能方案的选择，即最后选择的最佳方案。如果决策属于多级决策，则决策树的中间可以有多个决策点，以决策树根部的决策点为最终决策方案，通常用矩形来表示。

②状态节点（○），代表备选方案的经济效果（期望值），通过各状态节点的经济效果的对比，按照一定的决策标准就可以选出最佳方案。由状态节点引出的分支称为概率枝，概率枝的数目表示可能出现的自然状态数目每个分枝上要注明该状态出现的概率，通常用圆圈来表示。

③结果节点（△），将每个方案在各种自然状态下取得的损益值标注于结果节点的右端，通常用三角形来表示。

(2) 决策树优缺点

① 优点：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关特征数据。

② 缺点以及部分解决方案：对连续性的字段比较难预测；对有时间顺序的数据，需要很多预处理的工作；当类别太多时，错误可能就会增加的比较快；可能会产生过度匹配问题。

③ 适用数据类型：数值型和标称型。

(3) 决策树的一般流程

①收集数据：可以使用任何方法。
②准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化。
③分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期。
④训练算法：构造树的数据结构。
⑤测试算法：使用经验树计算错误率。
⑥使用算法：使用决策树可以更好地理解数据的内在含义。

3、朴素贝叶斯

(1) 朴素贝叶斯算法概述

贝叶斯分类是一系列分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。朴素贝叶斯分类器基于一个简单的假定：给定目标值时属性之间相互条件独立。独立性假设是指一个词的出现概率并不依赖于文档中的其他词。当然我们也知道这个假设过于简单。这就是之所以称为朴素贝叶斯的原因。尽管条件独立性假设并不正确，但是朴素贝叶斯仍然是一种有效的分类器。

(2) 朴素贝叶斯优缺点

① 优点：在数据较少的情况下仍然有效，可以处理多类别问题；使用概率有时要比使用硬规则更为有效；贝叶斯概率及贝叶斯准则提供了一种利用已知值来估计未知概率的有效方法；可以通过特征之间的条件独立性假设，降低对数据量的需求。
② 缺点以及部分解决方案：对于输入数据的准备方式较为敏感。
③ 适用数据类型：标称型数据。

(3) 朴素贝叶斯的一般流程

①收集数据：可以使用任何方法。
②准备数据：需要数值型或者布尔型数据。
③分析数据：有大量特征时，绘制特征作用不大，此时使用直方图效果更好。
④训练算法：计算不同的独立特征的条件概率。
⑤测试算法：计算错误率。
⑥使用算法：一个常见的朴素贝叶斯应用是文档分类。可以在任意的分类场景中使用朴素贝叶斯分类器，不一定非要是文本。

4、 支持向量机（SVM）

(1) SVM 的概述

支持向量机可以分析数据，识别模式，用于分类和回归分析。给定一组训练样本，每个标记为属于两类，一个 SVM 训练算法建立了一个模型，分配新的实例为一类或其他类，使其成为非概率二元线性分类。

(2) SVM 优缺点

① 优点：泛化错误率低，计算开销不大，结果易解释。
② 缺点：对参数调节和核函数的选择敏感，原始分类器不加修改仅适用于处理二类问题。
③ 适用数据类型：数值型和标称型数据。

(3) SVM 的一般流程

① 收集数据：可以使用任意方法。
② 准备数据：需要数值型数据。
③ 分析数据：有助于可视化分隔超平面。
④ 训练算法：SVM 的大部分时间都源自训练，该过程主要实现两个参数的调优。
⑤ 测试算法：十分简单的计算过程就可以实现。
⑥ 使用算法：几乎所有分类问题都可以使用 SVM，SVM 本身是一个二类分类器。

5、AdaBoost 元算法

(1) AdaBoost 算法的概述

自举汇聚法（bootstrap aggregating），也称为 bagging 方法，是在从原始数据集选择 n 次后得到 n 个新数据集的一种技术。新数据集和原数据集的大小相等。每个数据集都是通过在原始数据集中随机选择一个样本来进行替换而得到的。这里的替换就意味着可以多次地选择同一样本。这一性质就允许新数据集中可以有重复的值，而原始数据集的某些值在新集合中则不再出现。在 n 个数据集建好之后，将某个学习算法分别作用于每个数据集就得到了 n 个分类器。当我们要对新数据进行分类时，就可以应用这 n 个分类器进行分类。与此同时，选择分类器投票结果中最多的类别作为最后的分类结果。

boosting 是一种与 bagging 很类似的技术。不论是在 boosting 还是 bagging 当中，所使用的多个分类器的类型都是一致的。但是在前者当中，不同的分类器是通过串行训练而获得的，每个新分类器都根据已训练出的分类器的性能来进行训练。boosting 是通过集中关注被已有分类器错分的那些数据来获得新的分类器。

由于 boosting 分类的结果是基于所有分类器的加权求和结果的，因此 boosting 和 bagging 不太一样。bagging 中的分类器权重是相等的，而 boosting 中的分类器权重并不相等，每个权重代表的是其对应分类器在上一轮迭代中的成功度。

boosting 方法拥有多个版本，其中一个最流行的版本就是 AdaBoost。

(2) AdaBoost 算法的优缺点

① 优点：泛化错误率低，易编码，可以应用在大部分分类器上，无参数调整。
② 缺点：对离群点敏感。
③ 适用数据类型：数值型和标称型数据。

(3) AdaBoost 算法的一般流程

①收集数据：可以使用任意方法。
②准备数据：依赖于所使用的弱分类器类型，如果是单层决策树，可以处理任何数据类型。

③分析数据：可以使用任意方法。

④训练算法：AdaBoost 的大部分时间都用在训絲上，分类器将多次在同一数据集上训练弱分类器。

⑤测试算法：计算分类的错误率。

⑥使用算法：同 SVM 一样，AdaBoost 预测两个类别中的一个。如果想把它应用到多个类别的场合，那么就要像多类 SVM 中的做法一样对 AdaBoost 进行修改。

4.3.4 评价指标

1、准确率(Accuracy)

分对的样本数除以所有的样本数，即：准确（分类）率=正确预测的正反例数/总数。准确率一般用来评估模型的全局准确程度，不能包含太多信息，无法全面评价一个模型性能。

2、混淆矩阵(Confusion Matrix)

混淆矩阵中的横轴是模型预测的类别数量统计，纵轴是数据真实标签的数量统计。对角线表示模型预测和数据标签一致的数目，所以对角线之和除以测试集总数就是准确率。对角线上数字越大越好，在可视化结果中颜色越深，说明模型在该类的预测准确率越高。如果按行来看，每行不在对角线位置的就是错误预测的类别。总的来说，我们希望对角线越高越好，非对角线越低越好。

3、精确率(Precision)与召回率(Recall)

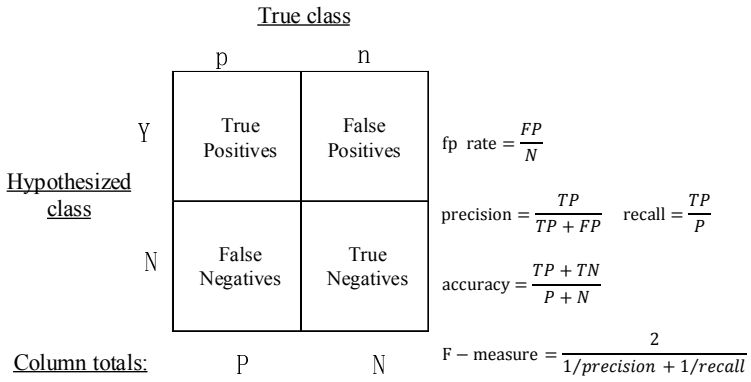

图 4.14 混淆矩阵和由此得出的通用性能指标

下面给出一些相关的定义。假设现在有这样一个测试集，测试集中的图片只由大雁和飞机两种图片组成，假设你的分类系统最终的目的是：能取出测试集中所有飞机的图片，而不是大雁的图片。

True positives：正样本被正确识别为正样本，飞机的图片被正确的识别成了飞机。

True negatives：负样本被正确识别为负样本，大雁的图片没有被识别出来，系统正确地认为它们是大雁。

False positives：假的正样本，即负样本被错误识别为正样本，大雁的图片被错误地识别成了飞机。

False negatives：假的负样本，即正样本被错误识别为负样本，飞机的图片没有被识别出来，系统错误地认为它们是大雁。

Precision 其实就是在识别出来的图片中，True positives 所占的比率。也就是本假设中，所有被识别出

来的飞机中，真正的飞机所占的比例。

$$\text{precision} = \frac{tp}{tp+fp} = \frac{tp}{n} \tag{4.53}$$

Recall 是测试集中所有正样本样例中，被正确识别为正样本的比例。也就是本假设中，被正确识别出来的飞机个数与测试集中所有真实飞机的个数的比值。

$$\text{recall} = \frac{tp}{tp+fn} \tag{4.54}$$

Precision-recall 曲线：改变识别阈值，使得系统依次能够识别前 K 张图片，阈值的变化同时会导致 Precision 与 Recall 值发生变化，从而得到曲线。

如果一个分类器的性能比较好，那么它应该有如下的表现：在 Recall 值增长的同时，Precision 的值保持在一个很高的水平。而性能比较差的分类器可能会损失很多 Precision 值才能换来 Recall 值的提高。通常情况下，文章中都会使用 Precision-recall 曲线，来显示出分类器在 Precision 与 Recall 之间的权衡。

4、平均精度(Average-Precision，AP)与 mean Average Precision(mAP)

AP 就是 Precision-recall 曲线下面的面积，通常来说一个越好的分类器，AP 值越高。

mAP 是多个类别 AP 的平均值。这个 mean 的意思是对每个类的 AP 再求平均，得到的就是 mAP 的值，mAP 的大小一定在[0,1]区间，越大越好。该指标是目标检测算法中最重要的一个。在正样本非常少的情况下，PR 表现的效果会更好。

5、IoU

IoU 值可以理解为系统预测出来的框与原来图片中标记的框的重合程度。计算方法即检测结果 Detection Result 与 Ground Truth 的交集比上它们的并集，即为检测的准确率 IOU 正是表达这种 bounding box 和 groundtruth 的差异的指标：

$$\text{IOU} = \frac{DetectionResult \cap GroundTruth}{etectionResult \cup GroundTruth} \tag{4.55}$$

6、ROC(Receiver Operating Characteristic)曲线与 AUC(Area Under Curve)

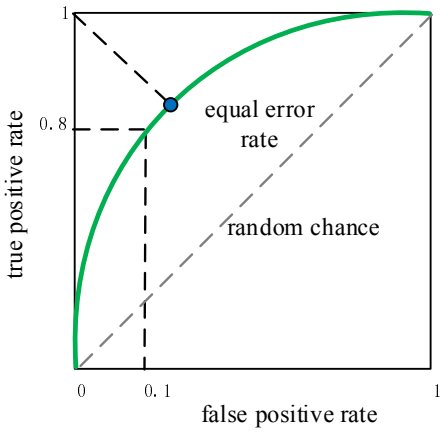

图 4.15 ROC 曲线图

- ROC 曲线：

横坐标：假正率(False positive rate，FPR)，FPR = FP / [FP + TN]，代表所有负样本中错误预测为正样本的概率，假警报率；

纵坐标：真正率(True positive rate，TPR)，TPR = TP / [TP + FN]，代表所有正样本中预测正确的概率，命中率。

对角线对应于随机猜测模型，而(0,1)对应于所有整理排在所有反例之前的理想模型。曲线越接近左上角，分类器的性能越好。

- ROC 曲线有个很好的特性：当测试集中的正负样本的分布变化的时候，ROC 曲线能够保持不变。在实际的数据集中经常会出现类不平衡(class imbalance)现象，即负样本比正样本多很多(或者相反)，而且测试数据中的正负样本的分布也可能随着时间变化。
- ROC 曲线的绘制：

（1）根据每个测试样本属于正样本的概率值从大到小排序；

（2）从高到低，依次将“Score”值作为阈值 threshold，当测试样本属于正样本的概率大于或等于这个 threshold 时，我们认为它为正样本，否则为负样本；

（3）每次选取一个不同的 threshold，我们就可以得到一组 FPR 和 TPR，即 ROC 曲线上的一点。

当我们将 threshold 设置为 1 和 0 时，分别可以得到 ROC 曲线上的(0,0)和(1,1)两个点。将这些(FPR,TPR)对连接起来，就得到了 ROC 曲线。当 threshold 取值越多，ROC 曲线越平滑。

- AUC(Area Under Curve)即为 ROC 曲线下的面积。AUC 越接近于 1，分类器性能越好。

物理意义：首先 AUC 值是一个概率值，当你随机挑选一个正样本以及一个负样本，当前的分类算法根据计算得到的 Score 值将这个正样本排在负样本前面的概率就是 AUC 值。当然，AUC 值越大，当前的分类算法越有可能将正样本排在负样本前面，即能够更好的分类。AUC 计算公式就是求曲线下矩形面积：

$$AUC = \sum_{i=2}^{m} \frac{(x_i - x_{i-1}) * (y_i + y_{i-1})}{2} \tag{4.56}$$

7、PR 曲线和 ROC 曲线

- ROC 曲线：

(1) 优点：当测试集中的正负样本的分布变化的时候，ROC 曲线能够保持不变。因为 TPR 聚焦于正例，FPR 聚焦于与负例，使其成为一个比较均衡的评估方法。在实际的数据集中经常会出现类不平衡（class imbalance）现象，即负样本比正样本多很多（或者相反），而且测试数据中的正负样本的分布也可能随着时间变化。

(2) 缺点：上文提到 ROC 曲线的优点是不会随着类别分布的改变而改变，但这在某种程度上也是其缺点。因为负例 N 增加了很多，而曲线却没变，这等于产生了大量 FP。像信息检索中如果主要关心正例的预测准确性的话，这就不可接受了。在类别不平衡的背景下，负例的数目众多致使 FPR 的增长不明显，导致 ROC 曲线呈现一个过分乐观的效果估计。ROC 曲线的横轴采用 FPR，根据 FPR，当负例 N 的数量远超正例 P 时，FP 的大幅增长只能换来 FPR 的微小改变。结果是虽然大量负例被错判成正例，在 ROC 曲线上却无法直观地看出来。

- PR 曲线：

PR 曲线使用了 Precision，因此 PR 曲线的两个指标都聚焦于正例。类别不平衡问题中由于主要关心正例，所以在此情况下 PR 曲线被广泛认为优于 ROC 曲线。

8、非极大值抑制（NMS）

Non-Maximum Suppression 就是需要根据 score 矩阵和 region 的坐标信息，从中找到置信度比较高的 bounding box。对于有重叠在一起的预测框，只保留得分最高的那个。

NMS 计算出每一个 bounding box 的面积，然后根据 score 进行排序，把 score 最大的 bounding box 作为队列中首个要比较的对象；然后计算其余 bounding box 与当前最大 score 与 box 的 IoU，去除 IoU 大于设定的阈值的 bounding box，保留小的 IoU 得预测框；然后重复上面的过程，直至候选 bounding box 为空。最终，检测了 bounding box 的过程中有两个阈值，一个就是 IoU，另一个是在过程之后，从候选的 bounding box 中剔除 score 小于阈值的 bounding box。需要注意的是：Non-Maximum Suppression 一次处理一个类别，如果有 N 个类别，Non-Maximum Suppression 就需要执行 N 次

4.4 机器学习经典方法

4.4.1 K 近邻算法

k 近邻(*k*-Nearest Neighbor,简称 *k*NN)学习是一种常用的监督学习方法，也可以说是最简单的机器学习算法，构建模型只需要保存训练数据集即可。想要对新数据点做出预测，算法会在训练数据集中找到最近的数据点，也就是它的“最近邻”。其工作机制非常简单：给定测试样本，基于某种距离度量找出训练集中与其最靠近的 *k* 个训练样本，然后基于这 *k* 个“邻居”的信息来进行预测.通常，在分类任务中可使用“投票法”，即选择这 *k* 个样本中出现最多的类别标记作为预测结果；在回归任务中可使用“平均法”，即将这 *k* 个样本的实值输出标记的平均值作为预测结果。同时还可基于距离远近进行加权平均或加权投票，距离越近的样本权重越大。

图 4.16 给出 *k* 近邻分类的一个示意图，显然，*k* 是一个重要参数，当 *k* 取不同值时，分类结果会有显著不同.另一方面，若采用不同的距离计算方式，则找出的“近邻”可能有显著差别，从而也会导致分类结果有显著不同。

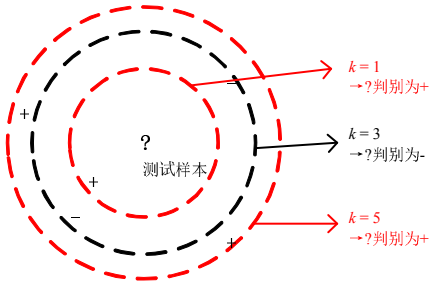

图 4.16　K 近邻分类示意图.虚线显示出等距线；测试样本在 *k*=1 或 *k*=5 时被判别为反例

k-近邻算法（kNN)的工作原理是：存在一个样本数据集合，也称作训练样本集，并且样本集中每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似数据（最近邻）的分类标签。一般来说，我们只选择样本数据集中前 *k* 个最相似的数据，这就是 *k*-近邻算法中 *k* 的出处，通常 *k* 是不大于 20 的整数。最后，选择 *k* 个最相似数据中出现次数最多的分类，作为新数据的分类。

1.　k 近邻分类

k-NN 算法最简单的版本只考虑一个最近邻，也就是与我们想要预测的数据点最近的训练数据点。预测结果就是这个训练数点的已知输出。图 4.17 给出了这种分类方法在 forge 数据集上的应用:

```
mglearn.plots.plot_knn_classification(n_neighbors=1)

‘’’
```

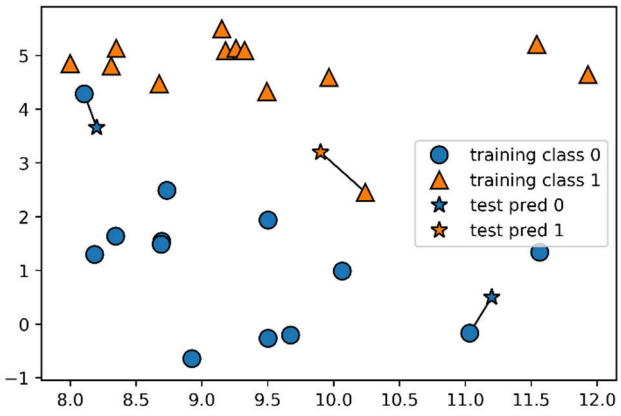

图 4.17　单一最近邻模型对 forge 数据集的预测结果

这里我们添加了 3 个新数据点(用五角星表示)。对于每个新数据点，我们标记了训练集中与它最近的点。单一最近邻算法的预测结果就是那个点的标签(对应五角星的颜色)。除了仅考虑最近邻，我还可以考虑任意个(*k* 个)邻居。这也是 k 近邻算法名字的来历。在考虑多于一个邻居的情况时，我们用“投票法”(voting)来指定标签。也就是说，对于每个测试点，我们数一数多少个邻居属于类别 0，多少个邻居属于类别 1。然

```
mglearn.plots.plot_knn_classification(n_neighbors=3)

‘’’
```

后将出现次数更多的类别(也就是 k 个近邻中占多数的类别)作为预测结果。下面的例子(图 4.18)用到了 3 个近邻：

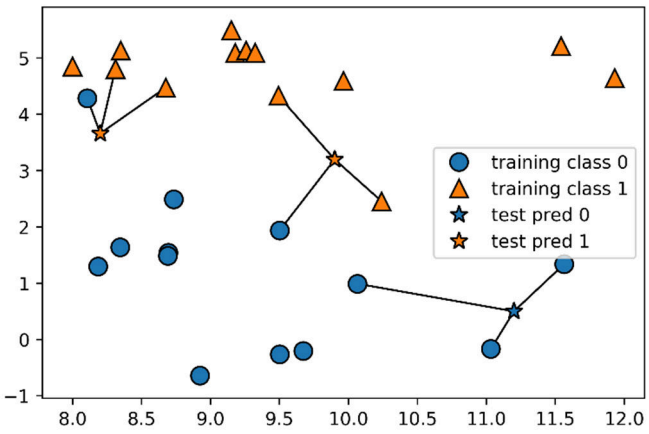

图 4.18 3 近邻模型对 forge 数据集的预测结果

和上面一样，预测结果可以从五角星的颜色看出。你可以发现，左上角新数据点的预测结果与只用一个邻居时的预测结果不同。虽然这张图对应的是一个二分类问题，但方法同样适用于多分类的数据集。对于多分类问题，我们数一数每个类别分别有多少个居，然后将最常见的类别作为预测结果。

现在看一下如何通过 scikit-learn 来应用 k 近邻算法。首先，将数据分为训练集和测试集，以便评估泛化性能：

```
from sklearn.model_selection import train_test_split
X,y = mglearn.datasets.make_forge()
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)

‘’’
```

然后，导入类并将其实例化。这时可以设定参数，比如邻居的个数。这里我们将其设为 3：

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)

‘’’
```

现在，利用训练集对这个分类器进行拟合。对于 KNeighborsClassifier 来说就是保存数据集，以便在预测时计算与邻居之间的距离：

```
clf.fit(X_train,y_train)

‘’’
```

调用 predict 方法来对测试数据进行预测。对于测试机中的每个数据点，都要计算它在训练集的最邻近点，然后找出其中出现次数最多的类别：

```
print("Test set prediction:{}".format(clf.predict(X_test)))

'''
运行以上程序可以得到输出:
Test set prediction:[1 0 1 0 1 0 0]
```

为了评估模型的泛化能力好坏，我们可以对测试数据和测试标签调用 score 方法：

```
print("Test set accuracy:{:.2f}".format(clf.score(X_test,y_test)))

'''
运行以上程序可以得到输出:
Test set accuracy:0.86
```

我们可以看到模型精度约为 86%，也就是说，在测试数据集中，模型对其中 86%的样本预测的类别都是正确的。

2. 分析 KNeighborsClassifier

对于二维数据集，我们还可以在 xy 平面上画出所有可能的测试点的预测效果。我们根据平面中每个点所属的类别对平面进行着色。这样可以查看决策边界(decision boundary)，即算法对类别 0 和类别 1 的分界线。

下列代码分别将 1 个、3 个和 9 个邻居的三种情况的决策边界可视化，见图 4.19：

```
import matplotlib.pyplot as plt

fig,axes = plt.subplots(1,3,figsize=(10,3))
for n_neighbors, ax in zip([1,3,9],axes):
#fit 方法返回对象本身，所以我们可以将实例化和拟合放在一行代码中
clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X,y)
mglearn.plots.plot_2d_separator(clf,X,fill=True,eps=0.5,ax=ax,alpha=.4)
mglearn.discrete_scatter(X[:,0],X[:,1],y,ax=ax)
ax.set_title("{} neighbor(s)".format(n_neighbors))
ax.set_xlabel("feature 0")
ax.set_ylabel("feature 1")
axes[0].legend(loc=3)

'''
```

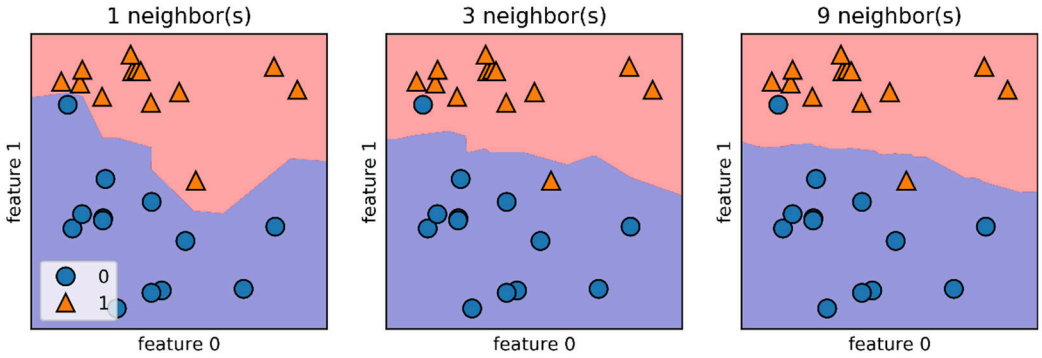

图 4.19 不同 n_neighbors 值得 k 近邻模型的决策边界

从左图可以看出，使用单一邻居绘制的决策边界紧跟着训练数据。随着邻居个数越来越多，决策边界也越来越平滑。更平滑的边界对应更简单的模型。换句话说，使用更少的邻居对应更高的模型复杂度（如图 4.19 右侧所示），而使用更多的邻居对应更低的模型复杂度（如图 4.19 左侧所示）。假如考虑极端情祝，即邻居个数等于训练集中所有数据点的个数，那么每个测试点的邻居都完全相同（即所有训练点），所有预测结果也完全相同（即训练集中出现次数最多的类别）。

我们来研究一下能否证实之前讨论过的模型复杂度和泛化能力之间的关系。我们将在现实世界的乳腺癌数据集上进行研究。先将数据集分成训练集和测试集，然后用不同的邻居个数对训练集和测试集的性能进行评估。输出结果见图 4.20：

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(
        cancer.data,cancer.target,stratify=cancer.target,random_state=66)
training_accuracy = []
test_accuracy =[]
#n_neighbors 取值从 1 到 10
neighbors_settings = range(1,11)
for n_neighbors in neighbors_settings:
    #构建模型
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train,y_train)
    #记录训练精度
    training_accuracy.append(clf.score(X_train,y_train))
    #记录泛化精度
    test_accuracy.append(clf.score(X_test,y_test))

plt.plot(neighbors_settings,training_accuracy,label="training accuracy")
plt.plot(neighbors_settings,test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbbors")
plt.legend()

'''
```

图像的 x 轴是 n_neighbors，y 轴是训练集精度和测试集精度。虽然现实世界的图像很少有非常平滑的，但我们仍可以看出过拟合与欠拟合的一些特征。仅考虑单一近邻时，训练集上的预测结果十分完美。但随着邻居个数的增多，模型变得更简单，训练集精度也随之下降。单一邻居时的测试集精度比使用更多邻居时要低，这表示单一近邻的模型过于复杂。与之相反，当考虑 10 个邻居时，模型又过于简单，性能甚至变得更差。最佳性能在中间的某处，邻居个数大约为 6。不过最好记住这张图的坐标轴刻度。最差的性能约为 88%的精度，这个结果仍然可以接受。

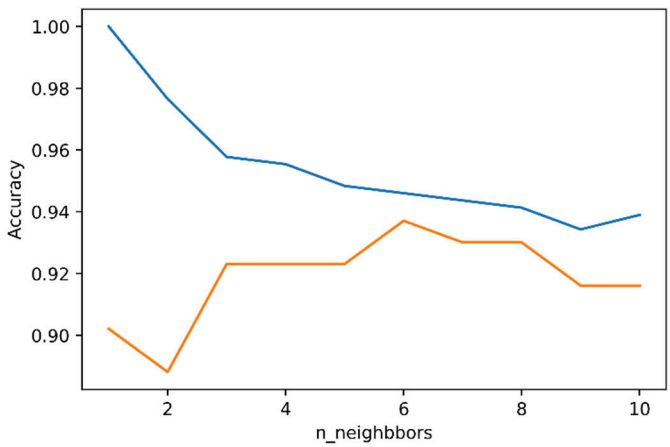

图 4.20　以 n_neighbors 为自变量，对比训练集精度和测试集精度

3.　K 近邻回归

K 近邻算法还可以用于回归。我们还是先从单一近邻开始，这次使用 wave 数据集。我们添加了 3 个测试数据点，在 x 轴上用绿色五角星表示。利用单一邻居的预测结果就是最近邻的目标值。在图 4.21 中用蓝色五角星表示：

```
import mglearn
mglearn.plots.plot_knn_regression(n_neighbors=1)

‘’’
```

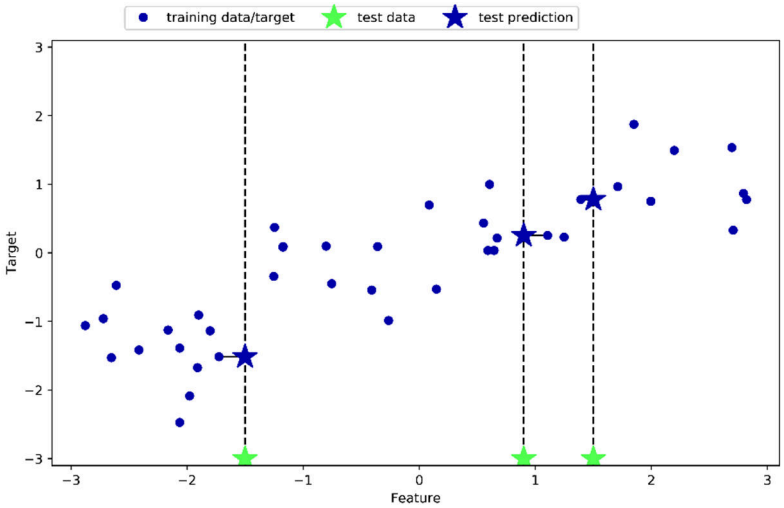

图 4.21 单一近邻回归对 wave 数据集的预测结果

同样，也可以用多个近邻进行回归。在使用多个近邻时，预测结果为这些邻居的平均值(图 4.22)：

```
import mglearn
mglearn.plots.plot_knn_regression(n_neighbors=3)

‘’’
```

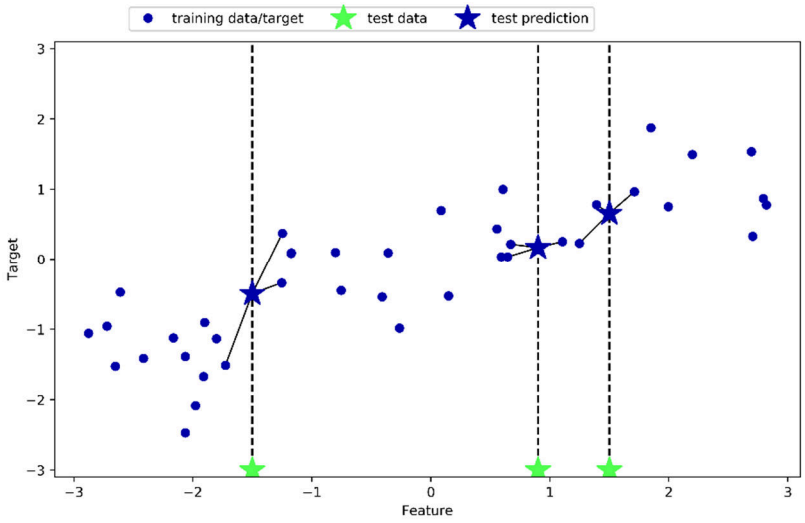

图 4.22　3 个近邻回归对 wave 数据集的预测结果

用于回归的 k 近邻算法在 scikit-learn 的 KNeighborsRegressor 类中实现。其用法与 KNeighborsClassifier 类似：

```
from sklearn.neighbors import KNeighborsRegressor
import mglearn

X,y = mglearn.datasets.make_wave(n_samples=40)

#将 wave 数据集分为训练集和测试集
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)

#模型实例化，并将邻居个数设为 3
reg = KNeighborsRegressor(n_neighbors=3)
#利用训练数据集和训练目标值来拟合模型
reg.fit(X_train,y_train)

'''
```

现在可以对测试集进行预测：

```
print("Test set predictions: \n {}".format(reg.predict(X_test)))

'''
运行以上程序可以得到输出：
Test set predictions：
[-0.05396539    0.35686046    1.13671923 -1.89415682 -1.13881398 -1.63113382
   0.35686046    0.91241374 -0.44680446 -1.13881398]
```

我们还可以用 score 方法来评估模型，对于回归问题，这一方法返回的是R^2分数。R^2分数也叫作决定系数，是回归模型预测的优度度量，位于 0 到 1 之间。R^2等于 1 对应完美预测，R^2等于 0 对应常数模型，即总是预测训练集响应(y_ train)的平均值：

```
print("Test set R^2:{:.2f}".format(reg.score(X_test,y_test)))

'''
运行以上程序可以得到输出：
Test set R^2:0.83
```

这里的分数是 0.83，表示模型的拟合相对较好。

4. 分析 KNeighborsRegressor

对于我们的一维数据集，可以查看所有特征值对应的预测结果(图 4.23)。为了便于绘图，我们创建一个由许多点组成的测试数据集：

```
import matplotlib.pyplot as plt
import mglearn
import numpy as np

X,y = mglearn.datasets.make_wave(n_samples=40)

#将 wave 数据集分为训练集和测试集
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0)

#模型实例化，并将邻居个数设为 3
reg = KNeighborsRegressor(n_neighbors=3)
#利用训练数据集和训练目标值来拟合模型
reg.fit(X_train,y_train)
print("Test set predictions:\n{}".format(reg.predict(X_test)))
print("Test set R^2:{:.2f}".format(reg.score(X_test,y_test)))

fig,axes = plt.subplots(1,3,figsize=(15,4))
#创建 1000 个数据点，在-3 和 3
line = np.linspace(-3,3,1000).reshape(-1,1)
for n_neighbors, ax in zip([1,3,9],axes):
#利用 1 个、3 个或 9 个邻居分别进行预测
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X,y)
    ax.plot(line,reg.predict(line))
    ax.plot(X_train,y_train,'^',c=mglearn.cm2(0),markersize=8)
    ax.plot(X_test,y_test,'^',c=mglearn.cm2(1),markersize=8)
    ax.set_title("{} neighbor(s)\n train score:{:.2f} test score:{:.2f}".format(
        n_neighbors,reg.score(X_train,y_train),reg.score(X_test,y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")
axes[0].legend(["Model predictions","Training data/target",
    "Test data/target"],loc="best")

'''
```

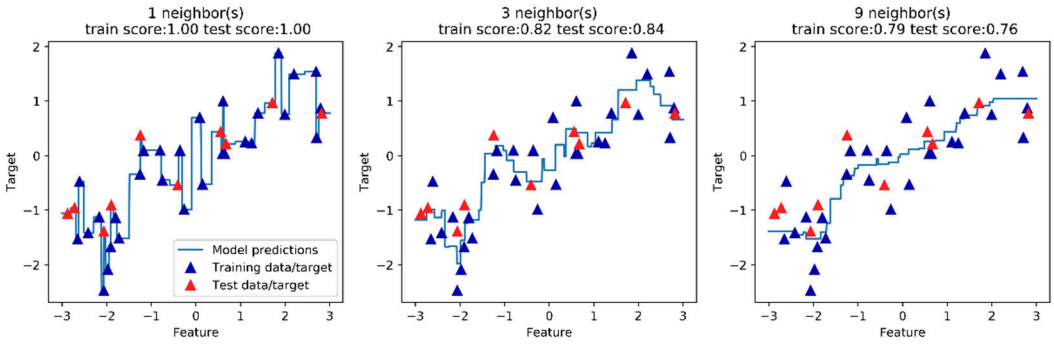

图 4.23 不同 n_neighbors 值得 k 近邻回归的预测结果对比

从图中可以看出，仅使用单一邻居，训练集中的每个点都对预测结果有显著影响，预测结果的图像经过所有数据点。这导致预测结果非常不稳定。考虑更多的邻居之后，预测结果变得更加平滑，但对训练数据的拟合也不好。

5. 优点、缺点和参数

一般来说，KNeighbors 分类器有 2 个重要参数：邻居个数与数据点之间距离的度量方法。在实践中，使用较小的邻居个数（比如 3 个或 5 个）往往可以得到比较好的结果，但你应该调节这个参数。选择合适的距离度量方法超出了本书的范围。默认使用欧式距离，它在许多情况下的效果都很好。

k-NN 的优点之一就是模型很容易理解，通常不需要过多调节就可以得到不错的性能。在考虑使用更高级的技术之前，尝试此算法是一种很好的基准方法。构建最近邻模型的速度通常很快，但如果训练集很大（特征数很多或者样本数很大），预测速度可能会比较慢。使用 k-NN 算法时，对数据进行预处理是很重要的。这一算法对于有很多特征（几百或更多）的数据集往往效果不好，对于大多数特征的大多数取值都为 0 的数据集（所谓的稀疏数据集）来说，这一算法的效果尤其不好。

虽然 k 近邻算法很容易理解，但由于预测速度慢且不能处理具有很多特征的数据集，所以在实践中往往不会用到。

4.4.2 决策树

1. 基本概念

决策树(decision tree) 是一类常见的机器学习方法.以二分类任务为例，我们希望从给定训练数据集学得一个模型用以对新示例进行分类，这个把样本分类的任务，可看作对"当前样本属于正类吗?"这个问题的"决策"或"判定"过程.顾名思义，决策树是基于树结构来进行决策的，这恰是人类在面临决策问题时一种很自然的处理机制.例如，我们要对"这是好瓜吗?"这样的问题进行决策时，通常会进行一系列的判断或"子决策"我们先看"它是什么颜色?"，如果是"青绿色"，则我们再看"它的根蒂是什么形态?"，如果是"蜷缩"，我们再判断"它敲起来是什么声音?"，最后?我们得出最终决策:这是个好瓜.这个决策过程如图 4.24 所示。

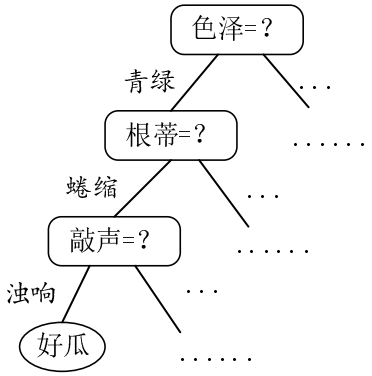

图 4.24 西瓜问题的一棵决策树

显然，决策过程的最终结论对应了我们所希望的判定结果，例如"是"或"不是"好瓜；决策过程中提出的每个判定问题都是对某个属性的"测试"，例如"色泽=?" "根蒂:?"；每个测试的结果或是导出最终结论，或是导出进一步的判定问题，其考虑范围是在上次决策结果的限定范围之内，例如若在"色泽=青绿"之后再判断"根蒂=?"，则仅在考虑青绿色瓜的根蒂。

一般的，一棵决策树包含一个根结点、若干个内部结点和若干个叶结点；叶结点对应于决策结果?其他每个结点则对应于一个属性测试;每个结点包含的样本集合根据属性测试的结果被划分到子结点中;根结点包含样本全集.从根结点到每个叶结点的路径对应了一个判定测试序列.决策树学习的目的是为了产生一棵泛化能力强，即处理未见示例能力强的决策树，其基本流程遵循简单且直观的"分而治之" (divide-and-conquer) 策略，如图 4.25 所示。

输入：训练集$D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$；
　　属性集$A=\{a_1,a_2,\cdots,a_d\}$。
过程：函数 TreeGenerate(D,A)
1：生成结点 node；
2：if D中样本全属于同一类别C then
3：　　将 node 标记为C类叶结点；return
4：end if
5：if $A=\emptyset$ OR D中样本在A上取值相同 then
6：　　将 node 标记为叶结点，其类别标记为D中样本数最多的类；return
7：end if
8：从A中选择最优划分属性a_*；
9：for a_*的每一个值a_*^v do
10：　　为 node 生成一个分支；令D_v表示D中在a_*上取值为a_*^v的样本子集；
11：　　if D_v为空 then
12：　　　　将分支结点标记为叶结点，其类别标记为D中样本最多的类；return
13：　　else
14：　　　　以 TreeGenerate$(D_v,A\{a_*\})$为分支结点
15：　　end if
16：end for
输出：以 node 为根结点的一棵决策树

图 4.25 决策树学习基本算法

显然，决策树的生成是一个递归过程。在决策树基本算法中，有三种情形会导致递归返回：(1)当前结

点包含的样本全属于同一类别，无需划分；(2)当前属性集为空，或是所有样本在所有属性上取值相同，无法划分；(3)当前结点包含的样本集合为空，不能划分。

在第(2)种情形下，我们把当前结点标记为叶结点，并将其类别设定为该结点所含样本最多的类别；在第(3)种情形下，同样把当前结点标记为叶结点，但将其类别设定为其父结点所含样本最多的类别。注意这两种情形的处理实质不同：情形(2)是在利用当前结点的后验分布，而情形(3)则是把父结点的样本分布作为当前结点的先验分布。

2. 划分选择

由算法 4.25 可看出，决策树学习的关键是第 8 行，即如何选择最优划分属性。一般而言，随着划分过程不断进行，我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的"纯度" (purity) 越来越高。

(1) 信息增益

"信息熵" (information entropy)是度量样本集合纯度最常用的一种指标。假定当前样本集合 D 中第 k 类样本所占的比例为 P_k $(k = 1,2,\dots,|\gamma|)$ ，则 D 的信息熵定义为：

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|\gamma|} p_k \log_2 p_k \tag{4.57}$$

$\mathrm{Ent}(D)$的值越小，则 D 的纯度越高。

假定离散属性 a 有 V 个可能的取值$\{a^1, a^2, \dots, a^V\}$，若使用 a 来对样本集 D 进行划分，则会产生 V 个分支结点，其中第 v 个分支结点包含了 D 中所有在属性 a 上取值为 a^v 的样本，记为 D^v 。我们可根据式(4.57)计算出 D^v 的信息熵，再考虑到不同的分支结点所包含的样本数不同，给分支结点赋予权重 $|D^v|/|D|$，即样本数越多的分支结点的影响越大，于是可计算出用属性 a 对样本集 D 进行划分所获得的"信息增益" (information gain)。

$$\mathrm{Gain}(D, \mathrm{a}) = \mathrm{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v) \tag{4.58}$$

一般而言，信息增益越大，则意味着使周属性 a 来进行划分所获得的"纯度提升"越大。因此，我们可用信息增益来进行决策树的划分属性选择，即在 4.58 算法第 8 行选择属性 $a_* = arg\max_{a \epsilon A} Gain(D, a)$ 。著名的 ID3 决策树学习算法[Quinlan，1986] 就是以信息增益为准则来选择划分属性。

以表 4.3 中的西瓜数据集 2.0 为例，该数据集包含 17 个训练样例，用以学习一棵能预测没剖开的是不是好瓜的决策树。显然， $|\gamma| = 2$。在决策树学习开始时，根结点包含 D 中的所有样例，其中正例占 $p_1 = \frac{8}{17}$，反例 $p_2 = \frac{9}{17}$ 。于是，根据式 (4.57) 可计算出根结点的信息熵为

$$\mathrm{Ent}(D) = -\sum_{k=1}^{2} p_k \log_2 p_k = -\left(\frac{8}{17}\log_2\frac{8}{17} + \frac{9}{17}\log_2\frac{9}{17}\right) = 0.998 \tag{4.59}$$

表 4.3 西瓜数据集

编号	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是

3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	稍晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

然后，我们要计算出当前属性集合{色泽，根蒂，敲声，纹理，脐部，触感}中每个属性的信息增益。以属性"色泽"为例，它有 3 个可能的取值：{青绿，乌黑，浅自}。若使用该属性对 D 进行划分，则可得到 3 个子集，分别记为： D^1 (色泽=青绿)， D^2 (色泽 2 乌黑)， D^3 (色泽=浅白)。

子集 D^1 包含编号为{1 ,4, 6, 10, 13, 17}的 6 个样例，其中正例占 $p_1 = \frac{3}{6}$，反例占 $p_2 = \frac{3}{6}$； D^2 包含编号为{2 ,3, 7, 8, 9, 15}的 6 个样例，其中正、反例分别占 $p_1 = \frac{4}{6}$， $p_2 = \frac{2}{6}$；D3 包含编号为{5 ,11, 12, 14, 16}的 5 个样例，其中正、反例分别占$p_1 = \frac{1}{5}$， $p_2 = \frac{4}{5}$。根据式(4.57)可计算出用"色泽"划分之后所获得的 3 个分支结点的信息熵为：

$$\mathrm{Ent}(D^1) = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1.000$$

$$\mathrm{Ent}(D^2) = -\left(\frac{4}{6}\log_2\frac{4}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) = 0.918$$

$$\mathrm{Ent}(D^3) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right) = 0.722$$

于是，根据式(4.58)可计算出属性"色泽"的信息增益为：

$$\begin{aligned}\mathrm{Gain}(D, \text{色泽}) &= \mathrm{Ent}(D) - \sum_{v=1}^{3}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v) \\ &= 0.998 - \left(\frac{6}{17}\times 1.000 + \frac{6}{17}\times 0.918 + \frac{5}{17}\times 0.722\right) \\ &= 0.109\end{aligned}$$

类似的，我们可计算出其他属性的信息增益：

$\mathrm{Gain}(D, \text{根蒂}) = 0.143$；$\mathrm{Gain}(D, \text{敲声}) = 0.141$；
$\mathrm{Gain}(D, \text{纹理}) = 0.381$；$\mathrm{Gain}(D, \text{脐部}) = 0.289$；
$\mathrm{Gain}(D, \text{触感}) = 0.006$.

显然，属性"纹理"的信息增益最大，于是它被选为划分属性。图 4.26 给出了基于"纹理"对根结点进行划分的结果，各分支结点所包含的样例子集显示在结点中。

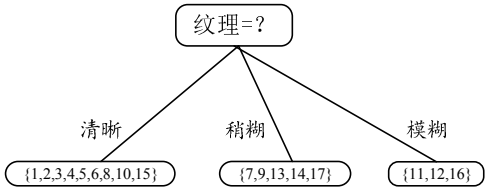

图 4.26 基于"纹理"属性对根结点划分

然后，决策树学习算法将对每个分支结点做进一步划分。以图 4.26 中第一个分支结点("纹理=清晰")为例，该结点包含的样例集合 D^1 中有编号为$\{1, 2, 3, 4, 5, 6, 8, 10, 15\}$的 9 个样例，可用属性集合为{色泽，根蒂，敲声，脐部，触感}。基于 D^1 计算出各属性的信息增益：

$$\text{Gain}(D^1, \text{根蒂}) = 0.043；\ \text{Gain}(D^1, \text{敲声}) = 0.458；$$
$$\text{Gain}(D^1, \text{纹理}) = 0.331；\ \text{Gain}(D^1, \text{脐部}) = 0.458；$$
$$\text{Gain}(D^1, \text{触感}) = 0.458.$$

"根蒂"、"脐部"、"触感" 3 个属性均取得了最大的信息增益，可任选其中之一作为划分属性。类似的，对每个分支结点进行上述操作，最终得到的决策树如图 4.27 所示。

(2)增益率

在上面的介绍中，我们有意忽略了表 4.3 中的"编号"这一列。若把"编号"也作为一个候选划分属性，则根据式(4.58)均可计算出它的信息增益为 0.998 ，远大于其他候选划分属性.这很容易理解"编号"将产生 17 个分支，每个分支结点仅包含一个样本，这些分支结点的纯度已达最大。然而，这样的决策树显然不具有泛化能力，无法对新样本进行有效预测。

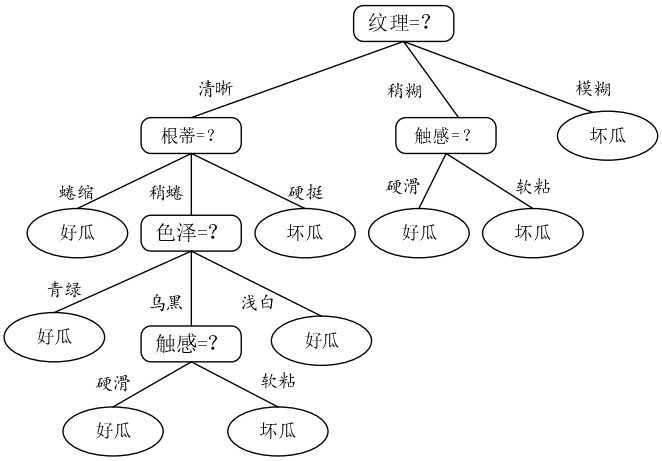

图 4.27 在西瓜数据集 2.0 上基于信息增益生成的决策树

实际上，信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响，著名的 C4.5 决策树算法[Quinlan，1993] 不直接使用信息增益，而是使用"增益率" (gain ratio)来选择最优划分属性。采用与式(4.59)相同的符号表示，增益率定义为：

$$\text{Gain\_ratio}(D^1,\text{a}) = \frac{\text{Gain}(D,\text{a})}{IV(a)} \tag{4.60}$$

其中

$$IV(a) = -\sum_{v=1}^{V}\frac{|D^v|}{|D|}\log_2\frac{|D^v|}{|D|} \tag{4.61}$$

称为属性 a 的"固有值" (intrinsic value) [Quinlan, 1993]. 属性 a 的可能取值数目越多(即 V 越大)，则 IV(a) 的值通常会越大。例如，对表 4.3 的西瓜数据集 2.0，有 IV(触感)= 0.874 (V = 2)， IV(色泽) = 1.580 (V = 3)，IV(编号) = 4.088 (V = 17).

3. 剪枝处理

剪枝(pruning)是决策树学习算法对付"过拟合"的主要手段。在决策树学习中，为了尽可能正确分类训练样本，结点划分过程将不断重复，有时会造成决策树分支过多，这时就可能因训练样本学得"太好"了，以致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。因此，可通过主动去掉一些分支来降低过拟合的风险。

决策树剪枝的基本策略有"预剪枝" (prepruning)和"后剪枝"(post-pruning) [Quinlan, 1993]。 预剪枝是指在决策树生成过程中，对每个结点在划分前先进行估计，若当前结点的划分不能带来决策树泛化性能提升，则停止划分并将当前结点标记为叶结点；后剪枝则是先从训练集生成一棵完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点。

如何判断决策树泛化性能是否提升呢?这可使用性能评估方法。本节假定采用留出法，即预留一部分数据用作"验证集"以进行性能评估。例如对表 4.3 的西瓜数据集 2.0，我们将其随机划分为两部分，如表 4.4 所示，编号为{1，2，3，6，7，10，14，15，16，17}的样例组成训练集，编号为{4，5，8，9，11，12，13}的样例组成验证集。

表 4.4 西瓜数据集 2.0 划分出的训练集(双线上部)与验证集(双线下部)

编号	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否
编号	**色泽**	**根蒂**	**敲声**	**纹理**	**脐部**	**触感**	**好瓜**
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是

8	乌黑	稍蜷	浊响	稍晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否

假定我们采用上节的信息增益准则来进行划分属性选择，则从表 4.4 的训练集将会生成一棵如图 4.28 所示的决策树。为便于讨论，我们对圈中的部分结点做了编号。

(1) 预剪枝

我们先讨论预剪枝。基于信息增益准则，我们会选取属性"脐部"来对训练集进行划分，并产生 3 个分支，如图 4.29 所示。然而，是否应该进行这个划分呢?预剪枝要对划分前后的泛化性能进行估计。

在划分之前，所有样例集中在根结点.若不进行划分，则根据算法 4.25 第 6 行，该结点将被标记为叶结点，其类别标记为训练样例数最多的类别，假设我们将这个叶结点标记为"好瓜"用表 4.4 的验证集对这个单结点决策树进行评估?则编号为{4，5，8}的样例被分类正确，另外 4 个样例分类错误，于是，验证集精度为 $\frac{3}{7}\times 100\% = 42.9\%$。

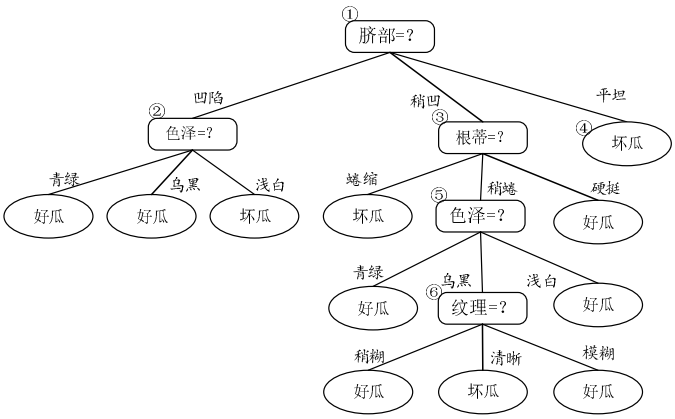

图 4.28 基于表 4.4 生成的未剪枝决策树

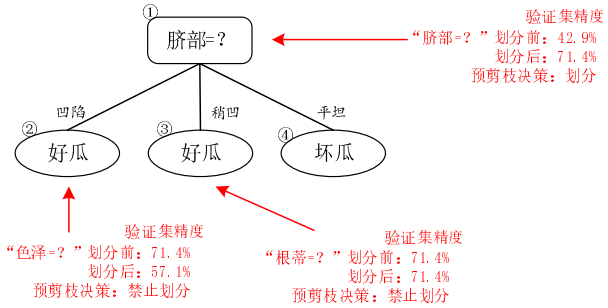

图 4.29 基于表 4.4 生成的预剪枝决策树

在用属性"脐部"划分之后，图 4.29 中的结点②、③、④分别包含编号为{1，2，3，14} 、{6，7，15，17}、{10，16}的训练样例，因此这 3 个结点分别被标记为叶结点"好瓜"、"好瓜"、"坏瓜"。此时，验证集中

编号为{4，5，8，11，12}的样例被分类正确，验证集精度为 $\frac{5}{7}\times 100\% = 71.4\% > 42.9\%$。于是，用"脐部"进行划分得以确定。

然后，决策树算法应该对结点②进行划分，基于信息增益准则将挑选出划分属性"色泽"然而，在使用"色泽"划分后?编号为{5}的验证集样本分类结果会由正确转为错误，使得验证集精度下降为 57.1%。于是，预剪枝策略将禁止结点②被划分。对结点③，最优划分属性为"根蒂"，划分后验证集精度仍为 71.4%。这个划分不能提升验证集精度，于是，预剪枝策略禁止结点③被划分。对结点④，其所含训练样例已属于同一类，不再进行划分。于是，基于预剪枝策略从表 4.4 数据所生成的决策树如图 4.29 所示，其验证集精度为 71. 4%。这是一棵仅有一层划分的决策树，亦称"决策树桩"(decisionstump)。

对比图 4.29 和图 4.28 可看出，于预页剪枝使得决策树的很多分支都没有"展开试时间开销。但另一方面，有些分支的当前划分虽不能提升泛化性能、甚至可能导致泛化性能暂时下降，但在其基础上进行的后续划分却有可能导致性能显著提高；预剪枝基于"贪心"本质禁止这些分支展开，给预剪枝决策树带来了欠拟含的风险。

(2) 后剪枝

后剪枝先从训练集生成一棵完整决策树?例如基于表 4.4 的数据我们得到如图 4.30 所示的决策树.易知 7 该决策树的验证集精度为 42.9%。

后剪枝首先考察图 4.28 中的结点⑥.若将其领衔的分支剪除，则相当于把⑤替换为叶结点.替换后的叶结点包含编号为{7，15} 的训练样本，于是，该叶结点的类别标记为"好瓜"，此时决策树的验证集精度提高至 57.1%. 于是，后剪枝策略决定剪枝，如图 4.7 所示.然后考察结点⑤，若将其领衔的子树替换为叶结点，则替换后的叶结点包含编号为{6 ， 7， 15} 的训练样例，叶结点类别标记为"好瓜'七此时决策树验证集精度仍为 57.1%. 于是，可以不进行剪枝.对结点②，若将其领衔的子树替换为叶结点，则替换后的叶结点包含编号为{1，2，3，14}的训练样例，叶结点标记为"好瓜"此时决策树的验证集精度提高至 71.4%。于是，后剪枝策略决定剪枝。对结点③和①，若将其领衔的子树替换为叶结点，则所得决策树的验证集精度分别为 71.4%与 42.9%，均未得到提高。于是它们被保留。

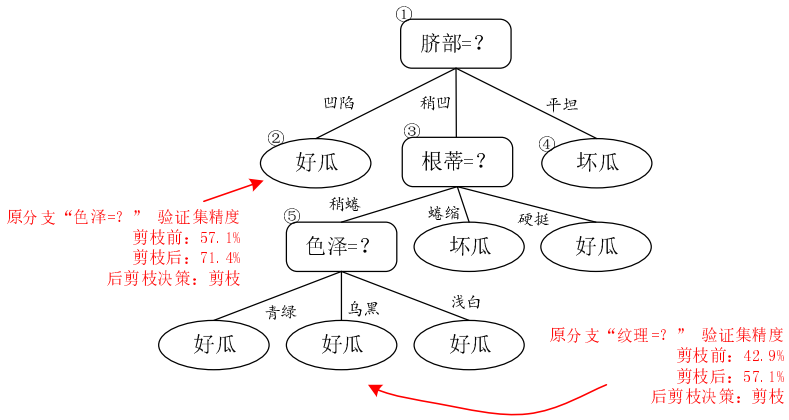

图 4.30 基于表 4.4 生成的后剪枝决策树

最终，基于后剪枝策略从表 4.4 数据所生成的决策树如图 4.30 所示，其验证集精度为 71.4%。

对比图 4.30 和图 4.29 可看出，后剪枝决策树通常比预剪枝决策树保留了更多的分支。一般情形下，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。但后剪枝过程是在生成完全决策树之后进行的，并且要白底向上地对树中的所有非叶结点进行逐一考察，因此其训练时间开销比未剪枝决策树和预剪枝决策树都要大得多。

4. 决策树实现

决策树是广泛用于分类和回归任务的模型。本质上，它从一层层的 if/else 问题中进行学习，并得出结论。这些问题类似于你在“20 Questions”游戏 9 中可能会问的问题。想象一下，你想要区分下面这四种动物：熊、鹰、企鹅和海豚。你的目标是通过提出尽可能少的 if/else 问题来得到正确答案。你可能首先会问：这种动物有没有羽毛，这个问题会将可能的动物减少到只有两种。如果答案是“有”，你可以问下一个问题，帮你区分鹰和企鹅。例如，你可以问这种动物会不会飞。如果这种动物没有羽毛，那么可能是海豚或熊，所以你需要问一个问题来区分这两种动物——比如问这种动物有没有鳍。

这一系列问题可以表示为一棵决策树，如图 4.31 所示。

```
mglearn.plots.plot_animal_tree()
```

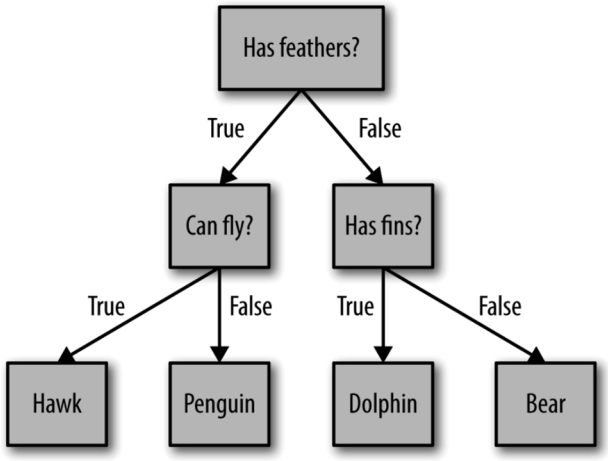

图 4.31 区分几种动物的决策树

在这张图中，树的每个结点代表一个问题或一个包含答案的终结点（也叫叶结点）。树的边将问题的答案与将问的下一个问题连接起来。用机器学习的语言来说就是，为了区分四类动物（鹰、企鹅、海豚和熊），我们利用三个特征（“有没有羽毛”“会不会飞”和“有没有鳍”）来构建一个模型。我们可以利用监督学习从数据中学习模型，而无需人为构建模型。

(1) 构造决策树

我们在图 4.32 所示的二维分类数据集上构造决策树。这个数据集由 2 个半月形组成，每个类别都包含 50 个数据点。我们将这个数据集称为 two_moons。

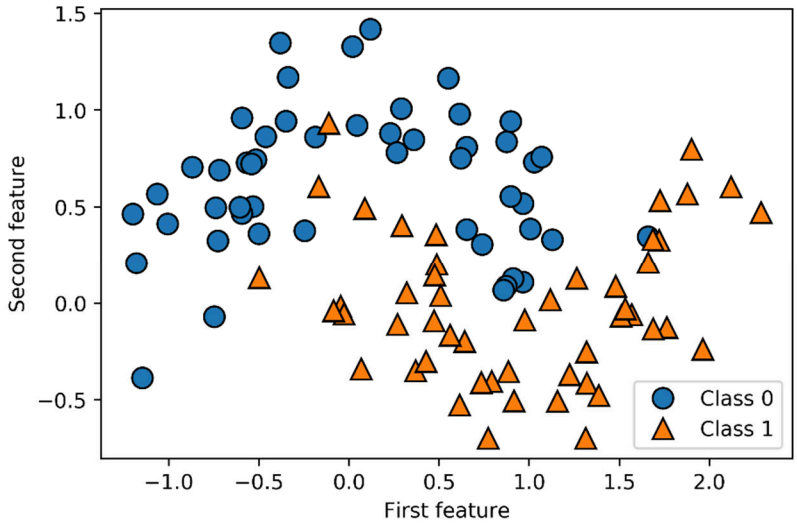

图 4.32：用于构造决策树的 two_moons 数据集

为了构造决策树，算法搜遍所有可能的测试，找出对目标变量来说信息量最大的那一个。图 4.33 展示了选出的第一个测试。将数据集在 x[1]=0.0596 处垂直划分可以得到最多信息，它在最大程度上将类别 0 中的点与类别 1 中的点进行区分。顶结点（也叫根结点）表示整个数据集，包含属于类别 0 的 50 个点和属于类别 1 的 50 个点。通过测试 x[1] <=0.0596 的真假来对数据集进行划分，在图中表示为一条黑线。如果测试结果为真，那么将这个点分配给左结点，左结点里包含属于类别 0 的 2 个点和属于类别 1 的 32 个点。否则将这个点分配给右结点，右结点里包含属于类别 0 的 48 个点和属于类别 1 的 18 个点。这两个结点对应于图 4.33 中的顶部区域和底部区域。尽管第一次划分已经对两个类别做了很好的区分，但底部区域仍包含属于类别 0 的点，顶部区域也仍包含属于类别 1 的点。我们可以在两个区域中重复寻找最佳测试的过程，从而构建出更准确的模型。图 4.34 展示了信息量最大的下一次划分，这次划分是基于 x[0] 做出的，分为左右两个区域。

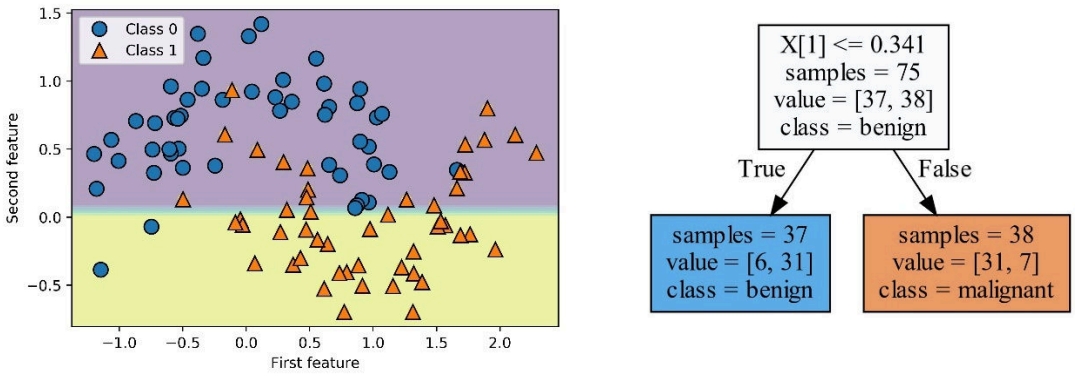

图 4.33 深度为 1 的树的决策边界（左）与相应的树（右）

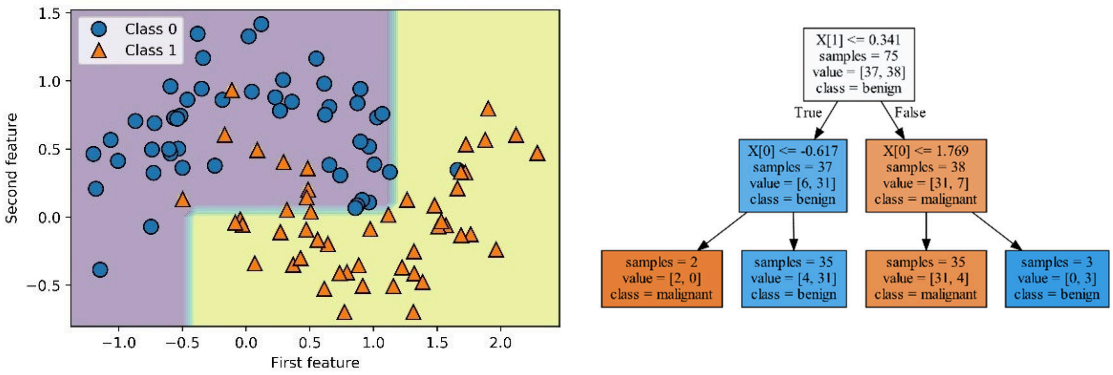

图 4.34 深度为2 的树的决策边界（左）与相应的树（右）

这一递归过程生成一棵二元决策树，其中每个结点都包含一个测试。或者你可以将每个测试看成沿着一条轴对当前数据进行划分。这是一种将算法看作分层划分的观点。由于每个测试仅关注一个特征，所以划分后的区域边界始终与坐标轴平行。对数据反复进行递归划分，直到划分后的每个区域（决策树的每个叶结点）只包含单一目标值(单一类别或单一回归值)。如果树中某个叶结点所包含数据点的目标值都相同，那么这个叶结点就是纯的(pure)。这个数据集的最终划分结果见图 4.35。

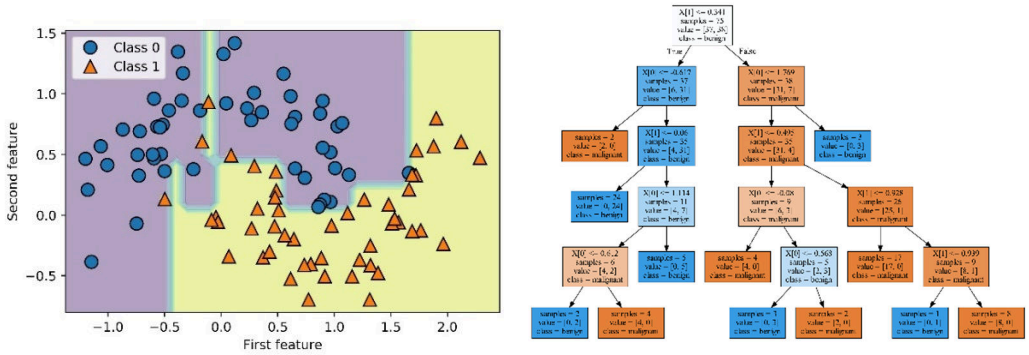

图4.35：深度为 9 的树的决策边界(左)与相应的树的一部分(右) (完整的决策树非常大，很难可视化)

想要对新数据点进行预测，首先要查看这个点位于特征空间划分的哪个区域，然后将该区域的多数目标值（如果是纯的叶结点，就是单一目标值）作为预测结果。从根结点开始对树进行遍历就可以找到这一区域，每一步向左还是向右取决于是否满足相应的测试。

决策树也可以用于回归任务，使用的方法完全相同。预测的方法是，基于每个结点的测试对树进行遍历，最终找到新数据点所属的叶结点。这一数据点的输出即为此叶结点中所有训练点的平均目标值。

(2) 控制决策树的复杂度

通常来说，构造决策树直到所有叶结点都是纯的叶结点，这会导致模型非常复杂，并且对训练数据高度过拟合。纯叶结点的存在说明这棵树在训练集上的精度是 100%。训练集中的每个数据点都位于分类正确的叶结点中。在图 4.35 的左图中可以看出过拟合。你可以看到，在所有属于类别 0 的点中间有一块属于类别 1 的区域。另一方面，有一小条属于类别 0 的区域，包围着最右侧属于类别 0 的那个点。这并不是人们想象中决策边界的样子，这个决策边界过于关注远离同类别其他点的单个异常点。

防止过拟合有两种常见的策略：一种是及早停止树的生长，也叫预剪枝（pre-pruning）；另一种是先构

造树，但随后删除或折叠信息量很少的结点，也叫后剪枝（post-pruning）或剪枝（pruning）。预剪枝的限制条件可能包括限制树的最大深度、限制叶结点的最大数目，或者规定一个结点中数据点的最小数目来防止继续划分。

scikit-learn 的决策树在 DecisionTreeRegressor 类和 DecisionTreeClassifier 类中实现。scikit-learn 只实现了预剪枝，没有实现后剪枝。

我们在乳腺癌数据集上更详细地看一下预剪枝的效果。和前面一样，我们导入数据集并将其分为训练集和测试集。然后利用默认设置来构建模型，默认将树完全展开（树不断分支，直到所有叶结点都是纯的）。我们固定树的 random_state，用于在内部解决平局问题：

```
from sklearn.tree import DecisionTreeClassifier
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))

'''
运行以上程序可以得到输出：
Accuracy on training set: 1.000
Accuracy on test set: 0.937
```

不出所料，训练集上的精度是 100%，这是因为叶结点都是纯的，树的深度很大，足以完美地记住训练数据的所有标签。测试集精度比之前讲过的线性模型略低，线性模型的精度约为 95%。

如果我们不限制决策树的深度，它的深度和复杂度都可以变得特别大。因此，未剪枝的树容易过拟合，对新数据的泛化性能不佳。现在我们将预剪枝应用在决策树上，这可以在完美拟合训练数据之前阻止树的展开。一种选择是在到达一定深度后停止树的展开。这里我们设置 max_depth=4，这意味着只可以连续问 4 个问题（参见图 4.33 和图 4.35）。限制树的深度可以减少过拟合。这会降低训练集的精度，但可以提高测试集的精度：

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))

'''
运行以上程序可以得到输出：
Accuracy on training set: 0.988
Accuracy on test set: 0.951
```

(3) 分析决策树

我们可以利用 tree 模块的 export_graphviz 函数来将树可视化。这个函数会生成一个.dot 格式的文件，这是一种用于保存图形的文本文件格式。我们设置为结点添加颜色的选项，颜色表示每个结点中的多数类别，同时传入类别名称和特征名称，这样可以对树正确标记：

```
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant","benign"],
feature_names=cancer.feature_names, impurity=False, filled=True)
```

我们可以利用 graphviz 模块读取这个文件并将其可视化（你也可以使用任何能够读取.dot 文件的程序），见图 4.36：

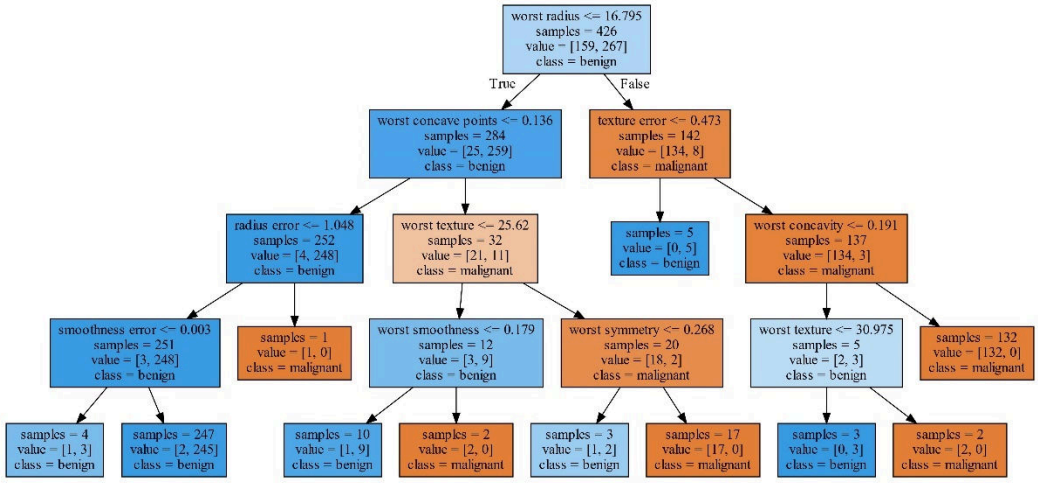

图 4.36 基于乳腺癌数据集构造的决策树的可视化

树的可视化有助于深入理解算法是如何进行预测的，也是易于向非专家解释的机器学习算法的优秀示例。不过，即使这里树的深度只有 4 层，也有点太大了。深度更大的树（深度为 10 并不罕见）更加难以理解。一种观察树的方法可能有用，就是找出大部分数据的实际路径。图 2-27 中每个结点的 samples 给出了该结点中的样本个数，values 给出的是每个类别的样本个数。观察 worst radius <= 16.795 分支右侧的子结点，我们发现它只包含 8 个良性样本，但有 134 个恶性样本。树的这一侧的其余分支只是利用一些更精细的区别将这 8 个良性样本分离出来。在第一次划分右侧的 142 个样本中，几乎所有样本（132 个）最后都进入最右侧的叶结点中。

再来看一下根结点的左侧子结点，对于 worst radius > 16.795，我们得到 25 个恶性样本和 259 个良性样本。几乎所有良性样本最终都进入左数第二个叶结点中，大部分其他叶结点都只包含很少的样本。

(4) 优点、缺点和参数

如前所述，控制决策树模型复杂度的参数是预剪枝参数，它在树完全展开之前停止树的构造。通常来说，选择一种预剪枝策略（设置 max_depth、max_leaf_nodes 或 min_samples_leaf）足以防止过拟合。

与前面讨论过的许多算法相比，决策树有两个优点：一是得到的模型很容易可视化，非专家也很容易理解（至少对于较小的树而言）；二是算法完全不受数据缩放的影响。由于每个特征被单独处理，而且数据的划分也不依赖于缩放，因此决策树算法不需要特征预处理，比如归一化或标准化。特别是特征的尺度完

全不一样时或者二元特征和连续特征同时存在时，决策树的效果很好。

决策树的主要缺点在于，即使做了预剪枝，它也经常会过拟合，泛化性能很差。因此，在大多数应用中，往往使用下面介绍的集成方法来替代单棵决策树。

4.4.3 线性回归

1. 基本形式

给定由 d 个属性描述的示例x $= (x_1; x_2; \cdots; x_d)$，其中$x_i$是在第 i 个属性上的取值，线性模型(linear model)试图学得一个通过属性的线性组合来进行预测的函数，即

$$f(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b \tag{4.62}$$

一般用向量形式写成

$$f(x) = w^Tx + b \tag{4.63}$$

其中 w= $(w_1; w_2; \cdots; w_d)$。W 和 b 学得之后，模型就得以确定。

线性模型形式简单、易于建模，但却蕴涵着机器学习中一些重要的基本思想。许多功能更为强大的非线性模型(nonlinear model)可在线性模型的基础上通过引入层级结构或高维映射而得。此外，由于 w 直观表达了各属性在预测中的重要性，因此线性模型有很好的可解释性(comprehensibility)。

本章介绍几种经典的线性模型，我们先从回归任务开始，然后讨论二分类和多分类任务。

2. 线性回归

给定数据集D $= \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$，其中$x_i = (x_1; x_2; \cdots; x_d)$，$y_i \in R$，“线性回归”(linear regression)试图学得一个线性模型以尽可能准确地预测实值输出标记。

我们先考虑一种最简单的情形：输入属性的数目只有一个。为便于讨论，此时我们忽略关于属性的下标，即D $= \{(x_i, y_i)\}_{i=1}^m$，其中$x_i \in R$。对离散属性，若属性值间存在“序”(order)关系，可通过连续化将其转化为连续值，例如二值属性“身高”的取值“高”“矮”可转化为$\{1.0, 0.0\}$，三值属性“高度”的取值“高”“中”“低”可转化为$\{1.0, 0.5, 0.0\}$；若属性值间不存在序关系，假定有 k 个属性值，则通常转化为 k 维向量，例如属性“瓜类”的取值“西瓜”“南瓜”“黄瓜”可转化为(0，0，1)，(0，1，0)，(1，0，0)。

线性回归试图学得：

$$f(x_i) = wx_i + b, \quad \text{使得} f(x_i) \approx y_i \tag{4.64}$$

如何确定 w 和 b 呢？显然，关键在于如何衡量$f(x)$与 y 之间的差别。均方误差是回归任务中最常用的性能度量，因此我们可试图让均方误差最小化，即：

$$\begin{aligned}(w^*, b^*) &= arg \min_{(w,b)} (f(x_i) - y_i)^2 \\ &= arg \min_{(w,b)} \sum_{i=1}^m (y_i - wx_i - b)^2 \end{aligned} \tag{4.65}$$

均方误差有非常好的几何意义，它对应了常用的欧几里得距离或简称“欧氏距离”(Euclidean distance)，基于均方误差最小化来进行模型求解的方法称为“最小二乘法” (least square method)。在线性回归中，最小二乘法就是试图找到一条直线，使所有样本到直线上的欧式距离之和最小。

求解 w 和 b 使$E_{(w,b)} = \sum_{i=1}^{m}(y_i - wx_i - b)^2$最小化的过程，称为线性回归模型的最小二乘“参数估计” (parameter estimation)。我们可将$E_{(w,b)}$分别对 w 和 b 求导，得到：

$$\frac{\partial E_{(w,b)}}{\partial \mathrm{w}} = 2\left(w\sum_{i=1}^{m}x_i^2 - \sum_{i=1}^{m}(y_i - b)x_i\right) \tag{4.66}$$

$$\frac{\partial E_{(w,b)}}{\partial \mathrm{b}} = 2\left(mb - \sum_{i=1}^{m}(y_i - wx_i)\right) \tag{4.67}$$

然后令式（4.66）和（4.67）为零可得到 w 和 b 最优解的闭式(closed-form)解：

$$\mathrm{w} = \frac{\sum_{i=1}^{m} y_i(x_i - \bar{x})}{\sum_{i=1}^{m} x_i^2 - \frac{1}{m}\left(\sum_{i=1}^{m} x_i\right)^2} \tag{4.68}$$

$$\mathrm{b} = \frac{1}{m}\sum_{i=1}^{m}(y_i - wx_i) \tag{4.69}$$

其中$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$为 x 的均值。

更一般的情形是如本节开头的数据集 D，样本由 d 个属性描述。此时我们试图学得：

$$f(x_i) = w^T x_i + b, \text{使得} f(x_i) \approx y_i \tag{4.70}$$

这称为“多元线性回归”(multivariate linear regression)。

类似的，可利用最小二乘法来对 w 和 b 进行估计，为便于讨论，我们把 w 和 b 吸收入向量形式$\hat{w} = (w; b)$，相应的，把数据集 D 表示为一个 m×(d+1)大小的矩阵 X，其中每行对应于一个示例，该行前 d 个元素对应于示例的 d 个属性值，最后一个元素恒置为 1，即：

$$\mathrm{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_m^T & 1 \end{pmatrix} \tag{4.71}$$

再把标记也写成向量形式$\mathrm{y} = (y_1; y_2; \cdots; y_m)$，则类似于式（4.65），有：

$$\hat{w}^* = arg\min_{\hat{w}}(y - X\hat{w})^T(y - X\hat{w}) \tag{4.72}$$

令$E_{\hat{w}} = (y - X\hat{w})^T(y - X\hat{w})$，对$\hat{w}^*$求导得到：

$$\frac{\partial E_{\hat{w}}}{\partial \hat{w}^*} = 2X^T(X\hat{w} - y) \tag{4.73}$$

令上式为零可得$\hat{w}$最优解的闭式解，但由于涉及矩阵逆的计算，比单变量情形要复杂一些。下面我们做一个简单的讨论。

当X^TX为满秩矩阵(full-rank matrix)或正定矩阵(positive definite matrix)时，令式（4.73）为零可得：

$$\hat{w}^* = (X^TX)^{-1}X^Ty \tag{4.74}$$

其中$(X^TX)^{-1}$是(X^TX)的逆矩阵。令$\hat{x}_i = (x_i, 1)$，则最终学得的多元线性回归模型为：

$$f(\hat{x}_i) = \hat{x}_i^T(X^TX)^{-1}X^Ty \tag{4.75}$$

然而，现实任务中X^TX往往不是满秩矩阵。例如在许多任务中我们会遇到大量的变量，其数目甚至超过样例数，导致 X 的列数多于行数，X^TX显然不满秩。此时可解出多个$\hat{w}$，它们都能使均方误差最小化。选择哪一个解作为输出，将由学习算法的归纳偏好决定，常见的做法是引入正则化(regularization)项。

线性模型虽简单，却有丰富的变化。例如对于样例(x，y)，y ∈ R，当我们希望线性模型（4.63）的预测值逼近真实标记 y 时，就得到了线性回归模型。为便于观察，我们把线性回归模型简写为：

$$\mathrm{y} = w^Tx + b \tag{4.76}$$

可否令模型预测值逼近 y 的衍生物呢？譬如说，假设我们认为示例所对应的输出标记是在指数尺度上变化，那就可将输出标记的对数作为线性模型逼近的目标，即

$$\ln y = w^Tx + b \tag{4.77}$$

这就是“对数线性回归”(log-linear regression)，它实际上是在试图让e^{w^Tx+b}逼近 y。式（4.77）在形式上仍是线性回归，但实质上已是在求取输入空间到输出空间的非线性函数映射，如图 4.37 所示，这里的对数函数起到了将线性回归模型的预测值与真实标记联系起来的作用。

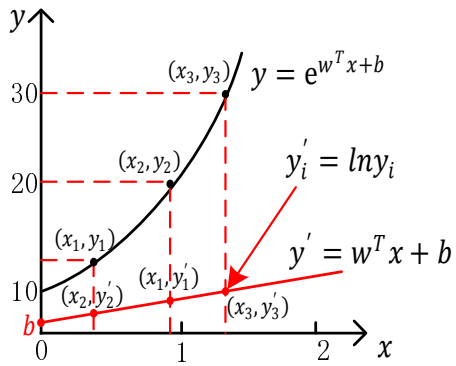

图 0.37 对数线性回归示意图

更一般地，考虑单调可微函数g(·)，令

$$\mathrm{y} = g^{-1}(w^Tx + b) \tag{4.78}$$

这样得到的模型称为“广义线性模型”(generalized linear model)，其中函数g(·)称为“联系函数 " (link function)。显然，对数线性回归是广义线性模型在$g(\cdot) = ln(\cdot)$时的特例。

3. 用于回归的线性模型

下列代码可以在一维 wave 数据集上学习参数 w 和 b：

```
import mglearn
mglearn.plots.plot_linear_regression_wave()

‘’’
运行以上程序可以得到输出:
w[0]: 0.393906    b: -0.031804
```

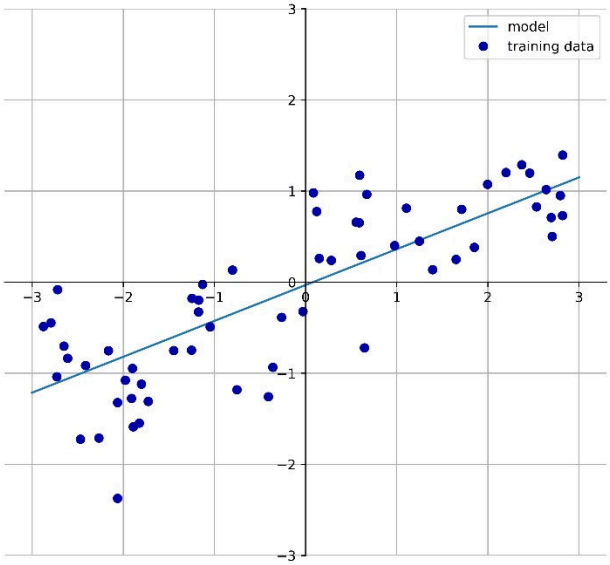

图 4.38 线性模型对 wave 数据集的预测结果

我们在图中添加了坐标网格，便于理解直钱的含义。从 w 可以看出，斜率应该在 0.4 左右，在图像中也可以直观地确认这一点。截距是指预测直线与 y 轴的交点：比 0 小，也可以在图像中确认。

用于回归的线性模型可以表示为这样的回归模型：对单一特征的预测结果是一条直线，两个特征时是一个平面，成者在更高维度（即更多特征）时是一个超平面。

如果将直线的预测结果与图 4.23 中 KNeighborsRegressor 的预测结果进行比较，你会发现直线的预测能力非常受限。似乎数据的所有细节都丢失了。从某种意义上来，这种说法是正确的，假设目标 y 是特征的线性组合，这是一个非常强的（也有点不现实的）假设。但观察一维数据得出的观点有些片面。对于有多个特征的数据集而言，线性模型可以非常强大。特别地，如果特征数量大于训练数据点的数量，任何目标 y 都可以（在训练集上）用线性函数完美拟合。

有许多不同的线性回归模型。这些模型之间的区别在于如何从训练数据中学习参数 w 和 b，以及如何控制模型复杂度。下面介绍最常见的线性回归模型。

4. 线性回归（又名普通最小二乘法）

线性回归，或者普通最小二乘法(ordinary least squares，OLS)，是回归问题最简单也最经典的线性方法。线性回归寻找参数 w 和 b，使得对训练集的预测值与真实的回归目标值 y 之间的均方误差最小。均方误差(mean squared error)是预测值与真实值之差的平方和除以样本数，线性回归没有参数，这是一个优点，但也因此无法控制模型的复杂度。

下列代码可以生成图 4.32 中的模型：

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import mglearn

X,y = mglearn.datasets.make_wave(n_samples=60)
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=42)
lr = LinearRegression().fit(X_train,y_train)
'''
```

“斜率”参数（w，也叫作权重或系数）被保存在 coef_属性中，而偏移或截距（b）被保存在 intercept_属性中：

```
print("lr.coef_:{}".format(lr.coef_))
print("lr.intercept_:{}".format(lr.intercept_))

'''
运行以上程序可以得到输出:
lr.coef_:[0.39390555]
lr.intercept_:-0.031804343026759746
```

intercept_属性是一个浮点数，而 coef_属性是一个 NumPy 数组，每个元素对应一个输入特征。由于 wave 数据集中只有一个输入特征，所以 lr.coef_中只有一个元素。

我们来看一下训练集和测试集的性能：

```
print("Training set score:{:.2f}".format(lr.score(X_train,y_train)))
print("Test set score:{:.2f}".format(lr.score(X_test,y_test)))

'''
运行以上程序可以得到输出:
Training set score:0.67
Test set score:0.66
```

R^2约为 0.66，这个结果不是很好，但我们可以看到，训练集和测试集上的分数非常接近。这说明可能存在欠拟合，而不是过拟合，对于这个一维数据集来说。过拟合的风险很小，因为模型非常简单（或受限）。然而，对于更高维的数集（即有大量特征的数据集），线性型将变得更加强大，过拟合的可能性也会变大，我们来看一下 LinearRegression 在更复杂的数据集上的表现，比如波士顿房价数据集。记住，这个数据集有 506 个样本和 105 个导出特征。首先，加载数据集并将其分为训练集和测试集。然后像前面一样构建线性回归模型：