TO MACHINE LEARNING

机器学习导论

[美] 米罗斯拉夫·库巴特（Miroslav Kubat）◎著
王勇　仲国强　孙鑫◎译

人工智能专家**米罗斯拉夫·库巴特教授**25年倾心打造

系统解读了有关机器学习的14个方面，快速读懂机器学习

全面揭开机器学习的奥秘

这本书通过大量容易操作的实践指导、容易理解的经典案例、容易激发学习兴趣的讨论等多种方式，介绍了机器学习的基本思想。本书的内容既包括贝叶斯分类器、最近邻分类器、线性和多项式分类器、决策树、神经网络以及支持向量机等工具的介绍，又包含了如何把这些简单工具通过“提升”（Boosting）的方式结合起来，怎样将它们应用于更加复杂的领域，以及如何处理各种高级的实践问题等。书中还有一章专门介绍了广为人知的遗传算法。

AN INTRODUCTION
TO MACHINE LEARNING

机器学习导论

[美] 米罗斯拉夫·库巴特（Miroslav Kubat）◎著
王勇　仲国强　孙鑫◎译

机械工业出版社
CHINA MACHINE PRESS

本书通过给出易操作的实践指导、采用简单的案例、激励学生讨论有兴趣的应用问题，用一种易于理解的方式介绍了机器学习的基本思想。本书主题包括贝叶斯分类器、最近邻分类器、线性和多项式分类器、决策树、神经网络以及支持向量机，且展示了如何把这些简单工具通过“提升”（Boosting）的方式结合起来，怎样将它们应用于更加复杂的领域，以及如何处理各种高级的实践问题。书中对广为人知的遗传算法也做了介绍。

图书在版编目（CIP）数据

机器学习导论 /（美）米罗斯拉夫·库巴特（Miroslav Kubat）著；王勇，仲国强，孙鑫译. —北京：机械工业出版社，2016.9（2018.5 重印）
书名原文：An Introduction to Machine Learning
ISBN 978 - 7 - 111 - 54868 - 3

Ⅰ. ①机… Ⅱ. ①米… ②王… ③仲… ④孙… Ⅲ. ①机器学习—研究 Ⅳ. ①TP181

中国版本图书馆 CIP 数据核字（2016）第 222726 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策划编辑：坚喜斌　　责任编辑：坚喜斌　陈瑞文
版式设计：张文贵　　责任校对：赵　蕊
责任印制：张　博
三河市宏达印刷有限公司印刷
2018 年 5 月第 1 版·第 4 次印刷
170mm × 240mm · 20.5 印张 · 303 千字
标准书号：ISBN 978 - 7 - 111 - 54868 - 3
定价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务
服务咨询热线：（010）88361066
读者购书热线：（010）68326294
（010）88379203
封面无防伪标均为盗版

网络服务
机 工 官 网：www.cmpbook.com
机 工 官 博：weibo.com/cmp1952
教育服务网：www.cmpedu.com
金　书　网：www.golden-book.com

献给我的妻子，瓦卢卡（Verunka）

推荐序

机器学习是人工智能领域的一个重要分支，其研究涉及代数、几何、概率统计、优化、泛函分析、图论、信息论、算法、认知计算等多个学科的知识，其应用不仅仅限于模式识别、计算机视觉、数据挖掘、生物信息学、智能控制等科学和工程领域，甚至在社会科学领域的研究中也有应用，如管理学、经济学和历史学等。目前，随着计算机科学和智能科学技术的进步，机器学习得到了快速发展，其方法被广泛应用到了各个领域。尤其是近些年，深度学习方法快速发展并在多个领域展示出优异性能，使机器学习和整个人工智能领域受到极大的关注。

机器学习是基于已有数据、知识或经验来设计模型或发现新知识的一个研究领域。20 世纪 50 ~ 70 年代是机器学习研究的初期，人们基于逻辑知识表示试图给机器赋予逻辑推理能力，取得了很多振奋人心的成果；20 世纪 80 年代，专家系统受到高度重视，为专家系统获取知识成为一个重要方向。20 世纪 80 年代中后期，人工神经网络由于误差反向传播（BP）算法的重新提出和广泛应用而形成一股热潮，但其地位在 90 年代后期被以支持向量机为核心的统计学习理论所取代。20 世纪 90 年代以后，受重视的机器学习方法还有集成学习、概率图模型、半监督学习、迁移学习等。2006 年，以加拿大多伦多大学的 G. Hinton 教授为代表的几位研究人员在深度学习方面取得巨大突破，在 Google、Microsoft、Facebook 等科技公司的推动下，深度学习借助于大数据和高性能计算的有利条件得到了广泛应用和高度关注。目前，搜索引擎、机器人、无人驾驶汽车等高科技产品都依赖于机器学习技术。机器学习，特别是深度学习，在语音识别、人脸识别、围棋、游戏等方面已经超过了人类水平，可以想象机器学习与人类的生产、生活之间的关系将会越来越紧密。

过去几十年，机器学习领域也出现了一些经典的著作或教材。1983 年，R. Michalski、J. Carbonell 和 T. Mitchell 主编的《机器学习：一种人工智能方法》一书出版，标志着机器学习成为人工智能的一个独立研究领域。《Machine Learning》期刊创刊于 1986 年，目前依然是机器学习领域的顶级期刊。1990 年，J. Carbonell

主编的《机器学习：范式与方法》对归纳学习、基于解释的学习、遗传算法和连接主义学习等机器学习范式及方法进行了深入探讨。T. Mitchell 于 1997 年出版的《机器学习》是一本经典的机器学习教材，其中文版已于 2003 年由机械工业出版社出版，但因为出版年限较早，许多内容已没有时效性。1998 年，V. Vapnik 出版的《统计学习理论》是一本完整阐述统计机器学习思想的名著。2001 年出版、2009 年再版的《统计学习基础：数据挖掘，推理和预测》是美国斯坦福大学教授 T. Hastie，R. Tibshirani 和 J. Friedman 的一部力作，其中对最为流行的机器学习方法进行了全面而深入的介绍，因其严谨的数学推导，该书不失为机器学习研究进阶的很好的读物。E. Alpaydin 所著的《机器学习导论》出版于 2004 年并于 2010 年再版，书中对基础的机器学习方法进行了介绍，是一本机器学习入门的很好的教材。C. Bishop 所著的《模式识别与机器学习》和 K. Murphy 所著的《机器学习：一个概率的视角》分别于 2006 年和 2012 年出版，两本书都从概率的角度全面而细致地介绍了许多经典的机器学习模型。C. Bishop 的《模式识别与机器学习》可帮助读者打下坚实的机器学习基础，而 K. Murphy 的书则相对介绍了更多较新的机器学习算法，甚至有一章专门介绍了深度学习方法。2012 年，李航老师出版了《统计学习方法》，2016 年，周志华老师出版了《机器学习》。这两本书中，《统计学习方法》主要集中于几种重要机器学习模型的介绍，而《机器学习》内容相对更加全面，深入浅出，堪称机器学习的中文经典著作。相对于以上这些机器学习书籍，M. Kubat 所写的这本《机器学习导论》更像是一本科普性质的读物，作者尽量避开复杂的数学公式，用生动形象的方式介绍机器学习算法，而且本书篇幅适当，又涵盖了几乎所有基本的机器学习方法，使得本书不仅适合作为本科学生机器学习课程的教材，也适合于想了解机器学习入门知识的普通读者。

本书的译者都是工作在机器学习教学与研究第一线的年轻老师，其中仲国强副教授过去是我的博士研究生，在模式识别和机器学习领域都有很扎实的研究基础。相信本书的中译本对于国内机器学习的教学和研究都会有所裨益，也为更多的人，尤其是初学者了解机器学习打开一扇门。

中国科学院自动化研究所副所长、模式识别国家重点实验室主任

刘成林

前　言

目前，机器学习慢慢走向成熟。你可能觉得这只是老生常谈，请让我做一个详细说明。

人们希望机器某一天能够自己学习，这个梦想几乎在计算机出现时就有了，也许更早。不过，长久以来，这仅仅是一个想象而已。罗森布拉特（Rosenblatt）感知器的提出曾经掀起过一股热潮，但是现在回想起来，这股热潮没能持续很长的时间。至于接下来的尝试，使情况发展得更糟糕，这个领域甚至没有再引起人们的注意，长期被忽视，因而无法取得重大突破，也没有这一类的软件公司，后续研究寥寥无几且得到的资金支持也不多。这个阶段，机器学习一直不被看好，像进入休眠期一样，在其他成功学科的阴影里生存。

然而，接下来发生的一切使这些颓势彻底改变了。

一群有识之士指出，在 20 世纪 70 年代的人工智能领域，基于知识的系统曾经风靡一时，但它们有一个弱点：“知识”从哪里来？当时主流的观点认为，应该让工程师和领域专家合作，用 if-then 的形式表示出来。但是实际情况差强人意，专家们发现很难把掌握的知识表达给工程师。反过来，工程师也不知道该问什么问题以及如何表示答案。尽管有几个广为人知的成功案例，但是其他大多数研究都试图建立知识库，并且成千上万的规则令人沮丧。

这些有识之士主张简单和直接的操作。如果难以准确地告诉机器如何处理某个问题，那么为什么不间接地给出指令，通过例子展示所需要的技能，计算机将通过这些例子来学习！

当然，这必须要有能够进行学习的算法才有意义，这也是困难所在。无论是罗森布拉特的感知器还是后来出现的技术都不太管用。然而，机器学习在技术方面的缺乏算不上是障碍，相反是一个挑战，并激发出了很多绝妙的点子。其中，使计算机有学习能力这个想法开创了一个激动人心的新领域，并引起了世人的

关注。

这一想法在1983年爆发了。一卷很厚的论文集——《机器学习：人工智能的方法》[㈠]中提出了很多各式各样的方法来求解这个谜题。在它的影响下，几乎一夜之间一个新的学科诞生了。3年后，后续著作一本接一本地出现。相关学术刊物也很快被创立，有着巨大影响力的年度学术会议相继召开。几十、或许是几百篇博士论文完成并通过答辩。

早期阶段，问题不仅是如何学习，而是学什么和为什么学。这段充满创造力的岁月让人难以忘怀，唯一有些遗憾的是很多非常好的想法后来被放弃了。实用主义占了上风，资源都被投向那些最有希望的方向。经过一段时间的发展，具体研究基本成形：知识系统if-then规则的归纳，分类归纳，程序基于经验来提高技能，Prolog程序自动调优，以及其他方面。相关的研究方向非常多，一些知名学者希望通过写书来引领未来的发展，这其中有些人做得很成功。

机器学习发展的一个重要的转折点是汤姆·米切尔（Tom Mitchell）的传奇教科书[㈡]。该书向博士生和科学家们总结了该领域的发展现状，慢慢地大学也用这本书作为研究生的教材。同时，研究方法也变得更加系统化。大量机器学习测试库被建立起来，用于比较性能或者学习算法的优劣。统计评估方法也被广泛地使用在评估过程中。相关流行程序的公开版本很容易获得，从事这个学科的人数增至数千，甚至更多。

现在，到了很多大学都为本科生开设机器学习课程的阶段。通常这些课程需要不同类型的教材。除了掌握基本技术以外，学生还需要了解不同方法的优点和缺点，以及不同情况下每种方法的独特之处。最重要的是，他们需要理解在特定情况下，哪些技术是可行的，哪些是不可行的。只有这样才能在解决具体问题时做出正确的选择。一本教材除了满足以上的各项要求外，还应该介绍一些数学概念，多包括一些实用的建议。

㈠ 米切尔斯基（R. Michalski），卡波内尔（J. Carbonell），米切尔（T. Mitchell）编辑。

㈡ T. Mitchell. Machine Learning［M］. New York: McGraw-Hill，1997.

关于教材，还要考虑材料的多少、结构以及风格，以便能够支持一个学期的导论课程。

第一个问题是材料的选择。当高科技公司准备成立机器学习研究团队时，大学就要向学生传授相应的知识和技能，以及对有关行业需求的理解。出于这个原因，本书重点介绍了贝叶斯分类器，最近邻分类器，线性和多项式分类器，决策树，神经网络的基础，以及提升（Boosting）算法的原理。本书用很大篇幅来描述具体应用的典型特征。在现实中，当面对有一定难度的任务时，一些基本方法和老师在实验环境下演示的结果可能不完全一样。因此在学习过程中，学生必须知道每种方法会发生什么。

本书共包括 14 章，每章覆盖一个专题。各章分成很多个小节，每节介绍一个关键问题。建议学生在做完每一节后面的 2 ~ 4 个“控制问题”后再学习下一节。这些问题用来帮助检查学生对学习材料的掌握情况。如果不会做这些题，则有必要重新阅读相关内容。

俗话说，实践出真知。每章结尾安排了必要的练习用于实际操作。如果接下来的思考实验能够全部完成，将有助于更深入地理解所学内容的各个方面。不过这些实验难度较大，只有付出很大努力才能获得正确的答案。所学的知识在上机实验中可被进一步巩固。编程对于学习同样也很重要。现在，人们都习惯从网上下载所需的程序，这是捷径，但本书不建议这样做，因为只有具体实现了程序的全部细节，才能领会机器学习技术的精妙之处。

目　录

机器学习导论

第1章　一个简单的机器学习任务

你会发现精确地描述你母亲的相貌，让朋友能在超市里认出她，是很难的。但如果你给他看几张你母亲的照片，他就能立刻找出所需要的特征。这就是人们常说的，一张图片，也就是一个样例，胜过千言万语。

这就是我们希望用技术去实现的。当不能足够精确地定义某些事物或概念时，我们希望以样例的方式把它们传输给机器。然而，计算机必须能将样例转换成知识才能奏效。所以，我们的兴趣在于机器学习（machine learning）的算法和技术，也是这本书的主题。

第 1 章将任务表示为一个搜索问题，并介绍了爬山搜索算法。爬山搜索算法不仅是我们解决机器学习任务的初步尝试，而且也为解决后面几章中的一些辅助问题提供便利的工具。在这些基础上，我们将继续探索一些能使学习过程苦中带乐的问题，包括性能标准、实验方法以及其他方面。

1.1 训练集和分类器

让我们首先介绍一些贯穿全书的问题和基本概念。

预分类训练样例。图 1.1 展示了 6 种乔尼（Johnny）喜欢和不喜欢的派。这些正例（positive examples）和负例（negative examples）的基本概念构成了一个训练集（training set），并以此由机器归纳出一个分类器（classifier）——一种能将今后任何的派归为正、负两个类别之一的算法。

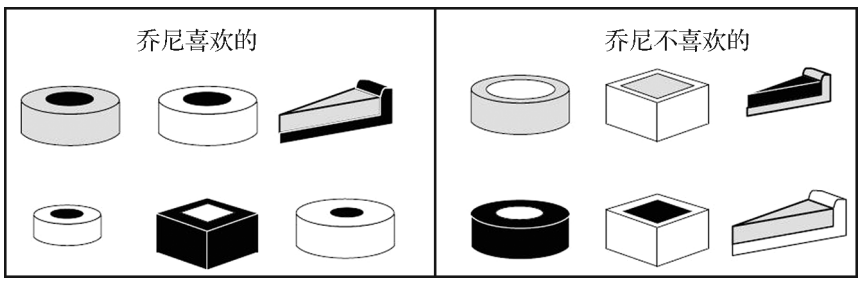


图 1.1　一个简单的机器学习任务：归纳学习出一个能将今后任何派归类为正例和负例的分类器，如归纳学习一个“乔尼喜欢的派”的分类器

毫无疑问，分类的数量可以更多。例如，一个分类器要有春、夏、秋、冬 4 个分类，才能辨别出一张风景照拍摄于哪个季节。iPad 上识别手写字迹的软件则至少需要 36 个分类：26 个字母和 10 个阿拉伯数字，文件分类系统可以识别成百上千种不同的主题。我们之所以只选择二分类域，只是因为它简单。

属性向量。为将训练样例（example）发送给机器，我们必须用恰当的方法描述它们。最常见的方法是使用所谓的**属性**（attributes）。在“派”问题域中，共有 5 种属性：形状（shape），取值包括圆形（circle）、三角形（triangle）和方形（square）；外壳尺寸（crust size），取值包括厚（thick）和薄（thin）；外壳色度（curst shade），取值包括白色（white）、灰色（gray）和深色（dark）；馅料尺寸（filling size），取值包括厚（thick）和薄（thin）；馅料色度（filling shade），取值包括白色（white）、灰色（gray）和深色（dark）。表 1.1 指定了图 1.1 中 12 个训练样例的属性值。例如，图 1.1 中左上角的派（在表 1.1 中称之为 ex1）可用下列组合属性描述：

$$(\text{shape} = \text{circle}) \text{ AND } (\text{crust} - \text{size} = \text{thick}) \text{ AND } (\text{crust} - \text{shade} = \text{gray})$$
$$\text{AND } (\text{filling} - \text{size} = \text{thick}) \text{ AND } (\text{filling} - \text{shade} = \text{dark})$$

表 1.1　以矩阵形式表示的 12 个训练样例

样例	形状（shape）	外壳（crust）		馅料（filling）		类别
		尺寸（size）	色度（shade）	尺寸（size）	色度（shade）	
ex1	circle	thick	gray	thick	dark	pos
ex2	circle	thick	white	thick	dark	pos
ex3	triangle	thick	dark	thick	gray	pos
ex4	circle	thin	white	thin	dark	pos
ex5	square	thick	dark	thin	white	pos
ex6	circle	thick	white	thin	dark	pos
ex7	circle	thick	gray	thick	white	neg
ex8	square	thick	white	thick	gray	neg
ex9	triangle	thin	gray	thin	dark	neg

（续）

样例	形状（shape）	外壳（crust）		馅料（filling）		类别
		尺寸（size）	色度（shade）	尺寸（size）	色度（shade）	
ex10	circle	thick	dark	thick	white	neg
ex11	square	thick	white	thick	dark	neg
ex12	triangle	thick	white	thick	gray	neg

归纳分类器。我们用训练集组成的输入来归纳分类器，但这是什么形式的分类器呢？假定我们让它符合布尔函数的形式，正例为真（true），负例为假（false）。当用训练集来检查表达式［（shape = circle） AND （filling-shade = dark）］时，我们会发现，所有的负例值都为假：因为在负例中，当样例形状为圆形时，没有灰色的馅。然而在正例中，有 4 个为真，其余两个为假。这说明分类器犯了两个我们可能无法容忍的错误，从而我们猜测存在更好的解决办法。

事实上，读者很容易证明下面的表达式在整个训练集中都不会出错：

$$[(\text{shape} = \text{circle})\,\text{AND}(\text{filling} - \text{shade} = \text{dark})]\,\text{OR}$$
$$[\text{NOT}(\text{shape} = \text{circle})\,\text{AND}(\text{crust} - \text{shade} = \text{dark})]$$

蛮力方法的问题。机器发现这种分类器会如何？蛮力（计算机最擅长的事情）在这里行不通。设想一下，用“派”问题中特定的属性可以识别出多少个不同的样例？对于 3 种不同的形状，每种形状就有两个不同的外壳尺寸可供选择，那么其组合数量为 $3 \times 2 = 6$。对这 6 种组合的每一种的下一个属性，即外壳色度，有 3 种不同的值，此时组合数量变为 $3 \times 2 \times 3 = 18$。将所有属性以此类推，会发现样例空间包含 $3 \times 2 \times 3 \times 2 \times 3 = 108$ 个不同的样例。

这些样例的每个子集——一共有 2^{108} 个子集！可能包含了代表某些人“好的派”概念的正例列表，且每个子集至少由一个布尔表达式来描述。显而易见，用训练集来检验每一个分类器是天方夜谭。

人工方法及搜索。虽然我们不确定如何创造出一个归纳分类器算法，但是可以尝试“手动地”发明一种分类器，并通过古老但好用的纸笔测试法来寻找灵

感。在这个过程中，我们先从试探性的初始版本入手，例如，shape = circle。用训练集检验该表达式后，我们发现 4 个正例和两个负例为真。显然，分类器需要“缩小”（特殊化）来排除那两个负例。特殊化的一个方法是添加连接词，例如，把 shape = circle 变成［(shape = circle) AND (filling – shade = dark)］。新的表达式虽然使所有负例都为假，但依旧有缺点，因为它只包含了 6 个正例中的 4 个（即 ex1、ex2、ex4、ex6）。因此，下一步应尝试一般化，如添加分离词：｛［(shape = circle) AND (filling – shade = dark)］OR (crust – size = thick)｝。我们不停地重复这个过程，直到找到一个 100% 准确的分类器为止（如果存在）。

这种自省法告诉我们，分类器可以通过一系列特殊化以及一般化的步骤，通过逐步调整一个给定的分类器来创建，直到符合某个预定义的要求。这对后期的研究极具鼓励意义。了解人工智能的读者会把这个过程看作在布尔表达式空间里的搜索。人工智能发展和探索了很多这样的搜索算法，建议对其中一些算法做一个初步了解。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在刚才的讨论中，什么是学习问题的输入和输出？
- 我们是如何描述训练样例的？什么是样例空间？可以计算它的大小吗？
- 在“派”域中找出一个布尔表达式，将表 1.1 中的所有训练样例正确分类。

1.2 一点题外话：爬山搜索

现在，我们把讨论过的搜索问题形式化，并介绍一种流行的算法，也就是所谓的爬山法（hill climbing）。人工智能将搜索定义为：从初态（initial state）开始，找到一系列步骤，通过一组中间的搜索状态（search states），到达预定义的

终态（final state）。从一种搜索状态过渡到另一种，每一步都由程序员预设的搜索算符（search operators）来完成。搜索算符应用的步骤遵循特定的搜索策略（search strategy），图 1.2 描述了这一原理。

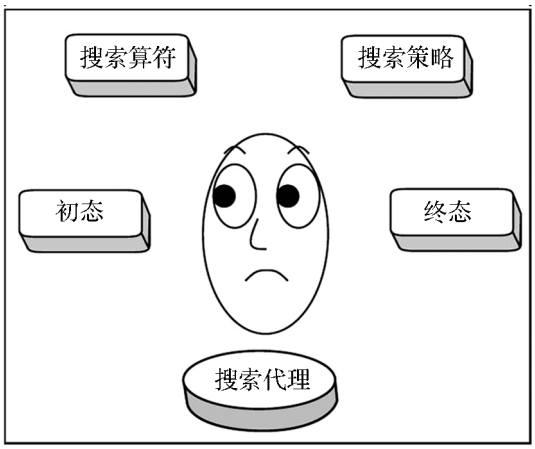


图 1.2　由初态、终态、搜索算符和搜索策略形式化的搜索问题

爬山法——一个示例。爬山法是一种流行的搜索策略。我们要用著名的脑筋急转弯游戏，也就是数字滑块拼图来说明爬山法的本质。该游戏通常的玩法是：木板上有排成 3 行 3 列的 9 个方格，其中 8 个放有用数字编号的滑块（整数 1 ~ 8），最后一个方格没有放置数字滑块。通过滑动邻近滑块到空方格里来转变搜索状态。游戏的最终结果是达到一个预指定的滑块排列。在图 1.3 所示的流程图中，我们从一个具体的初态开始，并在两个搜索算符里进行选择："上移滑块 6" 和 "左移滑块 2"。此时，可以借助评估函数（evaluation function），通过估计每个状态与目标的距离来指导选择。最简单的评价形式可能是数出滑块在到达最终目的地时所要滑过的方格数。在初态中，滑块 2、4 和 5 已经处于正确的位置；滑块 3 必须滑过 4 个方格；滑块 1、6、7、8 每个均要滑过两个方格。因此，距离总和为 $d = 4 + 4 \times 2 = 12$。

在图 1.3 中，两种搜索算符都能使初态到达一个距离终态 $d = 13$ 的状态。在缺乏其他的指导时，我们随机选择左移，使得空方格在第一行的中间位置。这时有 3 种移动可能，其中一种只能让我们返回初态，因此可以忽略掉。

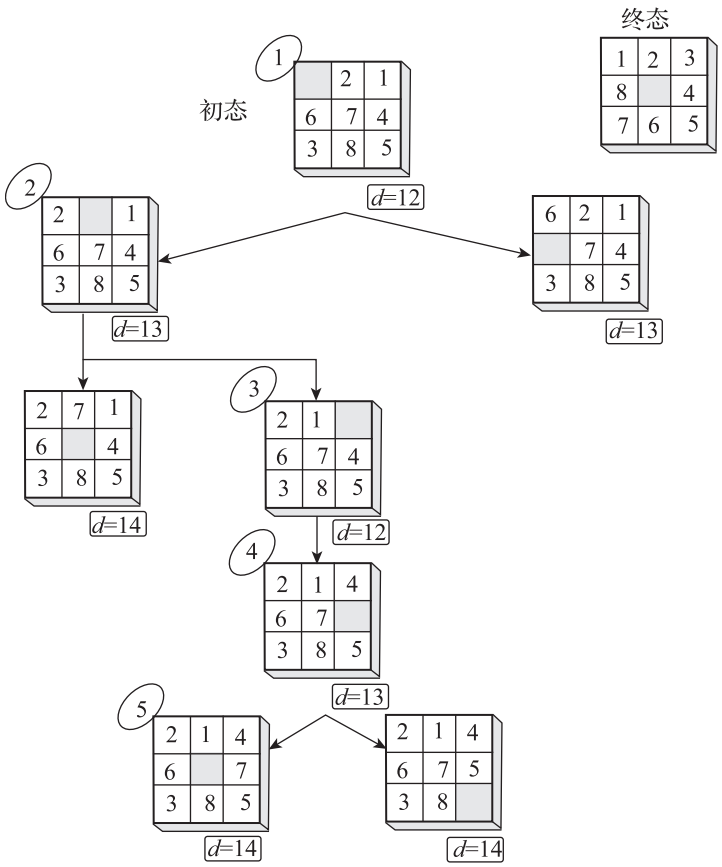


图 1.3　爬山法。圈出的整数代表搜索状态改变的顺序，*d* 是用特定评估函数计算出的该状态距终态的距离。遇到多选项（此处为两选项）则随机进行选择

至于剩下的两种可能性，一种能达到 $d = 14$ 的状态，另一种能达到 $d = 12$ 的状态。我们选择数值较小的后者。下一步很简单，只有一种移动方式能到达一个新的状态。之后，我们又面临着两种选择……持续这个搜索过程直到达到终态为止。

可选的终止标准和评估函数。我们也可以考虑其他的终止标准（termination criteria）。当超过最大期限时（我们不想让计算机一直运行下去）、当已知状态数量超过了某个界限时、当我们已经发现某种十分接近终态的状态时、当我们发现所有的状态都出现过时，诸如此类，这些标准都可以用来终止搜索过程。

实际的终止公式能反映特定应用的决定因素，并且有时需要结合上述的两个或多个标准。

另外，在数字滑块例子中用到的评估函数十分简单，刚够完成任务：让使用者把自己对问题的理解转化成一些概念，给那些需要的解决问题者提供线索。为了顺利解决实际问题，我们不得不想出一个更加缜密的函数。通常，人们可以设计出各种各样的方案，每个方案的步骤次序都不尽相同。有些很快就能找到解决办法，其他的则绕了远路。此时，搜索程序的性能取决于程序员正确选择搜索方案的能力。

爬山算法。表 1.2 中的伪代码总结了爬山算法。当然，细节部分取决于每个人设计程序的风格，但不同风格的代码几乎总会不约而同地包含一些经典的函数。其中一个是比较两种状态，如果相同则返回真；这便是程序确定达到终态与否的方法。另一个函数选取一个特定的搜索状态，并将其应用到所有搜索算符中，因此创造了一套完整的“子状态”。为避免无限循环，要有第三个函数来检验一个状态是否已经测试过了；第四个函数则负责计算给定状态距终态的距离；第五个会根据计算出的距离分类“子”状态；并把它们放在列表 L 的前端；最后一个函数检验状态是否符合终止条件[㊀]。

表 1.2　爬山搜索算法

1. 创建两个列表，L 和 L_{seen}。最初，L 只包含初态，L_{seen} 是空的
2. 令 n 是表 L 中的第一个元素，并与终态进行比较，不停地比较直到两者一致
3. 将 n 应用到所有可用的搜索算符中，从而得到了一系列新的状态，去除表 L_{seen} 中已经存在的状态，剩余的状态用评估函数分类，并置于表 L 的前端
4. 将 n 从表 L 中转移到表 L_{seen} 中，表 L_{seen} 中都是检验过的状态
5. 如果 $L=\varnothing$，则停止运算并报告错误；否则，返回第 2 步

最后一点需要注意的是：在图 1.3 的一些状态下，没有任何“子”状态可以改进“父”状态，只有在暂时的妥协后才能得到一个低值的 d。一个登山者也可

㊀ 简单来说，伪代码忽视了终止条件，因而没有达到终态。

能经历过这种情况：有时他不得不横穿一个山谷，才能继续攀登。顺便提一下，这个方法的名字源于登山的隐喻。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 人工智能是如何定义搜索问题的？我们如何理解“搜索空间”和“搜索算符”？
- 评估函数起怎样的作用？它是如何影响爬山搜索算法的？

1.3 机器学习中的爬山法

我们要探索一个具体的方法，将爬山法运用到机器学习中。

爬山法和乔尼的派。让我们从如何确定乔尼喜欢哪一个派的问题入手。输入一系列训练样例，并用可用属性描述每一个样例。输出——即终态，是一个布尔表达式，使所有训练集中的正例为真，负例为假。这个表达式包含了属性 - 值对、逻辑运算符（连接词、分离词和否定）和所需的括号。用评估函数衡量特定表达式在训练集中的错误率。就初态而言，任何随机生成的表达式都可以使用。在图 1.4 中，因为超过半数的正训练样例是圆形的，所以我们选择“shape = circle”作为初态。

至于搜索算符（search operator），一种方法是添加连接词，图 1.4 中也给出了说明：例如，最左边的子结点用［(shape = circle) AND (filling - shade = dark)］(见图 1.4，通常用“^”表示 AND）来替换“shape = circle”。注意，在玩具问题域中连接算符生成了多少种不同的表达式。任何其他属性 - 值对都可以连接“shape = circle”。因为剩余 4 个属性（除去 shape）分别有 2、3、2、3 个不同的取值，所以可以连接到“shape = circle”上的条件的总数为 $2 \times 3 \times 2 \times 3 = 36$[㊀]。或者，像最左边子结点的 3 种扩展所证明（见图 1.4）的一样，我们也可以添加分

㊀ 图 1.4 只展示了 36 个新建状态中的 3 个。

离词。其他算符的例子包括“去掉一个连接词”“去掉一个分离词”“加上一个否定词”“否定一个条件”，以及各种各样的括号调整等。总之，有几百种的搜索算符可以用到每一种状态及其产生的后续状态中。即使在非常简单的问题域中也难以管理。

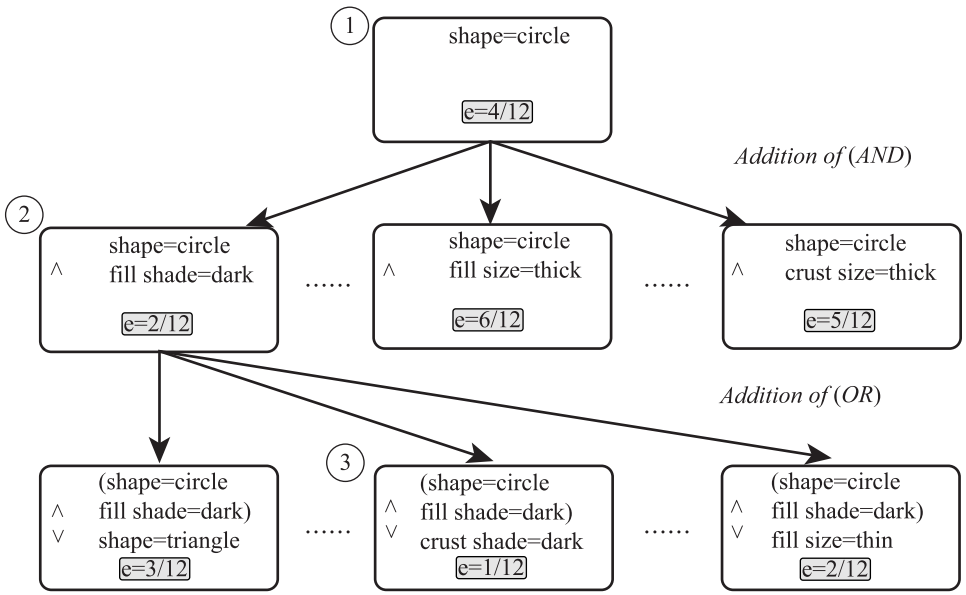


图 1.4　“派”域中的爬山搜索法

数值属性。在“派”问题域中，每一种属性取一些离散值中的一个值，但在实践中，一些属性可能是数值。例如，每个派都有价格，价格属性的值在连续域中。那么搜索会有什么不同呢？

为方便起见，假设只有两种属性：重量和价格。这样就可以用平面上的一个点来表示每个训练样例，如图 1.5 所示。读者可以发现，属于同一分类的样例倾向于分布在一个特定的区域上，用来划分单个区域的曲线从数学上可以被定义为直线、圆、多项式。例如，图 1.5 的右侧有 3 个不同的圆，每个圆都可以充当一个分类器：在圆中的样例被判断为正；否则判断为负。而且，在这些分类器中，有些分类器性能更优。那么，爬山法是如何找出这些最优的分类器的呢？下面是一种可能的情况。

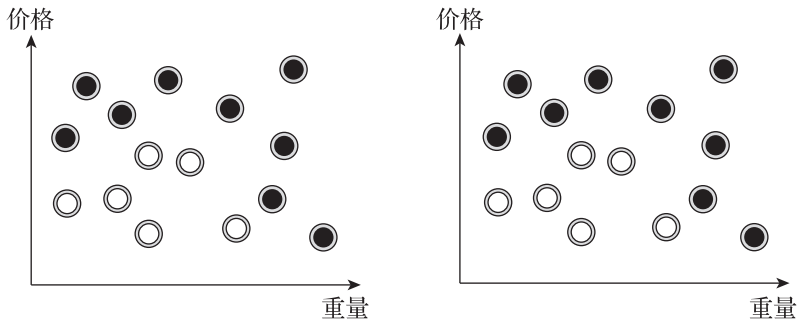


图 1.5　左边：具有连续属性的域；右边：一些“圆形”分类器

数值属性域中的爬山法。

初态（Initial State）。圆心和半径决定一个圆。我们可以随机挑选一个正例作为初态的圆心，并使最初的半径小到只能包含它本身这一个样例。

搜索算符（Search Operators）。这里可以使用两种搜索算符：一种可以增大圆的半径，一种可将圆心从一个训练样例换成另一个。在前一个搜索算符中，我们还需要决定将半径增大多少。一种观点认为，增大圆的半径到仅能再多容纳另外一个训练样例。起初，圆中只有一个训练样例。第一步过后，将会有两个、三个、四个等。

终态（Final State）。圆形并不是代表正域的理想形状。在这个问题中，我们找不到完全精确的形状，并且我们更愿意将终态定义为“能正确分类 95% 训练样例的分类器”。

评估函数（Evaluation Function）。正如之前所提及的，我们选择的评估函数要使错误率最低。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在机器学习中使用爬山法前，我们必须明确有关搜索的哪些方面?
- 哪些搜索算符可以用在“派”域和“圆”域中？如何定义评估函数、初态和终态?

1.4 分类器的性能

目前为止，我们通过比较已知分类的训练样例的类别和分类器给出的样例类别，来衡量错误率。实际上，我们的目标并不是重新分类那些类别已知的对象，而是对未来样例（future examples）进行分类，因为目前我们对其类别一无所知。我们通过实验来评估分类器的预期性能，并且有必要了解相应的评估原理。

独立的测试样例。最简单的做法就是将预分类样例分成两部分：训练集（training set），用来归纳分类器；测试集（testing set），用来评估样例（见图 1.6）。在“派”域中，有 12 个预分类样例，随机选择 8 个来归纳分类器，并检验剩余 4 个。如果分类器“猜”对了 3 个测试样例的分类（错了 1 个），那么其性能评估为 75%。

图 1.6 预分类样例被分为训练集和测试集

这种方法虽然看起来很合理，但存在一个很大的缺陷：随机挑选出的 8 个训练样例可能不足以代表所属的概念——对测试集（样例数甚至更少）也是如此。如果我们用蓝鲸、海豚和鸭嘴兽这组训练样例来归纳哺乳动物的定义，那么可能会误导学习者以为哺乳动物住在海里（蓝鲸、海豚），有的还会下蛋（鸭嘴兽），这个观点对生物学家来说是没办法接受的。但是，选择另一组训练样例可能会导致选择器符合最高标准。关键是，不同的训练集和测试集的划分产生不同的分类器，同样对未来性能的评估也不同。如果预分类样例数很少，那么这种现象就会更加明显。

如果我们想比较两种机器学习算法归纳出的产品质量，所谓的随机子抽样（random subsampling）可以缓解无代表性训练集的问题[㊀]。方法是多次随机划分样例，分成训练集和测试集，总是从第 i 组训练集归纳分类器，并在第 i 组测试集中测量错误率 Ei。认为产生较低 Ei 平均值的分类器的算法更优——它考虑了分类性能。

㊀ 后面我们将介绍其他的方法。

做解释的需要。在某些应用中，只给每一个样例确定分类是不够的，我们还想知道这样分类的理由。同样，如果一个外科医生仅仅用“我们的计算机是这样说的”来说服一个病人接受截肢手术，那么病人是不会同意的。但如何找到一个更好的解释呢？

在“派”问题域中，我们可以从布尔表达式本身收集到很多信息。例如，我们或许注意到了，形状为正方形且馅是白色的派被标记为负。结合这个观察结果和其他知识能启发我们：深色的馅可能象征着深红色，这是乔尼喜欢的；或者外壳是圆形的派比正方形的派口感更松脆等。比起分类本身，人们更想要用这种方法获得知识。

比较而言，在“圆”问题域中的分类器是一个数学表达式，它接收样例描述信息并仅仅反馈类标签，像一个“黑盒”，这未必是缺点。在一些应用中，解释只是一种补充；在其他的情况下，则显得多余。考虑这样一个分类器，它接收了一张手写文字的数字图像，就能反馈出图像所代表的字母。现在，用户想要使用该分类器将几页文本转化成为 Word 文档，但不会纠结于每个字符是如何转化的。

其他的解决方法。顺便说一句，我们应当注意到给定的数据可以归纳出**很多**明显完美的分类器。在“派”问题域中，训练集有 12 个样例，其余 96 个样例的分类未知。数学家会告诉我们有 2^{96} 个分类器能正确标记所有训练样例，但对 96 个未知样例的标记方式却各不相同。也许存在一个分类器可以正确标记未来的每一个样例——也存在另一个分类器可能标错所有样例。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在学习过程中没见过的样例，我们如何估计关于它的错误率？
- 为什么测试集上的错误率通常高于训练集的错误率？
- 给出一个域的例子，分类器必须解释自身的行为，再给出一个分类器不需要解释的域。
- “可以正确分类所有训练样例的分类器数目是一个组合数”是什么意思？

1.5 可用数据的困难

在一些应用中，训练集是手工产生的：专家准备一个样例，用类标签标记，选择属性，并明确每个样例中每个属性的值。在其他问题域中，过程是计算机化的。例如，公司想要预测员工离开的意愿。他们的数据库里有每个人的地址、性别、婚姻状况、职位、加薪、升值，还有员工是否在公司任职，如果不在，他们是哪天离开公司的。从这些信息中，程序可以获得属性向量，如果给定的员工自上次数据库更新后 1 年内离开公司，则标记为正。有时，属性向量能自动从数据库中提取，并被专家标记。或者，样例的一部分可以从数据库中获得，其余的则手工添加。通常，将两个或更多的数据库结合起来。诸如此类的变量其数量几乎是无限的。但不论样例来自何处，它们都可能存在一些缺陷，工程师必须理解这些缺陷的本质和产生的后果。

不相关属性。首先，有些属性十分重要，另一些则不是。当乔尼真的很喜欢深红色的馅时，这种喜好不会被厨师的鞋码所影响。这是需要考虑的问题：不相关（irrelevant）属性增加了计算开销，甚至有可能误导学习者。那么我们如何避免这个问题呢？

通常是无法避免的。当然，在一些手工生成的问题域中，专家应该知道哪些属性是真正起作用的，但在这里，情况并非那么简单。因此，在作者的“派”域中，他已经尽其所能地挑选那些他认为有用的属性。但由于不确定造成乔尼口味的真正原因，他可能会包含一些他认为必要的属性，但不能保证。相关性问题在从数据库中提取出的样例上发生得更加频繁。数据库产生的初衷是提供获得大量信息的渠道——通常其中只有一小部分适用于学习任务。至于是哪些部分，我们不得而知。

属性缺失。相反地，一些重要的属性很可能被遗漏。考虑到父母的经济条件，乔尼可能对价格贵的派有偏见。价格属性的缺失不可能归纳出优良的分类器：两个样例，尽管可用属性一致，但“缺失”的重要属性值可能不同。即使描述相同，一个样例被判定为正而另一个为负也就不足为奇。当出现这种情况时，我们

称训练集不一致（inconsistent）。这是很难避免的：不仅专家可能不知道价格属性的相关性，而且这个属性本身可能就是未知的。

冗余属性。在某种意义上，危害最小的是那些属性值可从其他属性获得的冗余属性。如果一个数据库包含了一个患者的出生日期和年龄，那么年龄是多余的，因为它可以通过今天的日期减去生日来计算。幸运的是，多余属性比不相关属性和缺失属性危害要小。

属性值噪声。因为不可靠的信息来源、糟糕的测量设备、拼写错误、用户的含混和其他许多理由的存在，属性值和类标签通常不可信。我们称之为数据受到各种各样的噪声（noise）干扰。

随机噪声（stochastic noise）是随机的。例如，我们的体重在一天中并不恒定，早晚的读数会不一样。人为的错误也会有影响，因为没时间测量患者的血压，偷懒的护士仅仅修改了以前的读数，当成今天的数据草草记下来。相较而言，系统噪声（systematic noise）对所有的数值的影响朝向一个方向；一个没有校准的温度计的读数总是偏低。在随意人为的情况下，给定的值没有任何实际意义。例如，从 EEG 电极松动的那一刻起，所有后面的读数都为 0。

类标签噪声。类标签也面临着相同的问题。可能专家推荐的标签尚未被恰当地记录下来。或许，一些样例恰好处于两个分类间的“灰色地带”，所以无法正确决定其分类标签。这两个案例都是随机噪声，后者可能只影响边界样例。类标签噪声也可以系统化：内科医生可能不愿意确诊一种稀有的疾病，除非有确凿的证据——它的类标签更倾向于为负。最终，在自动分类过程里出过错的域中，也会遇到随意人为分配类标签的问题。

类标签噪声可能比属性值噪声更危险。因此，在“圆”问题域中，一个位于正域内部的样例，不会因属性值轻微的改变而改变；只有边界样例会“越界”。相较而言，类标签噪声能使任何样例无效。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中

有任何困难，请回顾本节的相关内容。

- 解释下列几种属性：不相关、冗余和缺失。在“派”域中解释每种属性的含义。
- “不一致的训练集”是什么意思？由什么原因导致的？它是如何影响学习过程的？
- 我们知道的噪声有哪些种类？它们的来源可能是什么？

1.6 总结和历史简评

- 从包含预分类样例的训练集上归纳学习是研究得最深入的机器学习任务。
- 历史上，该任务被映射为搜索，即人们能够提出一种机制用以探索行之有效的搜索技术，如用初态、终态、中间状态、搜索算符和评估函数来定义的爬山法。
- 但是，机械地搜索不是最终的解决方法，后面我们将探索更有用的技术。
- 借助预分类测试数据可以评估分类器的性能。最简单的性能标准是错误率，即分类器错误分类的样例所占的比例。标准过程如图 1.7 所示。

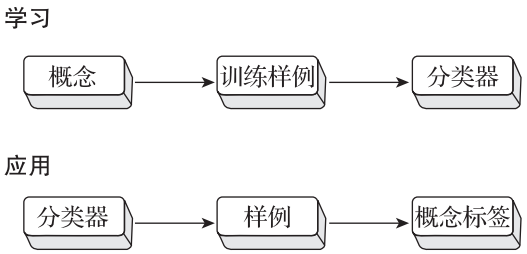


图 1.7 训练样例用于归纳学习分类器，分类器随后被用于分类未来的样例

- 两个在所有的训练集上都可以正确分类的分类器，可能在分类测试集时有

显著差别。

- 除了低错误率，某些应用还要求分类器解释这么分类的理由。
- 训练样例决定归纳学习出的分类器的质量。而训练样例的质量不仅取决于它的样例选择，还取决于描述样例的属性。有些属性是有关的，另外一些属性是不相关或多余的。很常见的是，关键属性会缺失。
- 属性值和类标签可能受到随机噪声、系统噪声和随机失真的干扰。在具体样例中的一个属性值可能是未知的。

历史简评。在 20 世纪 80 ~ 90 年代，将机器学习任务描述为搜索的观点十分流行。但几位“创始人”都有自己的独到见解。米切尔（Mitchell）通常被认为是第一个改进了基于搜索方法的人；同样有影响的是米歇尔斯基（Michalski）提出的 AQ 算法簇。米歇尔斯基、卡波纳（Carbonell）和米切尔共同编辑了论文集，并使这门学科有了长足的发展。他们构建了一代人的思维模式。

还有许多其他的搜索算法。有兴趣的读者可以参考人工智能的书籍，其中介绍最全面的也许是拉塞尔（Russell）和诺维格（Norvig），或科平（Coppin）的著作。

1.7 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 在数字滑块拼图问题中，给出一个比文中评估函数更优的函数。
2. 图 1.8 展示了一棵搜索树，其中，每一个结点代表一个搜索状态，并标记了相应的评估函数值。爬山搜索算法将以怎样的顺序访问这些状态？

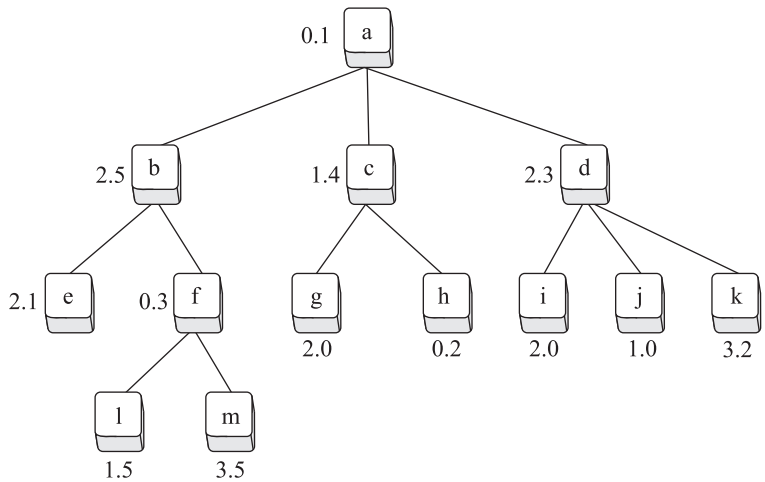


图 1.8　决定启发式搜索算法访问搜索状态的顺序（“盒子”旁边的数字给定了独立搜索状态的评估函数值）

3. 在“派”域中，假设评估函数计算的是正确分类训练集样例的比率。令表 1.1 中的第二个正例为初态。手工模拟使用了一般化和特殊化算符的爬山算法的过程。
4. 当样例被 10 个布尔属性描述时，问题域中的样例空间有多大？分类器的空间有多大？

思考题

1. 在“派”问题域中，所有分类器空间的大小是 2^{108}，并且样例空间的每个子集可以用一个不同的分类器表示。如果我们仅允许属性 – 值对间采用连接词的分类器，那么搜索空间将缩小多少？
2. “派”问题域中可能有哪些噪声？这些噪声可能源自于哪里？还有哪些问题致使训练集有瑕疵？
3. 一些分类器采用黑盒运行模式，不会提供很多解释。这便是“圆”问题域中的情况。举出一些黑盒分类器适用和不适用的问题域。
4. 选择一个你认为难以描述的分类，例如，识别复杂的生物对象（栎树、鸵鸟等）或识别一种音乐流派（摇滚、民谣、爵士等）。建议列出描述训练

样例的属性。这些属性的值容易获得吗？你认为在本章讨论的问题中哪个属性会使学习过程复杂化？

上机实验

1. 编写一个使用爬山算法解决数字化块拼图问题的程序。要求：给每个搜索状态挑选一个合适的代表，编写一个判断是否为终态的模块，并且实现搜索算符的使用。定义两个或三个可选择的评估函数，并观察它们如何产生顺序不同的搜索步骤。
2. 编写一个实现 1.3 节中“增大圆”算法的程序。要求：创建一个类似图 1.5 中二维样例的训练集。学习程序将运用爬山搜索法。评估函数将计算被分类器正确分类的训练集样例的比率。思考下列搜索算符：①增大/减小圆的半径；②使用不同的训练样例作为圆心。
3. 编写一个程序，实现描述“乔尼喜欢的派”时所用的搜索方法。要求：定义你自己的一般化和特殊化算符。评估函数使用从训练样例中观察到的错误率。

第 2 章　概率：贝叶斯分类器

基于已知属性值，预测一个样例的类别的最早尝试可以追溯到第二次世界大战之前，相对于计算机科学的发展，这可以说是“史前”了。当然，在那个时代，人们还没有使用“机器学习”这一术语，但是其工作的目标与本书所要实现的目标在本质上是相同的。

这里，求解问题的基本策略都是使用贝叶斯概率理论，计算所给对象属于各个类别的概率，然后选择概率最大者作为对这个样例所属类别的预测。

2.1 单属性的情况

我们从一个简单的以至于不太现实的情形开始：在一个域中，每个样例都可以用单一的属性来刻画。当然，在这一域建立的基本理论将被推广并应用到更为现实的域当中。

概率。用第 1 章的“玩具”问题可以很容易地解释一些基本概念。训练集合一共包含 12 个派（$N_{\text{all}}=12$），其中 6 个是正类样例（$N_{\text{pos}}=6$），剩余的 6 个是负类样例（$N_{\text{neg}}=6$）。假如每个样例都真实地反映了实际情况，那么乔尼喜欢一个随机挑选的派的概率是 50%：

$$P(\text{pos})=\frac{N_{\text{pos}}}{N_{\text{all}}}=\frac{6}{12}=0.5 \tag{2.1}$$

现在，我们仅考虑派的一个属性，如馅料的量。训练集中包含了 8 个有厚馅料的派（$N_{\text{thick}}=8$），其中 3 个被标记为正类（$N_{\text{pos}\mid\text{thick}}=3$）。这意味着“在给定是厚馅料的派的前提下，一个样例属于正类的条件概率”是 37.5%，这恰恰就是厚馅料样例中正类样例的相对频率所暗示的：

$$P(\text{pos}\mid\text{thick})=\frac{N_{\text{pos}\mid\text{thick}}}{N_{\text{thick}}}=\frac{3}{8}=0.375 \tag{2.2}$$

将条件概率应用于分类。这里，重要的是，相对频率只在给定属性值的派上计算得到。在同样的这 8 个派中，5 个属于负类，这意味着 $P(\text{neg}\mid\text{thick})=\frac{5}{8}=0.625$。观察到 $P(\text{neg}\mid\text{thick})>P(\text{pos}\mid\text{thick})$，我们可以得出结论：乔尼不喜欢厚

馅料的派的概率大于喜欢厚馅料的派的概率。因此，分类器将所有厚馅料的派标记为“乔尼喜欢的派”的负类样例看起来是有道理的。

注意，因为在计算条件概率 $P(\text{pos} \mid \text{thick})$ 时用到了更多的信息，所以它比先验概率 $P(\text{pos})$ 更加可信。这是很自然的。例如，在一个男女数目大致相同的托管中心，我们期望一个被随机挑选的孩子是男孩的概率是 $P(\text{boy}) = 0.5$，但是当我们听到有人叫这个孩子“乔尼（Johnny）”的时候，这个期望会有所提高，因为我们知道很少有女孩会叫“乔尼”。这就是 $P(\text{boy} \mid \text{Johnny}) > P(\text{boy})$ 的原因。

联合概率。两个事件同时发生的联合概率与条件概率是不能相互混淆的。务必使用正确的记号：在联合概率中，各项是用逗号分隔的，如 $P(\text{pos}, \text{thick})$；而在条件概率当中，各项用竖线分隔，如 $P(\text{pos} \mid \text{thick})$。对于一个随机挑选的派，$P(\text{pos}, \text{thick})$ 表示一个样例是属于正类的，而且是厚馅料的概率，然而，$P(\text{pos} \mid \text{thick})$ 表示在所有厚馅料的派中正类样例出现的概率。

一个具体的例子。图 2.1 展示了以上这些术语。长方形代表了所有的派。正

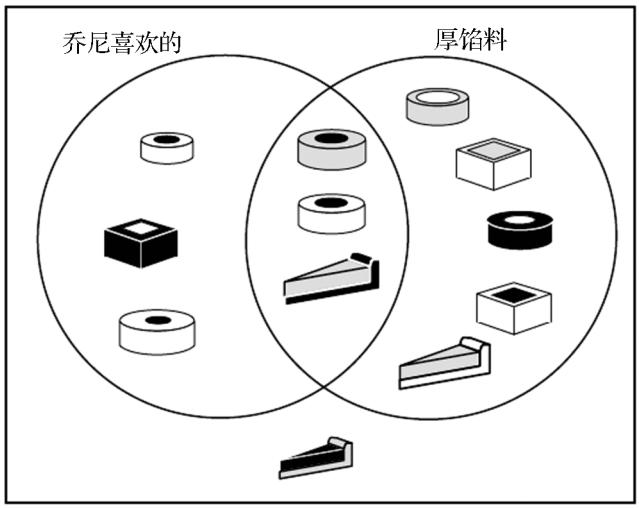


图 2.1　先验概率，$P(\text{pos}) = \frac{6}{12}$ 和 $P(\text{thick}) = \frac{8}{12}$；条件概率，$P(\text{pos} \mid \text{thick}) = \frac{3}{8}$ 和 $P(\text{thick} \mid \text{pos}) = \frac{3}{6}$；联合概率，$P(\text{pos}, \text{thick}) = \frac{3}{12}$

类样例位于一个圈中，厚馅料的派位于另一个圈中。两个圈的交集包含了 3 个同时满足这两个条件的样例。有一个派这两个条件都不满足，因此它位于这两个圆圈之外。条件概率，$P(\text{pos} \mid \text{thick}) = 3/8$，是用交集的大小除以厚馅料的派的集合大小计算得到的。联合概率，$P(\text{pos}, \text{thick}) = 3/12$，是用交集的大小除以整个训练集的大小计算得到的。先验概率，$P(\text{pos}) = 6/12$，是用正类样例集合的大小除以整个训练集的大小计算得到的。

从联合概率计算得到条件概率。图 2.1 告诉我们，联合概率可以由先验概率和条件概率计算得到，即 $P(\text{pos}, \text{thick}) = P(\text{pos} \mid \text{thick}) \cdot P(\text{thick}) = \frac{3}{8} \cdot \frac{8}{12} = \frac{3}{12}$；$P(\text{thick}, \text{pos}) = P(\text{thick} \mid \text{pos}) \cdot P(\text{pos}) = \frac{3}{6} \cdot \frac{6}{12} = \frac{3}{12}$。

注意，联合概率不会超过相应的条件概率的值，即 $P(\text{pos}, \text{thick}) \leqslant P(\text{pos} \mid \text{thick})$，这是因为联合概率是用条件概率乘以先验概率得到的，即 $P(\text{thick})$，它是不超过 1 的。

另一个需要注意的事实是 $P(\text{thick}, \text{pos}) = P(\text{pos}, \text{thick})$，因为它们都描述了同样的一件事情：厚馅料和正类样例同时发生的概率。相应地，前面两个方程的左边必定相等，这意味着下面的等式成立：$P(\text{pos} \mid \text{thick}) \cdot P(\text{thick}) = P(\text{thick} \mid \text{pos}) \cdot P(\text{pos})$。

这个等式两边同时除以 $P(\text{thick})$，我们就得到了著名的贝叶斯公式，它也是本章剩余内容的基础：

$$P(\text{pos} \mid \text{thick}) = \frac{P(\text{thick} \mid \text{pos}) \cdot P(\text{pos})}{P(\text{thick})} \tag{2.3}$$

如果我们推导出厚馅料的派属于负类的概率的类似公式，则有：

$$P(\text{neg} \mid \text{thick}) = \frac{P(\text{thick} \mid \text{neg}) \cdot P(\text{neg})}{P(\text{thick})} \tag{2.4}$$

通过比较式（2.3）和式（2.4）得出的值，我们可以知道哪个类，pos 或 neg，是更加可能的。事情其实比看上去的更加简单：既然分母 $P(\text{thick})$ 对于两个类（的公式）都是相同的，我们就可以忽略它，直接选择分子较大的类。

一个平凡的数值例子。表 2.1 中给出了由贝叶斯公式得到正确决策结果的一

个例子。为了简单起见，这个例子只处理一个平凡的情况，即每个样例由单一的布尔类型的属性来描述。事实上，这个单属性问题是如此简单，以至于我们可以很容易地从训练集中直接求得 $P(\text{pos} \mid \text{thick})$ 和 $P(\text{neg} \mid \text{thick})$，根本不用借助于强大的贝叶斯公式，这也使得验证结果的正确性变得很容易。

表 2.1　描述贝叶斯决策制定的原则

我们用单个属性——馅料的量，来描述每一个训练样例。这里，馅料的量或取值为 thick 或取值为 thin。我们希望机器能识别出正类（pos）。下面是 8 个已知的训练样例。

	ex1	ex2	ex3	ex4	ex5	ex6	ex7	ex8
Size	thick	thick	thick	thick	thick	thick	thick	thick
Class	pos	pos	pos	pos	neg	neg	neg	neg

每个属性值的概率和每个类别的概率都是通过它们的相对频率计算得到的。例如，8 个样例中的 3 个被描述为馅料的量是薄的，因此，$P(\text{thin}) = \frac{3}{8}$。

$$P(\text{thin}) = \frac{3}{8}, \qquad P(\text{thick}) = \frac{5}{8},$$

$$P(\text{pos}) = \frac{4}{8}, \qquad P(\text{neg}) = \frac{4}{8}。$$

一个具体的属性值在一个给定类别中出现的条件概率也是由相对频率决定的。从我们的训练集可以得到以下一些结果：

$$P(\text{thin} \mid \text{pos}) = \frac{2}{4}, \qquad P(\text{thick} \mid \text{pos}) = \frac{2}{4},$$

$$P(\text{thin} \mid \text{neg}) = \frac{1}{4}, \qquad P(\text{thick} \mid \text{neg}) = \frac{3}{4}。$$

基于这些值和贝叶斯公式，我们可以得到如下的条件概率：

$$P(\text{pos} \mid \text{thin}) = \frac{2}{3}, \qquad P(\text{pos} \mid \text{thick}) = \frac{2}{5},$$

$$P(\text{neg} \mid \text{thin}) = \frac{1}{3}, \qquad P(\text{neg} \mid \text{thick}) = \frac{3}{5}。$$

注意：$P(\text{pos} \mid \text{thin}) + P(\text{neg} \mid \text{thin}) = P(\text{pos} \mid \text{thick}) + P(\text{neg} \mid \text{thick}) = 1$。

基于以上结果，我可以得到结论：因为 $P(\text{pos} \mid \text{thin}) > P(\text{neg} \mid \text{thin})$，所以一个样例如果其馅料的量是薄的，则它就应该被分到正类中；相反，因为 $P(\text{neg} \mid \text{thick}) > P(\text{pos} \mid \text{thick})$，所以一个样例如果其馅料的量是厚的，则它就应该被分到负类中。

当样例是用两个或多个属性描述时，计算概率的方法在本质上相同，但我们还需要至少一个其他的技巧，这将在下一节中介绍。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 如何从条件概率与联合概率的关系推导出贝叶斯公式?
- 是什么使得贝叶斯公式如此有用？我们可以使用贝叶斯公式计算什么?
- 联合概率 $P(X, Y)$ 能否大于条件概率 $P(X \mid Y)$？在何种情形下，$P(X \mid Y) = P(X, Y)$?

2.2 离散属性值的向量

现在，我们来讨论如何将贝叶斯公式应用于样例由属性向量如 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ 所描述的域中。

多类问题。不像“派”域的只有正类和负类的情况，许多实际的应用都包含不止两个类。如果 c_i 是第 i 个类的标签，并且 $\boldsymbol{x}$ 是描述我们要进行分类的对象的向量，那么贝叶斯公式需要有如下形式：

$$P(c_i \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid c_i)P(c_i)}{(\boldsymbol{x})}$$

分母对于每一个类都是相同的，我们可以选择使得分子取得最大值的类——$P(\boldsymbol{x} \mid c_i)P(c_i)$。这里的 $P(c_i)$ 可以很容易地从训练集中 c_i 类的相对频率估计得到。然而，估计 $P(\boldsymbol{x} \mid c_i)$ 就没有那么简单了。

一个向量的概率。$P(\boldsymbol{x} \mid c_i)$ 是从 c_i 类中随机挑选一个被描述为向量 x 的样例的概率。它的值可以用相对频率来估计吗？不可以。在“派”这个域，样本空间包含 108 个不同的样例，其中训练集包含了 12 个。这 12 个向量每一个都对应于一个训练样例，而其他的向量（占到了绝大多数）都没有对应的样例。如果 x 在 6 个正类样例中，则它的相对频率是 $P(\boldsymbol{x} \mid \text{pos}) = 1/6$；如果不在，则它的相对频率是 $P(\boldsymbol{x} \mid \text{pos}) = 0$。任何训练样例都“天然地赋予”了类别标签，而训练集合之外的样例，对于任意的 c_i，$P(\boldsymbol{x} \mid c_i) = 0$。贝叶斯公式的分子会永远为 0，

即 $P(\boldsymbol{x} \mid c_i)P(c_i)=0$。这样，我们不可能选择出最有可能的类。很明显地，我们不能对只发生一次或一次都没有发生的事件计算其概率。

幸运的是，这对于单个属性是没有问题的。例如，形状是圆形（shape = circle）在正类样例中出现了 4 次，并且在负类样例中出现了两次。因此，相应的概率就是 $P(\text{shape}=\text{circle} \mid \text{pos})=4/6$ 和 $P(\text{shape}=\text{circle} \mid \text{pos})=2/6$。如果一个属性仅可取值为两三个数值，那么很可能每一个数值都在训练集中出现了多次。因此，这为概率的估计提供了更好的条件。

相互独立的属性。我们需要的是将各个属性的概率组合在一起以确定给定属性向量在给定类别的概率的一个公式——$P(\boldsymbol{x} \mid c_i)$。只要属性之间都是相互独立的，那么问题就简单了。如果 $P(x_i \mid c_j)$ 表示类别 c_j 中的一个样例的第 i 个属性值是 x_i 的概率，那么从 c_j 类中随机挑选一个描述为向量 $\boldsymbol{x}=(x_1, x_2, \cdots, x_n)$ 的样例的概率 $P(\boldsymbol{x} \mid c_j)$ 可以由式（2.5）计算得到：

$$P(\boldsymbol{x} \mid c_j) = \prod_{i=1}^{n} P(x_i \mid c_j) \tag{2.5}$$

一个对象将被标记为类 c_j，如果这个类使得如下形式的贝叶斯公式的分子取得最大值，则一个对象将标记为类。

$$P(c_j) \cdot \prod_{i=1}^{n} P(x_i \mid c_j) \tag{2.6}$$

朴素贝叶斯假设。有的读者可能会认为相互独立的假设不是很合情合理。确实，不同变量之间的相互影响能够避免吗？一个物体的重量随着其大小的增长而增长，一个人保健的情况可能与他的收入相关，一个物体的颜色可由其物理属性得知。总之，两个属性不以任何形式相互关联的域是很少的。无怪乎上面描述的方法被冠以一个不受恭维的名字——朴素贝叶斯。

然而，实际的情况却并不是很坏。违背“独立性要求”确实导致了概率预测的不准确性。但是，这并不意味着就会导致错误的分类结果。记得吗？$\boldsymbol{x}$ 只有在 $P(\boldsymbol{x} \mid c_i)P(c_i)$ 被最大化的时候才被标记为 c_i。如果一个类的计算结果是 0.8，而另一个类是 0.2，那么分类器的表现并不会因为概率估计有 10% 或 20% 的误差而发生改变。所以，当要求各属性在原则上都相互独立时，即使它们并不是真正的满足，我们依然可以得到比较理想的结果。

不能做相互独立假设的时候。我们不得不问，当变量之间的相互独立性不能被忽略的时候怎么办？一个数学家的第一反应是建议采用更加复杂的方法来估计 $P(\boldsymbol{x} \mid c_i)$。这样的方法确实存在，但它们的算法复杂度会随着属性的增多而增加，并且它们往往包含了我们很难确定的参数。追求实际的工程师们会怀疑这样做所带来的麻烦与好处可能是相抵的。

一个更实际的方法是通过适当的数据预处理来尽量减少属性之间的相关性。一个很好的方法就是将冗余的属性摈除，也就是那些已知的依赖于其他属性的属性。例如，如果属性集合中包含了年龄，出生日期以及当前的日期，那么对于朴素贝叶斯分类器很可能只使用年龄一个属性的效果会更好。

我们也可以尝试将两个或 3 个属性用一个人工设计的属性来代替。因此，在“派”这个域，烘焙师可能告诉我们馅料的厚薄和外壳的大小并不独立，馅料厚的派外壳较薄，反之亦然。这样，我们就可以将这两个属性值用一个新的属性来代替，并在计算上大获裨益。例如，这个新的属性叫“外壳馅料大小”，它仅有两个取值：厚外壳薄馅料和薄外壳厚馅料。

作为最后的策略，如果我们不想用多变量概率估计的高级方法，并且想尽量避免数据预处理，那么我们可以把贝叶斯分类器一起放弃，并采用本书后面章节介绍的其他机器学习范式。

一个数值例子。为了熟悉朴素贝叶斯分类的机制，读者可以浏览一下表 2.2 中的例子。这里，一个之前没有见过的派所属的类是基于表 1.1 中的训练集合得到的。我们用到了贝叶斯公式，并且假设各属性之间是相互独立的。

表 2.2　贝叶斯分类器：样例由相互独立的属性向量来描述

为了确定以下对象所属的类别，我们将贝叶斯公式应用于表格 1.1 的训练集：

x = [shape = square，crust – size = thick，crust – shade = gray，filling – size = thin，filling – shade = white]

这里一共有两个类：pos 和 neg。计算过程是为了分别求出每个类对应的贝叶斯公式的分子，然后选择值较高的那个类。在训练集中，每类都有相同数目的样例，即 $P(\text{pos}) = P(\text{neg}) = 0.5$。其他各项，$\prod_{i=1}^{n} P(x_i \mid \text{pos})$ 和 $\prod_{i=1}^{n} P(x_i \mid \text{neg})$，均可以由下面的条件概率

（续）

计算得到：

$P(\text{shape} = \text{square} \mid \text{pos}) = 1/6$，　$P(\text{shape} = \text{square} \mid \text{neg}) = 1/6$

$P(\text{crust} - \text{size} = \text{thick} \mid \text{pos}) = 1/6$，　$P(\text{crust} - \text{size} = \text{thick} \mid \text{neg}) = 1/6$

$P(\text{crust} - \text{shade} = \text{gray} \mid \text{pos}) = 1/6$，　$P(\text{crust} - \text{shade} = \text{gray} \mid \text{neg}) = 1/6$

$P(\text{filling} - \text{size} = \text{thin} \mid \text{pos}) = 1/6$，　$P(\text{filling} - \text{size} = \text{thin} \mid \text{neg}) = 1/6$

$P(\text{filling} - \text{shade} = \text{white} \mid \text{pos}) = 1/6$，　$P(\text{filling} - \text{shade} = \text{white} \mid \text{neg}) = 1/6$

基于这些数值，我们得到如下概率：

$$P(\text{x} \mid \text{pos}) = \prod_{i=1}^{n} P(x_i \mid \text{pos}) = \frac{1}{6} \cdot \frac{5}{6} \cdot \frac{1}{6} \cdot \frac{3}{6} \cdot \frac{1}{6} = \frac{15}{6^5}$$

$$P(\text{x} \mid \text{neg}) = \prod_{i=1}^{n} P(x_i \mid \text{neg}) = \frac{2}{6} \cdot \frac{5}{6} \cdot \frac{2}{6} \cdot \frac{1}{6} \cdot \frac{2}{6} = \frac{40}{6^5}$$

因为 $P(\boldsymbol{x} \mid \text{pos}) < P(\boldsymbol{x} \mid \text{neg})$，所以我们将 $\boldsymbol{x}$ 标记为负类。

计算过程用伪代码的形式总结在了表 2.3 中。

表 2.3　基于朴素贝叶斯原理分类

要进行分类的样本可以表示为 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$。

1. 对于每一个 x_i 和每一个类 c_j，用 x_i 在属于 c_j 类的训练样例中的相对概率来计算条件概率 $P(x_i \mid c_j)$。
2. 对于每个类 c_j，执行下面两个步骤：
   1）估计该类在训练集中出现的相对频率 $P(c_j)$。
   2）用属性相互独立的朴素假设来计算条件概率 $P(\boldsymbol{x} \mid c_j)$：

$$P(\boldsymbol{x} \mid c_j) = \prod_{i=1}^{n} P(x_i \mid c_j)$$

3. 选择使得 $P(c_j) \cdot \prod_{i=1}^{n} P(x_i \mid c_j)$ 的值最大的类。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在何种情况下我们可以假设各个属性是相互独立的？这个假设为我们估计 $P(\boldsymbol{x} \mid c_i)$ 带来了什么好处？
- 讨论与这个假设相矛盾的一些方面。

2.3 稀少事件的概率：利用专家的直觉

在第一个近似方法中，概率被认为与相对频率是一致的：在 100 次试验中观察到 x 出现了 30 次，我们就认为 $P(x)=0.3$。这是我们在前面章节中所做的。

公平地讲，然而，这样的估计只有在大量实验的支持下才可以被信任。可以想象，一个硬币被抛掷 4 次有 3 次正面朝上就断定 $P(\text{heads})=0.75$ 是仓促的；这个实验的物理原理要求一个公正的硬币应该有 50% 的机会是正面朝上的。这种先验期望能用来帮助我们改进在观测次数不充分的情况下的概率估计吗？

答案是，“是的，我们可以使用 m – 估计”。

m – 估计的本质。让我们用一个一面比另一面出现更频繁的不公平的硬币来阐明这个原理。在没有任何干预的情况下，正面朝上的先验期望是$\pi_{\text{heads}}=0.5$，一个辅助参数 m 的值可以帮助工程师说明对于类别预测程序有着怎样的信心，也就是先验期望在多大程度上是可信的（越大的 m 值表示越可信）。

让我们用N_{all}来表示硬币被抛掷的总次数，用N_{heads}表示硬币出现正面朝上的次数。将这些值与先验期望值和置信度组合起来可以用式（2.7）表示：

$$P_{\text{heads}}=\frac{N_{\text{heads}}+m\pi_{\text{heads}}}{N_{\text{all}}+m} \tag{2.7}$$

请注意，当$N_{\text{heads}}=N_{\text{all}}=0$ 时上式退化为先验期望π_{heads}。相反地，如果N_{heads}和N_{all}足够大以至于 m π_{heads}和 m 项可以忽略不计，它就收敛为它的相对频率。将$\pi_{\text{heads}}=0.5$ 和 $m=2$ 代入，可以得到：

$$P_{\text{heads}}=\frac{N_{\text{heads}}+2\times 0.5}{N_{\text{all}}+2}=\frac{N_{\text{heads}}+1}{N_{\text{all}}+2}$$

概率估计的一个示例。表 2.4 展示了 5 次实验过程中，如上的值是如何逐渐被计算出来的。读者可以看出，在较少次数实验的情况下，m – 估计比相对频率更符合常理。因此，两次实验后，m – 估计表明有 75% 的概率是正面朝上，而任何支持相对频率的不得不承认，基于前两次的实验，反面朝上的概率为 0。尽管如此，随着实验次数的增加，m – 估计和相对频率所估计的值将趋于收敛。

表 2.4　对于每次连续的实验，第二行给出了观察到的结果，第三行是正面出现的相对频率，最后一行是在$\pi_{\text{heads}}=0.5$，$m=2$ 时此概率的 m – 估计值

投掷次数	1	2	3	4	5
结果	正面	正面	反面	正面	反面
相对频率	1.00	1.00	0.67	0.75	0.60
m – 估计值	0.67	0.75	0.60	0.67	0.57

用户置信度的影响。让我们深入探讨一下用户置信度 m 的影响。如果我们对比下述两种不同的设定——左边的 $m=100$ 和右面的 $m=1$，则会有很多发现：

$$\frac{N_{\text{heads}}+50}{N_{\text{all}}+100}\text{和}\frac{N_{\text{heads}}+0.5}{N_{\text{all}}+1}$$

$m=100$ 的情形仅允许有大量的证据可获得时修改先验概率的估计值（$N_{\text{heads}}\gg 50$，$N_{\text{all}}\gg 100$）。对比之下，$m=1$ 的情形允许仅有几次实验即可否定用户的观点。

有超过两个结果的域。尽管我们目前只讨论了有两个输出结果的域，但是m – 估计也可用于有多个结果的域。投掷一个公平的骰子会出现 6 个不同的结果，我们期望所看到的概率是这样的：例如，三点的是$\pi_{\text{three}}=\frac{1}{6}$。令 $m=6$，我们得到下列式子：

$$P_{\text{three}}=\frac{N_{\text{three}}+m\pi_{\text{three}}}{N_{\text{all}}+m}=\frac{N_{\text{three}}+6\cdot\frac{1}{6}}{N_{\text{all}}+6}=\frac{N_{\text{three}}+1}{N_{\text{all}}+6}$$

再一次地，如果N_{all}数值很大以至于 $m=6$ 和$m\pi_{\text{three}}=1$ 可以被忽略，则公式将收敛于相对频率：$P_{\text{three}}=\frac{N_{\text{three}}}{N_{\text{all}}}$。如果我们不想这种情况过早地发生（或许是因为我们对于这个先验估计π_{three}有很大的信心），则可以选择一个更大的 m 值以阻止这种情况发生。

***m* – 估计的限制**。我们不应该忘记 m – 估计和它依赖的参数相关性很大。如果我们使用不现实的先验估计，则结果将会令人失望。假定$\pi_{\text{heads}}=0.9$，$m=10$，则式（2.7）转变成如下形式：

$$P_{\text{heads}}=\frac{N_{\text{heads}}+9}{N_{\text{all}}+10}$$

当我们用此公式重新计算表 2.4 的值时，发现 5 次实验之后概率被估计为 $P_{\text{heads}} = \frac{3+9}{5+10} = \frac{12}{15} = 0.8$，当然，与在 $\pi_{\text{heads}} = 0.5$ 时比较，它更为不合理，后者获得的 $P_{\text{heads}} = 0.57$。我们鼓励读者去核实当减小 m 的值时情况会有所改善。

数学合理性。我们做最后一个点评。数学家们强调所有的可能事件的概率总和应为 1：如果一个实验有 N 个不同的结果，P_i 是第 i 个结果的概率，则 $\sum_{i=1}^{N} P_i = 1$。很容易核实，式(2.7) 对所有的 m 值都满足这一条件。假定我们在处理一个只有两种结果的掷硬币域，如果先验期望值相加等于 1($\pi_{\text{heads}} + \pi_{\text{tails}} = 1$)，然后给定 $N_{\text{heads}} + N_{\text{tails}} = N_{\text{all}}$，则有如下推导：

$$\begin{aligned} P_{\text{heads}} + P_{\text{tails}} &= \frac{N_{\text{heads}} + m\,\pi_{\text{heads}}}{N_{\text{all}} + m} + \frac{N_{\text{tails}} + m\,\pi_{\text{tails}}}{N_{\text{all}} + m} \\ &= \frac{N_{\text{heads}} + N_{\text{tails}} + m(\pi_{\text{heads}} + \pi_{\text{tails}})}{N_{\text{all}} + m} = 1 \end{aligned}$$

有兴趣的读者会很容易地将其推广至任意有限个的类。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在什么情况下相对频率不适合用于预测离散的概率?
- 在式 (2.7) 中参数 m 有什么影响? 在什么情况下你更倾向于 m 取值大一些，什么情况下更倾向于小的 m 值?
- 在式 (2.7) 中先验估计 π_{heads} 有什么影响? m – 估计的可信度是如何被不切实际的 π_{heads} 值所影响的?

2.4 如何处理连续属性

目前为止，我们仅考虑了离散的属性，通过相对频率或 m – 估计来估计它们的概率。然而，这还远远不够。在许多应用中，我们所遇到的属性（如年龄、价

格或体重）是从连续域中取值的。

这时候，相对频率则不实用。虽然很容易确定一个工程专业学生是男性的概率是$P_{\text{male}}=0.7$，但是这个学生的体重为 184.5lb（1lb = 0.453kg）的概率却不能轻易地给出，因为体重可以取的值有无限多个，每一个的概率都是无穷小。这种情况下该怎么办？

离散化连续属性。一种可能的方法是离散化。这个最简单的“技巧”会把某属性的原始域一分为二。例如，把年龄属性替换为布尔属性，“老”对应于年龄大于 60 为真，否则为假。然而，有一部分可用信息会丢失：一个人或许是“老”的，但我们将不再知道他有多“老”，也不知道一个“老”人是否比另一个“老”人更“老”。

如果把原始域分成许多区间，而不是两个，如（0，10］，…，（90，100］，这种损失会得到缓解㊀。假定我们对应于每一个区间都有一个独立的格子，对于每个训练样例，若它的年龄值落在第 i 个区间中，则把一个小黑球放进第 i 个格子中。

完成这些之后，我们会得到图 2.2 描述的状况。图的上半部分展示了这些并排

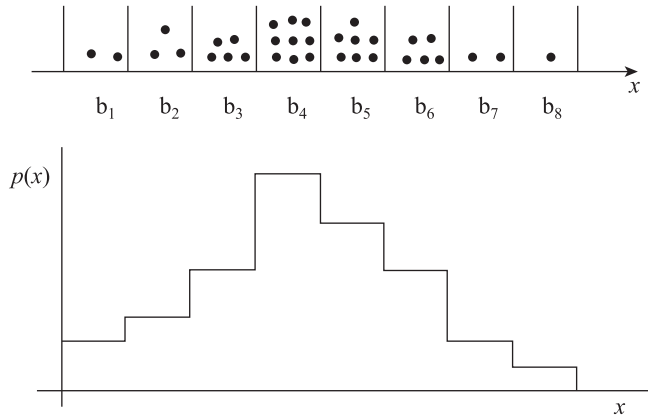


图 2.2　一个简单的离散化方法：将每一个子区间表示为一个独立的格子；下半部分展示了各个子区间上的直方图

㊀ 我们这里假定 100 是在训练集中观测的最大值。另外，我们的背景知识也告诉我们给定属性的值不会超过 100。

放置的格子，下半部分展示了以下述方式创建的一个分段常函数：如果 N 是这个训练集的大小，N_i 是第 i 个格子里的小球个数，那么在第 i 个区间里的常函数值即为 $\frac{N_i}{N}$ ——在第 i 个格子里面的小球的相对频率。因为在这个分段常函数之下的面积是 $\frac{\sum N_i}{N} = 1$，这个方法可以用来估计一个值落入给定的区间的概率，而不是一个具体的年龄值的概率。

概率密度函数。如果这个构造的阶梯函数看起来过于粗糙，假如在每个格子里的小球数又是足够用于可靠的概率估计的，我们可以通过把原始域分得更细从而有更多区间的办法来对它进行微调。如果这个训练集无限大，那么理论上说，我们可以继续减小区间的长度直到它们变成无限小。这种填格箱子训练的结果将不再是一个阶梯函数，而是一个连续的函数 $p(x)$，如图 2.3 所示。对它的解释和它被创造出来的方式一样：$p(x)$ 值越大，说明有很多样例的年龄值接近于 x；相反地，$p(x)$ 值越小，说明在 x 附近的年龄值越少。这就是我们称 $p(x)$ 为概率密度函数的原因，通常为了避免拗口，我们更倾向于使用它的首字母缩写 PDF。

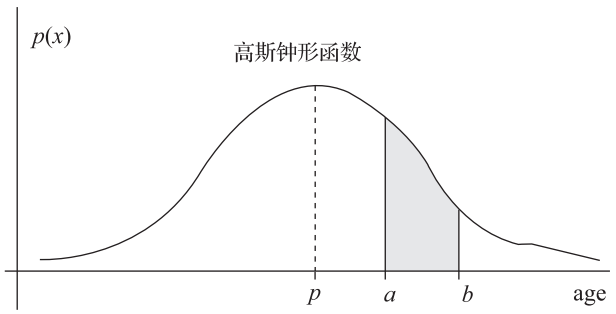


图 2.3　当使用概率密度函数的时候，我们用概率密度函数下方相对面积的大小来确定 $x \in [a, b]$ 的概率

让我们注意一下表示方法。x 的离散概率是用大写字母 $P(x)$ 来表示的，而在 x 处的概率密度函数是用小写字母 $p(x)$ 来表示的。如果我们想要指出这个概率密度函数是唯一的、来自于第 c_i 类的样本，则用一个下标来表示，即 $p_{c_i}(x)$。

连续属性的贝叶斯公式。概率密度函数的一个好处就是它让我们即便是在连

续属性的情况下也可以使用贝叶斯公式。我们只需用$p_{c_i}(x)$替代条件概率$P(x \mid c_i)$，用$p(x)$来替代$P(x)$即可。让我们从被分类对象仅由单一的连续属性x来描述的这一平凡的现象开始。这时，贝叶斯公式有如下形式：

$$p(c_i \mid x) = \frac{p_{c_i}(x) \cdot P(c_i)}{p(x)} \tag{2.8}$$

这里，$P(c_i)$由训练集里的第c_i类的相对频率估计得到，$p(x)$是从所有训练样本中得到的概率密度函数，$p_{c_i}(x)$是从训练集中属于第c_i类的样本得到的概率密度函数。

再次说明，分母可以被忽略，因为它对任何类来说都是相同的值。分类器简单地为每个类计算分子的值，$p_{c_i}(x) \cdot P(c_i)$，然后将对象标记为取得最大乘积的类。

重新审视朴素贝叶斯。当面对更加现实的情况，即样例由属性向量来描述，我们会利用像以前一样相同的“技巧”：做所有属性相互独立的假设。假如我们遇到一个被描述为$\boldsymbol{x} = (x_1, \cdots, x_n)$的样例，在$\boldsymbol{x}$处的概率密度函数可以用各属性的乘积来近似表示：

$$p_{c_j}(\boldsymbol{x}) = \prod_{i=1}^{n} p_{c_j}(x_i) \tag{2.9}$$

一位统计学家会给出理论上更加完善的公式，然而高度的复杂性往往并不能得出让人满意的结果。一方面，我们可能会对不精确的数字做精确的计算；另一方面，技术越复杂，公式被错误使用的危险程度也越高。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 概率密度函数（PDF）是什么？在贝叶斯分类的情况下它可以帮助我们做什么？
- 解释可以帮助我们获得概率密度函数的非正式定义的离散化机制。
- 在含有连续属性的域中，贝叶斯公式会发生怎样的变化？我们如何估计各项的值？

2.5 高斯钟形函数：一个标准的概率密度函数

一种近似计算概率密度函数的方法是用前边章节讲的离散化技术。此外，我们还可以利用已知的可以用于许多实际情形的标准化模型。在这些模型中用的最广的，可能就是以德国伟大的数学家命名的高斯函数。

描述它的图形和公式。图 2.3 中的曲线是一个例子，它的形状呈钟形，因此许多人把它叫作“钟形函数”。函数的最大值在均值处达到，即 $x=\mu$，同时曲线随着 x 与 μ 的距离的增加逐渐向下倾斜。对于体温这样的变量，它的概率密度的最大值在 $x=99.7$ °F 附近取得，有理由相信高斯函数是一个很合适的概率密度模型。

用数学方法表示，高斯函数可以由式（2.10）定义：

$$p(x)=k\cdot \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.10}$$

这里 e 是自然对数的底。

参数。注意到，由于指数项是负的，因此 x 和 μ 之间的差异越大，指数项的分子也越大，那么 $p(x)$ 的值就越小。分子 $(x-\mu)^2$ 是平方项，是为了确保在均值 μ 两侧斜率是以相同的值下降的，也就是说曲线是对称的。σ^2 是一个被称为方差的参数，它控制了倾斜的程度。大的方差意味着对 x 和 μ 之间的距离有较小的敏感性，因此是一个相对比较平的钟形曲线；反之，小的方差定义了一个较窄的钟形曲线。

k 的作用是保证概率论要求的钟形函数下方的面积为 1。相对容易证明，k 取式（2.11）给出的值时恰好满足这个条件：

$$k=\frac{1}{\sqrt{2\pi\sigma^2}} \tag{2.11}$$

设置参数值。为了在实际应用中近似 $p_{c_i}(x)$ 时使用这个模型，我们仅需要估计它的两个参数的值：μ 和 σ^2。这是很容易的。假如在训练样本中有 m 个来自于 c_i 类，如果 x_i 是第 i 个样本给定属性的属性值，那么均值和方差可以分别由式（2.12）和式（2.13）计算得到：

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.12}$$

$$\sigma^2 = \frac{1}{m-1}\sum_{i=1}^{m} (x_i - \mu)^2 \tag{2.13}$$

简单来说，高斯函数的中心 μ 是观察到的训练样例的属性值的算术平均，而方差是由 x_i 和 μ 的差的平方的平均值计算得到的。注意，在计算方差时，正如我们期望的，我们用 $m-1$ 来除求和项，而不是 m。这样做的目的是为了弥补 μ 本身也是估计得到的这一事实。因此，方差应该比用 m 除求和项要略微大一点。当然，这只是在训练集比较小的时候成立。对于大的 m，$m-1$ 和 m 的差别完全可以忽略不计。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 给出一个概率密度函数服从高斯分布的连续随机变量的例子。
- 哪些参数定义了钟形函数？我们如何从训练集中得到它们的值？
- 如何及为什么对钟形函数进行标准化？

2.6 用高斯函数的集合近似概率密度函数

虽然在许多实际应用中钟形函数是近似概率密度函数的一个很好的机制，但它并不是万能的。一些变量并不遵循它的分布，如考虑一组由小学生和他们的父母混合起来的人的体重的分布。如果我们用离散化的方法来构建概率密度函数，则会观察到两个峰值：一个对应于孩子，另一个对应于成人。如果父亲们的体重分布在一个比母亲们的体重较高的均值周围，则可能出现 3 个峰值；如果这些家庭来自于不同的种族，则可能出现更多的峰值。

组合高斯函数。毫无疑问地，一个单一的钟形函数会有表示不准确的情况。但是，如果我们把两个或更多个组合到一起呢？如果我们知道样例的不同来源，

则可以对每个数据源分别用一个高斯函数来表示，然后把这些高斯函数组合起来。这样可以解决我们的问题吗？

一个诚实的答案是，“是的，在这种理想的情况下”。但是，实际上，有关不同来源的现实知识是很难获得的。一个更好解决方案是将体重值随机地分为许多组。在最极端的情况下，我们甚至可以将每个样例视为它自己的一个分组，然后把这个样例的体重值标识为高斯中心，从而（对于 m 个样例）可以得到 m 个钟形函数。

结合它们的公式。假设我们想要近似连续属性 x 的概率密度函数。如果我们用μ_i 来标记 x 在第 i 个样例中的值，那么概率密度函数可以用 m 个函数的和来近似：

$$p(x) = k \cdot \sum_{i=1}^{m} \mathrm{e}^{-\frac{(x-\mu_i)^2}{2\sigma^2}} \tag{2.14}$$

像之前那样，这里的归一化常数 k，是为了确保函数曲线下方的面积是 1。当 k 按照式（2.15）计算时，这个目标可以实现：

$$k = \frac{1}{(2\pi)^{m/2}\sigma^m} \tag{2.15}$$

数学家告诉我们，如果 m 足够大，式（2.14）可以任意准确地近似那个概率密度函数。

描述这一点。图 2.4 用一个包含 $m=3$ 个样例的训练集阐明了这个方法，属性 x 的值分别为 $x_1=0.4$、$x_2=0.5$ 和 $x_3=0.7$。图 2.4 中，上面的 3 幅图展示了 3 个钟形函数，每个都分别以 3 个点中的一个为中心，方差都取 1（$\sigma^2=1$）；下面的一幅图显示了将式（2.14）和式（2.15）放在一起而组成的概率密度函数，密度分别为$\mu_1=0.4$、$\mu_2=0.5$ 和$\mu_3=0.7$，而且$\sigma^2=1$：

$$p(x) = \frac{1}{\sqrt{(2\pi)^3}} \cdot \left[\mathrm{e}^{-\frac{(x-0.4)^2}{2}} + \mathrm{e}^{-\frac{(x-0.5)^2}{2}} + \mathrm{e}^{-\frac{(x-0.7)^2}{2}} \right]$$

具体的参数值的影响。这样得到的概率密度函数的实用性（如贝叶斯公式的成功使用）取决于σ^2 的选择。在图 2.4 中，我们用了$\sigma^2=1$，但对于它在将来的应用中是否可用是没有保障的。为了能够进行适当的调整，我们需要了解它是如何影响组合起来的概率密度函数的形状的。

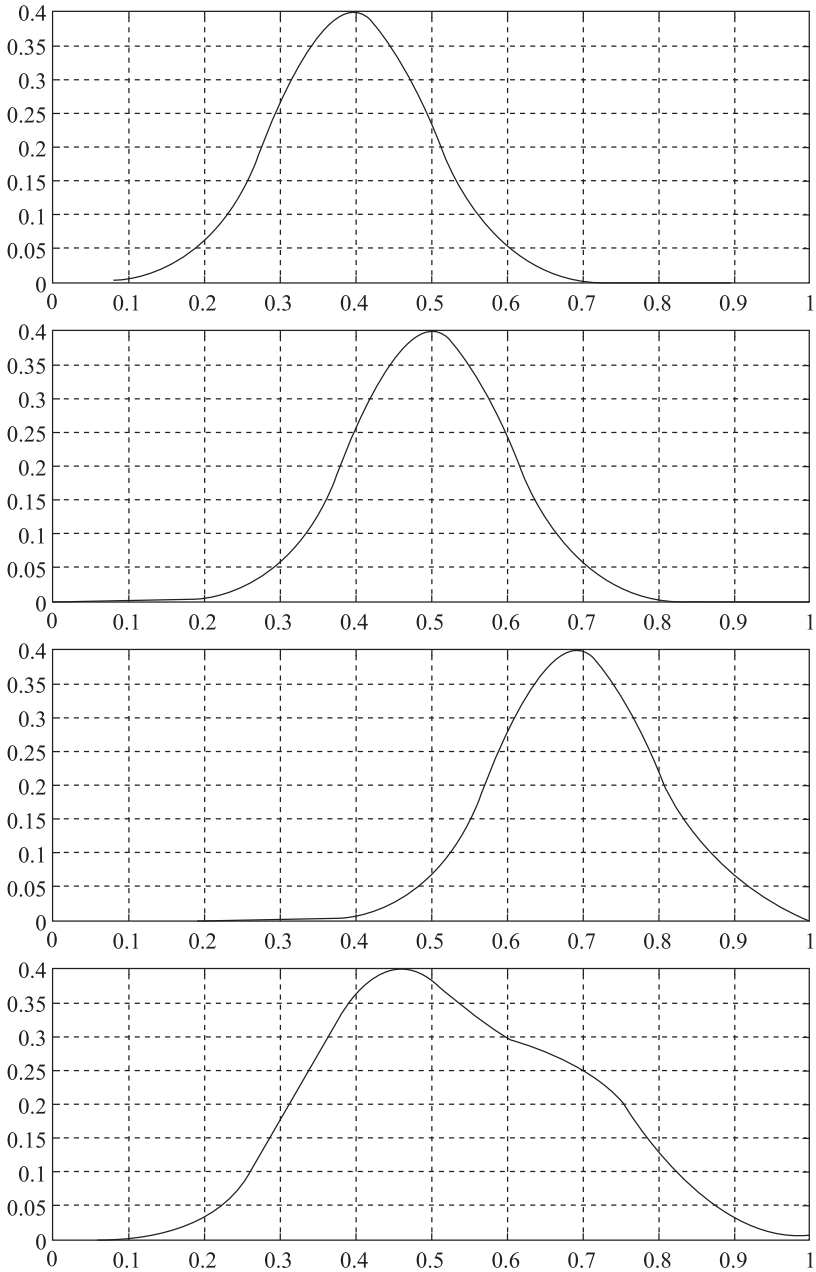


图 2.4　由如下 3 个样例的属性 x 的值组成的概率密度函数：$\mu_1 = 0.4$、$\mu_2 = 0.5$和$\mu_3 = 0.7$；上边 3 幅图展示了有贡献的 3 个高斯函数，下边一幅图展示了组合之后的效果；方差$\sigma^2 = 1$

检查高斯公式，我们意识到一个很小的σ^2值会造成对x和μ_i之差的很大的敏感度；单个的钟形函数会变“窄”，生成的概率密度函数会有多个延伸的“谷”分割成的陡峭的峰。相反地，一个高的σ^2的结果是一个几乎平坦的概率密度函数。为了找一个这两个极端的折中，如果我们使得σ^2依赖于样例之间的距离，则会得到比较好的结果。

最简单的解决方案是取$\sigma^2 = \mu_{\max} - \mu_{\min}$，这里$\mu_{\max}$和$\mu_{\min}$分别是$\mu_i$的最大值和最小值。如果你觉得这个方法太草率，则可以考虑用样例数归一化这个差值：$\sigma^2 = (\mu_{\max} - \mu_{\min})/m$。一个很大的训练集（大的$m$）会导致较小的变化，它会使得各个高斯成分变窄。最后，在某些域，我们可能会希望每一个有贡献的钟形函数都有一个自己的方差，而且是与离它最近的一个其他的高斯函数的中心的距离成比例的。然而，在这种情况下，我们就不能再用式（2.15）来设置k的值了。

一个数值例子。表2.5中的例子描述了包含一个小的训练集和一个待分类向量的具体实例的整个执行过程。我们鼓励读者阅读这个例子的所有细节，以清楚将公式放在一起的方式。也请参考图2.5中的图示。

表2.5　将朴素贝叶斯应用于有3个连续属性的域

假设我们给定了一个包含6个样例的训练集：ex_1，…，ex_6，每个样例都由3个属性来描述：at_1，at_2，at_3。

样例	at_1	at_2	at_3	类别
ex_1	3.2	2.1	2.1	pos
ex_2	5.2	6.1	7.5	pos
ex_3	8.5	1.3	0.5	pos
ex_4	2.3	5.4	2.45	neg
ex_5	6.2	3.1	4.4	neg
ex_6	1.3	6.0	3.35	neg

利用贝叶斯公式，找到$\boldsymbol{x}$=（9，2.6，3.3）最可能的类别。

我们的策略是来估计$p_{\text{pos}}(\boldsymbol{x}) \cdot P(\text{pos})$和$p_{\text{neg}}(\boldsymbol{x}) \cdot P(\text{neg})$。观察到$P(\text{pos}) = P(\text{neg})$，如果$p_{\text{pos}}(\boldsymbol{x}) > p_{\text{neg}}(\boldsymbol{x})$，则简单地用pos来标记$\boldsymbol{x}$，否则用neg。构建概

（续）

率密度函数的时候，我们基于属性相互独立的假设。由 2.2 节可知：$p_{\text{pos}}(\boldsymbol{x}) = p_{\text{pos}}(at_1) \cdot p_{\text{pos}}(at_2) \cdot p_{\text{pos}}(at_3)$ 和$p_{\text{neg}}(\boldsymbol{x}) = p_{\text{neg}}(at_1) \cdot p_{\text{neg}}(at_2) \cdot p_{\text{neg}}(at_3)$。

等号右边的项是由式（2.14）得到，这里$\sigma^2 = 1$，$m = 3$，相应地，$k = 1/\sqrt{(2\pi)^3}$。这样，对于这些项中的第一项，我们有：

$$p_{\text{pos}}(at_1) = \frac{1}{\sqrt{(2\pi)^3}} \left[\mathrm{e}^{-0.5(at_1 - 3.2)^2} + \mathrm{e}^{-0.5(at_1 - 5.2)^2} + \mathrm{e}^{-0.5(at_1 - 8.5)^2} \right]$$

注意，在正类样例中，at_1 的值分别有$\mu_1 = 3.2$，$\mu_2 = 5.2$ 和$\mu_3 = 8.5$，请看表达式中的指数项。函数剩余的 5 项可以类似得到，图 2.5 的最右一列画出了它们。

将 $\boldsymbol{x}$ 的坐标，即$at_1 = 9$，$at_2 = 3.6$ 和$at_3 = 3.3$，代入这些等式，可得 $p_{\text{pos}}(x) = 0.0561 \times 0.0835 \times 0.0322 = 0.00015$，$p_{\text{neg}}(x) = 0.0023 \times 0.0575 \times 0.1423 = 0.00001$。

观察到$p_{\text{pos}}(\boldsymbol{x}) > p_{\text{neg}}(\boldsymbol{x})$，因此我们用类别 pos 来标记 $\boldsymbol{x}$。

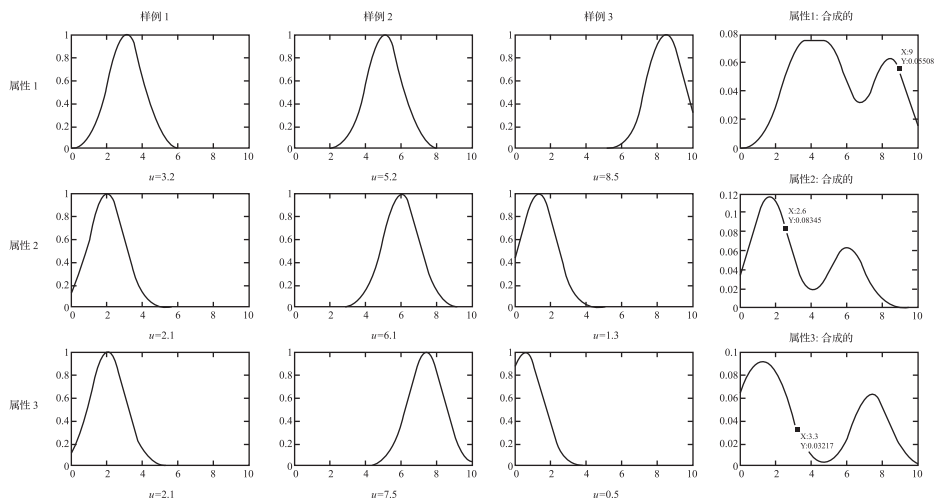


图 2.5　为正类和负类分别合成概率密度函数（取$\sigma^2 = 1$）；每一行代表了一个属性，而左边 3 列每一列都代表了一个样例；最右一列显示的是合成的概率密度函数

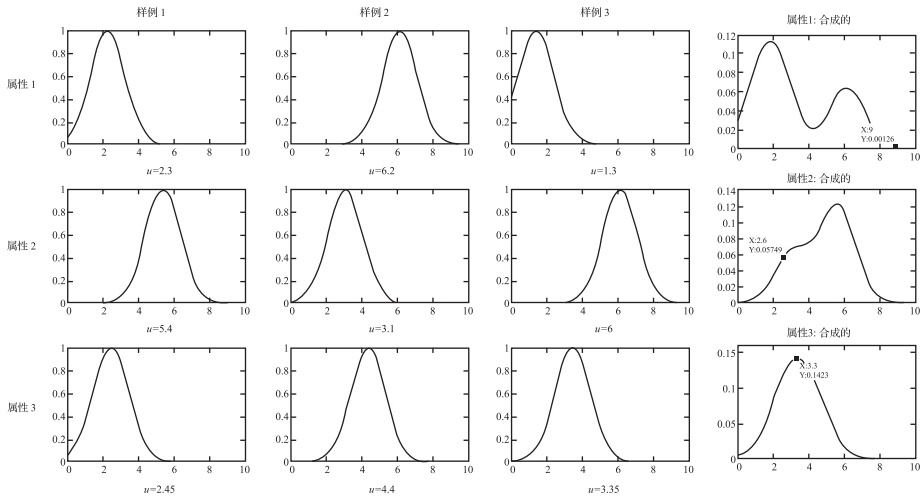


图 2.5　为正类和负类分别合成概率密度函数（取$\sigma^2=1$）；每一行代表了一个属性，而左边 3 列每一列都代表了一个样例；最右一列显示的是合成的概率密度函数（续）

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在什么情况下高斯函数是一个不够准确的概率密度模型?
- 为什么复合的概率密度函数需要用 k 进行标准化?
- 如何建立每个单独的钟形函数的中心和方差?

2.7　总结和历史简评

- 贝叶斯分类器对每个类，c_i，分别计算乘积 $P(\boldsymbol{x} \mid c_i)P(c_i)$，然后用使乘积取最大值的类别来标记样例 $\boldsymbol{x}$。
- 主要问题是如何计算概率：$P(\boldsymbol{x} \mid c_i)$。大部分的情况下，通过假设各个属性之间是相互独立的来简化这个工作，这时 $P(\boldsymbol{x} \mid c_i) = \prod_{j=1}^{n} P(x_j \mid c_i)$，这里 n 是属性的个数。

- 所谓的 m – 估计使得利用用户对一个事件概率的估计成为可能。这常发生在实验依据不充分的域中，在这种情况下相对概率是不可靠的。
- 在连续属性的域，离散概率和 $P(\boldsymbol{x} \mid c_i)$，分别由概率密度函数和$p_{c_i}(\boldsymbol{x})$ 代替。但分类的过程是一样的：样例被标记为使得乘积$p_{c_i}(\boldsymbol{x})P(c_i)$ 最大的类别。
- 近似估计概率密度函数的具体形状可以通过离散化、使用标准化的概率密度函数或高斯函数的和来实现。
- 概率估计还远不够完美，但即使在严格的理论假设不成立时，结果往往是令人满意的。

历史简评。最早使用贝叶斯决策理论进行分类的文章是尼曼（Neyman）和皮尔森[61]（Pearson）以及费舍尔[21]（Fisher），但是只有随着计算机的发展，这个域才获得了快速的发展，当时主要是由周[12]（Chow）主张的。第一个使用属性相互独立假设的是古德[28]（Good）。用钟形函数之和来近似概率密度函数的想法来自于帕珍[66]（Parzen）。

当提供了关于概率的完整信息时，可以保证贝叶斯分类器给出最佳的分类准确率。这就是为什么它有时被用来作为比较其他方法性能的参照的原因。

2.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 投掷一枚硬币 3 次，分别得到正面、正面和反面。使用 $m = 3$ 和$\pi_{\text{heads}} = \pi_{\text{tails}} = 0.5$，为这些输出计算 m – 估计。

2. 假设你有如下的训练样例，它们由 3 个属性所描述：x_1、x_2、x_3，被标注为c_1 类和c_2 类：

x_1	x_2	x_3	类别
2.1	0.2	3.0	c_1
3.3	1.0	2.9	c_1
2.7	1.2	3.4	c_1
0.5	5.3	0.0	c_2
1.5	4.7	0.5	c_2

使用这些数据，完成：

1）假如属性是相互独立的，近似计算如下的概率密度函数：$p_{c_1}(\boldsymbol{x})$，$p_{c_2}(\boldsymbol{x})$，$p(\boldsymbol{x})$。提示：利用叠加的钟形函数的思路。

2）利用上一步得到的概率密度函数，决定$\boldsymbol{x}=[1.4, 3.3, 3.0]$ 应该属于 c_1 类还是c_2 类。

3. 回到第 1 章中“派”的域，从训练集中移除一个正类样例，如左上角的那个（表 1.1 的第一行）。假如属性是相互独立的，计算这个特定的样例属于正类的概率。

思考题

1. 怎样在一个有 3 种可能输出：$[A, B, C]$，每个都有相同先验概率估计$\pi_A=\pi_B=\pi_C=1/3$ 的域中应用 m – 估计？如果你信赖对 A 的期望但是对 B 和 C 不确定呢？有没有一种能够在参数 m 的值上反映这种情况的方法？
2. 给出在什么条件下概率估计的准确率会受益于属性相互独立这一假设，讨论其优势和劣势。
3. 在一个有布尔属性、离散属性、连续属性的域中，如何计算输出类别的概率？讨论组合不同方法的可能性。

上机实验

1. 机器学习研究人员经常用公开的可用的基准域来测试他们的算法。这样的

域的一个大的数据库可以在以下地址找到：www. ics. uci. edu/ ~ mlearn/MLRepository. html。看一下这些数据并了解它们在属性个数、属性类型、大小等方面的不同。

2. 写一个计算机程序，它可以在所有属性都是离散值的域中利用贝叶斯公式计算类别的概率。将这个程序用于“派”的域。
3. 对于连续属性的情况，写一个计算机程序，如上面练习题第 3 题那样，它可以用表格的形式接受训练样例。基于这些，用这个程序近似概率密度函数，然后用它们来决定未来样例的类别。
4. 将上机实验第 3 题中编写的程序用于一些 UCI 库中的基准域上（从那些属性都是连续的库中选择），并注意这个程序在一些域上会比在另一些域上取得更好的结果。

第 3 章　相似性：最近邻分类器

两棵看起来非常相像的植物有可能是同一个物种；同样地，诉说相似症状的病人患同样的病也是非常常见的。总的来说，相似的事物经常是属于同一个类别的——这种观察形成了一种常用分类方法的基础：当判断对象 $\boldsymbol{x}$ 的类别时，找到与它最相似的训练样例，那么这个样例的类别即为对象 $\boldsymbol{x}$ 的类别。

这一章主要解释如何评价样例与样例之间的相似性，介绍为了达到分类的目的而使用这些相似性的具体机制，比较这种方法与贝叶斯分类器的性能，以及介绍一些克服内在弱点的方法。

3.1 k 近邻法则

我们如何确定一个对象更相似于 $\boldsymbol{x}$ 而不是 $\boldsymbol{y}$ 呢？有人可能会怀疑这样做的可能性。一匹马和一匹斑马相比，长颈鹿会更相似于一匹马吗？这类问题引起了质疑。在回答这些问题的时候，太多随意的和主观的因素会牵涉其中。

属性向量的相似性。正如本书所阐述的，机器学习的任务是把这些异议减到最少。分类器会基于属性的描述去比较对象而不是实际的对象本身——这似乎是一个可以解决的问题。因此，在第 1 章提到的玩具域里面，两个派的相似性可以通过计算它们相异的属性来确定：差异越小，相似度越高。表格 3.1 的第一行给出了对象 $\boldsymbol{x}$ 的属性值。对于 12 个训练样例，最右侧的一列显示了 $\boldsymbol{x}$ 与给定样例不同属性的个数。可以找到的最小值对应于ex_5，因此可以得出结论：对象 $\boldsymbol{x}$ 与ex_5是最相似的，同时意味着我们应该把 $\boldsymbol{x}$ 标记为 pos，即ex_5的类别。

表 3.1　计算离散属性向量之间不同属性值的个数

样例	Shape	Crust		Filling		Class	#差别
		Size	Shade	Size	Shade		
x	Square	Thick	Gray	Thin	White	?	—
ex1	Circle	Thick	Gray	Thick	Dark	pos	3
ex2	Circle	Thick	White	Thick	Dark	pos	4
ex3	Triangle	Thick	Dark	Thick	Gray	pos	4

（续）

样例	Shape	Crust		Filling		Class	#差别
		Size	Shade	Size	Shade		
ex4	Circle	Thin	White	Thin	Dark	pos	4
ex5	Square	Thick	Dark	Thin	White	pos	1
ex6	Circle	Thick	White	Thin	Dark	pos	3
ex7	Circle	Thick	Gray	Thick	White	neg	2
ex8	Square	Thick	White	Thick	Gray	neg	3
ex9	Triangle	Thin	Gray	Thin	Dark	neg	3
ex10	Circle	Thick	Dark	Thick	White	neg	3
ex11	Square	Thick	White	Thick	Dark	neg	3
ex12	Triangle	Thick	White	Thick	Gray	neg	4

处理连续的属性也是非常简单的。事实上，每个样例都能表示为 n 维空间中的一个点，这个事实使得计算每对样例之间的几何距离成为可能，例如，计算欧氏距离（3.2 节将会更多地叙述如何计算向量之间的距离）。再次说明，样例在空间中的距离越小，它们之间的相似度越高。最近邻分类器因此而得名：在样本空间中，与 x 在几何意义上距离最小的训练样例就是 x 的最近邻。

从单一邻居到 *k* 个邻居。在有噪声的域，最近邻的真伪是不可靠的。如果这个单一样本的类标本身就是不正确的呢？一个更加鲁棒的方法是找到多个最近邻而不是唯一的一个，然后让它们来投票。这就是 k 近邻分类器的实质，其中 k 是参与投票的近邻个数，通常由用户指定。表 3.2 中的伪代码总结了这一运算法则。

表 3.2　*k* 近邻分类器的一个最简单版本

假设我们有一种度量属性向量之间相似性的机制，以此来决定 x 的类别。

1. 在训练样本中，找到 x 的 k 个最近邻（与 x 最相似的样例）。
2. 假如 c_i 是这 k 个最近邻中包含最多的类别。
3. 则 x 的类别即为 c_i。

值得注意的是，当一个域中只存在两个类时，为了避免出现两个类数量均分的情况，k 应该取一个奇数。例如，一个4近邻分类器就可能遇到正类近邻的数目与负类近邻的数目相等的情况。而对于5近邻分类器，就不会出现这样的情况。当一个域中不止两个类时，即便 k 取奇数也不可避免出现这种平局的情况。例如，一个7近邻分类器可以找到3个属于 C_1 类的近邻、3个属于 C_2 类的近邻和一个属于 C_3 类的近邻。因此在设计分类器时，需要定义一种能够在 C_1 和 C_2 类中做出选择的机制。

一个图解。这个范例的某些“行为层面”可以通过一个假想域而变得非常明显且容易可视化，在这个假想域中每个样例都由两种数值属性来刻画。图3.1显示了一些正类和负类的训练样例，以及一些需要用 k 邻近分类器来决定其类别的对象（用黑色的圆点表示）。读者可以看到，对象1和对象2被同类的样例包围着，所以它们的分类是简单直接的。然而，对象3位于正类区域与负类区域之间的“无人之地”，所以即使是少量的属性噪声也能混淆对象的分类。因此这些边界样例的分类是不可靠的。

在图3.1的右侧，对象4在正类区域的内部，但是类别噪声却把它训练集里的近邻错误地标记为负类。在这里，如果使用1近邻分类器必然会导致错误的分类，但使用3近邻分类器却会给出正确答案，因为其他两个正类近邻的投票会超过一个负类近邻的投票。

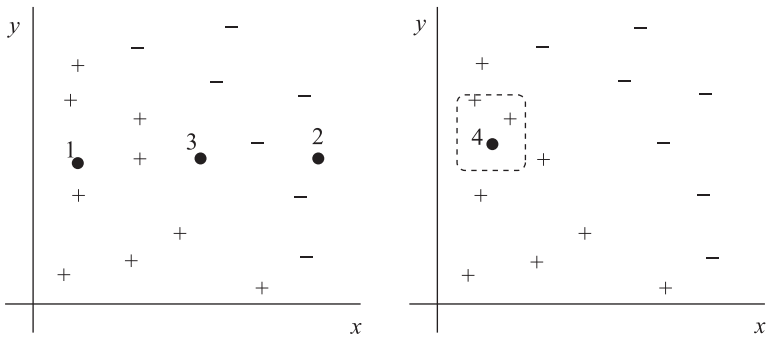


图3.1　位于边界区域的对象3是比较难分类的。在右侧所示的有噪声域，1近邻分类器会造成对象4错误分类，但是3近邻分类器会对对象4进行正确分类

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在属性都是离散值的情况下如何度量样例之间的相似性？在属性都是连续值的情况下呢？
- 在什么样的情况下，k 近邻分类器（$k>1$）会比 1 近邻分类器表现好？为什么？
- 为什么当一个域中存在两个类时，k 近邻分类器中的 k 应该为奇数？
- 属性噪声是如何影响一个边界样例的分类情况的？类别标号噪声有怎样的影响？

3.2 度量相似性

正如指出的那样，寻找对象 x 的最近邻的一个常用方法就是比较从 x 到每个训练样例之间的几何距离。图 3.1 用一个甚至可以用尺子直接量出距离的简单的域来说明这个原理。但是如果样例具有两个以上的属性，那么尺子就不能作为我们选择的工具了。此时我们需要的是一个可以通过属性值来计算相似性的表达式。

欧式距离（Euclidean Distance）。在一个二维平面内，$\boldsymbol{x}=(x_1, x_2)$ 和 $\boldsymbol{y}=(y_1, y_2)$ 两点之间的几何距离是可以通过勾股定理计算出来的，如图 3.2 所示，即 $d(\boldsymbol{x}, \boldsymbol{y})=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$。这个公式可以很轻易地推广到有 n 个连续属性的域，这里 $\boldsymbol{x}=(x_1, \cdots, x_n)$ 和 $\boldsymbol{y}=(y_1, \cdots, y_n)$ 之间的欧式距离定义如下：

$$d_E(\boldsymbol{x},\boldsymbol{y})=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \qquad (3.1)$$

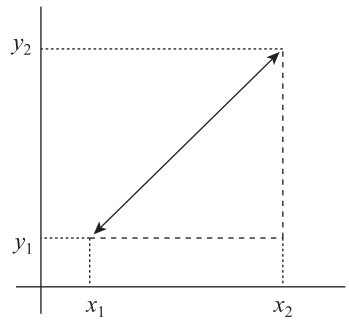


图 3.2　二维空间中，两个点之间的欧式距离等于直角三角形斜边的长度

表 3.3 展示了如何在 k 近邻分类器中使用这个

距离度量的方法，所使用的训练集包含 4 个样例，每个样例具有 3 个数值属性。

表 3.3　在一个三维欧式空间中使用最近邻准则

使用如下的训练集：其中包含 4 个样例，每个样例有 3 个数值属性，以此判断物体 $x=[2, 4, 2]$ 的类别。

ex_i 与 [2, 4, 2] 之间的距离		
ex_1	{[1, 3, 1], pos}	$\sqrt{(2-1)^2+(4-3)^2+(2-1)^2}=\sqrt{3}$
ex_2	{[3, 5, 2], pos}	$\sqrt{(2-3)^2+(4-5)^2+(2-2)^2}=\sqrt{2}$
ex_3	{[3, 2, 2], pos}	$\sqrt{(2-3)^2+(4-2)^2+(2-2)^2}=\sqrt{5}$
ex_4	{[5, 2, 3], pos}	$\sqrt{(2-5)^2+(4-2)^2+(2-3)^2}=\sqrt{4}$

通过计算 x 与所有训练样例之间的欧式距离，我们发现 x 的最近邻是ex_2。因为它的类标是 pos，所以最近邻分类器返回正类的类标。

3 近邻分类器也得到了一样的结果，因为两个 x 的最近邻（ex_1 和ex_2）均属于正类，仅有一个（ex_4）属于负类。

一个更通用的公式。读者可以看到根号下面的项是一个关于对应属性的平方距离的和［这些差值取平方项，能保证一定是正值，这样做的一个好处是防止了负的差值（$x_i-y_i<0$）会从正的差值（$x_i-y_i>0$）当中减去］。这个公式在数学上表示如下：

$$d_M(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} d(x_i,y_i)} \tag{3.2}$$

这就是当离散和连续属性混合在一起时一般的向量之间距离的计算方法（在符号 $d_M(x, y)$ 中，通过下标字母 M 来标识）。例如，我们可以把 $d(x_i, y_i)=(x_i-y_i)^2$ 用于连续向量，然而对于离散数学，如果 $x_i=y_i$，我们令 $d(x_i, y_i)=0$，如果 $x_i\neq y_i$，则令 $d(x_i, y_i)=1$。

注意，如果所有属性都是连续的，那么这个公式跟欧氏距离的计算公式是一样的；而如果所有属性都是离散的，那么这个公式则简单指出了两个向量不同属性值的个数。在纯布尔域，所有属性只能从正确和错误当中取值（我们分别将其缩写为 t 和 f），后一种的情形被称为海明距离（d_H）。例如，向量 $\boldsymbol{x}=(t, t, f, f)$ 与向量 $\boldsymbol{y}=(t, f, t, f)$ 的海明距离为 $d_H(\boldsymbol{x}, \boldsymbol{y})=2$。通常来说，式（3.2）用在

样例由离散和连续的属性混合起来描述的域中。

属性与属性间的距离可能被误导。我们必须注意，不能机械地运用式（3.2）而无视了给定域的特殊性。让我们简单地讨论两种容易出错的情况吧。

假定我们的样例是由3个属性来描述的：尺寸、价格、季节。其中，前两个显然是连续的，但最后一个是离散的。如果 $\boldsymbol{x}=(2, 1.5, \text{夏季})$ 和 $\boldsymbol{y}=(1, 0.5, \text{冬季})$，那么根据式（3.2），它们两者的距离计算如下：

$$d_M(\boldsymbol{x}, \boldsymbol{y})=\sqrt{(2-1)^2+(1.5-0.5)^2+1}=\sqrt{3}$$

我们首先来考虑一下第3个属性：夏季和冬季是不同的，如前所述，我们确定 d（夏季，冬季）=1。事实上，我们通常认为夏季与冬季的差异比夏季与秋季（相邻季节）的差异更大。至少就天气而言，春季与秋季比夏季与冬季是更加相似的。因此我们认为，只有0和1两个取值是明显不够的。我们应该考虑更多的中间值，而具体的选择取决于所给情境的特定需求。忽略此因素可能不会得到准确的结果。

在另一方面，将连续和离散的属性混合在一起也是非常危险的。不假思索地使用式（3.2）可能会导致尺寸这个属性的差异［例如，$\text{size}_1=1$ 和 $\text{size}_2=12$，意味着 $d(\text{size}_1, \text{size}_2)=11^2=121$］与季节的差异相比占据优势（事实上，它是不能超过1的）。这个结论与下一节要讨论的关于**尺度缩放**的问题是紧密相关的。

广义的距离。读者开始意识到相似性的问题并非是微不足道的。除了式（3.2），还有非常多的公式已经被提出，其中的一部分是非常复杂的（这些公式中，可能较著名的包括极距离、Minkowski度量和Mahalanobis距离）。尽管知道它们的存在是比较好的，但我们将不会在这里提及它们，因为我们不想过度地偏离我们的主题，但这足以说明距离度量必须满足以下要求：

1. 距离不可以是负的。
2. 相同的两个向量间的距离为0。
3. $\boldsymbol{x}$ 到 $\boldsymbol{y}$ 与 $\boldsymbol{y}$ 到 $\boldsymbol{x}$ 的距离是一样的。
4. 距离度量必须满足三角不等式：$d(\boldsymbol{x}, \boldsymbol{y})+d(\boldsymbol{y}, \boldsymbol{z})\geqslant d(\boldsymbol{x}, \boldsymbol{z})$。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 什么是欧式距离？什么是海明距离？它们都可以用于哪些域？距离与相似性是怎样联系起来的？
- 若域中的样例同时包含连续和离散属性，则写出所需的距离公式。讨论在现实中直接应用它会造成的困难之处。
- 任何距离度量都需要满足的基本性质有哪些？

3.3 不相关属性与尺度缩放问题

到目前为止，读者已经很好地理解了 k 近邻分类器的原理，并可以写出计算机程序来实现这一工具。但是，这还不够。如果只是机械地使用，那所设计出的软件很可能令人感到失望。所以，有必要知道这是为什么。

最近邻范例的基石是“所描述对象的向量之间的几何距离越小，则对象越相似”。我们必须注意不能让这个准则退化为一个教条。在一些情况下，几何距离又会形成误导。这两者有着一定的共同性。

不相关属性。如果认为所有的属性都是平等的，那就不对了，它们是不平等的。在机器学习中，一些属性是不相关的，因为它们的值与给定样例的类别是毫无关系的。但是，它们却会影响向量之间的几何距离。

一个简单的图示会阐明这一点。在图 3.3 中的训练集中，样例由两个数值属性来刻画：体温（横轴）和鞋号（纵轴）。黑点代表了要用 k 近邻分类器来预测为健康（pos）或生病（neg）的对象。

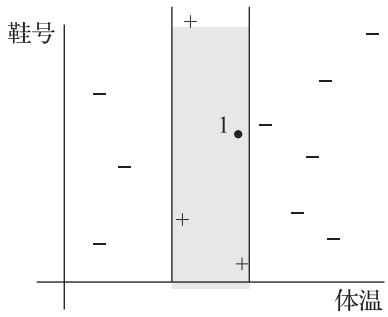


图 3.3 纵轴的属性对分类是无关的，但是它会影响样例之间的几何距离。对象 1 属于正类，尽管在这个二维空间中它的最近邻点属于负类

正如所料，所有的正类样例都位于“水平”属性上两个关键点所限定的阴影区域之内：温度高于最大值意味着高烧，低于最小值意味着低体温。对于“纵向”属性，我们可以看到正类样例和负类样例是散落在整个区间的，但是鞋号不能反映出一个人的健康信息。我们要分类的对象在阴影区域，尽管这个对象的最近邻属于负类，但是常识告诉我们它应该被标记为正类。

教训。如果仅用第一个属性，则两个样例之间的欧式距离是 $d_E(x, y) = \sqrt{(x_1 - y_1)^2} = |x_1 - y_1|$。如果两个属性都用，则欧氏距离会是 $d_E(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$。如果第二个属性是无关的，那 $(x_2 - y_2)^2$ 这一项是多余的，但是相反地，它对 k 近邻的相似性的计算是有影响的。这就是在图 3.3 中所发生的，也是为什么对象 1 被错误分类的原因。

不相关属性会造成多大的损失取决于有多少不相关属性被用来描述样例。在一个有几百个属性的域，仅有一个属性是无关的，那不必惊慌：单个的不良因素在任何意义上不太可能改变 $d_E(\boldsymbol{x}, \boldsymbol{y})$ 的值。但是，当不相关属性的比例逐渐提高事情就会发生变化了。如果大量的属性对于我们要识别的类别没有任何价值，那几何距离将变得几乎没有意义，分类器的表现也会逊色不少。

属性值的尺度。假设我们要评估两个样例的相似性：$\boldsymbol{x} = (\mathrm{t}, 0.2, 254)$ 和 $\boldsymbol{y} = (\mathrm{f}, 0.1, 194)$。它们都是用 3 个属性来描述的，其中第一个是布尔值，第二个是取值于 [0，1] 区间的连续值，第三个是取值于 [0，1000] 区间的连续值。使用式 (3.2)，读者很容易计算 x 和 y 之间的距离，得到如下结果：

$$d_M(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(1-0)^2 + (0.2-0.1)^2 + (254-194)^2}$$

检查这个表达式，我们会注意到第三项完全控制了计算结果，其他两项对结果毫无意义。不论我们怎样在它们的定义域调整取值，距离 $d_M(\boldsymbol{x}, \boldsymbol{y})$ 的值几乎不受影响。幸运的是，这种情况很容易得到修正。在训练集中，如果我们将第三个属性的值除以 1000，即将其值域“挤压”到 [0，1]，则属性之间的影响将变得更加平衡。我们可以看到，属性值的尺度会从根本上影响 k 近邻分类器的表现。

属性尺度缩放的另一个方面。对于属性尺度缩放还有更多内容要讲。考虑如

下的两个训练样例：ex_1 和ex_2，要决定其类别的对象 x：$ex_1 = [(10, 10), pos]$，$ex_2 = [(20, 0), neg]$，$\boldsymbol{x} = (32, 20)$。

距离分别是 $d_M(\boldsymbol{x}, ex_1) = \sqrt{584}$和 $d_M(\boldsymbol{x}, ex_2) = \sqrt{544}$。后者要小点儿，这样最近邻分类器将 $\boldsymbol{x}$ 标注为 neg。然而，假如第二个属性表示的是温度，单位是摄氏度（℃）。如果我们要用华氏温度（℉）来替换，则 3 个向量将变为：$ex_1 = [(10, 50), pos]$，$ex_2 = [(20, 32), neg]$，$\boldsymbol{x} = (32, 68)$。

重新计算距离，我们有 $d_M(\boldsymbol{x}, ex_1) = \sqrt{808}$和 $d_M(\boldsymbol{x}, ex_2) = \sqrt{1440}$。这次第一个距离要小点儿，因此最近邻分类器会将 $\boldsymbol{x}$ 标注为 pos。这看起来有点儿奇怪。除了选择了不同的温度单位，所有的样例完全一样，但是分类器的决策却发生了变化。

标准化属性尺度。解决这个问题的一个办法是标准化属性尺度：重新缩放使得所有的属性取值位于相同的区间，即 [0, 1]。截至目前所用到的若干机制，最简单的一个应该是：首先对于给定的属性找到它的最大值（MAX）和最小值（MIN），然后用 x 代替每一个属性值，代入到式（3.3）中：

$$x = \frac{x - \text{MIN}}{\text{MAX} - \text{MIN}} \tag{3.3}$$

下面用一个简单的例子来阐述一下。假设训练集中有 5 个样例，一个给定的属性分别有如下的值：[7, 4, 25, −5, 10]。

我们观察到 MIN = −5、MAX = 25。从各个值中减去 MIN，我们得到如下结果：[12, 9, 30, 0, 15]。

读者会发现新的最小值是 0，而新的最大值是 MAX − MIN = 25 − (−5) = 30。用 MAX − MIN 去除上面得到的结果，所有值都将落入区间 [0, 1]，即 [0.4, 0.3, 1, 0, 0.5]。

标准化的一个潜在的弱点。在许多实际应用中，标准化降低了错误率，特别是在原始属性的尺度显著不同的情况下。一个负面作用是样例的描述会变得扭曲。而且，使得所有数值都落入 0 ~ 1 之间的实用主义的决定可能是不合理的。例如，如果夏季和秋季的差为 1，则它会永远大于两个标准化的体温的差值。这是否重要取决于工程师的常识，包括工程师的经验以及所有的实验经历。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 为什么不相关属性的存在会影响 k 近邻分类器的表现？k 近邻分类器的表现与不相关属性的数目之间的关系是什么？
- 解释关于属性尺度缩放的基本问题。描述一个简单的标准化方法。

3.4 性能方面的考虑

花了这么多时间来探索 k 近邻分类器，读者不免会问：我真的需要关心这些吗？答案是肯定的，技术是很容易通过计算机编程来实现的，功能也很容易掌握。但是这样就代表它的分类性能已经足够好了吗？

最近邻分类器与理想贝叶斯的比较。我们给出一些对这个问题的思考。最终的度量 k 近邻分类器的潜能的准则是贝叶斯方法。我们可以回想一下，如果用于贝叶斯公式的概率和概率密度函数是已知绝对精确的，那么在理论上，对于给定的（噪声）数据，我们使用这个分类器（我们称之为理想贝叶斯）是可以达到最低错误率的。可以肯定的是，使用 k 近邻方法也不会相差很多。

这个问题已经从数学上做过严格分析，这里我们将给出结果。图 3.4 展示了在一定的理想条件下，如无限大的训练集以及无限的密度填充了整个样本空间的理想贝叶斯和 k 近邻的对比结果。实线代表了域中存在两个类的情形，即样例或属于正类或属于负类。我们可以看到，如果理想贝叶斯的错误率是 5%，则最近邻分类器（纵轴）的错误率是 10%。随着噪声量的增加，两种方法的差异在逐渐减小，当理想贝叶斯有 50% 的错误率时，两者的差异就会消失，当然，在这种情况下，训练样例的类别实际上是随机的，任何的自动分类的尝试都是没有用的。图中虚线代表的是域中存在多类别的情形，可见情况并没有任何好转。再一次地，理想贝叶斯在特定情形下的性能优于最近邻分类器。

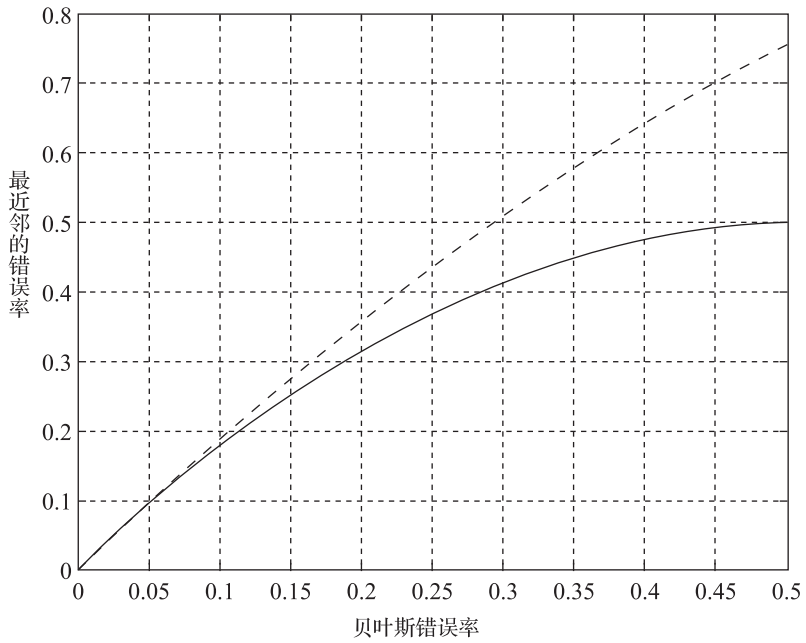


图 3.4　最近邻准则的理论错误率与理想贝叶斯的理论错误率（实线代表域中存在两个类的情形，虚线代表域中存在多个类的情形）

增加近邻个数。从最近邻分类器的角度看，图 3.4 的结果是比较令人失望的。另外，我们知道当我们使用更加通用的 k 近邻分类器（$k>1$）时，近邻分类器的性能会有所改善，因为一些噪声的近邻点参与投票时会被其他近邻点抑制。那么，数学上也会给这一预期提供一些支持吗？

是的，确实如此。在以上提到的理想情况下，已经证明错误率随着 k 值的增加在减小，直到 $k\to\infty$ 时收敛到理想贝叶斯的错误率。至少在理论上，最近邻分类器的性能是可以达到最大绝对值的。

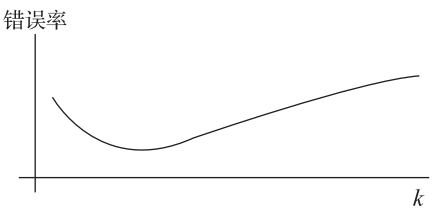


图 3.5　随着投票近邻个数（k）的增加，k 近邻分类器的错误率降低到一定水平后又开始上升

理论的现实约束。工程师们是不关心理论的要求的。在实际应用中，训练样例是非常稀疏地散落在样本空间的，而且增加投票近邻的个数会适得其反。更多的情况是，随着 k 值的增加错误率确有改进，但是相对于起点来说，改善只是一点点，正

如图3.5所描绘的，这里横轴代表了 k 的值，纵轴代表了在一个独立的测试集得到的错误率。

解释很简单：在样本空间中，相对较远的近邻所在的区域可能已经被其他类所占据，这样它们就会误导分类器。考虑一个极端情况：如果训练集中包含25个训练样例，那么25近邻分类器是没用的，因为它只是简单地标注了所有对象在训练集中出现的次数（最优的 k 值，也就是对应于最小错误率的 k 值，一般是通过实验获得的）。

维数灾难。显然，当对一个对象 x 进行分类时，一些它的近邻点并不足够相似于它，但却能够参与到投票中。这种情况经常发生在包含许多属性的域。假设每个属性的取值都限制在单位长度的区间［0，1］。使用勾股定理，很容易发现：在由属性所定义的 n 维空间中最大的欧式距离是 $d_{\mathrm{MAX}}=\sqrt{n}$。对于 $n=10^4$（在一些领域，如文本分类），这意味着 $d_{\mathrm{MAX}}=100$。考虑到没有属性的值会超过100这一事实，这个结论可能会令人吃惊。怪不得样例是趋向于稀疏的，除非训练集真的很大。

在这种情境下，常被提到的一个术语是维数灾难：随着属性个数的增加，需要以合适密度填充样本空间的训练样例的数目以快速增长，但可能因为增速太快而使得最近邻分类不再实用。

总结。尽管理想的 k 近邻分类器能够达到理想贝叶斯的分类性能，工程师需要注意这两种方法的现实局限性。在概率和概率密度函数完全缺失的情况下使用理想贝叶斯是不现实的。另外，k 近邻分类器往往会受到数据稀疏、不相关属性以及尺度问题的影响，具体的选择需要依据给定应用问题的特定需求来决定。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- k 近邻分类器与理想贝叶斯的分类表现对比情况是怎样的？分别对 $k=1$ 和 $k>1$ 的情况进行总结。这两个范式都依赖于什么样的理论假设？

- 在理论和实际应用方面，k 近邻分类器的表现是怎样依赖于 k 值的增长的？
- 怎样理解维数灾难？

3.5 加权最近邻

到目前为止，投票机制是被假设为民主的，每个近邻有相同的投票权。这看起来是个足够公平的安排了，但是从分类性能的角度考虑，我们还可以做得更好。

在图 3.6 中，我们需要确定对象 $\boldsymbol{x}$ 的类别，它用一个黑点表示。因为有 3 个近邻属于负类，两个属于正类，所以 5 近邻分类器会把它判定为负类。但是，这里看起来有点问题：3 个负类的近邻点离 $\boldsymbol{x}$ 特别远；这样，在物体的直接近邻区域，它们好像不应该和那两个正类样例一样有话语权。

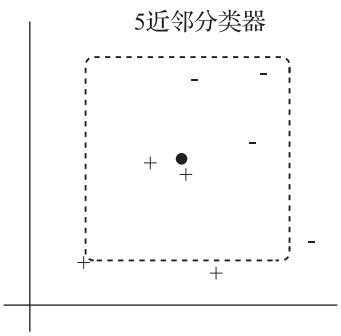


图 3.6　两个正类近邻应该比 3 个距离较远的负类近邻拥有更大权重的图示

加权最近邻。这样的情形促使了加权投票的引入，这样每一个近邻点的权重与它到 $\boldsymbol{x}$ 的距离成一定比例：近邻点越靠近，权重的影响越大。

我们用 w_1，…，w_k 来标记权重。加权 k 近邻分类器将正类的近邻的权重相加（用 $\sum^{+}$ 来标记结果），然后再将负类的近邻的权重相加（ $\sum^{-}$ ）。最终的判断取决于哪个值更大：如果 $\sum^{+} > \sum^{-}$，则 x 被认为是正类的，否则就认为它是负类的。以此推广到 $n>2$ 的情况。

为了图示的方便，假设在近邻中发现正类标号权重分别是 0.6 和 0.7，负类标号的权重分别是 0.1，0.2 和 0.3。那么加权 k 近邻分类器会选择正类，因为正类近邻的组合权重 $\sum^{+} = 0.6 + 0.7 = 1.3$ 大于负类近邻的组合权重 $\sum^{-} = 0.1 + 0.2 + 0.3 = 0.6$。正如图 3.6 所示的一样，因为正类近邻更加靠近 x，所以在投票中战胜了出现更加频繁的负类近邻。

一个具体的公式。让我们介绍一个简单的公式来计算权重。假设 k 个近邻点按照它们到 x 的距离排列，即 d_1，…，d_k，这样 d_1 是最近距离而 d_k 是最远距离。

第 i 个靠近的近邻点的权重可以按式（3.4）进行计算：

$$w_i = \begin{cases} \dfrac{d_k - d_i}{d_k - d_1} & (d_k \neq d_1) \\ 1 & (d_k = d_1) \end{cases} \tag{3.4}$$

显然，得到的权重将在 0～1 的范围内，0 对应于最远的近邻，1 对应于最靠近的近邻。这意味着这个方法实际上只考虑了 $k-1$ 个近邻点（因为 $w_k = 0$）。当然，这仅对于 $k > 3$ 是有意义的。如果我们用 $k = 3$，那么只有两个近邻点真的参与了进来，而且加权的 3 近邻分类器会退化成最近邻分类器。

表 3.4 描述了在一个简单的玩具域上的程序。

表 3.4　加权最近邻法则的示例

任务是用加权的 5 近邻分类器来决定对象 x 的类别。假设 x 与 5 个最近邻的距离分别为 $d_1 = 1, d_2 = 3, d_3 = 4, d_4 = 5, d_5 = 8$。因为最小值是 $d_1 = 1$，而最大值是 $d_5 = 8$，所以每一个权重可以这样计算：$w_i = \dfrac{d_5 - d_i}{d_5 - d_1} = \dfrac{8 - d_i}{8 - 1} = \dfrac{8 - d_i}{7}$。这样可以给出如下结果：$w_1 = \dfrac{8-1}{7} = 1, w_2 = \dfrac{8-3}{7} = \dfrac{5}{7}, w_3 = \dfrac{8-4}{7} = \dfrac{4}{7}, w_4 = \dfrac{8-5}{7} = \dfrac{3}{7}, w_5 = \dfrac{8-8}{7} = 0$。

如果两个最近邻属于正类其他的 3 个属于负类，则 x 将被分到正类，因为 $\sum^{+} = 1 + \dfrac{5}{7} > \sum^{-} = \dfrac{4}{7} + \dfrac{3}{7} + 0$。

另一个值得注意的事情是，如果所有的近邻点与 x 有相同的距离，那么按照式（3.4），它们将有同样的权重，即 $w_i = 1$。读者也可以容易地验证 $d_k = d_1$，当且仅当所有的 k 个近邻点与 $\boldsymbol{x}$ 有相同的距离。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 为什么我们要考虑每一个投票近邻点应该有不同的权重？
- 讨论文中推荐的用于计算单个权重的公式的表现形式。

3.6 移除危险的样例

如果所关心的是分类的程序，则每一个训练样例的价值可能是不同的：一些是所代表的类别中典型的，一些稍弱点，但是另外的一些可能完全就是误导。这就是为什么在使用训练集之前先要对它进行预处理：移除那些被认为是无效的样例。

图 3.7 所述的两种情形引出了两个对训练集进行预处理的具体方法。首先，一个样本被标注为一个类别，但是它被其他类别的样例所包围着，那么它很可能是由类别标号噪声所造成的。其次，位于两个类的边界区域的样例是不可靠的，因为它们的属性值可以被很小的噪声影响，而令它们的位置向着错误的方向发生改变，进而影响分类。这两种情况下的样例最好被移除。

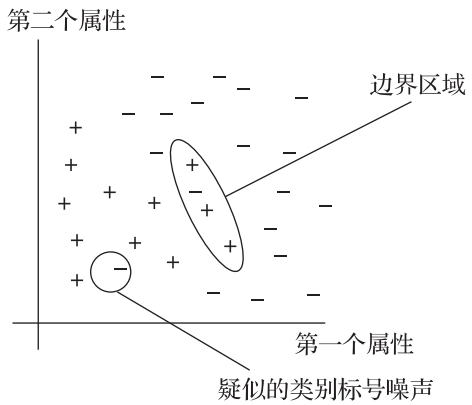


图 3.7　两种潜在有害类型的样例的图示（被不同类别的样例所包围的和位于边界区域的）

托梅克（Tomek）连接技术。然而，在危险样例被移除之前，它必须先被检测到。一种办法是通过托梅克连接技术，这个名字是以几十年前第一个使用它的数学家来命名的（顺便提一下，他是因其他不同的目的而使用托梅克连接的）。一个样例对，$\boldsymbol{x}$ 和 $\boldsymbol{y}$，如果同时满足如下 3 个要求，则我们就说它们形成了一个托梅克连接：

1．$\boldsymbol{x}$ 是 $\boldsymbol{y}$ 的最近邻。

2. $\boldsymbol{y}$ 是 $\boldsymbol{x}$ 的最近邻。

3. $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的类别不同。

这些条件是边界样例的特征，也是被其他类别的样例所包围的样例的特征，从训练集中移除这些样例对是有道理的。甚至这些还不够。有时候，移除存在的托梅克连接还会生成新的托梅克连接，因此，这个程序必须重复执行多次。

表 3.5 中的伪代码总结了这个算法，图 3.8 中展示了一些托梅克连接的例子。注意，这里只有这 3 对，没有其他样例对再满足托梅克连接的条件了。

表 3.5　确认（和移除）托梅克连接的算法

输入：N 个样例的一个训练集。

1. 令 $i=1$，T 为一个空集。
2. 令 $\boldsymbol{x}$ 是第 i 个训练样例，$\boldsymbol{y}$ 是 $\boldsymbol{x}$ 的最近邻。
3. 如果 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 属于同一个类别，则转到 5。
4. 如果 $\boldsymbol{x}$ 是 $\boldsymbol{y}$ 的最近邻，则令 $T=T\cup\{\boldsymbol{x},\boldsymbol{y}\}$。
5. 令 $i=i+1$。如果 $i\leqslant N$，则转到 2。
6. 从训练集中将所有在 T 中的样例全部移除。

但存在一个问题：因为令 $k>1$ 的主要原因就是减轻噪声的消极影响，然而一旦移除托梅克连接，参与投票的近邻个数 k 也会相应减少，甚至有可能出现使用 1 近邻分类器就可以取得 k 近邻分类器在整个原始训练集上所取得的分类性能。

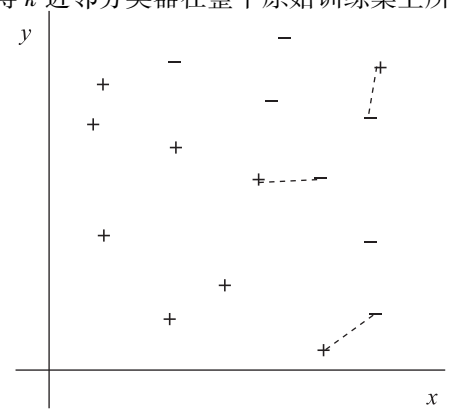


图 3.8　虚线表示了所有的托梅克连接。在每个托梅克连接中，每一个参与者都是它同伴的最近邻，而且两个样例是属于不同类别的

局限性。没有事物是完美的。托梅克连接技术并不能检测到所有导致错误的样例，相反地，一些被移除的样例可能是“无辜的”，是应该被保留的。但是经验证明移除托梅克连接确实改善了数据整体的质量。工程师需要注意以下两种特殊的情况：①当训练集非常小的时候；②当一类的样本数显著地多于另一类的时候。后一种情况会在 10.2 节再次讨论。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 是什么促使我们尝试清洗数据集的?
- 什么是托梅克连接?怎样从训练集中找到它们?为什么有时候程序必须重复执行多次?
- 移除托梅克连接是怎样影响 k 近邻分类器的?

3.7 移除多余的样例

一些训练样例并不会对分类器的性能产生消极影响，但是我们还是想移除它们。因为即使它们的存在不会降低错误率，但是增加了计算代价。

多余的样例。要在一个有10^6个训练样例，每个样例由10^4个属性描述的域中找最近邻（这样的域并不稀少），使用欧氏距离的程序需要执行$10^6 \times 10^4 = 10^{10}$次算术运算。当任务是要同时分类几千个物体时，算术运算的次数就是$10^{10} \times 10^3 = 10^{13}$次。这是很大的一个数字。

幸运的是，因为训练集往往都是冗余的，所以即使删除一些训练样例，也不会影响 k 近邻分类器的性能。甚至有时候移除大量的样例也没有太大影响，因为这些样例增加了分类的计算代价而对分类性能没有影响，这就是图 3.9a 所示的情况。

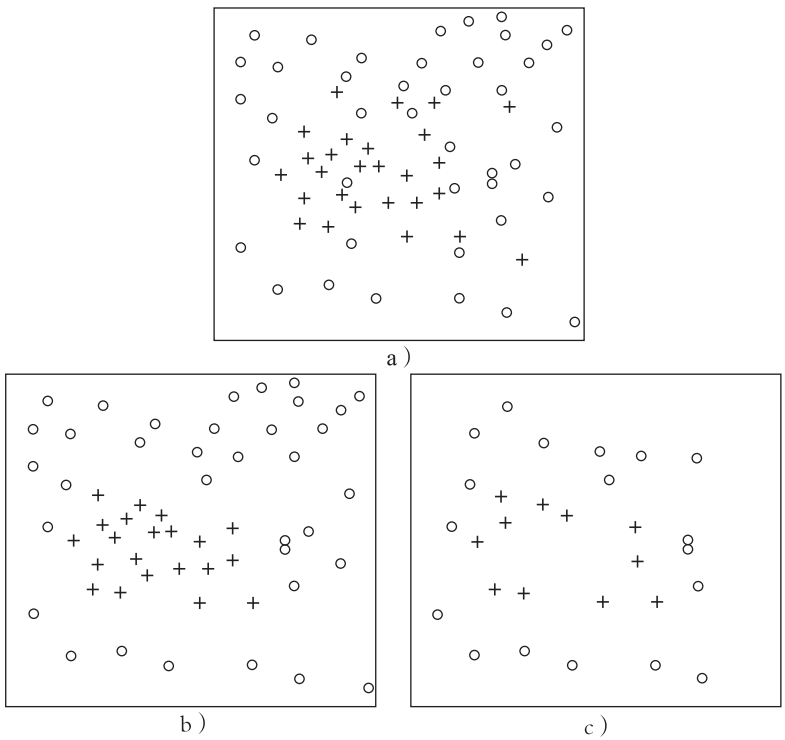


图 3.9　图示当边界、噪声和多余样例从训练集中移除之后会有什么变化
a）原始训练集　b）边界和噪声样例被移除　c）多余样例被移除

一致子集。在尝试降低冗余度的过程中，我们要将训练集 T 替换为它的一致子集 S。在这里指的是，如果用 S 替换 T 而不影响 k 近邻分类器返回的类别标号，则称 S 是 T 的一致子集。但是这样的定义并不是很实际，因为我们并不知道使用 T 或 S 时，k 近邻分类器在未知样例上会有怎样的性能。因此，我们来修改一下这个标准：对于任何的 $\boldsymbol{ex} \in T$，不管 k 近邻分类器是基于 $T - \{\boldsymbol{ex}\}$ 的还是基于 $S - \{\boldsymbol{ex}\}$ 的，ex 都获得相同的类标，则 S 便是 T 的一致子集。

通常一个现实的训练集会有很多个一致子集，并且子集越小越好。但是坚持要找到最小子集的完美主义者应该明白：要想总是找到最小子集需要付出很大的计算代价。考虑实际情况的工程师会怀疑这样做的合理性，他们更倾向于更加高效的算法，能够将原始集合缩小到合理的规模，尽管这样的形式化可能看起来不太科学。

构建一致子集。表 3.6 展示了一项实用的技术。首先，这个算法从每个类中随机选择一个样例。然后，这个初始子集 S 被最近邻算法用来评估所有训练样例的类标。在这一步，很可能有的训练样例被错误地分类。这些错误分类的样例会被加到 S 中，继续使用这个更大些的集合使整个程序再重复运行。直到集合 S 变得有足够的代表性，使得最近邻分类器能够正确地标记所有的训练样例，这时搜索停止。

表 3.6　通过移除一些多余的样例构建一个一致的训练子集的算法

1. 使得 S 包含训练集 T 中的一个正类样例和一个负类样例。
2. 基于 S 中的样例，用最近邻分类器对 T 中的样例重新分类。假设 M 是这时被错误分类的样例的集合。
3. 将 M 中的样例复制到 S 中。
4. 如果 S 中的元素未发生改变，则算法停止；否则转到步骤 2。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 从训练集中移除多余的样例有什么好处?
- 训练集的一致子集是什么意思? 为什么不一定总是要找到最小的一致子集?
- 解释构建一个合理规模的一致子集的简单算法的原理。

3.8 总结和历史简评

- 分类对象 $\boldsymbol{x}$ 的时候，k 近邻分类器在训练集中找到 k 个与 $\boldsymbol{x}$ 最相似的样例，然后在这些样例中选择最常见的类别标号。
- k 近邻分类器的分类性能很大程度上依赖于属性向量之间的相似性的计算。

可能建立向量$\boldsymbol{x}$和$\boldsymbol{y}$之间相似性最简单的方法就是用式(3.5)计算得到的几何距离：

$$d_M(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} d(x_i,y_i)} \tag{3.5}$$

本质上，对于连续向量，我们令$d(x_i,y_i) = (x_i - y_i)^2$；对于离散向量，如果$x_i = y_i$，我们令$d(x_i,y_i) = 0$，如果$x_i \neq y_i$，令$d(x_i,y_i) = 1$。

- 如果所描述样例的许多属性都是无关的，则k近邻分类器的性能会很差。另一个问题是关于属性值的尺度，这个问题可以通过将属性值标准化到单位区间来缓解。
- 因为训练集中的某些有害样例会增加分类错误率。还有的样例可能是多余的，因为它们增加了计算代价却不能改进分类性能。所以，有害的和多余的样例应该被移除。
- 在一些应用中，每一个近邻点是有相同的投票权的。在另一些应用中，根据样例与被分类对象间的距离，使用了加权投票机制。

历史简评。最近邻分类器的思想最早由菲克斯（Fix）和霍奇斯[23]（Hodges）提出，但是最早地、系统地分析是由卡渥（Cover）、哈特[16]（Hart）和卡渥[15]（Cover）给出的。随后，达萨拉斯（Dasarathy）的书[17]对它进行了详尽的研究。这里描述的加权k近邻分类器是由杜达尼[19]（Dudani）提出的。哈特[30]（Hart）描述了最早的用于找到训练集的一致子集的技术（本章中所介绍的是它的一个经过小的修改的版本）。托梅克连接的概念由托梅克[76]提出。

3.9 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 基于表 3.7 中的训练样例，使用最近邻分类器和 3 近邻分类器来决定 $\boldsymbol{y}$ = [1，22] 的类别。解释为什么这两个分类器在分类性能上会有不同。

表 3.7　用于练习的一个简单的训练样例的集合

x_1	1	1	1	2	3	3	3	4	5
x_2	1	2	4	3	0	2	5	4	3
类别	+	−	−	+	+	+	−	−	−

2. 基于表 3.7 中的样例，用 5 近邻分类器来分类物体 $\boldsymbol{y}$ = [3，3]。注意，其中两个最近邻属于正类而 3 个属于负类。加权的 5 近邻分类器会有什么变化吗？为了看清结果，请将样例的位置画出来。
3. 再一次基于表 3.7 中的训练样例：①是否存在托梅克连接？②是否可以通过移除至少一个多余的训练样例来找到训练集的一个一致子集？

思考题

1. 讨论将 k 近邻分类器应用于“派”域的可能性。对使用多少个近邻点，使用什么距离度量，是否根据距离来进行最近邻的投票等给出一些想法。
2. 提出最近邻准则的其他变体，可以基于下面的提示：
   1）引入其他的距离度量。不要忘记它们必须满足 3.2 节末尾所提到的公理。
   2）修改投票的方案：通过假设一些样例是由一位有知识的“老师”所创建的，其他的没有经过对它们的代表性的任何考虑而从数据库中得到。老师的样例显然应该拥有更大的权重。
3. 设计一个爬山搜索算法来移除多余样例。提示：初始的状态包含整个训练集，搜索算子每次移除一个训练样例（这样的移除不能影响分类器的表现）。
4. 设计一个爬山搜索算法来移除不相关的属性。提示：保留一部分训练样例，在其上测试最近邻分类器对不同属性子集的性能。

上机实验

1. 写一个程序，它的输入是训练集、一个用户设定的 k 值和一个物体 $\boldsymbol{x}$。输出是 $\boldsymbol{x}$ 的类别标号。
2. 将上机实验第 1 题中实现的程序应用于一些 UCI 库中的基准域（www. ics. uci. edu/ ~ mlearn/MLRepository. html）。总是拿出 40% 的样例并对它们在剩余的 60% 的样例上用最近邻分类器进行分类。
3. 创建一个合成的玩具域，它包含 1000 个样例，每个样例用取值于区间［0，1］中的一对属性来描述。在由这些属性值定义的正方形［0，1］×［0，1］内，按照你自己的选择定义一个集合图形，并将在其中的样例标注为正类，其他的样例标注为负类。由这个初始的没有噪声的数据集创建 5 个数据集，每一个通过改变 p% 的类别标号得到，$p \in \{5, 10, 15, 20, 25\}$（这样就得到了不同水平的类别标号噪声）。将每个数据集划分为两个部分，第一部分将由基于第二部分的 k 近邻分类器重新进行分类。观察不同的 k 值在不同水平的类别标号噪声条件下的不同表现。
4. 实现托梅克连接方法以移除有害的样例。使用 k 近邻分类器对移除后剩下的样例进行重复实验，比较得到的结果。观察对于不同的 k 值和不同的噪声水平，移除操作对于 k 近邻分类器的分类行为有怎样的影响。

第 4 章　类间边界：线性和多项式分类器

当将训练样例表示为 n 维样本空间的点时，我们可能会意识到正类样例和负类样例都倾向于聚集在一起，且二者所占据的区域是不同的。这个观察促使我们考虑分类的另一个方法。不同于之前所用到的概率和相似性，我们可以试着确定将两类分开的决策面。这样的话，一个非常简单的可行方法就是使用一个线性函数。更加灵活的是高阶的多项式，它们可以定义非常复杂的类间边界。不过，这些方法都必须谨慎地操作。

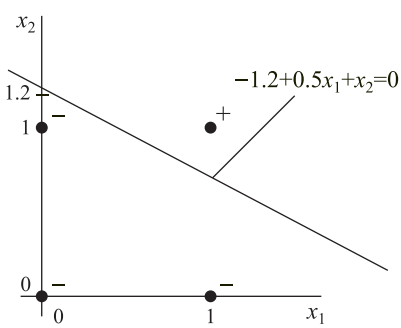
本章介绍两种简单的途径来从由布尔属性描述的样例中引出线性分类器，然后讨论如何在更一般的域中使用它们，如数值属性的域和多于两个类的域。然后，将所有的思想推广到多项式分类器上。

4.1 本质

从现在开始，我们仅考虑布尔域，这里每个属性或是 true 或是 false，为了使得这些属性能够融入代数函数，我们用整数来表示它们：用 1 表示 true，0 表示 false。

线性分类器。在图 4.1 中，一个样例被标记为正类，剩余的 3 个被标记为负类。在这个特殊的例子中，两类可以通过式（4.1）所定义的线性函数分割开：

$$-1.2+0.5x_1+x_2=0 \tag{4.1}$$



x_1	x_2	$-1.2+0.5x_1+x_2$	类别
1	1	0.3	pos
1	0	−0.7	neg
0	1	−0.2	neg
0	0	−1.2	neg

图 4.1 线性分类器在有两个类别和两个属性的域中的应用（用 1 代替 true，0 代替 false）

在这个表达式的左侧，x_1 和 x_2 代表了属性。如果我们用一个给定样例的具体值来替换 x_1 和 x_2，则表达式 $-1.2+0.5x_1+x_2$ 将变成一个正数或一个负数。然

后，这个数的符号决定了样例的类别。图 4.1 右侧的表格展示了左侧的 4 个样例是如何被分类的。

式（4.1）并不是唯一的可以解决这个问题的方法。其他的表达式，如 $-1.5+x_1+x_2$，也可以按照完全相同的方式来标记这 4 个样例。实际上，无限多的有如下一般形式的不同的分类器都可以取得一样的结果：

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

这很容易推广到有 n 个属性的域：

$$w_0 + w_1 x_1 + \cdots + w_n x_n = 0 \tag{4.2}$$

如果 $n=2$，则式（4.2）定义了一条直线；如果 $n=3$，则是一个平面；如果 $n>3$，则是一个超平面。顺便说一下，如果我们人为地引入一个第 0 个属性 x_0，它的值总是固定为 $x_0=1$，则式（4.2）可以重写为更加简洁的形式：

$$\sum_{i=0}^{n} w_i x_i = 0 \tag{4.3}$$

一些实际的问题。当编写计算机程序来运行线性分类器的时候，千万不能忘记决定怎样标记那些恰好位于超平面上的、使得表达式等于 0 的、很少的样例。普遍做法是随机地选择一个类别或选择那个在训练集中拥有更多代表的类别。

同样，我们也不能忘记如果两类不是线性可分的，那就没有一个线性分类器可以将正类和负类样本分开。因此，在图 4.1 中，如果我们将 $x=(x_1, x_2)=(0, 0)$ 的类别标号由负改为正，就没有一条直线可以成功分类了。我们将这个问题的进一步讨论推迟到 4.5 节。就目前来说，我们仅考虑类别是线性可分的域。

参数。分类器的表现决定于其中的系数——w_i。它们通常也被称为权重。机器学习的任务就是要找到它们合适的取值。

并不是所有的权重都扮演着相同的角色。从几何上来说，系数 w_1，…，w_n 定义了超平面关于坐标系的角度；w_0 称为偏差，决定了偏移量，即超平面到坐标系原点的距离。

偏差和阈值。在图 4.1 描述的例子中，偏差 $w_0=-1.2$。一个更大的值会使分类器从原点（[0, 0]）平移得更远，不过 $w_0=0$ 会使分类器与原点相交。如果我

们按照如下方式重写式（4.2），则我们对偏差项在分类器中的作用的直观理解会有所改进：

$$w_1 x_1 + \cdots + w_n x_n = \theta \tag{4.4}$$

式（4.4）右边的项：$\theta = -w_0$，是一个阈值，如果一个样例要被划分到正类，那么加权和一定要超过这个阈值。注意，这个阈值等于负的偏差项。

简单的逻辑函数。让我们通过（有点儿极端地）要求所有的属性都有相同的权重（即 $w_i = 1$）来简化我们考虑的问题。即使是在这样的约束之下，仔细地选择阈值也能获得一些有用的函数。例如，读者可以很容易地验证，以下的分类器返回正类当且仅当每一个属性都有 $x_i = 1$，这种情况被称为逻辑与：

$$x_1 + \cdots + x_n = n - 0.5。 \tag{4.5}$$

不同地，如果至少有一个属性是 $x_i = 1$，则下面的分类器就返回正类，这种情况被称为逻辑或：

$$x_1 + \cdots + x_n = 0.5 \tag{4.6}$$

最后，如果至少 k 个属性（在总共的 n 个属性中）是 $x_i = 1$，则下面的分类器就返回正类。这表示的是所谓的 n 中有 k（*k-of-n*）函数，“与”和“或”都是它的特殊形式：“与”是 n 中有 n（*n-of-n*），“或”是 n 中有 1（1-*of*-n）。

$$x_1 + \cdots + x_n = k - 0.5 \tag{4.7}$$

权重。现在我们已经理解了偏差的作用，那我们放弃所有权重都相等的约束，来看一下它们取具体值的结果。考虑由如下函数定义的线性分类器：

$$2 + 3x_1 - 2x_2 + 0.1x_4 - 0.5x_6 = 0 \tag{4.8}$$

在表达式中，第一个要注意的是在等号左侧，属性 x_3 和 x_5 的缺失。因为它们的权重为 0，即 $w_3 = w_5 = 0$，它们更倾向于是不相关的。对于其他属性，它们的影响决定于它们权值的绝对值和符号：如果 $w_i > 0$，那么 $x_i = 1$ 增加了表达式为正的机会；如果 $w_i < 0$，那么 $x_i = 1$ 增加了表达式为负的机会。注意，在这个由式（4.8）所定义的分类器的例子中，因为 $w_1 > w_4$，所以 x_1 比 x_4 更加支持正类。同样地，x_2 的影响要比 x_6 的影响大很多，仅仅是在相反的方向：减少总和的值，这个权重使得一个 $x_2 = 1$ 的样例更容易被认为是属于负类的。最后，非常小的 w_4 的

值使得属性 x_4 几乎是不相关的。

作为另外的一个例子，考虑由以下函数定义的分类器：

$$2x_1 + x_2 + x_3 = 1.5 \tag{4.9}$$

这里，通过 $x_1 = 1$ 的单独地出现（因为 $2x_1 = 2 \cdot 1 > 1.5$）或通过 $x_2 = 1$ 和 $x_3 = 1$ 的组合起来的贡献，都会使得阈值超过 1.5。这意味着，x_1 会在投票中胜过另外两个属性，即使是在 x_2 和 x_3 联合起来支持负类的情况下。

低的计算代价。注意这个方法相对比较低的计算代价。最近邻分类器需要评估许多的几何距离，然后在其中找到最小的一个，而线性分类器只需要决定一个相对简单的表达式的符号。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 写出在有 4 个布尔属性的域中定义线性分类器的一般表达式。为什么我们倾向于将 true 和 false 分别用 1 和 0 来表示？线性分类器如何决定一个样例的类别？
- 用一个线性分类器如何实现“与”函数、“或”函数和 n 中有 k 函数？
- 解释和讨论由表达式 $-2.5 + x_2 + 2x_3 = 0$ 所定义的线性分类器的行为。权重可以告诉我们关于各个属性的哪些信息？
- 比较线性分类器和最近邻分类器的计算代价。

4.2 加法规则：感知机学习

我们已经对线性分类器是如何工作的有了一定的了解，下面我们更进一步地观察如何从训练数据归纳学习出这个工具。

学习任务。我们假设每个训练样例 $\boldsymbol{x}$ 是由 n 个二值属性来描述的，属性值 $x_i = 1$ 或 $x_i = 0$。一个正类样例用 $c(\boldsymbol{x}) = 1$ 来标记，一个负类样例用 $c(\boldsymbol{x}) = 0$ 来标

记。为了确保我们不会将一个样例的真实类别与分类器返回的类别相混淆，我们用 $h(\boldsymbol{x})$ 来标记后者，这里 h 表明这是分类器的假设。如果 $\sum_{i=0}^{n} w_i x_i > 0$，则分类器假设样例属于正类，因此返回 $h(\boldsymbol{x}) = 1$。相反地，如果 $\sum_{i=0}^{n} w_i x_i \leqslant 0$，则分类器返回 $h(\boldsymbol{x}) = 0$。图 4.2 说明分类器将 x 标记为正类仅当倾向于这一类的累积的证据超过 0 时。

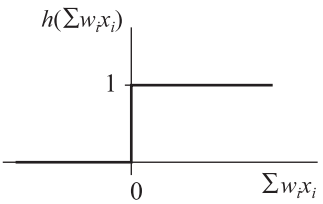


图 4.2　当 $\sum_{i=0}^{n} w_i x_i > 0$ 时，线性分类器输出 $h(\boldsymbol{x}) = 1$；当 $\sum_{i=0}^{n} w_i x_i \leqslant 0$ 时，输出 $h(\boldsymbol{x}) = 0$，分别表示样例属于正类或负类

最后，我们假设 $c(\boldsymbol{x}) = 1$ 和 $c(\boldsymbol{x}) = 0$ 的样例是线性可分的。这意味着，存在一个线性分类器，它可以正确标记所有训练样例，即 $h(\boldsymbol{x}) = c(\boldsymbol{x})$。机器学习的任务就是要找到权重 w_i，以达到这个目的。

从错误中学习。这里是归纳学习线性分类器的最普遍方法的本质。假设我们已经有了一个分类器的中间版本（甚至是不完美的）。当给定一个训练样例 $\boldsymbol{x}$ 时，分类器返回它的类标 $h(\boldsymbol{x})$。如果它不同于真实的类标，即 $h(\boldsymbol{x}) \neq c(\boldsymbol{x})$，这是因为权重还不够完美。因此，必须以一种可能会纠正错误的方式来修改它们。

这里是如何完成权重的修改呢？令真实的类别是 $c(\boldsymbol{x}) = 1$，这时，$h(\boldsymbol{x}) = 0$ 只能是 $\sum_{i=0}^{n} w_i x_i < 0$，说明权重都太小了。如果我们增大权重，那和式 $\sum_{i=0}^{n} w_i x_i$ 可能会超过 0，使得返回的类标为正，而成为正确的。注意，只要增加 $x_i = 1$ 的属性对应的权重即可。当 $x_i = 0$ 时，w_i 的值并不重要，因为任何数乘以 0 还是 0，即 $0 \cdot w_i = 0$。

相似地，如果 $c(\boldsymbol{x}) = 0$，$h(\boldsymbol{x}) = 1$，$x_i = 1$ 的属性对应的权重都应该减小，以

使和式小于0，即 $\sum_{i=0}^{n} w_i x_i < 0$。

权重调整公式。总结一下，一个训练样例 x 的出现会导致3种不同的情况发生。基于“从错误中学习”的技术按照下表来应对它们：

情　况	操　作
$c(\boldsymbol{x})=1$，$h(\boldsymbol{x})=0$	对 $x_i=1$ 的属性，增加 w_i 的值
$c(\boldsymbol{x})=0$，$h(\boldsymbol{x})=1$	对 $x_i=1$ 的属性，减少 w_i 的值
$c(\boldsymbol{x})=h(\boldsymbol{x})$	什么都不做

有趣的是，所有这些操作都可以根据式（4.10）来执行：

$$w_i = w_i + \eta \cdot [c(\boldsymbol{x}) - h(\boldsymbol{x})] \cdot x_i \tag{4.10}$$

我们来看一下这个权重调整公式的一些基本方面：

1. 纠正措施。如果 $c(\boldsymbol{x})=h(\boldsymbol{x})$，则括号中的项 $[c(\boldsymbol{x})-h(\boldsymbol{x})]=0$，$w_i$ 保持不变。如果 $c(\boldsymbol{x})=1$，$h(\boldsymbol{x})=0$，则括号中的项值为1，权重增加。如果 $c(\boldsymbol{x})=0$，$h(\boldsymbol{x})=1$，则括号中的项为负，权重减小。
2. 仅影响相关权重。如果 $x_i=0$，加到第 i 个权重上的项 $\eta \cdot [c(\boldsymbol{x})-h(\boldsymbol{x})] \cdot 0$ 是0。这意味着，式（4.10）只对 $x_i=1$ 对应的 w_i 起作用。
3. 变化的量。它由学习率 η 来控制，它的用户设定的值是从区间 $\eta \in (0, 1]$ 中选择得到。

注意，权重的修改是加法的，因为是有一项加到了前一个权重上。在4.3节，我们将讨论另一种可能性：一个乘法的公式。

感知机学习算法。式（4.10）构成了感知机学习算法的核心（它的作者是罗森布兰特（M. Rosenblatt），原本是将这个学习技术用在了一个他称之为感知机的设备上）。表4.1中的伪代码总结了它的算法步骤。原理很简单。一开始权重被初始化为小的随机值，然后以每次一个的方式加入训练样例。每个样例出现之后，分类器的权重就被提交给式（4.10）。最后的训练样例标志着一轮训练阶段的完成。除非分类器现在可以正确标注整个训练集，学习器将返回第一个样例，这样开始第二个轮次的学习，然后第三个……。基本上，为了达到目标，是需要若干轮的学习的。

表 4.1　感知机学习算法

假设：两类，即 $c(\boldsymbol{x})=1$ 和 $c(\boldsymbol{x})=0$，是线性可分的。

1. 初始化所有的权重 w_i 为小的随机数。选择一个合适的学习率：$\eta \in (0, 1]$。
2. 对于每个训练样例，$\boldsymbol{x}=(x_1, \cdots, x_n)$，它的类标 $c(\boldsymbol{x})$：
   1）如果 $\sum_{i=0}^{n} w_i x_i > 0$，则令 $h(\boldsymbol{x})=1$，否则 $h(\boldsymbol{x})=0$。
   2）使用公式更新权重，即 $w_i = w_i + \eta \cdot [c(\boldsymbol{x}) - h(\boldsymbol{x})] \cdot x_i$。
3. 如果对于所有训练样例 $c(\boldsymbol{x})=h(\boldsymbol{x})$，则算法停止；否则，返回第 2 步。

一个数值例子。表 4.2 展示了在一个玩具域上的程序执行过程，这个域有 3 个训练样例：$\mathbf{e}_1$，$\mathbf{e}_2$ 和$\mathbf{e}_3$，它们都是由两个二值的属性来描述的。对$\mathbf{e}_1$ 进行训练之后，因为 $h(x)=1$ 而 $c(x)=0$，所以权重应该减小（初始是随机的）；然而，它只发生在 w_0 和 w_1 上，因为 $x_2=0$。当训练$\mathbf{e}_2$ 之后，新版本的分类器的权重都增加，因为 $h(x)=0$ 而 $c(x)=1$，而且所有属性都有 $x_i=1$。训练$\mathbf{e}_3$ 之后，$h(x)=1$ 而 $c(x)=0$ 的结果导致了 w_0 的减小，但是因为 $x_1=x_2=0$，所以其他两个权重不变。现在，分类器正确地标记了所有的训练样例，因此这个过程可以终止了。

初始的权重与属性的个数。训练在一个阶段之后就产生了正确的分类器，但这仅仅是因为一些幸运的选择，它们扮演了关键的角色。我们简单对它们做下讨论。

首先是（随机的）初始权重的集合。不同的初始化可能会导致不同的学习阶段个数。多数情况下，分类器的初始版本是几乎没用的，在训练过程收敛到有效值之前需要大量的训练。不过，有时候第一个版本是相当好的，仅一个阶段的学习即可达到目标。在极端的情况下，存在极其微小的可能性，随机数生成器可以产生一个能完全正确地标记所有训练样本的分类器，而不需要任何训练。

表 4.2　图示感知机学习

令学习率 $\eta=0.5$，（随机产生的）初始权重为 $w_0=0.1$、$w_1=0.3$ 和 $w_2=0.4$，$x_0=1$。

任务：使用如下的训练集，感知机学习算法是要学习如何将负类样例 e_1 和 e_3 与正类样例 e_2 分割开。

样例	x_1	x_2	$c(\boldsymbol{x})$
$\mathbf{e}_1$	1	0	0
$\mathbf{e}_2$	1	1	1
$\mathbf{e}_3$	0	1	0

（续）

线性分类器关于 $\boldsymbol{x}$ 的类别的假设是：如果 $\sum_{i=0}^{n} w_i x_i > 0$，$h(\boldsymbol{x}) = 1$，反之，$h(\boldsymbol{x}) = 0$。每一个样例出现之后，所有权重按照同一个公式来调整，即 $w_i = w_i + 0.5 \cdot [c(\boldsymbol{x}) - h(\boldsymbol{x})] \cdot x_i$。

下表逐步地显示了在学习的过程中权重的变化情况。

	x_1	x_2	w_0	w_1	w_2	$h(\boldsymbol{x})$	$c(\boldsymbol{x})$	$c(\boldsymbol{x}) - h(\boldsymbol{x})$
随机分类器			0.1	0.3	0.4			
样例 $\mathbf{e}_1$	1	0				1	0	−1
新分类器			−0.4	−0.2	0.4			
样例 $\mathbf{e}_2$	1	1				0	1	1
新分类器		0.1	0.3	0.9				
样例 $\mathbf{e}_3$	0	0				1	0	−1
最后版本的分类器			−0.4	0.3	0.9			

最后版本的分类器：$-0.4 + 0.3x_1 + 0.9x_2 = 0$，不再错分任何一个训练样例了。因此，训练经过一轮就完成了。

另一个因素是属性向量的长度。一般来说，需要的训练阶段数趋向于关于属性个数线性增长（假如使用相同的学习率 η）。例如，在一个有 $3 \times n$ 个属性的域中需要的训练阶段数可能是一个有 n 个属性的域所需的训练阶段数的 3 倍。

学习率。学习率 η 扮演了一个很关键的角色。回到表 4.2 中的例子，读者会注意到权重的快速变化。e_1 出现之后，w_0 从 0.1 跳到了 −0.4，e_2 出现之后，又回到了 0.1，e_3 出现之后，又回到 −0.4。w_1 和 w_2 也经历了类似的变化。图 4.3 描述了这个现象。读者很容易可以验证 4 条线代表了分类器连续的 4 个版本。例如，请注意，从分类器 1 到分类器 2 的变化和从分类器 2 到分类器 3 的变化是多么富有戏剧性。

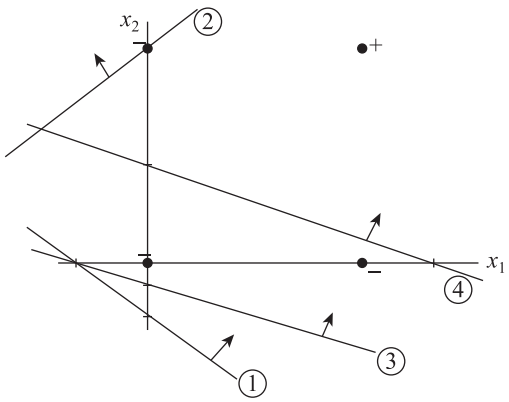


图 4.3　表 4.2 中的 4 个分类器。由初始权重定义的分类器用 1 来标记，数字 2 和 3 代表了两个中间步骤，4 是最后的解，箭头指示了正类样例所在的半平面

这种异常的敏感性来自于高的学习率 $\eta = 0.5$。一个小的值，如 $\eta = 0.1$，会减缓这种变化，因此使得学习更加平滑。但是如果我们做得太过了，如选择 $\eta = 0.001$，则训练过程就会变得非常慢，而且在所有训练样例都被正确分类之前会经过非常多轮的训练阶段。

如果解存在，那它一定会被找到。不要忘记，是在类分割超平面存在的前提下，不论初始权重、属性个数和学习率是什么，感知机学习算法保证可以在有限的步骤内找到类分割超平面。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在什么样的条件下，感知机学习保证可以找到一个将所有训练样例都正确标记的分类器?
- 算法什么时候减小分类器的权重，什么时候增加它们，什么时候不改变它们? 为什么只有对应的属性值 $x_i = 1$ 时才改进 w_i?
- 什么会影响算法收敛到一个解所需的学习轮数?

4.3 乘法规则：WINNOW

感知机学习通过应用加法规则来响应分类器的错误，也就是在权重上增加一个正的或负的项。一个明显的代替方式就是用乘法规则，在权重上乘以而不是加上某一项。WINNOW 就采取了这样的方法，表 4.3 中的伪代码总结了这个算法。

表 4.3 WINNOW 算法

假设：两类，即 $c(\boldsymbol{x})=1$ 和 $c(\boldsymbol{x})=0$，是线性可分的。

1. 初始化分类器的权重为 $w_i=1$。
2. 设置阈值 $\theta=n-0.1$（n 是属性的个数），并给参数 $\alpha>1$ 选择一个合适的值（通常 $\alpha=2$）。
3. 给出一个训练样例 $\boldsymbol{x}$，它的类别是 $c(\boldsymbol{x})$，分类器输出是 $h(x)$。
4. 如果 $c(\boldsymbol{x}) \neq h(\boldsymbol{x})$，对值为 $x_i=1$ 的属性对应的权重进行更新：
   1）如果 $c(\boldsymbol{x})=1$，$h(\boldsymbol{x})=0$，则 $w_i=\alpha w_i$。
   2）如果 $c(\boldsymbol{x})=0$，$h(\boldsymbol{x})=1$，则 $w_i=w_i/\alpha$。
5. 如果对于所有训练样例，$c(\boldsymbol{x})=h(\boldsymbol{x})$，则算法停止；否则，返回第 3 步。

原理和公式。基本方案与 4.2 节中的一样。一个训练样例 $\boldsymbol{x}$ 参与训练后，分类器返回对这个样例的类标的假设 $h(\boldsymbol{x})$。学习器比较这个假设与已知的真实类标 $c(\boldsymbol{x})$，如果两个类标不同，即 $c(\boldsymbol{x}) \neq h(\boldsymbol{x})$，则对值为 $x_i=1$ 的属性对应的权重按照如下方式进行修改（这里 $\alpha>1$ 是用户设定的一个参数）：

情　况	操　作
$c(\boldsymbol{x})=1$，$h(\boldsymbol{x})=0$	$w_i=\alpha w_i$
$c(\boldsymbol{x})=0$，$h(\boldsymbol{x})=1$	$w_i=w_i/\alpha$
$c(\boldsymbol{x})=h(\boldsymbol{x})$	什么都不做

我们鼓励读者来验证所有的这 3 个操作可以通过式（4.11）来执行：

$$w_i=w_i \cdot \alpha^{c(\boldsymbol{x})-h(\boldsymbol{x})} \tag{4.11}$$

一个数值例子。表 4.4 在一个简单的玩具域上展示了 WINNOW 的原理。训练集包含了所有可以用 3 个二值属性描述的样例。那些 $x_2 = x_3 = 1$ 的样例被标记为正类，其余的被标记为负类，不管（不相关的）属性的 x_1 的值是多少。

表 4.4　图示 WINNOW 的表现

任务：使用（下边）左边表格中的训练样例，归纳学习线性分类器。令 $\alpha = 2$，$\theta = 2.9$。注意，这里的训练通过两步来实现：e_5（假的负类）和 e_3（假的正类）的训练。这两步权重修改之后，所得到的分类器就可以对所有训练样例进行正确分类了。

	x_1	x_2	x_3	$c(\boldsymbol{x})$
e_1	1	1	1	1
e_2	1	1	0	0
e_3	1	0	1	0
e_4	1	0	0	0
e_5	0	1	1	1
e_6	0	1	0	0
e_7	0	0	1	0
e_8	0	0	0	0

	x_1	x_2	x_3	w_1	w_2	w_3	$h(\boldsymbol{x})$	$c(\boldsymbol{x})$
初始分类器				1	1	1		
样例 $\mathbf{e}_5$	0	1	1			0	1	
新的权重			1	2	2			
样例 $\mathbf{e}_3$	1	0	1			1	0	
新的权重			0.5	2	1			

在感知机学习中，权重是被初始化为小的随机值。然而，对于 WINNOW，它们都被初始化为 1。对于阈值，使用了稍微小于属性的个数的 $\theta = n - 0.1$。在表 4.4的玩具域中，这意味着 $\theta = 3 - 0.1 = 2.9$，因为 WINNOW 对一个属性是否无关没有先验认识。

当前 4 个样例参与训练之后，分类器的初始版本都可以正确地标记它们。第一个错误出现在 e_5出现时：对于这个正类样例，分类器错误地返回了负类的类标。因此，学习器增加了 $x_i = 1$ 的属性对应的权重（也就是 w_2 和 w_3）。这个新的分类器然后正确分类了所有剩余的样例，$e_6 \sim e_8$。在第二个学习阶段，分类器在 e_3 上犯了错误，造成了一个假的正类预测。相应于这个错误，算法减少了 w_1 和 w_3 的权重（但没有 w_2，因为 $x_2 = 0$）。这次最后的权重修改之后，分类器就可以正确标记整个训练集了（类似于感知机学习的情况，我们可以考虑使用第 0 个属性，

$x_0 = 1$，它的初始权重是 $w_0 = 1$）。

注意，这时不相关属性 x_1 的权重是要小于相关属性的权重的。实际上，通过显著降低不相关属性的权重来削弱它们的影响，即“摈除它们”，是这个技术的主要优势之一。

参数 α。参数 α 以类似于感知机学习中的学习率的方式来控制学习器对错误的敏感性。主要的不同就是 $\alpha > 1$ 的要求，这保证了在假的负类的情况下权重 w_i 的增加和假的正类的情况下权重 w_i 的减少。这个参数的具体取值是完全任意的。如果它只是略微超过 1（如 $\alpha = 1.1$），那么权重的更新会非常慢，从而导致慢的收敛速度。增加 α 的值会加快收敛，但是会有得不到解的风险。理想的值最好通过实验获得；好的结果经常是在 $\alpha = 2$ 时取得的。

没有负的权重？让我们指出一个 WINNOW 和感知机学习的根本不同。因为初始为正的权重总是乘以 α 或 $1/\alpha$，所以它们永远不会降到 0，更别说负数了。这就是说，除非采取合适的措施，否则一大类的线性分类器会因此消失：那些有负值或以 0 作为系数的。

如果我们将每一个原始属性用一对新的属性来替换，则可以弥补这个缺点：一个取原来的属性值，另一个取相反的值。在一个原来有 n 个属性的域，总的属性数将是 $2n$，第 $n+i$ 个属性的值 x_{n+i}，就是 x_i 取反。例如，假设一个样例由下面的 3 个属性值来描述：$x_1 = 1$，$x_2 = 0$，$x_3 = 1$。在新的表示中，同一个样例将由 6 个属性来描述：$x_1 = 1$，$x_2 = 0$，$x_3 = 1$，$x_4 = 0$，$x_5 = 1$，$x_6 = 0$。

对于这些值，WINNOW 会找到 6 个权重，即 w_1，…，w_6，或者可能是 7 个，如果包括 w_0 的话。

与感知机比较。对比感知机，WINNOW 看起来在有不相关属性的域会收敛得更快一些，因为不相关属性的权重会很快下降到很小的值。但是，WINNOW 和感知机都不能识别（和去除）冗余属性。在两个属性总是有相同值的情况下，$x_i = x_j$，对于它们学习过程将会收敛到同样的权重，使得它们看起来一样重要，即使很显然它们中只有一个是有用的。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- WINNOW 的权重更新机制采用了什么样的公式？为什么这个公式被称为是乘法的？
- 用 $\alpha > 1$ 来乘以或除以权重有什么不足之处？这种不足该如何补救？
- 总结 WINNOW 和感知机学习的不同点。

4.4 多于两个类的域

超平面有两个面，它可以将正类样例和负类样例分割开，对于两类问题，这样就可以了；当面临多类问题时，超平面这样的工具看起来就不能再发挥作用了。或者，它也能？

二值分类器组。超过单个人能力的问题可以由一个团队来解决。图 4.4 展示了在这种情境下一种常用的方法。这个团队包含 4 个二值分类器，每个针对于 4 类中的一类，即 $C_1 \sim C_4$。理想地，一个 C_i 类的样例经过训练后，第 i 个分类器返回 $h_i(\boldsymbol{x}) = 1$，而其他分类器返回 $h_i(\boldsymbol{x}) = 0$，再一次强调，前提是每个类别可以从其他类别线性地分割开。

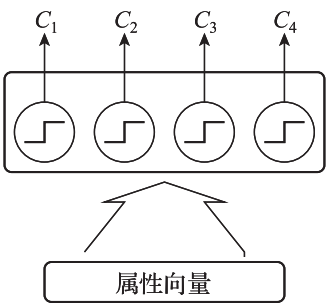


图 4.4 将一个四类的问题转化为四个 2 类的问题。

修改训练数据。然而，要获得这种值得称赞的效果，每一个分类器都需要被适当地训练。这里，前几节介绍的算法在这里可以用上。仅需要增加的“技巧”是需要相应地修改训练数据。

表 4.5 描述了这个原理。左侧是原始的训练集 T，这里每一个样例被标注为 4 类中的一类。右侧是 4 个导出的集合，即 $T_1 \sim T_4$，每一个包含了同样的 6 个样例，但是它们被重新进行了标记，原始训练集 T 中代表 C_i 类的一个样例在 T_i 中被标记为 $c(\boldsymbol{x}) = 1$，在其他集合中被标记为 $c(\boldsymbol{x}) = 0$。

表 4.5　一个 4 类的训练集 T 被转化为 4 个二值的训练集（$T_1 \sim T_4$）

T		T_1		T_2		T_3		T_4	
e_1	C_2	e_1	0	e_1	1	e_1	0	e_1	0
e_2	C_1	e_2	1	e_2	0	e_2	0	e_2	0
e_3	C_3	e_3	0	e_3	0	e_3	1	e_3	0
e_4	C_4	e_4	0	e_4	0	e_4	0	e_4	1
e_5	C_2	e_5	0	e_5	1	e_5	0	e_5	0
e_6	C_4	e_6	0	e_6	0	e_6	0	e_6	1

需要一个主分类器。训练集T_i 会被呈现给一个学习器，这个学习器会从中归纳学习出一个专门针对相应类别的线性分类器。但是这并不是故事的结尾。训练集可能会很差地代表相应的类别，它们可能被噪声污染，甚至可能不能满足线性可分性的要求。作为这些复杂性的结果，归纳学习到的分类器可能是相互“重叠的”，因为它们中的两个或多个对同一个样例 $\boldsymbol{x}$ 都会返回$h_i(\boldsymbol{x})=1$，从而让人误认为 x 属于多于一个类。这就是为什么一个“主分类器”是必需的，它的任务就是从返回的类别中选择出最可能正确的那一个。

这并不困难。读者可能会记起，一个线性分类器会标记 $\boldsymbol{x}$ 为正类，如果 x 的属性值的加权和超过0，即 $\sum_{i=0}^{n} w_i x_i > 0$，则这个和（对于返回$h_i(\boldsymbol{x})=1$ 的每一个分类器经常是不同的）可以理解为支持相应类别的证据的量。那么主分类器可以简单地倾向于给出的最高的 $\sum_{i=0}^{n} w_i x_i$ 值的二值分类器所返回的类别。表 4.6 图示了这个原理，这里两个分类器 C_2 和 C_3，都返回了 $h(\boldsymbol{x})=1$。从这两个当中，主分类器选择了 C_2，因为它的加权和大于 C_3 的加权和。

表 4.6　图示主分类器的应用：从多个候选中选取样例的类别

假设有 4 个如下表所列的权重定义的二值分类器（第 i 个分类器用于第 i 个类别）。主分类器应该怎样标记样例 $x=(x_1, x_2, x_3, x_4)=(1, 1, 1, 0)$？

类别	分类器					$\sum_{i=0}^{n} w_i x_i$	$h(x)$
	w_0	w_1	w_2	w_3	w_4		
C_1	−0.5	2	0.5	−1	−5	1	0
C_2	0.5	1.5	−1	3	1	4	1
C_3	1	−2	4	−1	0.5	2	1
C_4	−2	1	1	−3	−1	−3	0

最右一列说明两个分类器，C_2 和 C_3，都返回了 $h(x)=1$。在它们两个当中，C_2 的 $\sum_{i=0}^{n} w_i x_i$ 值更大。因此，主分类器标注 x 为 C_2。

一个现实的约束。这里要写一个小小的免责声明。这个在多类域使用线性分类器的方法仅在类别数是适当的情况下是可靠的，如 3 ~ 5 类。在有许多类的域，导出的训练集T_i是不平衡的，因为多数的样例都是$c(x)=0$的，只有少数是$c(x)=1$的。正如 10.2 节我们将会学到的，除非采取恰当的措施，否则不平衡的训练集在有噪声的域很可能会造成学习的困难。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在一个N类的域中使用N个线性分类器的时候，为了归纳学习每个二值分类器，应该如何构建训练集T_i?
- 在两个或多个二值分类器都返回$h(\boldsymbol{x})=1$的情况下，一个样例的类别该如何确定?

4.5 多项式分类器

现在是时候放弃正类样例和负类样例线性可分的要求了，因为在大多数情况下它们并不是线性可分的。不仅仅是因为线性可分性会被噪声所破坏。一个类别所占据的区域的形状就会使得用线性的决策平面将它分割开是不可能的。因此，在图 4.5 所示的训练集中，没有线性分类器可以成功地将两类分割开。这是一个只有非线性曲线才能达到的结果，如图中所示的抛物线。

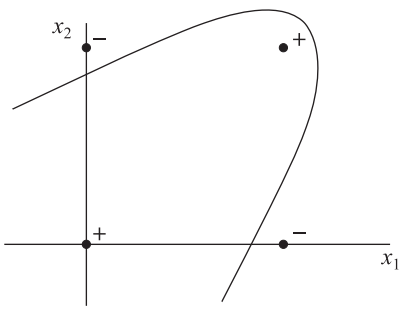


图 4.5 在一些域中，没有线性分类器可以将正类和负类样例分割开，只有非线性分类器可以做到

现在，我们要问怎样归纳学习出更加复杂的非线性分类器？毫无疑问，它们是存在的。例如，数学家告诉我们，任何n维函数都可以

被足够高阶的多项式以任意精度来逼近。因此，让我们看一下为了达到分类的目的应该如何使用以及推导这些多项式。

二阶多项式。好的消息是，多项式的系数可以使用线性分类器中所用过的同样的技术进行推导。现在，我们解释一下。

为了清楚起见，我们从只包含两个布尔属性（x_1 和 x_2）的一个简单域开始。那么，二阶多项式函数可以定义如下：

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2 = 0 \tag{4.12}$$

表达式的左边是几项的和，它们有一个共同点：一个权重 w_i 乘以一个乘积 $x_1^k x_2^l$。在第一项中，我们有 $k + l = 0$，因为 $w_0 x_1^0 x_2^0 = w_0$；下面的是 $k + l = 1$ 的项，具体来说是 $w_1 x_1^1 x_2^0 = w_1 x_1$ 和 $w_2 x_1^0 x_2^1 = w_2 x_2$；然后以 3 个 $k + l = 2$ 的项结束，分别为 $w_3 x_1^2$、$w_4 x_1 x_2$ 和 $w_5 x_2^2$。需要记住的是，二阶多项式展开当指数的和达到 2 时结束。

当然，一些系数可以是 $w_i = 0$，使得一些项“不可见”，如 $7 + 2x_1 x_2 + 3x_2^2$ 中，x_1、x_2 和 x_1^2 的系数是 0。

***r* 阶多项式**。更加常见的是 *r* 阶多项式（还是在二维的域中），由多个加权项 $w_i x_1^k x_2^l$ 的和构成，而且 $k + l = j$，$j = 1$，…，r。

读者很容易再往下走一步，并写出在有多于两个属性的域中定义 *r* 阶多项式的一般公式。提示，在任何单项式中指数之和不能超过 r。

转化为线性分类器。无论多项式是多少阶的，无论属性的个数是多少，机器学习的任务就是找到权重，能够使得正类和负类样例分开。各项都是非线性的（指数的和有时会超过 1）这个看起来不利的条件，可以通过乘子很容易地消除，它是一种返回输入的逻辑连接的策略：如果所有输入都是 1，则值为 1；如果至少一个输入为 0，则值为 0。借助于它的帮助，我们可以将每一个属性的乘积替换为一个新的属性z_i，因此，式（4.12）可以重写为如下形式：

$$w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5 = 0 \tag{4.13}$$

例如，这意味着，$z_3 = x_1^2$ 和$z_4 = x_1 x_2$。注意，这个技巧已经将原来的有两个属性（x_1 和 x_2）的非线性问题转化为一个关于 5 个新创建属性——$z_1 \sim z_5$的线性问题了。

图 4.6 展示了一个二阶多项式在一个有 3 个属性的域中的使用情况。

因为每个样例的z_i值都是已知的，所有权重可以通过感知机或 WINNOW 没有任何困难地获得。当然，我们一定不能忘记，只有所选阶数的多项式是能够将两类分割开的，我们才能用这些技术找到解。

关于 3 个属性的二阶多项式函数由如下函数定义：

$$0 = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_1^2 + w_5 x_1 x_2 + w_6 x_1 x_3 + w_7 x_2^2 + w_8 x_2 x_3 + w_9 x_3^2$$

使用乘子，我们可得：

$$0 = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5 + w_6 z_6 + w_7 z_7 + w_8 z_8 + w_9 z_9$$

下图是带有乘子的整个“设备”的图示。在得到线性分类器之前，每一个信号z_i要乘以相应的权重w_i。

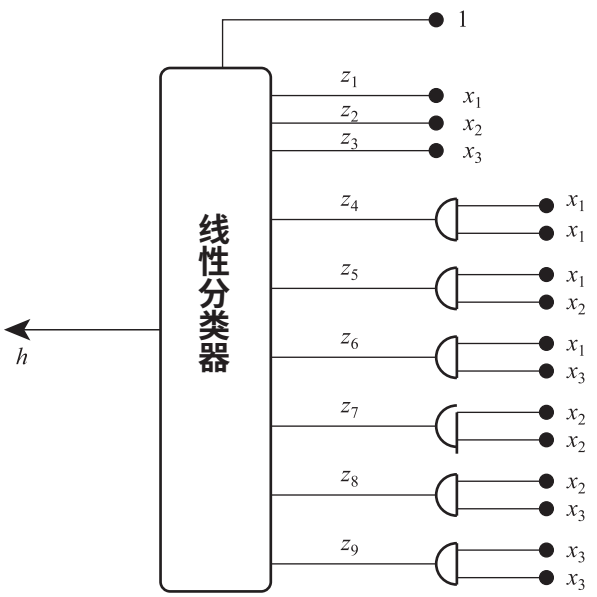


图 4.6　借助于预处理数据的乘子，一个多项式分类器可以转化为一个线性分类器

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 我们什么时候需要非线性分类器？特别地，在什么情况下会用多项式分类器？

- 写出定义一个多项式分类器的数学表达式。这里，什么“技巧”使得我们可以使用线性分类器用到的相同的学习技术？

4.6 多项式分类器的特殊方面

为了能够成功地应用一个机器学习技术，工程师不仅要理解它的长处，还要知道它的局限、不足和缺陷。对于多项式，这里有几点值得注意。让我们简单地讨论一下。

过拟合。多项式分类器倾向于过拟合带有噪声的训练数据。因为过拟合的问题在其他机器学习范式里也会遇到，所以我们需要详细地讨论一下它的本质。为了清晰起见，我们放弃所有属性都是布尔型的要求；取而代之，我们使用二维连续域，它很容易可视化。

图 4.7 中的 8 个训练样例分属于两类。在其中的一类，所有样例都是正例；在另一类中，所有样例都是负例。为了将两类分开，进行了两种尝试。左侧的是一个线性分类器，忽略了有一个训练样例被错分的小瑕疵；右侧的采用了多项式分类器，试图避免在训练集上犯任何错误。

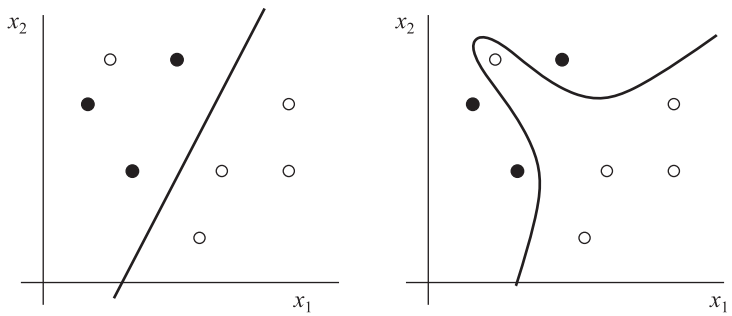


图 4.7　两类是线性可分的，但是噪声导致了一个负类样例被错分为正类；右侧的高阶多项式分类器过拟合了数据，忽视了噪声存在的可能性

一个不可避免的权衡。两者哪个更好呢？这不是一个简单的问题。一方面，两类可能是线性可分的，一个正类样例存在于负类的区域，仅可能是由类标的噪声所引起的。如果这是事实，则线性分类器在训练集上所犯的一个错误是不重要

的，然而多项式分类器很深地切入了负类的区域，会导致有些样本被错分。另一方面，也存在可能，那个异常值确实是正类的合法值，虽然这种情况很少。这种情况下，使用多项式就是合理的。不过，整体来说，单个的异常值是噪声的假设比是特殊的正确值的想法更有可能是正确的。

一个实际的训练集会包括不止一个，可能是几个甚至许多样例看上去都位于样本空间中错误的地方。分类器试图去建模的类间边界可能实际上是扭曲的，尽管有多扭曲依赖于每个人的猜想。工程师可能因为线性分类器分类太粗糙而放弃它，而选择灵活性更高的多项式。这就是说，一个非常高阶的多项式即使在一个有非常大噪声的训练集上都能避免任何错误发生，但它也会在将来的数据上表现得非常糟糕。理想的解决方法通常是在线性分类器和高阶多项式之间找到合适的位置。最好的选择可以通过实验来决定。

权重的数量。要训练的权重的总的数量决定于属性向量的长度和多项式的阶数。一个简单的分析可以得到，对于 n 个属性和 r 阶多项式的情况，这个数量由下面的组合表达式决定：

$$N_w = \left(\frac{n+r}{r} \right) \tag{4.14}$$

当然，对于数值很大的 n，N_w 会不切实际地高。例如，即使对于有 $n = 100$ 个属性和多项式阶数为 $r = 3$ 的相对适度的情况，要训练的权重的个数就是 $N_w = 161,700$。对于现在的计算机，所导致的计算代价是难以应付的。更糟的是，对带噪声的训练数据的过拟合的危险；多项式简直太富有灵活性了。下一段将告诉我们到底有多灵活。

容量。图 4.1 中的平凡域包含 4 个样例。假如每个样例都可以标记为正类或负类，那么对这个训练集我们有 $2^4 = 16$ 种不同的设定类标的方法。在这 16 种方法中，只有两种代表了两类不能线性地分开的情况——在这个域中，线性不可分性是稀少的事件。但是在更加一般的包含 m 个由 n 个属性描述的样本中，这种情况还有多典型？一个样例的随机标记导致线性可分类别的概率有多大？

数学家找到了一条用在 n 是合理的高的（如 10 个或更多个属性）域的简单准则：如果样例数 m 少于属性数的两倍（$m < 2n$），两个类标的随机分布导致线性可

分的概率是接近100%的。相反地，当 $m > 2n$ 时，概率几乎为0。从这个意义上说，“线性分类器的容量是属性数的两倍”。

这个结果对于多项式分类器同样适用。这里，属性的角色由基于乘子得到的项 z_i 来扮演。它们的个数 N_w 由式（4.14）得到。我们已经看到 N_w 可以是非常大的，这使得其容量也很大。在 $n = 100$、$r = 3$ 的情况下，权重的个数是161,700。这意味着只要样例个数是少于323,400的，则3阶多项式就可以将两类分割开（不管是否有噪声）。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 详细描述术语“过拟合”，解释为什么这个现象（和它的结果）对于多项式分类器是很难避免的。
- 在 n 个属性的域中，r 阶多项式可以训练的权重个数的上界是多少？
- 线性或多项式分类器的容量是多少？关于线性可分性容量告诉了我们什么？

4.7 数值域和支持向量机

现在我们已经认识到多项式并不需要新的机器学习算法，我们可以回归到线性分类器，一个还没有穷尽的话题。现在我们要放弃自己强加的布尔属性的限制，并开始考虑连续属性的可能性了。我们还可以依赖上边描述的两个训练算法吗？

数值域的感知机。对于感知机的情况，答案很简单：是的，可以使用同样的权重更新公式。然而，实际的经验表明最好还是把所有属性都映射到单位区间，即 $x_i \in [0, 1]$。

为了方便读者，这里我们重复一下权重调整的公式：

$$w_i = w_i + \eta\ [c(\boldsymbol{x}) - h(\boldsymbol{x})]x_i \tag{4.15}$$

学习率 η 和真实与假设的类标的差值 $[c(x) - h(x)]$ 都与之前有相同的意义

和影响，变化的仅仅是 x_i 的角色。在具有布尔属性的情况下，x_i 的值决定了权重是否要变化。但是，在这里更应该说是权重在多大程度上被影响了：属性值越大，影响越大。

乘法规则。对于 WINNOW，基本上可以用与二值属性情况下相同的学习公式：

$$w_i = w_i \, \alpha^{c(\boldsymbol{x}) - h(\boldsymbol{x})} \tag{4.16}$$

这就是说，当使用式（4.16）的时候还需要小心一点。之前，只需要修改属性值 $x_i = 1$ 对应的权重即可。现在既然属性值来自于一个连续的域，就需要有一些改变，一种可能性是如下的规则：仅当第 i 个属性值为 $x_i > 0.5$ 时更新权重 w_i。

我们应该注意的是，即使是在两类是线性不可分的情况下，如存在噪声，这两个算法（感知机学习和 WINNOW）通常也可以找到一个相对低错误率的解。

哪个线性分类器更好？然而，现在需要讨论一个重要的问题。图 4.8 展示了 3 个线性分类器，每一个都完美地将正类和负类样例分割开了。既然知道在训练集上好的表现并不能保证在将来也有高的性能，那么我们不得不问：这 3 个中的哪个在未来的样例上可能会获得最高分？

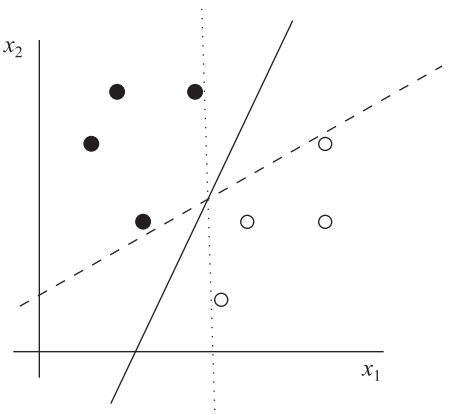


图 4.8　线性可分的类别可以用无限多种不同的方法分割开。问题是，哪一个在训练集上完美的分类器可以在将来的数据上做得最好

支持向量机。研究这个问题的数学家发现了一个答案。看一下图 4.8，我们可以发现虚线的分类器与两侧的最近的样例都是接触的，我们说它的间隔是小的。相反地，实线分类器的间隔是大的：最近的正类样例都在线的一侧，而且在线的另一侧，最近的样例都是比其他分类器的情况要远很多。事实证明，间隔越大，分类器在将来数据上表现好的几率也越大。

图 4.9 描述了支持向量机的技术。实线是最好的分类器，图中也显示出两条与分类器相平行的细线，每一个都与分类器有相同的距离。读者可以看到，它们

经过了离分类器最近的样例，这些样例被称为支持向量（因为毕竟每一个样例都是一个属性向量）。

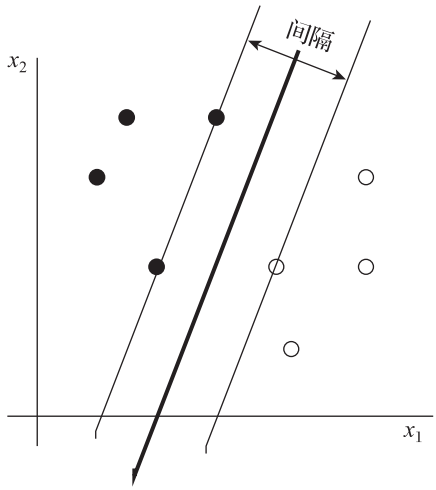


图 4.9　支持向量机技术要寻找最大化间隔的一个分割超平面

机器学习的任务就是要找到最大化间隔的支持向量。找到最优的支持向量的集合的具体机制超出了一个介绍性文档的范围。一个最简单的技术就是直接尝试所有可能的 n 元样例，然后测量每一个这样的选择所给出的间隔。然而，在有许多样例的域中这是不现实的。因此，多数情况下，工程师们依赖于互联网上免费提供的一些软件包。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 感知机学习和 WINNOW 可以用于数值域吗？该怎样用？
- 假如有无穷多个线性分类器都可以将正类样本与负类样本分割开（假设这样的分割存在），它们中的哪一个可以期望在新的数据上能给出最好的结果？
- 什么是支持向量？间隔最大化意味着什么？

4.8 总结和历史简评

- 线性和多项式分类器都定义了一个决策平面来将正类样例和负类样例分割开。特别地，线性分类器按照如下表达式的符号来标记样例：

$$w_0 + w_1 x_1 + \cdots + w_n x_n$$

具体的行为由权重w_i来决定。机器学习的任务就是要找到合适的权重。

- 这一章中的学习技术依赖于“从错误中学习”的准则。训练样例都是一个接着一个地呈现给学习器的。每一次学习器错分了样例，就更新权重。当全部的训练集都训练过了，一个学习阶段即完成。通常都是需要多个学习阶段的。
- 这里考虑了两个权重修改的技术：加法规则的感知机学习和乘法规则的WINNOW。
- 在多于两个类的域，可以对每类使用一个特定分类器。然后，用一个“主分类器”来选择那个使得 $\sum_{i=0}^{n} w_i x_i$ 有最大值的分类器所返回的类别。
- 在有非线性类别边界的域，有时候可以使用多项式分类器。一个二维域的二阶多项式由如下表达式定义：

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2$$

- 如果非线性项（如 $x_1 x_2$）由（借助于乘子）新创建的属性，如$z_3 = x_1^2$ 和 $z_4 = x_1 x_2$来代替，则多项式的权重可以通过线性分类器中相同的学习算法得到。
- 多项式分类器趋向于过拟合带噪声的训练数据，这主要是由大量的可训练的权重引起的高灵活性所导致的。
- 潜在的最好的类别分割超平面（在无穷多个候选当中）可以由支持向量机（SVM）技术获得，它最大化了超平面与最近的正类和负类样例之间的距离。

历史简评。感知机学习的准则是由罗森布兰特[68]（Rosenblatt）提出的，而

WINNOW 是由利特斯通[47]（Littlestone）提出和分析的。线性和多项式分类器的容量的问题是卡渥[14]（Cover）分析出来的。作为第 7 章会涉及的计算学习理论的结果，瓦普尼克[80]（Vapnik）提出了支持向量机的原理。

4.9 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 写出线性分类器的等式来实现如下函数：
   - 在布尔属性 x_1，…，x_5中至少两个是 true。
   - 在布尔属性 x_1，…，x_6中至少 3 个是 true，并且至少一个是 false。
2. 回到表 4.2 中的样例。从一个不同于表格中所用的权值的初始化开始，手工模拟感知机学习算法的学习过程。也试一下另外一个学习率。
3. 使用 WINNOW 重复同样的练习。不要忘记为感知机中负的权重引入额外的“属性”。
4. 写出在二维空间定义一个三阶多项式的等式。如果我们想用感知机学习算法来训练权重，则需要多少个乘子（每个至多对应于 3 个输入）？

思考题

1. 怎样使用线性分类器来识别不相关的属性？提示，试着在属性的不同子集上运行学习算法，然后观察固定的几轮之后得到的错误率。
2. 解释为什么应用于一对样例（一个属于正类，另一个属于负类）上的最近邻分类器定义了一个线性分类器。基于这个观察提出一个机器学习算法来实现线性分类器的归纳学习。推广这个学习过程到 n 维域。

3. 什么时候线性分类器会比多项式分类器在独立的测试样例上可能表现出更好的分类性能？
4. 有时候，由于类别标号的噪声，一个线性可分的域会变成线性不可分。思考一个技术以移除这样的噪声样例。提示，你可能会用到一个在 k 近邻分类器中已经遇到过的一个思路。

上机实验

1. 实现感知机学习算法并在以下训练集上运行它，这个训练集包含由 4 个属性描述的 6 个样例（3 个正的和 3 个负的）：

x_1	x_2	x_3	x_4	类别
1	1	1	0	**pos**
0	0	0	0	**pos**
1	1	0	1	**pos**
1	1	0	0	**pos**
0	1	0	1	**pos**
0	0	0	1	**pos**

因为两类不是线性可分的，所以可以观察到线性分类器不能达到 0 错误率。

2. 创建一个包含 20 个样例，每个样例由 5 个二值属性描述（x_1，…，x_5）的训练集。至少有 3 个属性值为 1，$x_i = 1$ 的样例被标记为正类，其他样例被标记为负类。将这个训练集作为输入，用感知机归纳学习出一个线性分类器。用不同的学习率 η 做实验。画出一个函数，横轴代表 η，纵轴代表为了正确分类所有训练样例所需要呈现给分类器的样例数。讨论所得结果。
3. 使用与前一个作业相同的域（5 个布尔属性，相同的正类定义）。对每个样例增加 N 个布尔属性，这些增加的布尔属性的值由一个随机数生成器来给出。将 N 从 1 变换到 20。观察，基于 N，为了得到 0 错误率需要呈现的样例数。

4. 再一次使用相同的域，但是通过改变随机选择的样例的属性值来增加属性噪声（保持类标不变）。观察可以得到的最小错误率。
5. 用不同大小的训练集来重复以上 3 个作业，并在（总是相同的）学习阶段没有见过的测试样例集上评价结果。
6. 用 WINNOW 来代替感知机学习，重复以上 4 个实验。从所需要的计算代价的角度比较实验结果。这些代价可以用在训练集上收敛到 0 错误率所需要的学习轮数来度量。
7. 定义一个包含 3 个从单位区间［0，1］取值的数值属性的域。产生 100 个训练样例，将使得表达式 $1 - x_1 + x_2 + x_3$ 为正的样例标记为正类。使用权值更新法则修改为如下版本的“感知机学习算法”：

   1）$w_i = w_i + \eta\ [c(\boldsymbol{x}) - h(\boldsymbol{x})]x_i$。

   2）$w_i = w_i + \eta\ [c(\boldsymbol{x}) - h(\boldsymbol{x})]$。

   3）$w_i = w_i + \eta\ [c(\boldsymbol{x}) - \sum w_i x_i]\ x_i$。

   4）$w_i = w_i + \eta\ [c(\boldsymbol{x}) - \sum w_i x_i]$。

   它们都能在训练集上收敛到 0 错误率吗？比较收敛速率。
8. 创建一个训练集，每个样例由 6 个布尔属性（x_1，…，x_6）来描述。将每个样例标注为如下定义的 4 类中的一类。

   1）C_1：至少 5 个属性有 $x_i = 1$。

   2）C_2：3 个或 4 个属性有 $x_i = 1$。

   3）C_3：两个属性有 $x_i = 1$。

   4）C_4：一个属性有 $x_i = 1$。

   使用感知机学习，每一个对应于 4 个类中的一个。

   作为一个变体，使用不同数目的不相关属性，数目从 0 改变到 20。观察还能否在训练集上得到 0 错误率。记录在一个独立的测试集上所观察到的假的正类和假的负类的数目。
9. 设计实验展示出：如果增加类别数，则图 4.4 中并行连接的 K 个二值分类器的表现会下降。在有噪声的情况下，这个观察有多显著？

10. 从 UCI 库（www. ics. uci. edu/ ~ mlearn/MLRepository. html）中选择布尔域问题，在其上运行线性分类器的归纳学习，比较结果。
11. 在 UCI 库中选择的域上做实验，观察学习率 η 对感知机学习算法收敛速度的影响。
12. 比较线性和多项式分类器的表现。观察前者怎样在简单的域取胜、后者怎样在高度非线性的域取胜。

机器学习导论

第 5 章　人工神经网络

多项式分类器可以对任何形状的决策面进行建模，但是它们的实际有效性却是有限的，因为它们很容易过拟合带噪声的训练数据，而且有时包含特别多的可训练参数。更为普遍使用的是人工神经网络，许多被称为神经元的简单单元通过加权连接相互结成拥有非常高性能的更大的结构。

神经网络领域太丰富了，由于篇幅限制很难全面覆盖。因此，我们仅提供两种常用类型的基本信息：多层感知机和径向基函数网络。本章将描述它们如何分类样例和从训练数据中归纳学习它们的一些基本机制。

5.1 作为分类器的多层感知机

在这一章中，我们假设所有属性都是连续的。而且，假设它们已经被标准化，所有取值总是落入区间 $[-1, 1]$，这是实用的（尽管不是严格必需的）。

神经元。神经元是一个多层感知机的基本单元，它的函数非常简单。到达输入的一个加权和要经过一个传递函数。若干不同的传递函数都可以用，本章倾向于使用的一个是所谓的 S 形函数，它由式(5.1)定义，这里是 $\sum$ 输入的加权和：

$$f(\sum) = \frac{1}{1 + e^{-\Sigma}} \tag{5.1}$$

图 5.1 展示了代表传递函数的曲线。读者可以看到，$f(\sum)$ 随着 $\sum$ 的值的增加单调递增，但是不会离开开区间(0,1)，因为 $f(-\infty) = 0$ 而 $f(\infty) = 1$。纵轴与 $f(0) = 0.5$ 相交。我们假设所有神经元都有相同的传递函数。

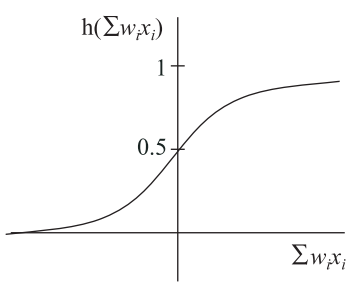


图 5.1　一个常用的传递函数——S 形函数

多层感知机。图 5.2 中的神经网络就是著名的多层感知机。由椭圆表示的神经元排列在输出层和隐含层（当我们从上面来看这个网络时，隐含层是被输出层遮挡了的）。简单起见，我们仅考虑有一个隐含层的网络，需要记住的是使用两层甚至三层的情况也很普遍，但是很少有使用更多层的。

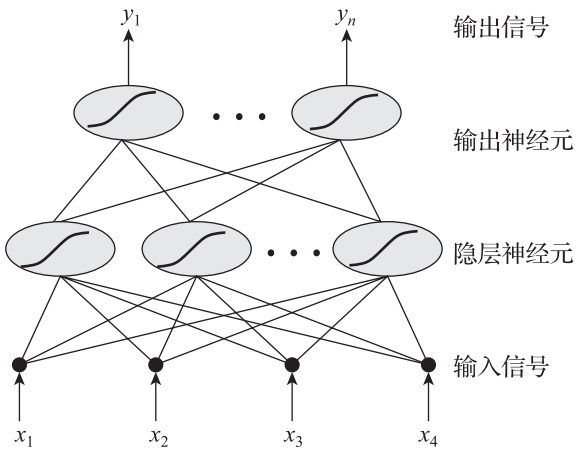


图 5.2　一个包含两个相互连接层的神经网络示例

在同层的神经元之间是没有连接的，相邻两层之间是全连接的。重要的是，每一个神经元与神经元之间的连接都与一个权重相关联。从第 j 个隐层神经元到第 i 个输出层神经元之间的连接的权重记为 $w_{ji}^{(1)}$，从第 k 个属性到第 j 个隐层神经元之间的连接的权重记为 $w_{kj}^{(2)}$。注意，第一个指标总是指连接的“开始”，第二个是它的“结束”。

前向传播。我们通过一个样例 $x=(x_1, \cdots, x_n)$ 来展示神经网络，这个样例的属性值要沿着网络中的连接传到各个神经元。值 x_k 乘以与连接相关联的权重，第 j 个隐层神经元接受加权和 $\sum_k w_{kj}^{(2)} x_k$ 作为输入，然后传输这个和式到 S 形函数 $f\left(\sum_k w_{kj}^{(2)} x_k\right)$，第 i 个输出层神经元接受由隐层而来的加权和式，并再一次地将它传输给传递函数。这就是第 i 个输出是怎样得到的。属性值从网络的输入到输出这样的传播过程被称为前向传播。

每个类别都被分配了它自己的输出神经元，第 i 个输出神经元所返回的值代表了支持第 i 类的证据的量。例如，如果 3 个输出神经元得到的值为 $y=(0.2, 0.6, 0.1)$，则分类器会将给定的样例标记为第二类，因为 0.6 大于 0.2 和 0.1。

本质上，这种两层的感知机计算了式（5.2），这里 f 是神经元中所使用的 S 形传递函数（参加式（5.1）），$w_{kj}^{(2)}$ 和 $w_{ji}^{(1)}$ 分别是指向隐层和输出层的连接，x_k 是给定样例的属性值：

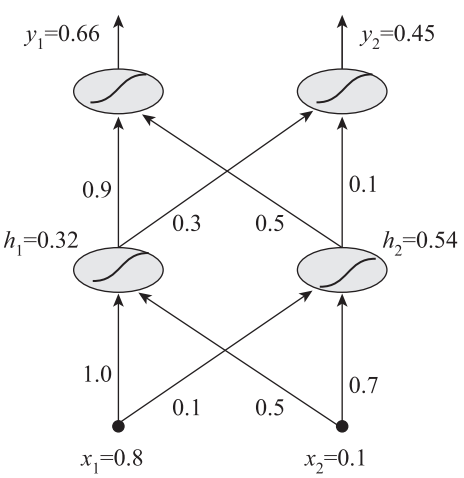
$$y_i = f\Big(\sum_j w_{ji}^{(1)} f\Big(\sum_k w_{kj}^{(2)} x_k\Big)\Big) \tag{5.2}$$

一个数值例子。表 5.1 中的数值例子描述了前向传播的原理。一开始，属性向量 x 出现。在达到隐含层神经元之前，属性值要乘以相应的权重，并且加权和要传给 S 形函数。然后，结果（$h_1 = 0.32$ 和 $h_2 = 0.54$）乘以下一层的权重，并传给输出神经元，这里它们再次被传输给 S 形函数。这就是两个输出值 $y_1 = 0.66$ 和 $y_2 = 0.45$ 是怎么得到的了。支持“左侧”输出神经元的类别的证据要高于支持“右侧”输出神经元的类别的证据。因此，分类器选择左边神经元的类别。

通用分类器。数学家曾证明，选择正确的权重和隐层神经元个数，式（5.2）可以以任意精度近似任何实际的函数。这个所谓的通用性理论的结果就是，原则上，多层感知机可以用来解决任何分类问题。尽管如此，这个理论并没有告诉我们，需要多少隐层神经元以及各个权值应该是多少。用另一句话说就是，我们知道结果存在，但是对找到它却没有任何保证。

表 5.1　多层感知机中前向传播的例子

任务：通过以下网络前向传播 $x = (x_1, x_2) = (0.8, 0.1)$。



解：

隐层神经元的输入：

$z_1^{(2)} = 0.8 \times (-1.0) + 0.1 \times 0.5 = -0.75$

$z_2^{(2)} = 0.8 \times 0.1 + 0.1 \times 0.7 = 0.15$

（续）

隐层神经元的输出：

$$h_1 = f(z_1^{(2)}) = \frac{1}{1+e^{-(-0.75)}} = 0.32$$

$$h_2 = f(z_2^{(2)}) = \frac{1}{1+e^{-0.15}} = 0.54$$

输出层神经元的输入：

$$z_1^{(1)} = 0.54 \times 0.9 + 0.32 \times 0.5 = 0.65$$

$$z_2^{(1)} = 0.54 \times (-0.3) + 0.32 \times (-0.1) = -0.19$$

输出层神经元的输出：

$$y_1 = f(z_1^{(1)}) = \frac{1}{1+e^{-0.65}} = 0.66$$

$$y_2 = f(z_2^{(1)}) = \frac{1}{1+e^{-(-0.19)}} = 0.45$$

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释由一个连续值属性向量描述的样本是怎样通过多层感知机前向传播的。网络的输出如何解释？
- 什么是传递函数？写出定义S形传递函数的公式并描述它的形状。
- 什么是通用性理论？它告诉了我们什么，没告诉我们什么？

5.2 神经网络的误差

我们在后续章节再介绍训练多层感知机的技术（找到它的权重）。在解决这个问题之前，有必要通过仔细看一下分类的方法和评价分类准确率的方式来做些准备工作。

错误率。假如一个训练样例 $\boldsymbol{x}$ 和它的已知的类别 $c(\boldsymbol{x})$ 被呈现给了一个已有的多层感知机。前向传播过程会给出一个类标 $h(\boldsymbol{x})$。如果 $h(\boldsymbol{x}) \neq c(\boldsymbol{x})$，就造成了一个错误。这也可能发生在其他样例上，我们想要知道的是这会以怎样的频率发生。我们想知道错误率——对于一个给定的样例集，它是由错误数除以样例数得到的。

例如，如果分类器将200个样例中的30个错误分类了，那么错误率就是$\frac{30}{200}=0.15$。

然而，这并没有给出网络分类器表现的整幅图片。错误率没有反映出来的就是S形函数度量错误大小的能力。

举一个例子来澄清这一点。假设我们有两个不同的网络可以选择，它们每一个都有对应于标记为C_1，C_2和C_3的类别的3个输出神经元。对于一个样例x，我们假设第一个网络的输出是$\boldsymbol{y}_1(\boldsymbol{x})=(0.5, 0.2, 0.9)$，第二个的输出是$\boldsymbol{y}_1(\boldsymbol{x})=(0.6, 0.6, 0.7)$。这意味着两个网络都会将$\boldsymbol{x}$标记为第三个类别，$\boldsymbol{h}_1(\boldsymbol{x})=\boldsymbol{h}_2(\boldsymbol{x})=C_3$。如果正确的答案是$c(\boldsymbol{x})=C_2$，那么两个网络都犯错误了，但是看起来错误却不是相同的。读者会注意到，第一个网络关于类别是C_3是“非常确定”的(因为0.9是明显大于另外两个输出的，即0.5和0.2)，但是第二个网络不是这么确定，输出值（0.6，0.6，0.7）的差别是如此之小，以至于怀疑C_3只是靠运气取胜的。因为对不正确类别较弱的保证，所以第二个网络比第一个犯的错更小一些。

一个更加合适的误差函数——均方误差（*MSE*），可以捕获到这种情况。

目标向量。但是，在给出均方误差的定义之前，我们必须先介绍另一个重要的概念，即目标向量，它也依赖于具体的样例$\boldsymbol{x}$。我们用$\boldsymbol{t}(\boldsymbol{x})$来标记它。在一个$m$类的域，目标向量$\boldsymbol{t}(\boldsymbol{x})=(t_1(\boldsymbol{x}), \cdots, t_m(\boldsymbol{x}))$包含$m$个二值数字。如果一个样例属于第$i$类，那么$t_i(\boldsymbol{x})=1$，这个向量的其他元素均为$t_j(\boldsymbol{x})=0$（这里$j\neq i$）。例如，假设存在3个不同的类，$C_1$，$C_2$和$C_3$，已知$\boldsymbol{x}$属于$C_2$类。理想情况下，第二个神经元应该输出1，而其他两个神经元应该输出0（更加准确地，因为S形函数取值于开区间（0，1），所以输出会靠近1和0）。因此，目标为$\boldsymbol{t}(\boldsymbol{x})=(t_1, t_2, t_3)=(0, 1, 0)$。

均方误差。均方误差由输出向量与目标向量元素之间的差值来定义：

$$MSE=\frac{1}{m}\sum_{i=1}^{m}(t_i-y_i)^2 \tag{5.3}$$

当计算网络的*MSE*时，我们需要为每个输出神经元计算它的输出与目标向量对应元素的差值。注意，为了确保负的差值不会被从正的差值里减去，括号中的项(t_i-y_i)是求平方的。

回到以上提到的两个神经网络的例子，如果目标向量是 $\boldsymbol{t}(\boldsymbol{x})=(0,1,0)$，则均方误差如下：

$$MSE_1=\frac{1}{3}[(0-0.5)^2+(1-0.2)^2+(0-0.9)^2]=0.57$$

$$MSE_2=\frac{1}{3}[(0-0.6)^2+(1-0.6)^2+(0-0.7)^2]=0.34$$

正如所期望的，$MSE_2<MSE_1$，这与我们的直觉，第二个网络在 $\boldsymbol{x}$ 上比第一个网络错得更少一些，是一致的。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 为什么我们会说传统的错误率关于一个神经网络的分类准确率不能提供足够的信息?
- 解释神经网络的输出，样例的类别和目标向量的不同之处。
- 写出定义错误率和均方误差的公式。

5.3 误差的反向传播

在多层感知机中，影响网络表现的参数是权重的集合，$w_{ji}^{(1)}$ 和 $w_{kj}^{(2)}$。机器学习的任务就是要为这些权重找到能够优化网络分类性能的值。正如线性分类器的情况，这可以通过训练实现。这节主要介绍一种常用的可以完成这个任务的技术。

一般的情形。原则上，学习的过程与前面章节介绍的是一样的。一开始，权重被初始化为小的随机数，典型地，取值于区间 $(-0.1,0.1)$。这步之后，训练样例一个接着一个出现，每一个都前向传播到网络的输出层。这个输出与样例的目标向量之间的差异告诉我们怎样修改权重。权重修改之后，下一个样例出现。当到达最后一个训练样例时，一轮学习完成。在多层感知机中，成功训练需要的学习轮数是远远大于线性分类器的情况的：它可以是几千几万甚至更多。

梯度下降。在进入权重调整的具体公式的讲解之前，我们需要对问题的本质有个更好的理解。图 5.3 会帮助我们。纵轴代表了均方误差，它表示为（沿着横轴画的）网络权重的一个函数。为了图解的方便，我们假设只有两个权重。当然，退一步说，这是不现实的。但是如果我们想要的就是一个有启发性的例子，我们不能简单地提供更多维的情况——我们几乎不能看的十维的样子，对吧？这里，我们需要知道的信息是，误差函数可以被想象成一种“风景画”，它的“山谷”代表了函数的局部最小值。其中最深的是全局最小值，在理想情况下，这也是训练过程想要达到地方；更特别地，应该找到的是对应于全局最小值的那组权重。

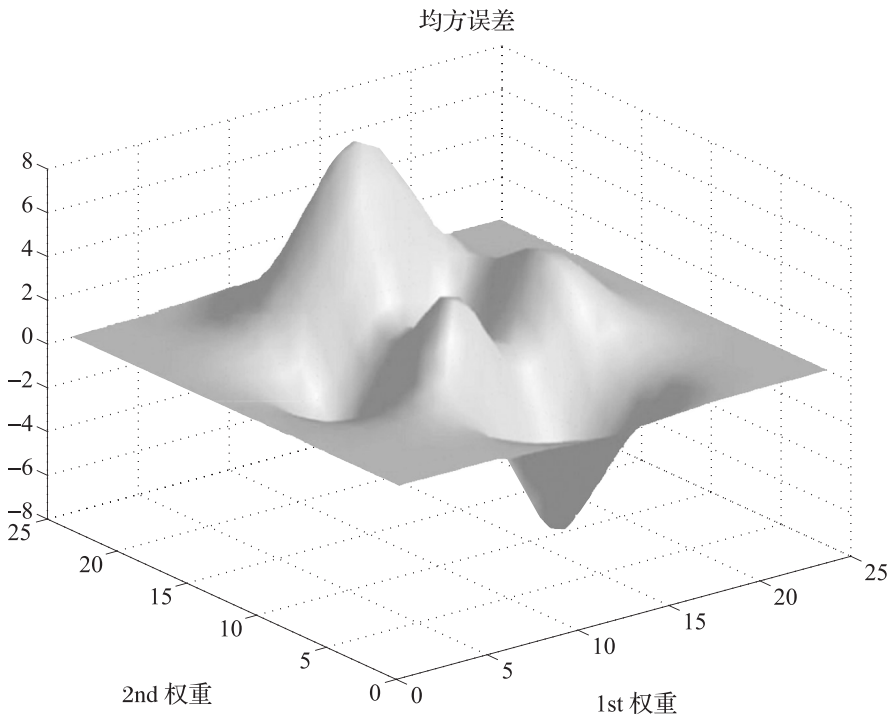


图 5.3　对于一个给定的样例，每一个权重的组合都暗示了一定的均方误差，训练过程应该尽快地降低这个误差

图 5.3 告诉我们任何一对权重为每个给定的训练样例都定义了在风景画中的一个位置，特别是在某个斜面上的。任何的权重改变都会产生沿着水平的两个坐标轴的不同坐标，也对应于误差函数曲面上的不同位置。这个新的位置到底在哪里，是在斜面的上部还是下部，依赖于每个权重变化的多少和方向。例如，同时

给 w_1 和 w_2 增加0.1对于均方误差仅会有微小的降低，但是给 w_1 增加0.3给 w_2 增加0.1会降低很多。

在这里讨论的技术中，我们希望权重改变是沿着误差函数最陡方向下降的。回忆第1章中的术语，这是爬山搜索要做的事情。为此，用在多层感知机中的最著名的技术就是误差的反向传播。

误差的反向传播。确切的权值调整公式可以由式（5.2）通过计算函数的梯度推导出来。但是，这本书是写给实践者的，而不是数学家的，因此我们将跳过推导过程，转而集中于解释学习过程的行为。

首先，假设各个神经元对整体的误差的贡献不同是合理的。它们中的一些比另外的一些“破坏游戏”更加严重。如果这是事实，则读者会同意，指向这些神经元的连接比那些指向问题更小的神经元的连接应该有更大的权重改变。

幸运的是，每个神经元对于整体误差的“责任”大小可以容易地通过计算得到。一般来说，公式的具体选择依赖于使用了什么传递函数。对于S形函数（参见式（5.1））的情况，责任可以按如下方式计算。

输出层神经元：$\delta_i^{(1)} = y_i(1-y_i)(t_i - y_i)$。

这里，$(t_i - y_i)$ 是第 i 个输出和相应目标值的差。这个差值被乘以 $y_i(1-y_i)$，而这项的最小值在当 $y_i = 0$ 或 $y_i = 1$ 时达到（对于 x 是否应该标记为第 i 类的一个“强硬意见”），当 $y_i = 0.5$ 时这项达到最大，这种情况下“意见”可以被认为是中立的。注意，$\delta_i^{(1)}$ 的符号仅决定于 $(t_i - y_i)$，因为 $y_i(1-y_i)$ 总是正的。

隐含层神经元：$\delta_j^{(2)} = h_j(1-h_j)\sum_i \delta_i^{(1)} w_{ji}$。

隐层神经元的责任通过反向传播前一步中得到的输出神经元的责任来计算，这就是 $\sum_i \delta_i^{(1)} w_{ji}$ 项所扮演的角色。注意，每一个 $\delta_i^{(1)}$（第 i 个输出神经元的责任）都被乘以关联第 i 个输出神经元和第 j 个隐层神经元的连接的权重。加权和被乘以 $h_j(1-h_j)$，除了用 h_j 替换了 y_i，本质上与前一步使用的是相同的项。

权重更新。现在我们已经知道了各个神经元的责任，我们已经准备好更新指向它们的连接的权重了。类似于感知机学习，使用如下加法规则。

输出层神经元：$w_{ji}^{(1)} = w_{ji}^{(1)} + \eta\, \delta_i^{(1)}\, h_j$。

隐含层神经元：$w_{kj}^{(2)} = w_{kj}^{(2)} + \eta\, \delta_j^{(2)} x_k$。

因此，权重修正的程度决定于 $\eta\,\delta_i^{(1)}\,h_j$ 或 $\eta\,\delta_j^{(2)}x_k$。这里有两点需要观察。第一，神经元的责任 $\delta_i^{(1)}$ 或 $\delta_j^{(2)}$ 被乘以学习率 η，理论上说，η 应该从单位区间中选出，$\eta\in(0,1)$，但是实际实现的时候经常使用很小的值，特别地小于 0.1。第二，神经元的责任 $\delta_i^{(1)}$ 或 $\delta_j^{(2)}$ 也被分别乘以 $h_j\in(0,1)$ 和 $x_k\in[0,1]$。因此，修正是非常小的，但是它的作用是相对的。如果要加的项的值是，如 0.02，那么小的权重，如 $w_{ji}^{(1)}=0.01$，会比大的权重如 $w_{ji}^{(1)}=1.8$ 受到更显著的影响。

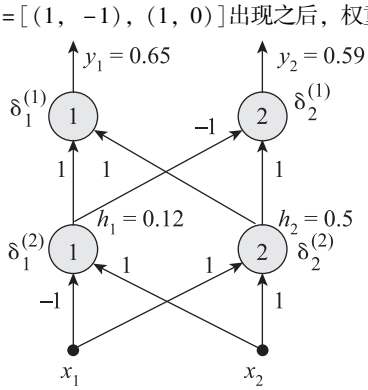
表 5.2 中的伪代码总结了整个训练过程。读者也可以从仔细阅读表 5.3 中的例子获益，这个例子给出了对应于每个训练样例权重是如何更新的所有必要的细节。

表 5.2　有一个隐含层的神经网络中的误差反向传播

1. 将样例 x 呈现给输入层，并在网络中传播它。
2. 设 $\boldsymbol{y}=(y_1,\cdots,y_m)$ 是输出向量，$\boldsymbol{t}(\boldsymbol{x})=(t_1,\cdots,t_m)$ 是目标向量。
3. 对于每个输出神经元，计算它对网络误差的责任 $\delta_i^{(1)}$：$\delta_i^{(1)}=y_i(1-y_i)(t_i-y_i)$。
4. 对于每个隐含层的神经元，计算它对网络误差的责任 $\delta_j^{(2)}$。完成这步的时候，使用前一步得到的输出神经元的责任 $\delta_i^{(1)}$。$\delta_j^{(2)}=h_j(1-h_j)\ \sum_i\delta_i^{(1)}w_{ji}$。
5. 使用如下公式更新权重，这里 η 是学习率。
   输出层：$w_{ji}^{(1)}=w_{ji}^{(1)}+\eta\,\delta_i^{(1)}\,h_j$；$h_j$ 是第 j 个隐层神经元的输出。
   隐含层：$w_{kj}^{(2)}=w_{kj}^{(2)}+\eta\,\delta_j^{(2)}x_k$；$x_k$ 是第 k 个属性值。
6. 除非一个终止准则被满足，否则返回步骤 1。

表 5.3　反向传播的图示

任务：在如下神经网络中，令传递函数为 $f(\sum)=\dfrac{1}{1+e^{-\sum}}$。使用误差反向传播（取 $\eta=0.1$）展示在样例 $[\boldsymbol{x},\boldsymbol{t}(\boldsymbol{x})]=[(1,-1),(1,0)]$ 出现之后，权重是如何改变的。



（续）

前向传播。

上图展示了前向传播之后的状态，离开隐含层和输出层神经元的信号通过如下计算得到：

$$h_1 = \frac{1}{1+e^{-(-2)}} = 0.12$$

$$h_2 = \frac{1}{1+e^{0}} = 0.5$$

$$y_1 = \frac{1}{1+e^{-(0.12+0.5)}} = 0.65$$

$$y_2 = \frac{1}{1+e^{-(-0.12+0.5)}} = 0.59$$

误差反向传播。

目标向量是 $t(x)=(1, 0)$，而输出向量是 $y=(0.65, 0.59)$，反向传播的任务是要建立每个神经元对输出误差的责任。如下是关于输出神经元的计算：

$$\delta_1^{(1)} = y_1(1-y_1)(t_1-y_1) = 0.65\ (1-0.65)(1-0.65) = 0.0796$$

$$\delta_2^{(1)} = y_2(1-y_2)(t_2-y_2) = 0.59\ (1-0.59)(0-0.59) = -0.1427$$

使用这些值，我们可以计算因此神经元的责任。注意，我们先计算对于两个隐层神经元的每一个（标记为Δ_1 和Δ_2）的加权和，即 $\sum_i \delta_i^{(1)} w_{ij}^{(1)}$。

$$\Delta_1 = \delta_1^{(1)} w_{11}^{(1)} + \delta_2^{(1)} w_{12}^{(1)} = 0.0796 \times 1 + (-0.1427) \times (-1) = 0.2223$$

$$\Delta_2 = \delta_1^{(1)} w_{21}^{(1)} + \delta_2^{(1)} w_{22}^{(1)} = 0.0796 \times 1 + (-0.1427) \times 1 = -0.0631$$

$$\delta_1^{(2)} = h_1(1-h_1)\Delta_1 = 0.12\ (1-0.12) \times 0.2223 = -0.0235$$

$$\delta_2^{(2)} = h_2(1-h_2)\Delta_2 = 0.5\ (1-0.5) \times (-0.0631) = 0.0158$$

一旦责任都已知了，权重修改就是直接的了。

- $w_{11}^{(1)} = w_{11}^{(1)} + \eta\,\delta_1^{(1)}\,h_1 = 1 + 0.1 \times 0.0796 \times 0.12 = 1.00096$
  $w_{21}^{(1)} = w_{21}^{(1)} + \eta\,\delta_1^{(1)}\,h_2 = 1 + 0.1 \times 0.0796 \times 0.5 = 1.00398$
  $w_{12}^{(1)} = w_{12}^{(1)} + \eta\,\delta_2^{(1)}\,h_1 = -1 + 0.1 \times (-0.1427) \times 0.12 = -1.0017$
  $w_{22}^{(1)} = w_{22}^{(1)} + \eta\,\delta_2^{(1)}\,h_2 = -1 + 0.1 \times (-0.1427) \times 0.5 = 0.9929$
- $w_{11}^{(2)} = w_{11}^{(2)} + \eta\,\delta_1^{(2)}\,x_1 = -1 + 0.1 \times (-0.0235) \times 1 = -1.0024$
  $w_{21}^{(2)} = w_{21}^{(2)} + \eta\,\delta_1^{(2)}\,x_2 = 1 + 0.1 \times (-0.0235) \times (-1) = 1.0024$
  $w_{12}^{(2)} = w_{12}^{(2)} + \eta\,\delta_2^{(2)}\,x_1 = 1 + 0.1 \times 0.0158 \times 1 = 1.0016$
  $w_{22}^{(2)} = w_{22}^{(2)} + \eta\,\delta_2^{(2)}\,x_2 = 1 + 0.1 \times 0.0158 \times (-1) = 0.9984$

权重都更新了，网络就可以接收下一个样例了。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 为什么本节的训练技术被称为“误差反向传播”？建立“神经元的责任”的根本原因是什么？
- 讨论计算各层神经元的责任的公式的表现。
- 解释用来更新权重的公式的表现。说明这些公式的一些关键的方面。

5.4 多层感知机的特殊方面

因篇幅所限，我们不能深入调研使得多层感知机的训练更像艺术而胜于科学的许多特征了。为了验证所有这些特征，可能会需要另一个至少与本章同样长度的章节。尽管如此，如果训练想要成功，一些关键方面的知识是必不可少的。因此，我们对更加重要的一些方面做个简单的综述。

计算代价。误差反向传播的计算代价很高。一个样例出现了，就得计算每一个神经元的责任，然后相应地修改权重。这要对所有训练样例重复进行，通常需要很多轮。为了理解所有这些的真实代价，考虑一个要分类由 100 个属性所描述的样例的网络，这是一个相当现实的情形。如果有 100 个隐层神经元，那么这个低一些的层中权重的个数是 $100 \times 100 = 10^4$。这是每个训练样本出现之后权重改变的数量。注意，在这些计算中，如果类别数较少，则上层的权重可以忽略。例如，在一个有 3 类的域中，上层的权重数量为 $100 \times 3 = 300$，相对于10^4它是一个很小的数字。

假如训练集包含10^5个样例，而且训练需要持续10^4轮。这时，权重更新的数量是$10^4 \times 10^5 \times 10^4 = 10^{13}$。这看起来已经很多了，但是许多应用甚至是有更高要求的。因此，许多相当灵巧的会使训练更加高效的方法被提了出来。然而，这些都超出了我们这里所感兴趣的范围。

重新考虑目标值。简单起见，截至目前我们都假设每一个目标值是 1 或 0，这

可能不是最好的选择。首先，一个神经元的输出y_i永远不会达到这些值。更进一步地，在这两个极端的附近权重的改变是微小的，因为当y_i接近0或1时，输出神经元的责任的计算，$\delta_i^{(1)} = y_i(1-y_i)(t_i-y_i)$，仅返回一个非常接近0的值。最后，我们知道，分类器会选择返回最大值的输出神经元对应的类别。因此，各个神经元的输出精度并不重要，更重要的是与其他输出的比较结果。如果前向传播给出$\boldsymbol{y}=(0.9, 0.1, 0.2)$，那么样例一定被标记为第一类（支持$y_i=0.9$对应的类别），这个决定不会被小的权重改变所影响。

考虑到这些争论，更合适的目标值推荐如下：例如，如果样例属于第i类，则$t_i(\boldsymbol{x})=0.8$，否则$t_i(\boldsymbol{x})=0.2$。假设有3个类别：C_1，C_2和C_3，而$c(\boldsymbol{x})=C_1$。这种情况下，目标向量将被定义为$\boldsymbol{t}(\boldsymbol{x})=(0.8, 0.2, 0.2)$。0.8和0.2都在S形函数相对较高敏感性的区域，而且这样会消除上一段中引发的多数的担心。

局部极小值。图5.3描绘了当多层感知机训练采用梯度下降方法时的主要缺点。权重以沿着最陡斜面下降的方式改变。但是一旦到达一个局部极小值的底部，就没有任何地方可去了，这有点儿尴尬，毕竟，最终目标是要到达全局最小值。这里需要两个事物：第一个，识别局部极小值的一个机制；第二个，从落入局部极小值中恢复出来的方法。

一种在训练过程中实时发现局部极小值的可能性是记录均方误差，在每轮学习之后将它在整个训练集上求和。正常情况下，这个和从一轮到另一轮逐渐下降。一旦它看起来到达了一个高原，几乎观察不到任何误差的下降，则学习过程就有陷入了局部极小值的可能。

克服这个困难的技术通常依赖于自适应的学习率（参见下文），以及增加新的隐层神经元（参见第5.5节）。一般来说，在有很多隐层神经元的网络中，这个问题并不严重。同样，如果所有的权重都很小，如取值于区间$(-0.01, 0.01)$，则局部极小值趋向于较浅且不那么频繁。

自适应的学习率。当描述误差反向传播的时候，我们假设用户设定的学习率η是一个常数。但是，这在实际应用中是很少发生的。更多的情况是，训练开始于一个大的η，然后它逐渐随着时间下降。这样做的动机很容易猜到。开始的时候，较大的权重修改会降低学习轮数，而且甚至会帮助学习器“跳过”一些局部

极小值。但是，随后，这个大的 η 就可能导致“越过”全局最小值，这也就是为什么它的值应该被减小了。如果我们将学习率表达为时间的函数 $\eta(t)$，这里 t 说明了现在训练是在哪一轮中，然后，如下的负指数公式会逐渐降低学习率（α 是负指数的斜率，$\eta(0)$ 是学习率的初始值）：

$$\eta(t) = \eta(0)\ \mathrm{e}^{-\alpha t} \tag{5.4}$$

可能需要注意的是，一些先进的权重改变公式是可以反映出“当前趋势”的。例如，使用“惯性”是非常常见的：如果之前两次权重改变都是正的（或都是负的），就有理由增加权重改变的步长；相反地，如果一个正的改变之后是一个负的改变（或相反），则为了避免越过了，就应该减小步长。

过训练。多层感知机与多项式分类器享有同一个令人不愉快的属性。从理论上说，它们能够对任何决策面建模，而这使得它们易于过拟合训练数据。读者会记得过拟合典型的意思是，对带噪声的训练样例完美地分类，但不可避免的是，随后在未来的数据上令人失望的表现。

对于小的多层感知机，这个问题并不令人痛苦，它们并不足以灵活到过拟合。但是随着隐层神经元个数的增加，网络获得了灵活性，过拟合就变成一个真正要担心的事情了。然而，正如我们会在下章所学的，这并不意味着我们应该总是倾向于小的网络！

有一个简单的方法可以发现训练过程是否达到了“过拟合”阶段。如果训练集足够大，则可以将 10% ~20% 的样例放在一边。在误差反向传播过程中始终不用它们；确切地说，每轮学习之后，计算在这些保留的样例上的均方误差的和。一开始，这个和会趋于减少，但是至多到某一个时刻；然后，它又开始增长，警示工程师训练已经开始过拟合数据了。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 关于误差反向传播技术的计算代价你都知道些什么?
- 解释为什么本章推荐的目标向量的值要取自于{0.8，0.2}而不是{1，0}。

- 讨论局部极小值的问题。为什么它会成为训练中的一个问题？如何降低陷入一个局部极小值的危险？
- 自适应的学习率有什么好处？关于它的什么公式受到了推荐？
- 关于神经网络的训练会导致过拟合训练数据的危险你都知道些什么？

5.5 结构问题

到目前为止，有一个重要的问题被忽略了：要使用多少个隐层神经元？如果只有一个或两个，网络会缺乏灵活性；它不仅不能对一个复杂的决策面进行建模，而且这个网络的训练也很容易陷入一个局部极小值。在另一个极端，使用数千个神经元不仅会增加计算的代价，因为要训练这么多神经元，这个网络也会比所需求的更加灵活。所以，它很容易过拟合数据。通常，在这种情况下，需要找到一些折中的办法。

表现相对于规模。假设你决定要运行下面的实验。可获得的已预先分类的样例的集合被分为两个部分，一个用来训练，另一个用来测试。训练应用于若干个神经网络，每个都有不同个数的隐层神经元。训练这些网络直到观察不到训练集上错误率的下降。这步之后，度量测试数据上的错误率。

最优的神经元个数。当画在一幅图中，结果一般看起来很像图5.4所描绘的

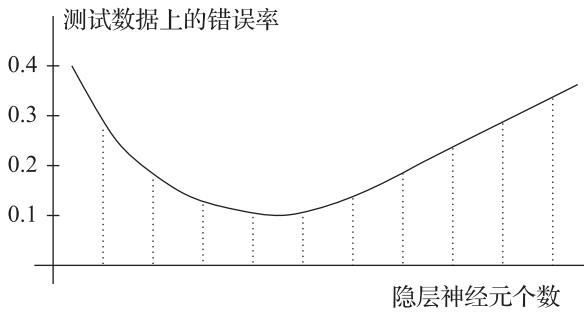


图5.4　在测试样例上测得的错误率依赖于隐含层中神经元的个数

情况。这里，横轴代表隐层神经元的个数，纵轴是在测试集上测得的错误率。典型地，在非常小的网络的情况下，错误率会比较高，因为这些网络缺乏合适的灵

活性，而且也承受着陷入局部极小值的危险。如果我们增加隐层神经元的个数，则这两个劣势都可以减轻。正如图中所画，越大的网络呈现出越低的错误率。但是，太大的网络却易于过拟合。这就是为什么隐层结点数多于一个特定值后，测试集上的错误率又开始增长了（图中右侧的尾巴）。

这条曲线准确的形状依赖于训练数据的复杂度。有时候，当网络包含 3 ~ 5 个隐层神经元的时候，错误率就达到了最小。但是，在其他域，仅当使用了几百个隐层神经元的时候，才能达到最小值。还有的值得一提的是当训练样例完全无噪声的情况。在这种域中，过拟合不会成为一个问题，曲线右侧的尾巴可能都不会向上增长。

寻找合适的规模。上面描述的情形代价太高，在现实应用中不会用到。回过来，我们并不知道我们是否需要仅仅几个、十几个还是上百个神经元，所以，在建立起理想的规模之前，我们可能不得不重复地运行计算量非常大的训练算法很多次。取而代之，我们希望有一种能够更加快速地找到合适规模的技术。

表 5.4 中的伪代码总结了一种这样的技术。思路是从一个非常小的仅包含几个隐层神经元的网络开始。每轮学习之后，学习算法检查在训练集上观察到的均方误差的和。这个误差的和应该是随着学习轮数的增加而保持下降的，但是仅到一个确定点之前。到达这个点的时候，或者是因为它不够充分的灵活性导致了不可能正确分类，或者是因为它落入了一个局部极值点，网络的性能不再增长。当观察到这个的时候，加入一些带随机初始化的权重的神经元，继续训练。

表 5.4　逐步寻找一个好的隐含层规模

1. 一开始，仅使用几个隐层神经元，如 5 个。
2. 训练网络直到均方误差看起来不再改进。
3. 这时候，向隐含层加入几个（如 3 个）带随机初始化的权重的神经元，继续训练。
4. 重复 2 ~ 3 步直到满足终止性条件。例如，新的加入不再能导致显著的误差下降，或隐含层超过了用户设定的最大规模。

通常，加入的灵活性使得进一步的误差下降成为可能。如图 5.5 所示，错误率在两种情形下“暂停”，加入新的隐层神经元提供了必要的新的灵活性。

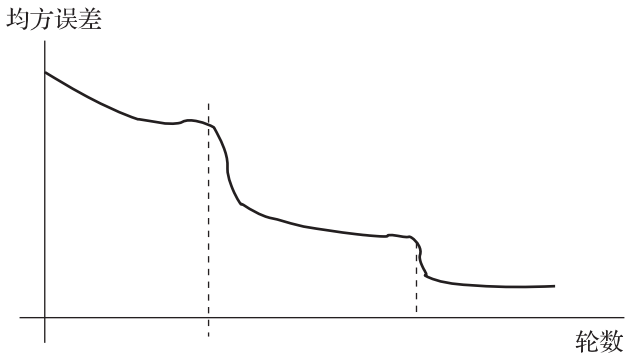


图 5.5　当训练集上的均方误差看起来不再下降，可以通过加入新的隐层神经元获得进一步的改进（在本例中，这被执行了两次）

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 讨论图 5.4 中曲线的形状。非常小和非常大的网络都有什么缺点?
- 解释寻找隐含层合理规模的算法，它面临的主要困难是什么?

5.6　径向基函数网络

多层感知机一个输出神经元的表现与线性分类器的表现是类似的。这样，在类别线性不可分的域中，它可能会被认为效果很差。但是，在神经网络领域，这样的限制不一定是有害的。事情是这样的，原始的样例被隐含层的 S 形函数进行了变换。结果，输出层的神经元处理的是这些变换得到的新的“属性”。在训练过程中，这些变换了的样例（隐层神经元的输出）可能变得线性可分了，以至于输出层神经元可以没有任何困难地将两类分割开。

替代物。也存在变换属性值的其他方法：通过使用所谓的径向基函数（RBF）作为隐层神经元的传递函数。这是图 5.6 所描绘的网络的情形。一个输入样例通过一组神经元进行传输，每个神经元返回一个标记为的φ_j值。

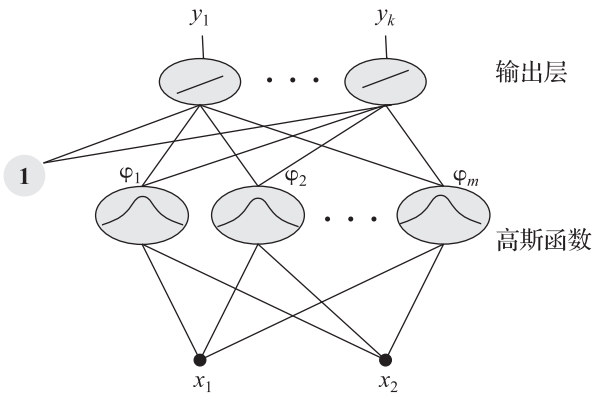


图 5.6　径向基函数网络

径向基函数——RBF。本质上，这是基于我们已经在第 2 章了解到的正态分布。假设描述样本的属性都取值于合理大小的区间，如 $[-1, 1]$。给定一个方差 σ^2，式（5.5）定义了中心位于 $\boldsymbol{\mu}_j = [\mu_{j1}, \cdots, \mu_{jn}]$ 的 n 维高斯曲面：

$$\varphi_j(\boldsymbol{x}) = \exp\left\{ - \frac{\sum_{i=1}^{n} (x_i - \mu_{ji})^2}{2\sigma^2} \right\} \tag{5.5}$$

在某种意义上，$\varphi_j(\boldsymbol{x})$ 度量了样例向量 $\boldsymbol{x}$ 和高斯中心 $\boldsymbol{\mu}_j$ 之间的相似性：两者之间的距离越大，$\varphi_j(\boldsymbol{x})$ 的值越小。如果要对 $\boldsymbol{x}$ 进行分类，网络首先重新描述它为 $\boldsymbol{\varphi}(\boldsymbol{x}) = [\varphi_1(\boldsymbol{x}), \cdots, \varphi_m(\boldsymbol{x})]$。然后，计算第 i 个输出神经元的输出信号，$y_i = \sum_{j=0}^{m} w_{ji}\varphi_j(\boldsymbol{x})$，这里 w_{ji} 是第 j 个隐层神经元到第 i 个输出层神经元之间连接的权重（w_{0i} 连接到一个固定的 $\varphi_0 = 1$）。这个输出信号被解释为支持第 i 类的证据的数量，样例被标记为 $y_i = \max_k \{y_k\}$ 的第 i 类。

输出层权重。建立输出层权重 w_{ij} 是相对简单的。因为只有一层的权重要训练，我们可以也使用第 4 章描述的感知机学习算法，并将它应用于描述经过隐层神经元中的 RBF 函数变换的样例。

高斯中心。通常的做法是用各个训练样例作为高斯中心 $\boldsymbol{\mu}_j$。如果训练集比较小，则我们可以简单地使用每个训练样例对应一个隐层神经元。但是，在许多现实的应用中，训练集是非常大的，这意味着有数千甚至更多个隐层神经元。意识

到这样大的网络是难处理的，许多工程师倾向于仅选择训练样例的一个小的子集作为高斯中心。很多时候，随机选择就够了。另一种可能性是找到几组“相似的”的向量，然后为每个 RBF 神经元使用一个组的中心。

基于 RBF 的支持向量机。RBF 神经元将原始的样例描述转换为包含值φ_1，…，φ_m的一个新的向量。很多时候，这个转换增加了转换后的样例会是线性可分的机会。这使得将它们输入一个权重通过感知机学习训练得到的线性分类器成为可能。

这可能是一个合适的位置来提一下，也是很常见的，将 4.7 节介绍的支持向量机的想法应用于变换后的样例。在这种情况下，得到的机器学习工具通常被称为基于 RBF 的支持向量机。特别是在类间边界是高度非线性的域中，这个分类器比普通的（线性）支持向量机更加强大。

计算代价。RBF 网络的训练包含两步。第一步，选择径向基函数（高斯）的中心。第二步，通过训练得到输出神经元的权重。因为只有一层要训练，训练过程比多层感知机的计算量要小很多。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释径向基函数网络的原理。它在哪些方面不同于多层感知机?
- 在径向基函数网络中有多少权重需要训练？如何完成训练?
- 对于建立高斯隐层，有哪些可能性?

5.7 总结和历史简评

- 多层感知机的基本单元是神经元。神经元接受一个输入的加权和，并传输这个和到一个传递函数。一些不同的传递函数都可用。本章选择使用的是如下等式所定义的 S 形函数,这里是$\sum$ 输入的加权和：

$$f(\Sigma)=\frac{1}{1+e^{-\Sigma}}$$

通常，所有的神经元都使用相同的传递函数。

- 这一章所描述的多层感知机的简单版本由一个输出层和一个隐含层的神经元组成。相邻层的神经元是全连接的，但是同层神经元之间没有连接。一个呈现给网络输入的样例要被前向传播到它的输出，原则上，执行如下函数：

$$y_i=f\left(\sum_j w_{ji}^{(1)} f\left(\sum_k w_{kj}^{(2)} x_k\right)\right)$$

这里，$w_{ji}^{(1)}$和$w_{kj}^{(2)}$分别是输出层和隐含层神经元的权重，f是S形函数。

- 多层感知机的训练是通过一个被称为误差反向传播的技术实现的。对于每一个训练样例，这个技术首先确定每个神经元各自对于网络整个误差的责任，然后按照这些责任更新权重。

这里的那些责任是这样计算的：

输出层神经元：$\delta_i^{(1)}=y_i(1-y_i)(t_i-y_i)$。

隐含层神经元：$\delta_j^{(2)}=h_j(1-h_j)\sum_i \delta_i^{(1)} w_{ji}$。

这里的权重是这样更新的：

输出层：$w_{ji}^{(1)}=w_{ji}^{(1)}+\eta\,\delta_i^{(1)}\,h_j$。

隐含层：$w_{kj}^{(2)}=w_{kj}^{(2)}+\eta\,\delta_j^{(2)} x_k$。

- 这里讨论了误差反向传播的一些方面。这些方面当中，最重要的是计算代价，局部极小值的存在，自适应学习率，过拟合的危险和怎样决定隐层规模的问题。
- 一个替代物是径向基（RBF）网络。对于隐层神经元中的传递函数，这里使用了高斯函数。输出层神经元经常使用的是阶梯函数（原则上是线性分类器），或简单地输入一个线性函数。
- 在RBF网络中，每一个高斯中心都是一个训练样例。如果有太多样例的话，可以进行随机选择。在这个简单版本中，高斯标准差设为$\sigma^2=1$。
- RBF网络的输出层神经元可以用感知机学习来训练，而且仅有这一层的权重需要训练。

- 有时候，支持向量机的思想会被应用于隐层神经元的输出上。这个得到的工具被称为基于 RBF 的支持向量机。

历史简评。很著名地，闵斯基（Minsky）和帕波特[56]（Papert）提出的怀疑性的观点延迟了神经网络的研究。两位知名的作者所表达的悲观可能是为什么布里森（Bryson）和胡[11]（Ho）所提出的一个早期版本的神经网络训练方法被忽视的主要原因，沃布斯[82]（Werbos）的一个独立的成功尝试也享受到了同样的命运。只有拉莫哈特[70]（Rumelhart）和麦克克里兰德（McClelland）的开创性的书卷发表之后，算法才被独立地重新提出，人工神经网络领域才成为一个受人尊敬的学科。阿施[1]（Ash）提出了逐渐成长的多层感知机。布鲁姆汉德（Broomhead）和洛[10]（Lowe）首先在神经网络领域提出了径向基函数的思想。

5.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 回到表 5.3 中对误差反向传播的描述。仅使用一支笔、一张纸和一个计算器，重复一个稍微不同的训练样例的计算：$\boldsymbol{x}=(-1,\ -1)$，$t(\boldsymbol{x})=(1,\ 0)$。
2. 像前面的例子一样，手动模拟误差反向传播，重复以下两种情况的计算。
   1）高的输出层权重：$w_{11}^{(1)}=3.0$，$w_{12}^{(1)}=-3.0$，$w_{21}^{(1)}=3.0$，$w_{22}^{(1)}=3.0$。
   2）小的输出层权重：$w_{11}^{(1)}=0.3$，$w_{12}^{(1)}=-0.3$，$w_{21}^{(1)}=0.3$，$w_{22}^{(1)}=0.3$。
   观察每种情况下权重的相对变化。
3. 考虑一个包含10^5个样例，每个样例由 1000 个属性描述的训练集。通过10^5轮学习训练一个有 1000 个隐层神经元和 10 个输出神经元的多层感知机的计算代价是多少？

思考题

1. 怎样推广误差反向传播技术使其可以用于多于一个隐含层的多层感知机？
2. 5.1 节建议所有属性都应该标准化到区间 [-1.0, 1.0]。如果属性没有这样标准化，那么网络的分类和训练会受到怎样的影响？（提示：与 S 形函数有关）
3. 讨论径向基函数和多层感知机的分类过程的相似和不同之处。
4. 比较径向基函数和多层感知机的优势和劣势。

上机实验

1. 对提前定义的输出和隐层神经元个数，写一个程序实现误差反向传播，使用固定的学习率 η。
2. 将上机实验第 1 题中实现的程序应用于 UCI 库中的一些基准域（www. ics. uci. edu/ ~ mlearn/MLRepository. html）。用不同的 η 值进行训练，并观察它们如何影响收敛的速度。
3. 对于一个给定的数据集，用多层感知机中不同数量的隐层神经元进行实验，并观察它们是如何影响网络的学习能力的。
4. 再一次用不同个数的隐层神经元进行实验。这次，集中于计算代价。在网络收敛之前，需要多少轮的学习？并且检查错误率的演变情况。
5. 写一个程序，对于一个给定的训练集，它可以建立一个径向基函数网络。对于大的训练集，随机选择样本来定义高斯中心。
6. 将上机实验第 5 题中实现的程序应用于 UCI 库中的一些基准域。

机器学习导论

第 6 章　决策树

之前各章讨论的分类器希望所有属性值都同时给定。但是，这样的情境有它自身的缺点。一个想要掌握病人状况本质的内科医生，一开始时除了了解一些主观的症状，其他什么都没有掌握。因此，她开展实验测试，并且基于这些结果，可能还需要做其他测试，以缩小诊断的范围。那么，在任何时刻，医生仅考虑那些加入之后对她当前的信息和理解有意义的“属性”。从一开始就要求进行所有可能的（成千上万种的）实验测试是可笑的。

这里，得到的教训是，详尽的信息往往不是立即可得的，甚至可能是不需要的。根据情况所需每次选择一个属性，分类器可能就会做得很好。把这种情形作为目标的最常用的工具是决策树。本章讲述它的分类性能，然后介绍从数据中归纳学习一棵决策树的一项简单技术。读者还会学到如何从树的剪枝中获益和如何将归纳学习到的树转化为一些规则。

6.1 作为分类器的决策树

表 6.1 中所示的训练集包含 8 个样例，每个样例由 3 个属性描述，而且已被标记为正例或负例。为了简化对基本概念的解释，我们假设所有的属性都是离散的。随后，当相关原理变得清楚了，我们会很轻易地推广相应方法使其能够用于连续属性或连续与离散属性混合的域中。

表 6.1 由 3 个符号属性描述的、已被分类为正例和负例的 8 个训练样例

样例	crust **size** （外壳大小）	shape（形状）	filling **size** （馅料的量）	类别
e1	big	circle	small	**pos**
e2	small	circle	small	**pos**
e3	big	square	small	**neg**
e4	big	triangle	small	**neg**
e5	big	square	big	**pos**
e6	small	square	small	**neg**

（续）

样例	crust **size**（外壳大小）	shape（形状）	filling **size**（馅料的量）	类别
e7	small	square	big	**pos**
e8	big	circle	big	**pos**

决策树。图 6.1 中展示了一些示例的决策树，它们可以处理表 6.1 中的数据。内部结点代表了属性值测试，边指示了在多种测试结果的情况下如何继续进行，叶子包含了类别标签。一个要分类的样例首先输入到最顶端的结点，即根结点所

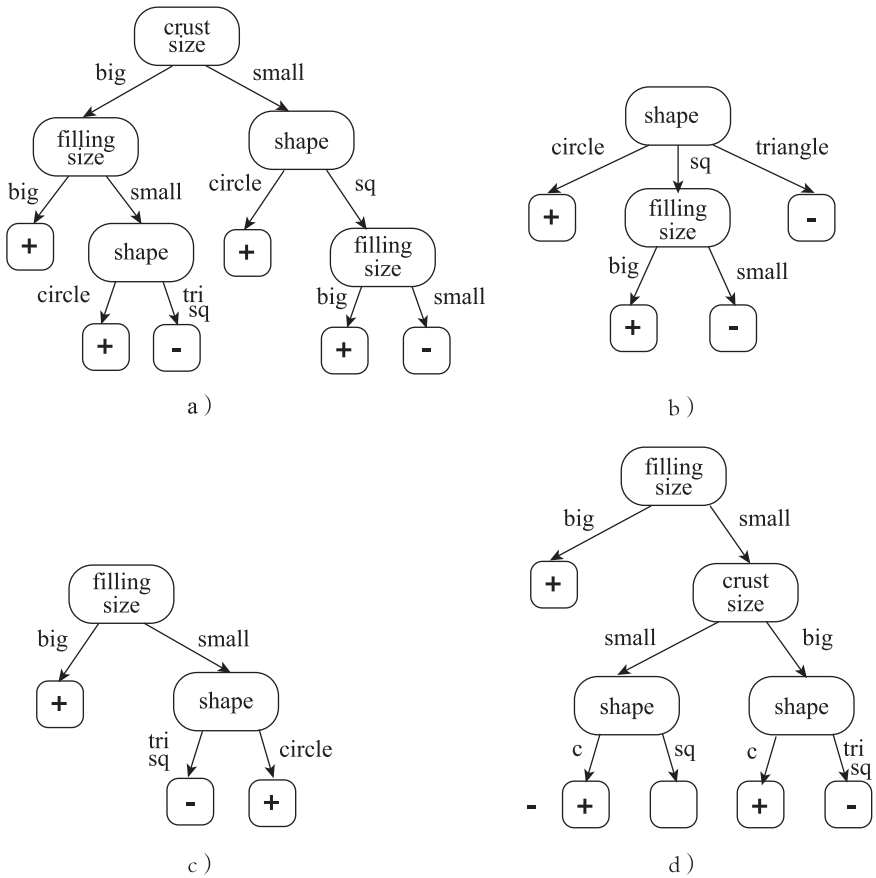


图 6.1　针对于“派”的域的示例决策树。注意它们在规模和测试顺序方面的差异。它们中的每一个都正确地分类了表 6.1 中列出的所有训练样例

规定的测试中。然后，这个测试的结果决定了这个样例要沿着哪个边向下传送，这个过程一直持续，直到到达某个叶子结点。之后样例就被标记为与这片叶子相关联的类别。

我们用图 6.1b 中的树来介绍这个过程。根结点考察形状（shape），它的 3 个值每个都用一条边来代表。在样例 $e1$、$e2$ 和 $e8$ 中，形状的值是圆形的（shape = circle），它对应于左侧的边。样例沿着这条边向下传送，结束于标记为正类（pos）的一个叶结点。这实际上就是这 3 个样例共同的类别。对 $e4$，形状是三角形的（shape = triangle），对应的边结束于一个标记为负类（neg）的叶结点——再一次地，正确的类别。对于样例 $e3$，$e5$，$e6$ 和 $e7$，这里形状是正方形的（shape = square），情况有一些复杂。对于这个特定的值，边没有结束于一个叶结点，而是一个含有测试的结点，它需要知道馅料的量（filling – size）的值。对 $e5$ 和 $e7$ 的情况，这个值为大的（big），它指向一个标记为正类的叶结点。另外两个样例，$e3$ 和 $e6$，这个值为小的（small），则指向了一个标记为负类的叶结点。

我们展示了图 6.1b 中的决策树可以为所有训练样例找到正确的类别。通过做一些练习，请读者来验证图 6.1 中的其他树也可以成功（从某种意义上，决策树可以看作一种简单的数据压缩机制）。

可解释性。对比这个分类器与之前介绍的分类器，我们可以看到一个明显的优势：可解释性。如果有人问为什么样例 e_1 被认为是正例，答案是“因为它的形状是圆形的”。其他分类器不会提供这种解释。特别地，神经网络是一个真正的黑盒子：当给定一个样例，它简单地返回一个类别而从不提供任何关于为什么这个特定类标要优先于其他类标的深刻见解。贝叶斯和线性分类器的情况也不会好很多。仅有近邻分类器提供了一种表面上的相当基本的讨论。例如，有人会说“x 应该被标记为正类是因为这是与 x 最接近的训练样例的类别”。但是，这样的陈述与决策树所做的准确的基于属性的解释相差甚远。

可以更进一步地解释决策树是一个规则的集合，例如，“如果形状是正方形的而且馅料的量是大的，那么这个样例属于正类”。一个检查这些规则的领域专家可能会决定这些规则是否直观上是吸引人的，以及它们是否与他或她对要处理的问题的“人类的理解”一致。专家甚至可能会关于这些树给出一些改进的建议，例

如，通过指出一些不正确的测试，这些不正确的测试存在于数据结构中仅仅是因为数据中的一些随机的规律性。

缺失的边。读者会记起，在线性分类器的情况，一个样例可能恰好位于类别分割超平面上，这时类别几乎是随机选定的。在决策树中，相似的事情偶尔也会发生。假设图6.1a中的树用来决定如下样例的类别：

(crust – size = small)　AND(shape = triangle)　AND(filling – size = small)

外壳的大小是小的，形状是三角形的，而且馅料的量是少的。

我们一步一步遵循学习过程进行。根结点需要知道外壳的大小（crust – size）。知道值是小的（small），分类将样例从右侧的边向下传送，到达对形状（shape）的测试。这里，只有两个结果看起来是可能的：圆形的（circle）或正方形的（square），但是没有三角形的（triangle）。原因是，创建这棵树的人并不知道一个外壳小（crust – size = small）的物体会是三角形的：训练集中就不存在这种东西。因此，看起来没有什么理由来创建相应的边。即使创建了这条边，也不清楚它应该指向哪里。

用计算机程序实现这个分类器的工程师必须确定程序“知道”在存在“缺失的边”的情况下该做什么。随机地选择类别或倾向于出现最频繁的类别是最显然的可能。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 描述用决策树来分类样例的机制。用图6.1中的决策树和表6.1中的训练样例描绘决策树分类的过程。
- “决策树对一个具体类别的选择是可以解释给用户的”，这个陈述是什么意思？前面章节讨论的分类器是否有相似的可能？
- 在什么情况下，决策树不能决定一个样例的类别？如果你是程序员，你会如何处理这种情况？

6.2 决策树的归纳学习

我们从一个非常粗糙的归纳学习算法开始。将它应用到一个训练集，我们将会得到很多的替代的决策树。一个简短的讨论会让我们相信，在这些决策树中，稍小的更受青睐。这个观察会促使产生一个改良版本的技术，并因此为后面的章节打下基础。

分而治之。我们试着手工创建一棵决策树。假设我们决定根结点要测试形状（shape）的值。在训练集中，可以找到 3 个不同的结果：圆形的（circle），三角形的（triangle）和正方形的（square）。对于每一个，分类器需要从根结点引出一条边。第一条边，定义为形状是圆形的（shape = circle），有样例$T_C = \{e1, e2, e8\}$ 满足；第二条边，定义为形状是三角形的（shape = triangle），有样例$T_T = \{e4\}$ 满足；最后一条边，定义为形状是正方形的（shape = square），有样例$T_S = \{e3, e5, e6, e7\}$ 满足。3 条边中每一条都开始于根结点终止于其他结点，可能是一个属性测试结点或包含类别标号的叶子结点。

观察发现，T_C 中的所有样例都属于正类，我们让这条边指向一个标记为正类（pos）的叶子结点。类似地，T_T中样例满足的边应该指向一个标记为负类（neg）的叶子结点。仅在最后一条边的情况会出现一定的困难，因为T_S由两类混合而成。为了分割开它们，我们需要将另一个测试，如馅料的量（filling - size），放置于边的末端。这个属性可以获得两个值：小的（small）和大的（big），它们将T_S划分为两个子集。其中，$T_{S-S} = \{e3, e6\}$ 由馅料的量是小的（filling - size = small）来刻画；另一个，$T_{S-B} = \{e5, e7\}$ 由馅料的量是大的（filling - size = big）来刻画。T_{S-S}中的所有样例都属于正类，而T_{S-B}中的所有样例都属于负类。这允许我们以叶结点来结束这两条边，并且前一个标记为正类（pos），后一个标记为负类（neg）。这时，构建树的过程可以终止了，因为每一个呈现给创建的分类器的训练样例都到达了一个叶结点。

读者会注意到，树的每个结点可以与一些样例相关联，这些样例或经过它或终结于它。开始于根结点，每一个测试将训练集划分为不相交的子集，然后这些

子集被进一步划分为更小的子集，这样一直下去，直到在一定意义上讲，每个子集都变成“纯的”，它的所有样例都属于同一类。这就是为什么这个方法有时候被称为分而治之的技术。

代替的树。在如上描述的过程中，形状（shape）和馅料的量（filling - size）的（相当任意的）选择导致了图 6.1b 所示的决策树。为了熟悉其中的机制，鼓励读者用其他的替代物，如将外壳大小（crust - size）或馅料的量（filling - size）用于根结点的测试和在稍低层考虑不同的测试的选择，来进行使用。这样将会创建出非常多的不同的决策树——图 6.1 中描绘了其中的一些。

在这样一个非常简单的玩具域中都能找到这么多解，是一个非常值得思考的问题。有没有一种办法可以决定哪棵树更好？这样，分而治之技术的一个改进版本应该通过设计而不是仅靠机会得到一棵“好的”树。

树的规模。图 6.1 中最小的数据结构包含两次属性测试，最大的包含 5 次。这种不同对分类器的性能可能会有很大的影响。然而，在继续讨论这个现象的不同方面之前，我们强调树中结点的个数不是其规模的唯一标准。测试的次数也很重要，当分类普通样例的时候必定要进行测试。

例如，在一个形状（shape）几乎总是圆形的（circle）或三角形的（triangle）域中（仅有很少的正方形的（square）），图 6.1b 的树执行的平均测试次数将仅仅稍微超过 1，因为形状为圆形的（shape = circle）和形状为三角形的（shape = triangle）都直接指向了带类别标号的叶子结点。但是如果形状普遍都是正方形的（square），则平均测试次数会接近 2。那么，经常地，稍大的树比稍小的树可能会需要更少次的测试。

小树相对于大树。关于为什么更倾向于小的决策树有很多原因。原因之一是可解释性。人类专家发现一个仅包含几次测试的决策树是容易分析和解释的，甚至是容易修正的。树越大，做这些越困难。

小的决策树的另一个优势是它们对处理不相关的和多余的信息的倾向。图 6.1a 中相对大的树使用了全部的 3 个属性，而图 6.1b 中稍小的一个在分类训练集方面表现一样好但没有考虑外壳的大小（crust - size）。这样的经济性在一些属性值获取代价高或花费时间多的领域是有用的。

最后，大的树倾向于过拟合训练样例。这是因为分而治之方法持续地将训练集划分为越来越小的子集，这些划分的次数等于树中属性测试的次数。最终，得到的训练子集变得太小，以至于类别的划分仅仅是利用在剩余的正例和负例中一个基于机会或噪声获得不同值的属性来进行的。

小的决策树的归纳学习。在描述手工构建树的过程中分而治之技术的性能时，我们随机地挑选属性。这样做的时候，我们观察到一些选择会比另一些导致更小的树。显然，属性之间在承载了多少信息上是不同的。例如，形状（shape）可以直接将一些样例标记为正例（如果值为圆形的（circle））或负例（如果值为三角形的（triangle））；但是外壳的大小（crust - size）却做不到，除非有其他属性的协助。

假如有一种度量每个属性提供的信息量的方法（实际上这样的机制是存在的，见6.3节），我们已经准备好将决策树的归纳学习技术通过伪代码形式化出来，读者可以在表6.2中看到它。

表6.2 决策树的归纳学习

令 T 是训练集。

grow（T）：

1. 找到属性 *at*，它关于类别标号贡献了最多的信息类。
2. 将 T 划分为子集 T_i，每一个用 *at* 的一个不同的值来刻画。
3. 对于每个 T_i：
   1）如果 T_i 中所有的样例都属于同一个类别，则创建一个标记为这个类别的叶子结点。
   2）否则，递归地应用同样的过程到每个训练子集上，即 *grow*（T_i）。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释为归纳学习决策树的分而治之技术的原理。
- 相比于稍大的决策树，小的决策树有什么优势?
- 什么决定了由分而治之技术得到的决策树的大小?

6.3 一个属性承载了多少信息

为了创建一棵相对小的决策树，分而治之技术依赖于一个关键的组成部分，即决定各个属性所承载的关于类别标号的信息有多少。这一节就介绍一种计算这个量的机制。

一个消息的信息含量。假设我们已知训练样例被标记为 pos 或 neg，这两类的相对频率分别为p_{pos}和p_{neg}（回忆一下，pos 的相对频率是标记为 pos 的样例（在训练集中）的百分比，它代表了一个随机抽取的样例属于正类的概率）。我们选择一个随机的训练样例。“这个样例的类别是 pos”，这个消息传递出了多少信息？

答案取决于p_{pos}。所有样例都已知为正例的极端的情况下，$p_{\text{pos}}=1$，这个消息并没有告诉我们任何新的信息。即使没有告诉这个消息，读者也已经知道这个样例是正例了。如果已知两类被相同地代表，即$p_{\text{pos}}=0.5$，情况就要改变了。这里，猜想并不比投掷一枚硬币效果好，因此这个消息提供了一些信息。如果已知大部分样例都是负例，如$p_{\text{pos}}=0.01$，那读者几乎可以确定选出来的样例也将会是负例；但告诉他并非如他所想的消息就会令其感到意外了。而且，p_{pos}的值越小，这个消息所提供的信息越多。

当量化这样一条消息的信息含量时，式（6.1）很方便：

$$I_{\text{pos}}=-\log_2{}^{\ominus}p_{\text{pos}} \tag{6.1}$$

这里的负号弥补了$p_{\text{pos}}\in(0,1)$的对数总是负数的事实。表 6.3 展示了一些典型的p_{pos}值的信息含量。注意，信息量的单位是比特（bit）。另一点评论：对数的底是 2，相当常见的是将其写为 $\log^{\ominus}p_{\text{pos}}$而不是更加严谨的$\log_2 p_{\text{pos}}$。

㊀ 本书原著中将 lbx 统写为 $\log_2 x$ 的形式，全书余同，故本书中将 x 的以 2 为底的对数统写为 $\log_2 x$。

㊁ 本书原著中将 lgx 缩写为 logx 的形式，全书余同，故本书中将 x 的以 10 为底的对数缩写为 logx。

表 6.3 消息“这个随机抽取的样例是正例”的信息含量的一些值（用 bit 度量）。注意，对$p_{pos}=0$，消息是不可能出现的

p_{pos}	$-\log_2 p_{pos}$
1.00	0bit
0.50	1bit
0.25	2bit
0.125	3bit

熵（平均信息含量）。一条消息的信息含量有这么多。然而，假设多次重复实验。两种消息都会出现，“样例是正例”和“样例是负例”，第一个有概率p_{pos}，第二个有概率p_{neg}。那这些消息平均的信息含量可以通过式（6.2）得到，这里每条消息的信息含量都由它的概率进行了加权（参数 T 指的是训练集）：

$$H(T)=-p_{pos}\log_2 p_{pos}-p_{neg}\log_2 p_{neg} \tag{6.2}$$

细心的读者会抗议说 0 的对数是没有定义的，如果$p_{pos}=0$ 或$p_{neg}=0$，则式（6.2）可能是没用的。幸运地，简单的分析（使用极限和洛必达法则）会告诉我们，随着 p 趋于 0，$p\log p$ 也趋于 0，这意味着 $0\cdot\log 0=0$。

$H(T)$ 被称为 T 的熵。当$p_{pos}=p_{neg}=0.5$ 时，它的值达到最大，$H(T)=1$（因为$-0.5\cdot\log 0.5-0.5\cdot\log 0.5=1$）；当$p_{pos}=1$ 或$p_{neg}=1$ 时，它的值降到最小，$H(T)=1$（因为 $0\cdot\log 0+1\cdot\log 1=0$）。顺便提一下，$p_{pos}=1$ 或$p_{neg}=1$ 的情况被视为完美的规律性，因为所有的样例都属于同一个类别；相反地，$p_{pos}=p_{neg}=0.5$ 的情况被视为完全缺乏规律性。

一个属性贡献的信息量。熵（规律性的缺失）的概念会帮助我们解决一个主要的问题：一个离散属性 at 的值的知识告诉我们多少关于样例类别的信息？

让我们提醒自己，at 将训练集 T 划分为子集T_i，每一个T_i 用一个不同的 at 值来刻画。相当自然地，每个子集会以其自己的两个类别（p_{ipos}和p_{ineg}）的概率（用相对频率估计的）为标志。基于这些知识，式（6.2）会给出相应的熵，即$H(T_i)$。

现在，令 $|T_i|$是T_i 中的样例数，$|T|$是整个训练集 T 中的样例数。一个随机抽取的训练样例在T_i 中的概率可以估计如下：

$$P_i = \frac{|T_i|}{|T|} \tag{6.3}$$

我们已经准备好计算子集的熵的加权平均：

$$H(T, at) = \sum {}_i P_i \cdot H(T_i) \tag{6.4}$$

得到的结果 $H(T, at)$，是一个系统的熵，这里每个训练样例的类别标号和 at 的值都是已知的。那么，at 贡献的信息量就是考虑它之前与之后熵的差值：

$$I(T, at) = H(T) - H(T, at) \tag{6.5}$$

数学家能容易地证明这个差值不会是负数的。考虑 at，信息只能获得而不会失去。然而，在某些非常稀少的情况下，如 $I(T, at) = 0$，这意味着没有信息获得。

将式（6.5）分别应用于各个属性，我们可以发现它们中的哪个提供了最多的信息量，并且让它作为表6.2所示算法的第1步中“根结点”测试的最佳选择。

表6.4中的伪代码总结了刚才描述的过程。这个过程开始于计算系统的熵，这里仅有类别的百分比是知道的。下一步，算法计算每个属性所承载的信息增益。提供了最高信息增益的属性被认为是最佳的。

表6.4 找到最有信息量的属性的算法

1. 使用正例和负例的百分比，p_{pos} 和 p_{neg}，计算训练集 T 的熵：$H(T) = -p_{\text{pos}} \log_2 p_{\text{pos}} - p_{\text{neg}} \log_2 p_{\text{neg}}$。
2. 对于每个将 T 以相对大小 P_i 划分为子集 T_i 的属性 at，执行以下步骤：
   1）计算每个子集 T_i 的熵。
   2）计算平均熵：$H(T, at) = \sum {}_i P_i \cdot H(T_i)$。
   3）计算信息增益：$I(T, at) = H(T) - H(T, at)$。
3. 选择有最高信息增益的属性。

图示。表6.5展示了在表6.1的域中怎样使用这个机制选择最有信息量的属性。首先，建立没有属性的系统的熵 $H(T)$。然后，我们观察到，例如，形状（shape）属性会将训练集划分为3个子集。计算它们的熵的平均值 $H(T, \text{shape})$，以及 $H(T)$ 与 $H(T, \text{shape})$ 之间的差值，给出了这个属性所承载的信息量。对外壳的大小（crust – size）和馅料的量（filling – size）重复这个过程，并比较得到的

结果，我们会发现形状（shape）比其他两个属性贡献了更多的信息，这就是为什么我们为根结点的测试选择形状（shape）属性。

顺便说一下，这就是图 6.1b 中的决策树是怎么得到的。

表 6.5　图示寻找最有信息量的属性

样例	crust **size**（外壳大小）	shape（形状）	filling **size**（馅料的量）	类别
*e*1	big	circle	small	**pos**
*e*2	small	circle	small	**pos**
*e*3	big	square	small	**neg**
*e*4	big	triangle	small	**neg**
*e*5	big	square	big	**pos**
*e*6	small	square	small	**neg**
*e*7	small	square	big	**pos**
*e*8	big	circle	big	**pos**

这是训练集的熵，这里只有类别标签是已知的：

$$H(T) = -p_{\text{pos}}\log_2 p_{\text{pos}} - p_{\text{neg}}\log_2 p_{\text{neg}} = -\left(\frac{5}{8}\right)\log\left(\frac{5}{8}\right) - \left(\frac{3}{8}\right)\log\left(\frac{3}{8}\right) = 0.954$$

下一步，我们计算由形状（shape）的值定义的子集的熵：

$$H(\text{shape} = \text{square}) = -\left(\frac{2}{4}\right)\log\left(\frac{2}{4}\right) - \left(\frac{2}{4}\right)\log\left(\frac{2}{4}\right) = 1$$

$$H(\text{shape} = \text{circle}) = -\left(\frac{3}{3}\right)\log\left(\frac{3}{3}\right) - \left(\frac{0}{3}\right)\log\left(\frac{0}{3}\right) = 0$$

$$H(\text{shape} = \text{triangle}) = -\left(\frac{0}{1}\right)\log\left(\frac{0}{1}\right) - \left(\frac{1}{1}\right)\log\left(\frac{1}{1}\right) = 0$$

从这些值，我们可以得到系统的平均熵，这里类别标签和形状（shape）的值是知道的：

$$H(T, \text{shape}) = \left(\frac{4}{8}\right)\cdot 1 + \left(\frac{3}{8}\right)\cdot 0 + \left(\frac{1}{8}\right)\cdot 0 = 0.5$$

为其他两个属性重复相同的过程，我们得到如下结果：

$$H(T, \text{crust} - \text{size}) = 0.951$$

$$H(T, \text{filling} - \text{size}) = 0.607$$

这些值给出了如下的信息增益：

$$I(T, \text{shape}) = H(T) - H(T, \text{shape}) = 0.954 - 0.5 = 0.454$$

（续）

$I(T,$ crust – size$)$	$=H(T)-H(T,$ crust – size$)$	$=0.954-0.951=0.003$
$I(T,$ filling – size$)$	$=H(T)-H(T,$ filling – size$)$	$=0.954-0.607=0.347$
我们断定形状（shape）属性贡献了最大的信息量。		

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 我们谈论“一个消息所承载的信息量”时，我们的意思是什么？这个量是怎样确定的，用什么单位？
- 什么是熵？它是怎样与训练集中的正例和负例出现的频率关联起来的？
- 当评估一个属性所贡献的信息量时，我们怎样使用熵？

6.4 数值属性的二元划分

前一节中的基于熵的机制要求所有的属性都是离散的。但是，通过一些小的修改，同样的方法也可以应用到连续属性上。我们需要做的只是将它们转换为布尔属性。

将一个连续属性转换为布尔型的。我们用 x 来标记连续属性。技巧是选择一个阈值 θ，然后确定如果 $x<\theta$，则新创建的布尔属性的值为 true，否则为 false（反之亦然）。

足够简单。但是，具体选择什么样的 θ 值呢？毫无疑问，有很多可选。这里是一个可能性。

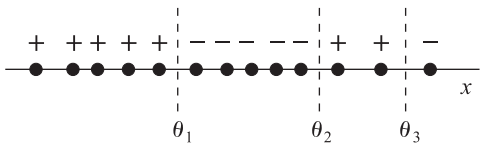
假设 N 个训练样例的每一个中 x 都有不同的值。我们将这些值按升序排列，标记最小的为 x_1，最大的为 x_N。任何一对相邻的值，x_i 和 x_{i+1}，都定义了一个阈值，$\theta_i=(x_i+x_{i+1})/2$。例如，在一个有 4 个样例的训练集中，x 有值 3、4、6 和 9，我们会考虑$\theta_1=3.5$、$\theta_2=5$ 和$\theta_3=7.5$。对于这 $N-1$ 个阈值中的每一个，我们计算所定义的布尔属性贡献的信息量，然后选择信息增益最大的那个阈值。

候选阈值。刚才描述的方法因其过高的计算代价会受到批评。实际上，在一个包含十万个样本，每个样本由 100 个属性描述（并不特别）的域中，有$10^5 \times 10^2 = 10^7$个不同阈值的信息含量要被计算。幸运的是，数学家曾证明这些阈值中的大部分都可以被忽略。这将计算代价缩到了很小的一部分。

表 6.6 描述了其中的原理。在上边的部分，x 的 13 个值从左到右依次排序，每一个都被标记为有相应值的训练样例的类别（正类或负类）。这里的规则是：最佳阈值总不会是在标记为相同类别的值之间。这意味着仅考察不同类别标号的值之间的位置所贡献的信息就足够了。在表 6.6 所示的特定的例子中，仅有 3 个候选阈值，即θ_1、θ_2 和θ_3，需要考察（其中θ_1 是最好的）。

表 6.6　图示寻找最佳阈值

属性 x 的值按照升序排列。候选阈值位于标记为不同类别标号的值之间。



这里是忽略属性值之后训练集的熵：

$$H(T) = -p_+ \log p_+ - p_- \log p_- = -\left(\frac{7}{13}\right)\log\left(\frac{7}{13}\right) - \left(\frac{6}{13}\right)\log\left(\frac{6}{13}\right) = 0.9957$$

这里是由 3 个候选阈值定义的训练子集的熵：

$$H(x < \theta_1) = -\left(\frac{5}{5}\right)\log\left(\frac{5}{5}\right) - \left(\frac{0}{5}\right)\log\left(\frac{0}{5}\right) = 0$$

$$H(x > \theta_1) = -\left(\frac{2}{8}\right)\log\left(\frac{2}{8}\right) - \left(\frac{6}{8}\right)\log\left(\frac{6}{8}\right) = 0.8113$$

$$H(x < \theta_2) = -\left(\frac{5}{10}\right)\log\left(\frac{5}{10}\right) - \left(\frac{5}{10}\right)\log\left(\frac{5}{10}\right) = 1$$

$$H(x > \theta_2) = -\left(\frac{2}{3}\right)\log\left(\frac{2}{3}\right) - \left(\frac{1}{3}\right)\log\left(\frac{1}{3}\right) = 0.9183$$

$$H(x < \theta_3) = -\left(\frac{7}{12}\right)\log\left(\frac{7}{12}\right) - \left(\frac{5}{12}\right)\log\left(\frac{5}{12}\right) = 0.9799$$

$$H(x > \theta_3) = -\left(\frac{0}{1}\right)\log\left(\frac{0}{1}\right) - \left(\frac{1}{1}\right)\log\left(\frac{1}{1}\right) = 0$$

与各个阈值相关联的平均熵为：

$$H(T, \theta_1) = \left(\frac{5}{13}\right) \cdot 0 + \left(\frac{8}{13}\right) \cdot 0.8113 = 0.4993$$

$$H(T, \theta_2) = \left(\frac{8}{13}\right) \cdot 1 + \left(\frac{8}{13}\right) \cdot 0.9183 = 0.9811$$

（续）

$H(T, \theta_3) = \left(\frac{8}{13}\right) \cdot 0.9799 + \left(\frac{8}{13}\right) \cdot 0 = 0.9045$

各个候选阈值所带来的信息增益为：

$I(T, \theta_1) = H(T) - H(T, \theta_1) = 0.9957 - 0.4993 = 0.4964$

$I(T, \theta_2) = H(T) - H(T, \theta_2) = 0.9957 - 0.9811 = 0.0146$

$I(T, \theta_3) = H(T) - H(T, \theta_3) = 0.9957 - 0.9045 = 0.0912$

阈值θ_1给出了最高的信息增益。

一棵数值决策树的根结点。表6.7的伪代码总结的算法为所有属性都连续的一个域中的一棵决策树的根结点决定了最佳的属性测试。注意，这个测试包含一个数值对［at_i，θ_{ij}］，这里at_i是所选择的属性，θ_{ij}是为这个属性找到的最佳阈值。如果一个样例的第i个属性值小于这个阈值，即$at_i < \theta_{ij}$，就到决策树的左边分支中；否则，选择右边分支。

表6.7　找到最优数值属性测试的算法

1. 对于每个属性at_i：
   1）基于at_i的值对训练样例排序。
   2）决定候选阈值θ_{ij}，选择那些位于有不同标号的样例之间的阈值。
   3）对于每个θ_{ij}，决定所创建的布尔属性贡献的信息量。
2. 选择提供了最高信息增益的对［at_i，θ_{ij}］。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释为什么本节建议将一个连续属性的域划分为两个子域。
- 什么数学发现减少了需要考察的阈值个数？

6.5　剪枝

6.2节称赞了小的决策树的好处——可解释性，移除了不相关的和多余的属

性，降低了过拟合的危险。这些争论促使了信息论在归纳学习决策树的过程中的实验；它们也促使了下面的步骤——剪枝。

本质。图 6.2 会帮助我们解释其中的原理。左边是原始的决策树，它的 6 个属性测试分别命名为t_1，…，t_6。右边是剪枝后的版本。注意，原树中以测试结点t_3为根的子树在剪枝后的树中被替换为了一个标记为负类的叶子结点；以测试结点t_6为根的子树被替换为了一个标记正类的叶子结点。读者可以看到这一点：剪枝就是将一棵或多棵子树用叶子替换，每一个标记为在原来分类器中到达被移除子树的训练样例中最普遍的类别。

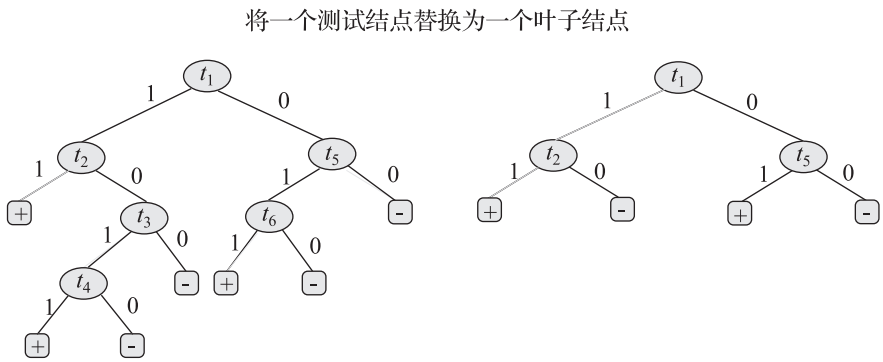


图 6.2　一个简单的剪枝方法会将一棵子树替换为一片叶子

一个纯粹主义者可能会抗议：归纳学习的机制是要创建一棵在训练样例上得到 0 错误率的决策树，但是在剪枝的树中这种完美可能就丢失了！但是实用思想的工程师却并不惊慌。最终的目标又不是分类训练样例（毕竟他们的类别已经知道了）。准确地说，我们需要的是一个可以标记未来样例的工具。而且经验说明合理的剪枝通常会改进分类性能。

误差估计。剪枝通常由一系列步骤来实现：首先将一棵子树替换为一片叶子，然后另一棵，如此下去，只要按照一些合理的准则这种替换看起来是有利的。“有利”这个词意味着，警示我们小的树的优势不应该比降低分类性能更重要。

这给我们带来了误差估计的问题。图 6.2 描述了它的原理。令 m 是到达左侧决策树的测试结点t_3的训练样例数。如果我们用一个叶子结点来替换以t_3为根的子树（正如在右侧的树中发生的），则这 m 个样例中会有一些可能被错误分类。

用 e 来记这些错误分类样例的个数，我们试着通过相对频率 e/m 来估计一个样例在这个叶子上被错误分类的概率。但是，承认小的 m 值可能导致有问题的估计，我们更倾向于式（6.6），这里 N 是训练样例的总数：

$$E_{\text{estimate}} = \frac{e+1}{N+m} \tag{6.6}$$

细心的读者可能会想起（或重读）2.3 节说过的稀有事件的概率估计的困难。

对整棵树的误差估计。我们再次回到图 6.2。左边的树有两棵子树，一棵以 t_2 为根结点，另一棵以 t_5 为根结点。令 m_2 和 m_5 分别是到达 t_2 和 t_5 的训练样例数；令 E_2 和 E_5 是由式（6.6）得到的两棵子树的误差估计。对于总共的 $N = m_2 + m_5$ 个训练样例，整棵树的错误率由两棵子树的加权平均估计得到：

$$E_R = \frac{m_2}{N}E_2 + \frac{m_5}{N}E_5 \tag{6.7}$$

当然，在多于两棵子树的情况下，加权和要在所有子树上边计算。这应该没什么大的困难。

对于 E_2 和 E_5 的值，它们来自于特定子树的错误率，而这些也是来自于子树的子树的错误率，如此下去，沿着所有路径向下直到最底层的测试。误差估计过程是一个递归的工作。

假设要剪枝的树是以 t_3 为根结点的那个，它恰好是测试结点 t_2 的两个子结点之一。对 t_2 的误差估计可以通过 E_3 和 t_2 的另一个子结点（标记为正类的叶结点）的误差估计的加权平均来计算。然后，得到的估计会与上面所示的 E_5 进行组合。

后剪枝。这个术语是指决策树在它完全从数据中归纳学习到之后再进行剪枝的情况（另一个方法是下一部分介绍的主题）。我们已经知道，本质上就是将一棵子树用一片标记为到达这片叶子的训练样例中出现最多的类别的叶子来代替。因为经常有几棵（或许多）子树可以被代替，所以必须做出一个选择；而一个选择的存在意味着我们需要一个准则来引导我们的决定。

这里是一种可能性。我们知道，剪枝有可能会改变分类器的性能。一种评估这种改变的方法就是比较剪枝之后和之前决策树的误差估计：

$$D = E_{\text{after}} - E_{\text{before}} \tag{6.8}$$

我们从可用的剪枝备选方案中选择差值$D_{\min}$最小的那个，但是仅当$D_{\min} < c$时才进行剪枝，这里c是一个用户设定的阈值，它说明为了换取树的紧致性有多少性能下降是可以忍受的。然后，重复这个机制，决策树随之变得越来越小，终止条件也随之通过常数c得到加强。这样，在图6.2中，第一个剪枝步骤可能会移除以t_3为根结点的子树。第二步，以t_6为根结点的子树。这里，过程终止，因为任何进一步地尝试剪枝都会导致产生误差估计增长太大的一棵树，最后的（剪枝后的）树的误差估计与图6.2左侧原来的树的误差估计的差值超过了用户的阈值，即$D > c$。

表6.8中的伪代码总结了这个准则。

表6.8　决策树剪枝算法

c——一个用户设定的常数：

1. 估计原始决策树的错误率，记为E_{before}。
2. 估计通过剪枝原始树的各种备选方法得到树的错误率。
3. 找到剪枝之后估计的错误率，增加最小的剪枝，即找到$D_{\min} = E_{\text{after}} - E_{\text{before}}$，仅当$D_{\min} < c$时，选择这个剪枝。
4. 只要$E_{\text{after}} - E_{\text{before}} < c$，重复步骤2和3。

在线剪枝。在分而治之过程中，每一个后续属性都将训练样例集划分成越来越小的子集。不可避免地，在树的较低层支持测试的选择的证据是微弱的。例如，当仅有两个训练样例（一个正例和一个负例）到达一个树结点时，一个完全无关的属性可能仅靠巧合成功将正例和负例分割开。从添加这个测试到决策树所得到的“好处”仅仅是对训练集的过拟合。

在线剪枝的动机就是要确定防止这种情况发生。这里的准则是：如果训练子集小于一个用户设定的最小值m，则停止树的进一步扩展。

剪枝的影响。剪枝到什么时候由两个参数控制：后剪枝中的c和在线剪枝中的m。这两种情况下，都是参数的值越大树越小。

剪枝会增强决策树在未来样例上的分类性能，主要原因是低层测试有较差的统计支持，移除这些低层测试通常会减小过拟合的危险。但是，这仅在一定程度上适用。如果过分地剪枝（极端情况下），则会导致决策树被一个标记为多少类

的叶子所代替。这样的分类器是不会有用的。

图 6.3 展示了逐渐增加的剪枝对分类性能的影响。横轴画出了由 c 或 m 或它们两者控制的剪枝的程度；纵轴代表了在训练集和一个构造的测试集上测得的错误率（后者包含了在学习中没有用到，但是其类别标号已知的样例）。

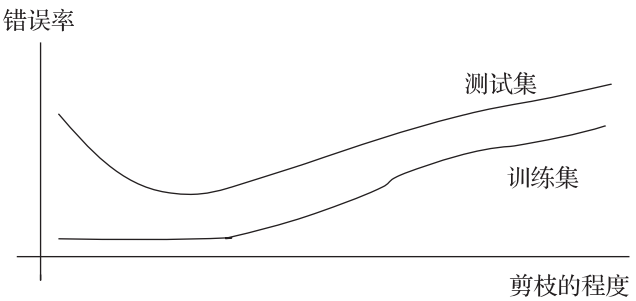


图 6.3　随着剪枝程度的增加，测试集上的错误率一般是先降低然后再开始上升；训练集上的错误率一般是单调地增加

在训练集的曲线上，当没有剪枝的时候错误率最低。但是，更加有趣的是测试集的曲线。它的形状告诉我们，没有剪枝的树在测试数据上往往得分很低，这可以通过没有剪枝的树趋向于过拟合训练集得到解释，这种现象可以通过增加剪枝的程度来减少。然而，过多的剪枝会将确实承载了有用信息的属性测试移除，而这对于分类性能有着不利的影响。

顺便说一下，关于潜在的数据，这两条曲线可以告诉我们很多信息。在一些应用中，即使很小的剪枝都会损害测试数据上的错误率，例如，一个没有噪声的域中的一个相对较小的训练集。

另一个需要注意的事情是，测试集上的错误率几乎总是大于训练集上的错误率。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 决策树剪枝有哪些潜在的好处？

- 我们如何估计树在未来数据上的错误率？写出公式并解释如何使用它。
- 描述后剪枝和在线剪枝的原理。
- 什么参数控制了剪枝的程度？它们如何影响在训练集上的错误率？如何影响在测试集上的错误率？

6.6 将决策树转换为规则

相比于其他分类器，决策树的一个优势就是它的可解释性。任何序列的沿着从根结点到一片叶子的路径的测试都代表了一个 if – then 规则，而且这个规则解释了为什么分类器将一个给定的样例标记为这个或那个类别。

由决策树产生的规则。读者会发现可以很容易地将一棵决策树转换为一个规则的集合。注意怎样通过一系列边到达一片叶子就足够了，这里边的特定选择决定于沿着这条路径遇到的属性测试的结果。因此，每片叶子都与一个具体的测试结果的发生相关联。

为了清楚地说明，我们写出对应于图 6.1a 中决策树得到正类（pos）的完整的规则集合：

1. 如果外壳大小是大的（crust – size = big）且馅料的量是大的（filling – size = big），那么属于正类（pos）。

2. 如果外壳大小是大的（crust – size = big），馅料的量是小的（filling – size = big）且形状是圆形的（shape = circle），那么属于正类（pos）。

3. 如果外壳大小是小的（crust – size = small）且形状是圆形的（shape = circle），那么属于正类（pos）。

4. 如果外壳大小是小的（crust – size = small），形状是正方形的或三角形的（shape = circle OR triangle）且馅料的量是大的（filling – size = big），那么属于正类（pos）。

5. 其他属于负类（neg）。

注意最后一行的默认类：负类（neg）。如果所有规则都失败了，则一个样例就会被标注为默认类。这意味着规则的 if 部分的值为假（false）。我们注意到，在

二类域，我们仅需要考虑导致正类（pos）的规则，其他类就是默认选项。我们也可以用另外一个方式来做，仅考虑负类的规则，使得正类（pos）成为默认类。这实际上更加经济，因为仅有两片叶子被标注为负类，因此仅有两条对应的规则。鼓励读者通过简单的练习写出这两条规则。

无论如何，教训是清楚的：有 K 个类的一个域中，仅需要 $K-1$ 个类的规则，最后一个类作为默认类即可。

修剪规则。之前描述的树的后剪枝机制将一棵子树代替为一片叶子。这意味着稍低层的测试是先被剪枝的，剪枝不会让稍高层结点的移除早于它下边的结点的移除。在线剪枝也存在类似的情况。

然而，一旦树被转换为规则，剪枝就获得了灵活性：任何规则的 if 部分的任何测试都是要移除的潜在候选项；整个规则都可以被移除。通过表 6.9 中的伪代码所总结的规则修剪算法可以实现这些，表 6.10 通过一个例子描述了规则修剪算法。这里，初始的规则集由图 6.2 左边部分的（现在已经熟悉了的）树得到。第一个修剪步骤移去了那些看起来对整体分类性能贡献不多的测试；下一步删除了最弱的规则。

但是，我们得赶紧承认，这加入的灵活性的代价就是计算成本的显著增加。

表 6.9　规则修剪算法

将决策树重写为一个规则集。
令 c 是一个用户设定的用来控制修剪程度的常数。
1. 在每条规则中，计算移除各个测试所带来的误差估计的增加。
2. 选择出误差估计增加（$D_{\min}$）最小的移除。仅当$D_{\min} < c$ 时，移除测试。
3. 在规则集中，搜索要移除的最弱规则。
4. 选择出默认类别。
5. 按照规则的强度（覆盖多少训练样例）对其进行排序。

表 6.10　图示规则修剪算法

假设图 6.2 左边部分的决策树已经被转换为了如下的规则集（当所有规则的 if 部分都为假（false），neg 是所用的默认标号）。

（续）

$t_1 \wedge t_2$	$\rightarrow$pos
$t_1 \wedge \neg t_2 \wedge t_3 \wedge t_4$	$\rightarrow$pos
$\neg t_1 \wedge t_5 \wedge t_6$	$\rightarrow$pos
其他 neg	

假设对规则中测试的评估得出的结论是第二个规则的t_3和第三个规则中的t_5的移除不会较大地增加误差估计。我们得到如下的修改的规则集。

$t_1 \wedge t_2$	$\rightarrow$pos
$t_1 \wedge \neg t_2 \wedge t_4$	$\rightarrow$pos
$\neg t_1 \wedge t_6$	$\rightarrow$pos
其他 neg	

下一步揭示出第二个（已经修改的）规则可以移除而不会有较大的误差估计增加。移除之后，规则集如下。

$t_1 \wedge t_2$	$\rightarrow$pos
$\neg t_1 \wedge t_6$	$\rightarrow$pos
其他 neg	

至此完成了修剪。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释将决策树转换为一个规则集的机制。会得到多少条规则？这种转换背后的动机是什么？
- 默认类这个词是什么意思？怎样将它选出来？
- 讨论规则修剪的可能性。从什么意义上，我们可以声称规则修剪比决策树剪枝提供了更大的灵活性？为了这增加的灵活性需要付出什么代价？

6.7 总结和历史简评

- 在决策树中，每次对一个属性值进行测试，每次测试的结果指示出了下一步应该发生什么：或者是另一个属性测试，或者如果到达了一片叶子，就决定类别标号。可以说决策树包含了一些部分有序的测试的集合，每个序列的测试定义了以一片叶子终结的树枝。
- 从一个一般的训练集，可以创建出许多备用的决策树。作为一个准则，稍小的树更受青睐，它们的主要优势有可解释性、移除不相关的和多余的属性、较低的过拟合带噪声的训练数据的危险。
- 从数据中归纳学习决策树的最经典程序以递归的方式进行，总是寻找关于类别标号承载了最大信息量的属性。这个方法趋向于使得归纳学习到的决策树更小。“最好的”属性是利用信息论中的简单公式找到的。
- 决策树归纳学习的一个重要方面是剪枝。主要的动机是为了确保所有的树枝都由充分的证据所支持。进一步地，剪枝降低了树的规模，这是有一定好处的（参见上文）。存在两种通用类型的剪枝方法：①后剪枝，首先构建一棵完整的树，然后进行剪枝；②在线剪枝（它可能有一点误称），一旦用来决定下一个属性测试的训练子集变得太小了就停止树的构建。在这两种情况下，修剪的程度都是由用户设定的参数（分别记为 c 和 m）来控制的。
- 一棵决策树可以被转换为一个规则集，这个规则集可以被进一步修剪。在一个有 K 类的域中，指定 $K-1$ 个类的规则就足够了，剩下的类会变成默认类。规则往往更容易解释。规则修剪算法有时会导致更紧致的分类器，尽管会显著地增加计算代价。

历史简评。决策树的思想首先由胡沃兰德（Hoveland）和哈德（Hund）于20世纪50年代晚期提出。他们的工作随后在哈特（Hunt）、马林（Marin）和斯通[34]（Stone）的书中得到了总结，这本书报道了若干次实施作者的概念学习系统（Concept Learning System，CLS）的经验。弗里德曼（Friedman）、本特雷

(Bentley) 和芬克尔[24] (Finkel) 独立地开发了一个类似的方法。布雷曼[9] (Breiman) 等使得相关研究达到了一个早期的制高点，他们的研究描述了 CART 系统。昆兰[63,64] (Quinlan) 将这个思想引进了机器学习领域。可能昆兰[65] (Quinlan) 最著名的实现是 C4.5。本章就是基于 C4.5 的一个简化的版本。

6.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 图 6.4 中，有 8 个由两个属性：大小 (size) 和颜色 (color)，描述的训练样例，类别标号是材料：或者是木制的 (wood)，或者是塑料的 (plastic)。
   1) 当仅考虑类别标号 (忽略属性值) 的时候，训练集的熵是什么？
   2) 使用 6.3 节中的基于信息的机制，决定两个属性中的哪个更好地预言了类别。

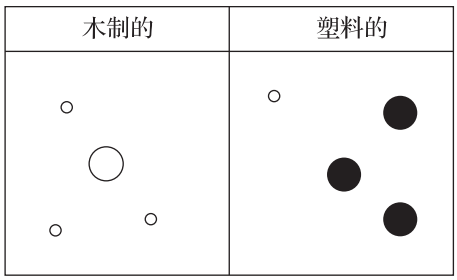


图 6.4 使用代表木制的和塑料的圆圈的另一个例子

2. 移除图 6.1a 中决策树右下角的关于馅料的量 (filling – size) 的测试。基于表 6.1 中的训练集，这步“剪枝”前后的误差估计分别是什么样的？

3. 选择图6.1中的一棵决策树并将其转换为一个规则集。选取其中的一条规则，并决定移除哪个测试估计的误差会有最小的增加。
4. 考虑10个训练样例的一个集合。假设一个连续属性有如下值：3.6，3.2，1.2，4.0，0.8，1.2，2.8，2.4，2.2，1.0。假设这些样例中的前5个和最后一个是正例，其他的样例都是负例。这个属性值范围的最优二值划分是什么？

思考题

1. 评估决策树所用的基准性能准则是错误率和树的规模（结点数）。但是，在某些领域，这些可能并不合适。指出或者决策树的规模或者它的错误率可能并不重要的领域。提示，考虑错误决策的代价和获得属性值的代价。
2. 一个领域的什么可能的特点会使得决策树的表现明显好于作为基准的最近邻分类器？提示，考虑噪声、不相关属性或训练集的大小等特点；然后自己决定它们中的每一个会对分类器的表现可能有怎样的影响。
3. 在什么样的数据上，线性分类器可能会比决策树做得更好？给出至少两个刻画这种属性的特征。可以基于前一个问题中的一些提示。
4. 找到了前面两个问题的答案，你应该可以得出逻辑上的结论：将决策树归纳学习和线性分类器归纳学习应用于给定数据，它们的表现会暴露出数据的什么特征？
5. 这一章描述的决策树关于给定样例的类别仅能给出“脆弱的”是或不是（yes-or-no）的决定（从这个意义上，有人会争辩，贝叶斯分类器或多层感知机是更加灵活的）。利用缓解这个弱点的方式，建议一种机制来修改决策树框架，使其对于每个样例不仅给出类别标号，而且给出分类器对这个类别标号的置信度。

上机实验

1. 实现决策树归纳学习的基准算法，并在UCI库（www.ics.uci.edu/~mlearn/MLRepository.html）中一些选择的域上测试它的性能。对比所得到

的结果与 k 近邻分类器得到的结果。

2. 实现本章描述的一个简单的剪枝机制。从 UCI 库中选择一个数据文件并运行，使用并观察剪枝的不同程度怎样影响训练和测试集上的错误率。
3. 从 UCI 库中选择一个充分大的域，保留 30% 的样例用来测试，分别用剩下的 10%，…，70% 的样例做训练。画出一张图，横轴给出了样例数，纵轴给出了归纳学习所花费的计算时间。画出另一张图，纵轴给出在测试集上的错误率。讨论这些得到的结果。

第 7 章　计算学习理论

众所周知，对于学习而言一个完备的理论是极其重要的。事实上可学习性强的数学模型有助于我们更好地理解如何根据数据选择好的分类器。相反地，对于那些学习能力较弱的模型，得到的结果往往是令人失望的。尽管本书一直想避免过度数学化，但是我们必须介绍一下计算学习理论的基本概念。

计算学习理论的核心是 PAC 学习，PAC 学习的提出使得量化学习能力成为可能。本章的第一节将介绍 PAC 学习在离散化属性数据域中的应用，并推导出数据集规模与分类器错误率的简单关系。紧接着的两节，将简单地讨论一下相应的公式。对于属性值连续的数据域，我们将介绍 VC 维的概念。

7.1 PAC 学习

对于机器学习，计算学习理论中最具有价值可能便是“可能近似学习”，有时也简称 PAC 学习。首先，让我们来解释下相关的基本原则，然后对公式进行推导，总结一些实践建议。

假设与定义。为了简化分析过程，一些假设条件是非常有必要的。首先，训练样本以及测试样本都是完全无噪声数据；其次，所有属性都是离散的；然后，以属性值的逻辑表达式构建分类器，正样本的逻辑值为真，负样本为假；最后，至少存在一个逻辑表达式能够对所有样本进行正确分类[㊀]。

每个逻辑表达式，都可以看作对分类器的一个假设。所有的假设构成一个假设空间 H，假设空间的大小（不同的假设的数量）以 $|H|$ 表示。假如对假设空间没有特别声明，我们默认 $|H|$ 为有限值。

不够精确的分类器仍可能在训练样本中取得成功。训练样本一般很少容括类别所有的细微之处。尽管一个分类器在训练样本中能够正确预测所有类标，但仍然可能在测试阶段取得很差的预测结果。随着我们增加训练样本，测试阶段预测出错的频率一般也随之降低，这是因为这些增加的样本很可能反映了原始样本缺

㊀ 读者注意到第 1 章介绍的“派”数据域都满足这些要求。

少的样本分布。在机器学习领域，有一条简单的经验法则：训练样本越多，学习到的分类器分类效果越好。

在测试阶段成功地预测样本的类标需要多少训练样本呢？为了找到答案，我们首先考虑这样一个分类器，其中该分类器在整个样本空间的错误率大于预定义的阈值。换句话说，分类器能够正确预测一个随机样本的概率小于 $1-\epsilon$。现在考虑有 m 个随机样本的情况，此时分类器能正确预测这 m 个随机样本的概率满足如下不等式：

$$p \leqslant (1-\epsilon)^m \tag{7.1}$$

该不等式意味着：一个包含 m 个样本的训练集够被一个错误率大于 ϵ 的分类器完全分类正确的概率是 $p \leqslant (1-\epsilon)^m$。当然在实际情况中，这个概率是很低的。例如，如果 $\epsilon = 0.1$，$m = 20$，此时概率 $p < 0.122$。假如我们增加训练样本数到 100（保持分类器的错误率下界 $\epsilon = 0.1$ 不变），概率 p 减少至 10^{-4} 以下。这确实已经很小，但是小概率并不等价于不可能。

去除弱分类器。假定错误率大于 ϵ 的分类器被认为是不可接受的。那么这些从训练集推导出性能较差的分类器，能够对整个训练集进行正确分类的机会有多大呢？

对于包含 $|H|$ 个分类器的假设空间 H，现在我们考虑在 m 个训练样本中评估每个分类器的理论可能性，并且保留那些没有分类出错的分类器。在这些“幸存者”中，令人遗憾的是，有一些分类器在训练样本中是零错误率，然而在整个样本空间中错误率实际上超过了 ϵ。假设这种分类器的个数为 k。

如何确定 k 的值？我们可以在整个实例空间依次测试每个分类器。然而在实际过程中，这很不现实。我们唯一能确定的 $k \leqslant |H|$，幸运的是，在实际中 $|H|$ 或已知或者能够计算出来[㊀]。

这里，我们重新定义概率 P 为在这 k 个分类器中至少有一个分类器在 m 个训练样本中取得零错误的概率。此时 P 的上界满足如下不等式：

㊀ 回顾下第 1 章的“派”分类问题，则假设空间大小 $|H| = 2108$，其中 296 个分类器能在训练集中实现正确分类。

$$p \leqslant k(1-\epsilon)^m \leqslant |H|(1-\epsilon)^m \tag{7.2}$$

通过上式，我们为在排除所有错误率超过 ϵ 的分类器之后，分类器在 m 个训练样本中能够成功正确分类概率定义了一个上界。

方便起见，对式（7.2）进行相应的修改。数学家都知道 $1-\epsilon < e^{-\epsilon}$，意味着 $(1-\epsilon)^m < e^{-m\epsilon}$。进行指数变化后，$P$ 的上界如下：

$$p \leqslant |H| e^{-m\epsilon} \tag{7.3}$$

假定我们希望这个概率低于某些用户自己设置的阈值 δ：

$$|H| e^{-m\epsilon} \leqslant \delta \tag{7.4}$$

对式（7.4）左右两边取对数，并进行重新排列，得到式（7.5），具体将在下面的部分进行讲解。

$$m > \frac{1}{\epsilon}\left(\ln |H| + \ln \frac{1}{\delta}\right) \tag{7.5}$$

“可能近似正确”学习。我们对推导过程进行详细的讲解，主要还是帮助读者能更好地理解其中一些容易被混淆的变量的具体含义。为此，我们特意在表 7.1 中总结了涉及的变量，方便读者快速浏览。

表 7.1　PAC 可学习性中使用的变量

m	…	训练样本数目
$\|H\|$	…	假设空间大小
ϵ	…	分类器错误率上限
δ	…	分类器在一定错误率下可训练的概率
		大于 ε 被认为在训练集上是 error-free 的

现在，我们已经具备定义一些重要概念的条件。一个错误率低于 ε 的分类器可以被认为是近似正确的。δ 定义为近似正确的分类器能够由 m 个训练样本推导出来的概率（m 为有限值）。因此，完整的概念为可能近似正确学习，简称为 PAC 学习。在本章中，我们一般称一个类不具有 PAC 学习能力，意味着没有足够多的训练样本能够满足（ϵ, δ）的要求，或者尽管有足够多的样本，训练分类器所花的时间却遥遥无期。

解释。式（7.5）意味着，要想以不低于 δ 的概率获得错误率低于 ϵ 的分类

器，我们至少需要 m 个训练样本。这个结论并不取决于你使用哪种机器学习技术，仅仅依赖于分类器假设空间 $|H|$ 的大小。

特别注意，m 与 $\frac{1}{\epsilon}$ 呈线性增长关系。例如，我们加强分类器的错误率约束，把 $\epsilon = 0.2$ 调低到 0.1，仍要求以相同的概率 δ 获取分类器，此时我们需要先前两倍的样本数量。同时，读者也应该留意到对于 δ 的变化 m 是比较不敏感的，只是与 $\frac{1}{\delta}$ 呈对数增长关系。

然而，我们千万不要忘记这里的推导过程是基于最坏的打算。因此，对于完美主义者来说，这里的上界是不够紧致的。例如，在推导过程中，允许存在 $k = |H|$ 的可能性，在大多数实际应用中，这过于悲观了。式（7.5）不应该这样理解：告知技术员到底需要多少训练样本。倒不如说是用于比较分类器学习能力的一条准则。在下一节中，我们将继续探讨这个概念。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 式（7.5）中 4 个变量的具体含义是什么?
- PAC 学习的概念是什么？在什么情况下，可以称一个类不具有 PAC 学习能力?
- 对式（7.5）进行推导。讨论这个结果的含义以及实践意义并解释为什么它可以视为最坏情况分析?

7.2 PAC 可学习性的实例

式（7.5）告诉我们在给定 ϵ 和 δ 的情况下，模型的可学习能力仅仅依赖于假设空间的大小。接下来我们将以一两种具体的分类器为例进行讲解。

布尔型属性的合取：假设空间。假定所有的属性都属于布尔型，而且所有数

据都是无噪声的，我们定义样本的类别取决于属性的逻辑组合：如果结果为真，则认为样本为正样本，否则为负样本。例如，训练样本的预测类标可以由如下逻辑表达式决定：

$$\text{att1} = \text{true AND att2} = \text{false}$$

这意味着一个样本被标记为正样本，则必须满足属性 1 为真，属性 2 为假，剩余属性无须考虑。假如样本不同时满足这两个条件，则被标记为负样本。

以机器学习的术语来说，这里的任务就是找到一个逻辑与表达式，能够对训练集实现正确分类任务。此时，所有满足我们定义的与运算构成的整个假设空间 H 有多大呢？

上面指定的逻辑与运算中，每个属性或为真或为假，或者不相干。这使得属性 1 有 3 种可能性，属性 2 也有 3 种可能性，以此类推直到最后一个属性 n。因此，假设空间满足 $|H|=3^n$。

布尔型属性的合取：PAC 可学习性。假设有一个无噪声的训练样本集可以用于训练符合上述范式的分类器[㊀]。为了满足在 7.1 节中提到的最后一条假设条件，我们认为至少存在一个与运算能够对所有训练样本实现正确分类。

因为 $\ln|H|=\ln 3^n = n\ln 3$，由式（7.5）可得：

$$m > \frac{1}{\epsilon}\left(n\ln 3 + \ln\frac{1}{\delta}\right) \tag{7.6}$$

通过式（7.6）我们可以得到训练样本数目的一个相对保守的下界，该下界满足以 δ 的概率使得训练出来的分类器在整个样本空间的错误率不超过 ϵ。需要注意的是，训练样本数量 m 与属性个数 n 呈线性关系。理论上来讲，保持 ϵ 与 δ 值不变，如果属性数目加倍，则需要的训练样本也需加倍。

任意布尔函数：假设空间。让我们拓展上述分类器至任何布尔函数，也就是现在考虑所有的 3 种基础逻辑运算（与运算、或运算、非运算）以及它们之间的自由组合。当然，此处我们也是假定所有样本是无噪声数据且包含了 n 个布尔型

㊀ 例如，在 1.2 节中介绍的变种爬山算法便可以处理这种情况。

属性。那么此时假设空间又是多大呢?

从 n 个布尔型属性，我们能够获得 2^n 个不同的样本。这定义了样本空间的大小。对于样本空间中的任意一个子集，至少存在一个逻辑函数，使得该子集内部样本的逻辑值为真，该子集外部样本的逻辑值为假。从分类角度来看，假如两个逻辑函数对任意一个样本的预测结果相同，则可称这两个逻辑函数是一致的，因为对于任意样本的预测意见总是一致的。因此根据逻辑函数对样本空间的划分，则逻辑函数的个数应该与样本空间的子集个数一致。

我们都知道，对于一个包含 X 个元素的集合拥有 2^X 个子集。因为我们指定的样本空间包含 2^n 个样本，则样本空间的子集个数为 2^{2^n}，同时这也是我们假设的空间大小。

$$|H| = 2^{2^n} \tag{7.7}$$

任意布尔函数：PAC 可学习性。由于 $\ln|H| = \ln 2^{2^n} = 2^n \ln 2$，由式（7.5）可得：

$$m > \frac{1}{\epsilon}\left(2^n \ln 2 + \ln\frac{1}{\delta}\right) \tag{7.8}$$

从式（7.8）我们可以得到以下结论：给定合适的 ϵ 和 δ，所需训练样本数目的下界与属性的个数呈指数增长关系。如此增长速度，在实践中应该尽量避免。例如，如果我们只增加一个属性，则 $\ln|H|$ 值加倍，因为 $2^{n+1}\ln 2$ 是 $2^n \ln 2$ 的两倍。同理，假如我们增加 10 个属性，则 $\ln|H|$ 至少增加千倍。综上所述，这足够使我们相信这种形式的分类器不具有 PAC 可学习性。

友情提示。我们必须谨慎使用结论，盲目地生搬硬套并不可取。例如，在前面推导式（7.5）的过程中，我们考虑的是最坏的情况，因此得到的下界约束是相对保守的。事实上，在实际应用过程中，要想获得可信赖的分类器并不需要那么多样本。

在工程师寻找恰当的学习技术来解决问题的过程中，式（7.5）主要的贡献在于帮助工程师比较不同分类器的可学习性。例如，属性的与运算作为分类器可以从适量的训练样本中学习而来，而任意布尔函数作为分类器却不能学习到。

一些其他推论，我们将在 7.3 节中详细介绍。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 属性的与运算作为分类器，假设空间有多大？代入式（7.5）中结果如何？
- 其他一些分类器，你能求得假设空间的大小吗？
- 解释一般形式的布尔函数为什么不具有 PAC 可学习性？

7.3 一些实践和理论结果

本章第 1 节的理论分析为学习任务以及线索提供了更广的视角，具有不可忽视的意义。不仅这些作为智慧结晶被理论家津津乐道，作为实践指导更是令工程师欣喜若狂。现在让我们来快速浏览一番。

偏好与可学习性。在 7.2 节中我们将分类器约束为属性的与运算，并探讨其 PAC 可学习性，这种约束称之为特定类别的偏向。如果允许某些属性可被视为不相干的，则我们可计算得到假设空间 $|H|=3^n$。假如进一步强化偏好：分类器必须考虑所有属性，则假设空间减小至 $|H|=2^n$。这是因为此时每个属性只有真和假两种可能，而在先前情况下，允许第三种情况（不相干）存在。

这里还存在着其他形式的偏好约束，有些约束合取项的个数，有些偏爱析取项，或者预定义的合取项与析取项的混合模式，或者强加其他约束条件。不管怎样，在我们的讨论中，不同的偏好只是造成假设空间大小不一样而已。正如我们前面看到的，假设空间的大小会影响可学习性。

没有学习不带有偏好。偏好无处不在，考虑任意布尔函数，我们可知假设空间的 $\ln|H|$ 与属性的个数呈指数增长关系，符合 $\ln|H|=2^n\ln 2$ 这种形式的一般不具有 PAC 可学习性。这个解释非常简单，无约束的假设的空间太过庞大，有很大的可能性出现这种情况：学习到分类器，在训练集性能很好，而在测试集中性能却很差。更直白地说，学习到的分类器不可信。

需要注意的是，在机器学习之旅中，工程师只有给假设空间添加有意义的偏

差约束才能取得成功。偏差约束，理所当然地不能偏向错误的方向。读者应该能回忆起，我们的分析基于假设空间确实包含问题解，且所有样本是无噪声的等前提条件[㊀]。

奥卡姆剃刀定理。在寻找答案的旅途上，工程师经常会面临在多个偏差中选择的问题。例如，我们可以定义属性的合取形式来学习分类器，其中每个属性都起一定作用。现在我们放宽约束条件，允许有些属性不参与合取。在前一种情形中，我们得到 $\ln|H|=n\ln 2$，而后一种情形是 $\ln|H|=n\ln 3$，显然有更大的假设空间。此时，我们有两种正确的偏差约束，每一个定义了不同大小的假设空间。哪个偏差约束会更合适呢？

细心的读者，我想此时心中已有答案。由式（7.5）可知，训练出成功的分类器所需要的训练样本数依赖于 $\ln|H|$。对于给定的 ϵ 和 δ，越小的下界约束意味着需要更少的训练样本。选择那些假设空间较小的偏差约束将有助于工程师解决问题。

随便说一下，在几个世纪前，科学家们已经开始运用与之类似的准则：如果对于同一现象有两种不同的假说，我们应该采取比较简单的那一种。这条准则也被称为奥卡姆剃刀原理。这条准则，是以一名神学家命名的。神学家奥卡姆逝世于 14 世纪，并率先提出在近现代科学中使用这条准则。这里使用剃刀的含义是，当制订假设时我们最好剔除多余的信息。

在机器学习领域，科学家们已经找到证明这条准则有效性的形式化方法，甚至能够量化奥卡姆剃刀的有效范围，这真是个伟大的成就。

不相干与冗余属性。现在我们以 7.2 节提出的分类问题为例，可知所需训练样本数目的下界 m 依赖于属性的个数 n。以属性合取为分类器，我们可以得到 $\ln|H|=n\ln 3$；给定合适的 ϵ 和 δ，所需训练样本数与属性个数呈线性增长关系。

同理，我们能总结出另一个观点，不相干或冗余属性的存在会影响类的可学习能力。属性数目越高，也就是 n 越高，意味着在保持 ϵ 和 δ 不变的情况下需要更多的训练样本。因此，当我们如果有机会剔除没那么有效的属性时，最好还是

㊀ 当这些假设条件不满足时的分析过程，远远超过介绍性书籍的范围。

去掉。

这些考虑也正好解释了，为什么当只有很少部分属性带有相关信息时，我们往往很难学到好的分类器，如文本分类就是个很好的例子。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 是否理解了“没有一个学习不存在偏向”的含义？在什么情况下工程师添加偏向约束可能会影响算法的训练？
- 解释奥卡姆剃刀原理。数学家们又是以何种方式来支持这个哲学准则的？
- 式（7.5）揭露了不相干和冗余属性有哪些作用？

7.4 VC 维与可学习性

目前为止，我们仅仅考虑了假设数量是有限的数据域。现在我们开始考虑假设数量为无限的情况，这在实际应用中经常无法避免，如有些属性值是连续的。在这种假设空间为无限空间下，比较两类分类器的学习能力几乎是无法实现的任务。

我想读者都会同意分类器之间的灵活性都是有一定差异的，如某些分类器（多项式）明显比一些分类器（线性）更加灵活。那么灵活性是否也影响可学习性呢？确实有影响。让我们快速浏览一下理论学家在这种情况下是如何量化可学习性的。

散点训练集。现在考虑在二维空间中，存在着如图 7.1a 所示的 3 个数据点。我们可以看到无论以何种方式将正标签和负标签分配给样本，我们总能找到一个线性分类器成功地分开两类。此时我们称这个样本集能够被线性分类器“打散”。

然而，如果 3 个数据点的分布如图 7.1b 所示，则不能被线性分类器打散。假如我们标记中间的样本为正样本，标记剩余的两个点为负样本，显然不会存在线性分类器能够成功区分两类。换言之，这个特别的数据集合不能被线性分类器打散。

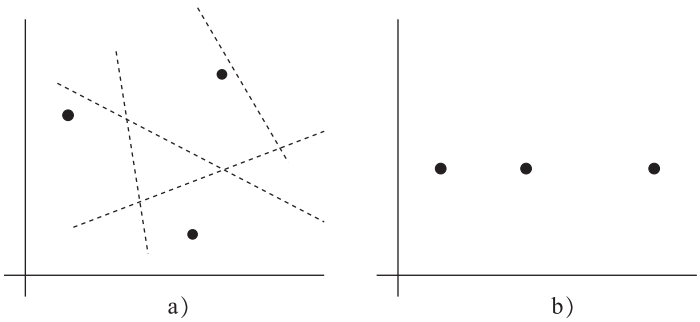


图 7.1　在分图 a 中包含 3 个点的数据能够被线性分类器“打散”；而分图 b 中集合不能被打散，因为假如中间的数据点与剩余的数据点不属于同一类，则线性不可分

给定一个数据集，不同的分类器具有不同的区分能力。例如，抛物线能够区分如图 7.1b 所示共线的 3 个数据点，对于拥有 4 个数据点的集合，只要不都共线则都能被区分。对于其他分类器，如高次多项式，能够区分任意大小的数据集。

Vapnik-Chervonenkis 维。Vapnik-Chervonenkis 维简称 VC 维。分类器的 VC 维就是它能打散或区分的最大样本数目。

在前面我们能看到，在二维空间中，线性分类器不能打散共线的 3 个数据点，但是只要不共线便能被打散。现在我们考虑 4 个数据点，不管怎样排列数据点位置，我们都能找到一组类标配置，使得数据点不能被线性分类器完全分开。正如定义所说，“能打散的最大样本数目”，我们能推断出在二维空间中线性分类器的 VC 维等于 3。

这里，有一点需要记住，VC 维的值反映和量化了给定分类器的几何属性。

连续域的可学习性。VC 维的概念使我们在连续域中研究可学习性成为可能。现在让我们来介绍一条经过稍微修改的著名定理，并省略了以下不相干的技术：

假设分类器的 VC 维为 d，并且假设空间 H 为有限空间，如果目标类与某些假设 $h \in H$ 一致，且样本数目 m 满足式（7.9），则将获得 $1-\delta$ 的置信度使学习到的分类器错误率低于 ϵ。

$$m \geqslant \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta},\ \frac{8d}{\epsilon}\log\frac{13}{\epsilon}\right) \tag{7.9}$$

式（7.9）意味着 m 的下界为 $\frac{4}{\epsilon}\log\frac{2}{\delta}$ 或 $\frac{8d}{\epsilon}\log\frac{13}{\epsilon}$ 中较大的那个值。对于这个结果，工程师可能会错误地理解为在训练样本中分类正确的分类器就是值得信赖的，而无须考虑机器学习算法。

特别注意的是，PAC 学习所需的训练样本数一般与 VC 维呈线性增长关系。然而，这不是件值得开心的事。那是因为，许多真实分类器的 VC 维度往往会随着属性的增加而迅速变得很大，这些情形我们将在下文详细介绍。

VC 维的一些例子。我们在表 7.2 中，列举了一些线性和多项式分类器的 VC 维的值。读者也需要对分类器复杂度与可学习性之间的妥协关系好好思索一番。因为分类器越复杂，意味着假设空间中越有可能包含了正确的解，同时过于复杂的分类器会使所需的训练样本数又快速增长，以至于我们根本没有能力学习到可信赖的分类器。

表 7.2　一些假设类型在 R^n 中的 VC 维

假设类型	VC 维
超平面	$n+1$
超球面	$n+2$
二次方程	$\frac{(n+1)(n+2)}{2}$
r 阶多项式	$\binom{n+r}{r}$

事实确实如此，在高次多项式的情形中，我们对训练样本的数量要求往往是不切合实际的。例如，一个二次多项式在 100 维的样本空间中，它的 VC 维可以由下式计算而来：

$$d=\frac{102\cdot 101}{2\cdot 1}=5151$$

这明显比线性分类器大得多，但仍处于可接受的范围。不管 ε 和 δ 取何值，只要 d 值充分大，则式（7.9）中的第一项总是会小于第二项。

如果，我们增加多项式分类器的次数到 4，此时分类器的 VC 维将高得离谱：

$$d=\frac{104\cdot 103\cdot 102\cdot 101}{4\cdot 3\cdot 2\cdot 1}=4598126$$

因此，我们应该尽量避免使用高阶多项式作为分类器。此外，神经网络和决策树被认为具有更少开销的 VC 维（见第 5 章和第 6 章），这正是为什么工程师如此偏爱他们的原因。

友情提示。与式（7.5）一样，为了简化分析过程，式（7.9）也是我们在考虑最坏的情况下推导出来的。当然对于这条结论，我们不能只是简单地理解为，在实际应用中需要多少样本。

非常重要的是，在实际应用中样本不是随意标记的。因为同一类的样本之间总是存在一些相似性，并且在实例空间中倾向于聚集在一起。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 给定合适的 ε 和 δ，成功训练所需的样本与分类器的 VC 维呈什么关系?
- 求二维连续空间中，线性分类器的 VC 维是多少?
- 以 n 元 r 次多项式作为分类器，如何计算 VC 维？如何理解“多项式分类器具有很高的 VC 维，反而不适合作为分类器”?

7.5 总结和历史简评

- 计算学习理论为了训练出可信赖的分类器，对训练样本数提供了确切的条件约束。样本约束仅仅依赖于两个参数：错误率阈值 ϵ，以及 m 个样本训练出低于错误 ϵ 的分类器的概率 δ。
- 如果样本属性值都是离散的，则对于给定的 ϵ 和 δ，用以成功训练出分类器所需最少的样本数由假设空间的大小 $|H|$ 决定。计算公式如下：

$$m > \frac{1}{\epsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right) \tag{7.10}$$

- 如果样本有些属性值是连续的，则对于给定的 ϵ 和 δ，用以成功训练出分类器所需最少的样本数由分类器的 VC 维决定。特别地，如果我们以 d 代

表 *VC* 维的值，则所需训练样本可通过式（7.11）计算：

$$m \geqslant \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8d}{\epsilon}\log\frac{13}{\epsilon}\right) \tag{7.11}$$

- 式（7.5）和式（7.9）都是在最坏情况下推导出来的。事实上，相对小一些的训练集已经足以训练出优秀的分类器。这些公式的主要好处还是方便大家比较不同机器学习算法间的学习能力。

历史简评。PAC 学习的主要思想由瓦利亚特[78]（Valiant）提出。关于 VC 维的经典论文由瓦普尼克（Vapnik）和泽范兰杰斯[81]（Chervonenkis）发表。在此基础上，瓦普尼克[79]（Vapnik）又进行了更细致的描述。运用 VC 维来研究可学习性，并研究奥卡姆定理，都属于布鲁默[5]（Blumer）等人的工作。随后更加紧致的边界被发现，如哈威泰勒（Hawe – Taylor）、安东尼（Anthony）和比格斯（Biggs）等人的成果。线性分类器与多项式分类器的 VC 维能够由科弗[14]（Cover）的成果推导而来。读者要是有兴趣的话，可以阅读科恩斯（Kearns）和瓦齐拉尼[36]（Vazirani）的名著，定能获益匪浅。

7.6 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 假定样本空间由第 1 章提到的用以描述“派”的属性构成。如果以属性间的合取为分类器，假设空间多大？现在考虑两种情况，一种情况允许某些属性是不相干的，另一种情况是所有属性必须包含在合取式中。
2. 考虑 7.2 节中提到属性为布尔型的合取问题，如果我们把 ϵ 的值由 0.2 更改为 0.5，δ 值保持不变，此时我们至少需要多少样本？相反地，我们只改变 δ，需要的样本数是如何变化的？

3. 再次考虑，样本属性值都为布尔型，分类器由属性间合取构成的问题。假如我们的分类器由 3 个属性的合取构成，此时假设空间有多大？例如，下面就是一个分类器的具体实例：

$$\text{att1} = \text{true AND att2} = \text{false AND att3} = \text{false}$$

4. 假设我们拥有一个 20 维属性值是连续的数据域。现在请在此数据域中，计算二次函数作为分类器的 VC 维，并与三次多项式做比较。假如，此时工程师发现，样本中有一半的属性是不相干的，移除那些不相干的属性，此时两个分类器的 VC 维又是如何变化的？
5. 比较具有 8 个属性的布尔函数和具有 4 个数值属性的二次方程作为分类器的可学习性。

思考题

1. 在式（7.5）中，δ 代表了在给定训练数据集中能成功训练出分类器的概率。在什么情况下我们可以不考虑 δ 的影响？
2. 从 PAC 的角度来看，不相干的属性与冗余的属性有区别吗？
3. 我们可以看到对于有些分类器，假如不添加偏好约束便不具有 PAC 可学习性。推荐一个学习过程，使得布尔型表达式作为分类器具有可学习性（考虑多个偏向约束）。
4. 一些机器学习的学者，考虑通过离散化处理将连续型属性转化为离散型数据。通过这种方式，将属性的取值范围分成多个区间，每个区间视为一个布尔属性（给定的数值在或者不在区间内）。现在考虑属性取值范围［0，100］的两种划分方式。第一种方式，将取值范围划分成［0，50］和［51，100］两个区间；第二种方式，将取值的范围划分成若干个相同大小的子空间，即［0，10］，…，［91，100］。从 PAC 学习的角度来探讨这两种方式各自的利与弊。

第 8 章　几个有帮助的案例

对于那些希望成为机器学习专家的人，仅仅掌握一些基本技巧是不够的。与书中的讲解相比，机器学习在实际操作中要复杂得多，工程师们往往会被超出预想的障碍困住，同样，实际操作也会改变每个人对分类器的认识，例如，分类器究竟要干什么？这样做的原理是什么呢？这些问题的答案都会和书上的内容有着极大的不同。就像在其他领域中的技术科学一样，没有一定的创造力是很难成功的。

自己实际尝试的经验或前人尝试后得到的经验都会对你有很大的帮助。本章就是讲解一些实际经验，对一些精心挑选的案例进行解读，使读者熟悉机器学习在现实应用中会遇到的一些典型问题，并学习实际操作中解决它们的方法。

8.1 字符识别

前边章节中关注的识别技术是很难编写计算机程序的，但是可以通过预分类训练样本的方法来实现。阅读文章甚至阅读手写文章的能力都可以归为这一种识别技术。它的应用是非常广泛的，例如，自动地预处理各种形式的文字、将记事本上书写潦草的文字转换为文本编辑器使用的字体的软件、各种各样的将纸质报纸转化为电子版的项目。仅仅一两代人以前，这类的事业还被认为是不可能的，但现在没有人觉得这些事情有什么超凡之处了。

任务描述。找到一种便于计算机识别的字符描述方式是十分困难的。为了更好地明白其中的原理，我们以数字“5”和“6”以及手写字母“t”和“l”的区分过程为例进行讲解。学习过程中你会发现，计算机对字符的识别是非常困难的，而将我们脑中的想法转换为一个计算机可以理解的代码是更加困难的。

正是这些困难促使我们使用机器学习的方法来解决问题。如果我们准备好了训练样本，并把它们表示成预分类的属性向量形式，那么前面的章节中讲到的一些技术可以生成分类器。这个过程似乎简单到让人不屑于研究，但它的重要性不言而喻，让我们来看一下相关内容。

样本和属性向量。首先，技术员必须定义一些描述样本的属性。图 8.1 给出了一种可用的方法。第一步确定数字所在的矩形区域；第二步将矩形区域划分为 $6 \times 4 = 24$ 个大小相同的小区域，用每个小区域的平均亮度表示小区域的特征。某一小区域有更多的墨水意味着它的亮度低，它的属性值会更低。相反，如果某个小区域没有任何墨水，那么它的属性值将是最大的。当然，这只是一个原理，没有任何规定要求必须有 24 个属性。实际上，如果期望更精细的识别，则需要划分更小的小区域，那么用于描述样本的属性个数也会更多。

在实际应用中，分类器不仅要识别单个的字符（如图 8.1 中的“3”），而且要“读”包含许多字符的文本。当然，这并不会增加任何的难度，只要将单个的字符分离出来，然后用相同的原则分别处理即可。

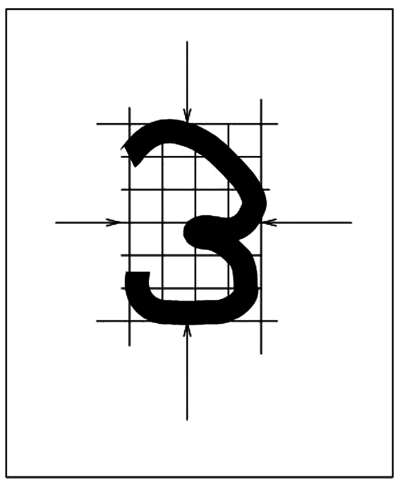


图 8.1　将手写字符图片转换为 24 个连续属性值向量的一种简单方法

目前网上有成千上万的手写字符数据集可以下载，其中一些就是利用上述方法进行描述的，当然也有一些使用了其他方法。

选择分类器。当我们知道如何表示训练样本以后，还需要继续选择分类技术。我们首先要明白的是：通过图 8.1 所示的方法获取的属性向量中，只有一小部分（有时候可能并没有）是不相关或冗余的，也就是说这个问题无须担心（除非属性的个数增加到远远多于图 8.1 所示）。另一个影响我们选择机器学习范式的因素

是：各个类之间是否是线性可分，因此在线性分类器的选择上会有所犹豫。最后一点，分类器是不需要对结果进行解释的。如果程序的目的是阅读且将手写字体转换为文本编辑器中的文字，那么用户不会想知道其中某个字符归类为“P”而不是“D”的原因。

基于以上观察，简单而又易于实现的 k-NN 分类器是一个不错的选择。谨慎一点的技术员可能会担心成千上万的训练和测试样本导致的计算成本。但我们设置的属性数量是适中的，这些成本不可能过高。从实际应用角度来看，整个算法的“高昂代价”主要取决于单个字符的划分和字符图像转换为属性向量的相关计算。只要这两种“代价”与分类“代价”差不多，则分类器是完全可以接受的。

实际上，最近邻分类器是最常用的方案，在本应用中，错误率通常小于 2%。对于一些太过潦草的手写样本，错误率会有所升高。但是，我们也不能对机器学习算法太过严厉，要知道有时候一个经验丰富的药剂师也是难以辨别一副潦草的手写药方的。

类别个数。如果我们假设所有的字符都是大写的，分类器要识别 10 个数字和 26 个字母，一共 36 个类别。如果小写字母和大写字母同时出现，那么总类别将达到 $10+2\times26=62$，同时我们也可以添加特殊字符，例如“?”“!”“$”等。得到一个相对大量的类别群其实并不难，而且也值得我们关注一下。

最需要考虑的问题是如何对分类器进行评价，仅仅使用错误率这一个参数是不够的。因为一个分类器若能正确识别 98% 的字符，它的性能已经好得令人叹为观止了。但是，错误率无法告诉我们错误的分布情况。通常情况下，一些字符总能被正确识别，另外一些字符的识别是有困难的，我们应当把精力更多地放到识别起来更困难的这些字符上。例如，一对长得相似的字符往往会被混淆，经验丰富的技术员往往想要知道是哪些“字符对”导致了学习过程中的混淆，以便通过对这些难以识别的类增加额外的训练来解决这个问题。

然而，相比于其他字母，有一些字母较少见。在这种情况下，如果不采取特殊的预防措施，这些“稀有的”类别将会被机器学习算法“忽视”，关于这个问题本书将在 10.2 节做更详细的讨论。

分类器应该被允许拒绝识别某个样本。辨识某个人的笔迹是很困难的一件事，有些字母的模糊程度让阅读者都无法辨认。然而第 3 章介绍的 *k*-NN 分类器却未被这些模糊的字符吓住：无论这个字母写得多么潦草，它总会找到一个最相似的类别并将它归于这一类。

实际经验表明这种情形是很糟糕的，得到一个错误分类产生的坏影响远远大于不能判断该样本的类别产生的影响。例如，在一个自动读取邮政编码的设备中，读取一个错误的数字将会导致一封信件被送到了错误的目的地，进而会引起延迟交付。另一方面，如果分类器没有给出任何答案，那么员工则不得不自己去读取。那么人工处理的代价会小于机器处理得到错误地址的代价。

因此分类器必须有拒绝识别样本的功能，特别是没有足够的证据支持它属于某个类别的样本时。对于 *k*-NN 分类器实现上述功能的最简单的方法就是在最后确定的类别获得的票数和第二名获得的票数之间设置一个最小的阈值。例如，在 7-NN 分类器中，最后确定的类别得到的票数是 4，而第二名获得了 3 票，那么这个样本由于不能明确究竟属于哪个类而被拒绝识别。

在其他一些例子中类似的功能很容易实现，例如，贝叶斯分类器或神经网络：分类器对某一个样本最有可能的两个类别的概率值或输出信号进行简单的比较，如果它们的差距没有超过预定的阈值，则拒绝对这个样本分类。

错误率 VS 丢弃率。分类器拒绝分类模糊的样本会减小错误率；另一方面，拒绝次数太多对分类器而言也不是一件好事。如果所有的样本都被拒绝了会怎么样？那样错误率将为零，但是用户会质疑：这个工具有什么实际的用处呢？

我们得到的经验就是，技术员需要考虑拒绝率和错误率之间的权衡。增加拒绝率可能会减少错误率，但过度增加拒绝率则会使分类器毫无价值。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 阐述如何利用属性向量表示手写字符。

- 寻找最合适的机器学习工具时，需要考虑哪些方面（以及为什么）？
- 应用中类别数量过多会产生什么直接影响？为什么这里说错误率不能解释一切？
- 为什么分类器可以丢弃某些样本？讨论丢弃率和错误率之间的平衡关系；讨论性能和实用性之间的相互作用。

8.2 溢油检测

图 8.2 是由卫星拍摄的海平面的雷达图像。在浅灰色的背景下，我们可以看到几个显著的深色区域：或大或小，显眼或不显眼，还会有各种形状。我们主要感兴趣的是右上角区域里的那个显眼且修长的目标：即石油泄露区域，因为一个游轮船长为了丢弃船底下的残渣而在大海上非法倾倒导致的。同样地，这个特殊的溢油引起了海岸警卫队的极大关注。

对于所有相关的海域（如毗邻的主要港口），卫星拍摄了很多这样的“快照”，并把它们发给了地面合作站。专家们在这里仔细研究这些图像来搜索石油非法泄露的迹象。不论何时，只要专家们发现一个，就会派遣一架飞机，携带光谱仪来验证这个迹象是否真的是溢油（这个任务对于卫星来说是无法完成的），进而收集证据，甚至有机会抓到肇事者。

不幸的是，专家的人工费用是昂贵的，并且不是随时可用的。他们可能因为休假、生病或者其他一些原因不能参与进来。此外，大量的图片识别工作是极其无聊的，容易出现辨识错误。这就是要开发一个能自动识别漏油过程的计算机程序的原因。

图 8.2 是从众多照片中选出来的一张，主要原因是它极其罕见的清晰性。事实上，未经训练的我们用肉眼也能看出来，图 8.2 中所包含的石油泄露区是不同于其他黑暗区域的。即便如此，我们也发现可用于计算机程序识别“漏油事件”的辨识特征是十分困难的。就实际目标而言，任务将更加困难。无论如何，为漏油事件编写具有识别能力的程序都是一个相当大的挑战。

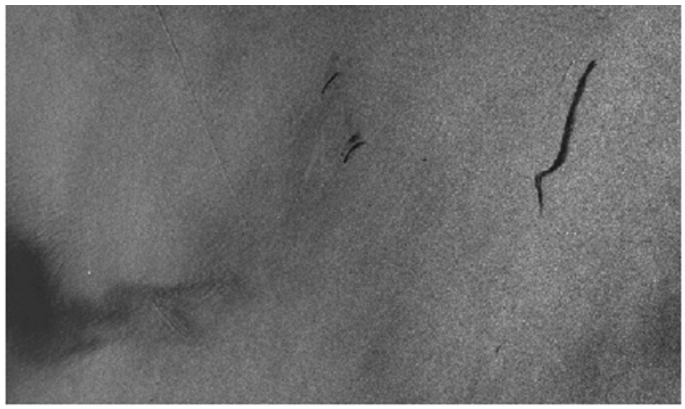


图 8.2　海面的雷达图像，右上角蠕动的细长的黑色区域代表环境灾害：石油泄露（库巴特[43]等）

再一次，机器学习让我们看到了希望，我们试图让机器自主开发一些必要的技能，从训练样本中归纳出方法。一般来说这种尝试可以总结为以下几步。

1）收集一些包含石油泄露的雷达图像。

2）使用具有标示能力的图像处理软件，在这些照片中找出显著的黑色区域。

3）请专家从中标记这些石油泄露的正面样本，其他黑暗区域（所谓的“长得相似的区域”）为负面样本。

4）用属性向量描述所有样本，让机器学习程序从样本中训练产生所需的分类器。

类似于前面的应用，我们将仔细研究这个案例中的一些关键点以吸取相关经验。

属性和类标。最新的图像处理技术很容易发现图像中的黑暗区域。要想让它们更好地为机器学习技术所使用，我们需要用属性来描述所有的图像以便于辨识溢油区域。最理想的是属性值能够不受所给定对象的大小和方向影响。

这个项目中使用的属性，包括区域的平均亮度、平均边缘梯度（用来量化边缘的清晰度）、目标短轴长度和长轴长度之间的比例、背景强度差异、边缘梯度的差异等。总之，通过各种特别的方式可以构造出 40 多个类似的属性。很难说其中哪些是真的有用，因为专家们对属性各自的相关性和冗余性也无法达成共识。我

们把最后的机会留给了机器学习程序。

把训练的样本分门别类地标记也不是件容易的事情，图 8.2 中的例子是较为容易归类的，在其他图像中可能会更加模糊。大多时候专家最可能会说的是“是的，这看起来像是溢油事件”或者“我觉得这就是我们正在找的”。因此类标的正确性非常不稳定，从而产生类别扰乱。类别扰乱的出现降低了我们的期望：分类器只能在训练样本数据上得到好结果。

选择分类器。这些样本被 40 多个属性描述，而其中的大部分可能是不相关或冗余的。这是一个严重的状况，读者应该记得无关和冗余属性的存在会使得一些分类器表现不佳。在这个特定的应用中，首先要使用一个决策树来避免这个问题，消除那些没有在任何树中出现过的属性。这是一种非常合乎逻辑的做法。毕竟，属性包含的信息决定了其是否能够包含在树中。由于无关属性中的信息含量非常少，用决策树方法来做识别是十分可靠的，同样在某种程度上冗余属性也可以被消除。

当那些经过决策树筛选保留下来的属性被应用到 k-NN 分类器时，分类器的性能应当是非常不错的。那么溢油问题只要有一个简单的机器学习技术就能很好地解决了：只要有的选，技术人员总是倾向于使用更简单的工具。

选择 k-NN 分类器也有其他几方面的考虑。这是许多实例中的典型应用，下面我们来详细地讨论一下。

误差成本。在这里，性能评价不像之前章节那么简单。一方面，错误率在该应用中可能会误导海洋监管者，产生各种各样的附带伤害和损失。这在溢油检测中是非常明显的，一个没有污染的地区被判为污染（下文称为假阳性错误），会导致飞机白白地飞往一个可疑的“无辜”地区，这意味着时间和资源的浪费（注意，消耗的不仅是金钱也是信任，因为这些错误会降低使用者对分类器的信任）。另一方面，一个污染的地区如果被判断为没有污染（下文称为假阴性错误），意味着会有一个没被发现的环境危害，其后果不可预料（金钱、环境和政治都会受到影响）。鉴于这一情况，我们可以看到这两类错误是非常不同的，我们甚至无法对它们进行比较，这也增加了完成该案例的难度。

在测试数据集上的实验表明，分类器犯的错误大多数为假阳性错误（即假警报）。至于假阴性错误，其实是非常罕见的。那么根据这个观察来修改分类器以改

变这两类错误的比例是否可行呢？

我们很难孤立地从本应用的短暂需求来回答这个问题。财政上的约束可能偶尔会迫使用户接受环境危害的风险，因为预算不能容忍错误警报。从而用户希望减少假阳性的频率，即使这意味着石油泄漏存在但没有检测到的可能性增加（假阴性错误）。

当然，这种情况在经济繁荣时期将会改变，此时用户不想错过任何一个石油泄漏，他们可以接受更高频率的假阳性错误从而保证非常少的假阴性错误。

倾向于学习一个或多个类。针对这些需要权衡的问题，要让用户有机会调整分类器，来调整一种或多种类型错误的频率。

正如前面提到的，由于该案例使用 *k*-NN 分类器，因此这个要求很容易满足：诀窍在于控制支持两类的决策的边界。方便起见，假设我们使用了一个 7-NN 分类器。如果我们规定如果某一样本至少有 5 个最近邻是正样本时，才能将它分类标记为正样本，那么可以有效地减少假阳性错误的数量；任何样本不能满足这个条件则被视为负样本。与之相对的是，如果我们想降低假阴性错误的频率，我们让分类器满足至少 3 个最近邻是正样本时返回正样本标签。用户对错误类型的偏好通过最近邻中正样本所占的比例来改变。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 技术人员是如何处理冗余或不相关属性的？是如何识别那些少而有用的属性的？
- 在训练集中，什么可以被认为是可靠的类别？分类器的期望性能是什么意思？
- 讨论这个案例中两种错误的代价：假阳性和假阴性。可以使用相同的计量单位做比较吗？为什么用户需要一个机制通过付出一种错误的代价来降低另一种错误？
- 解释 *k* 近邻分类器用来增加或减少其中一种错误机制的本质。

8.3 睡眠分类

整个晚上，我们经历不同的睡眠阶段，如深度睡眠、浅度睡眠或快速眼动睡眠（简称 REM，当我们做梦时出现）。为了确定在一个具体的睡眠过程中的这些阶段，科学家运用了先进设备做了细致观察：记录眼球运动的眼电图、记录肌肉收缩的肌电图和通过在头皮上贴上电极记录的大脑神经信号。基于所有这些仪器的结果，医学专家可以判断任何一个给定的时刻处于哪个阶段，甚至可以画出睡眠时相序列图，如图 8.3 所示。

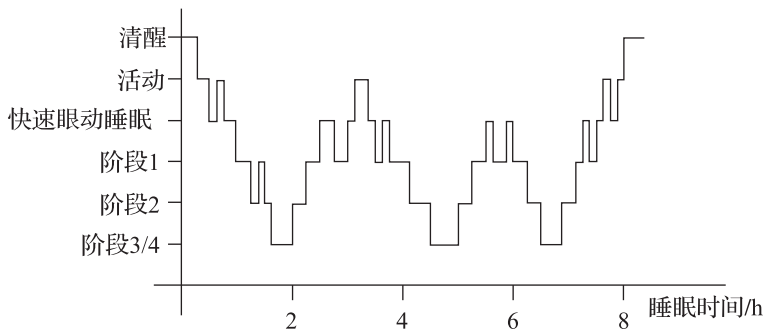


图 8.3　一个 8h 睡眠过程的时相序列图记

可以看到深度睡眠在这 8h 的睡眠时间里只出现了 3 次，并且不会持续太长的时间。还要注意“移动阶段”，这发生在试验志愿者从一侧转向另一侧以及移动头部或手臂时。

为什么重要。医疗过程中有时需要能够识别特定的睡眠阶段。一个典型的例子就是所谓的婴儿猝死综合症（SIDS）：婴儿在没有任何明显原因的情况下死亡。在医院中，被怀疑处于危险当中的新生儿必须处在检测中，从而可以随时使其苏醒。幸运的是，婴儿猝死综合症总是发生在快速眼动睡眠阶段，这就意味着没有必要对婴儿实施全时间段的监护，只是在 REM 这个阶段风险会增大。举例来说，一个设备可以识别快速眼动睡眠阶段，这就可以提醒护士：婴儿在接下来的 5min 可能需要更多的关注。

在夜间睡眠阶段的分布情况可以用于诊断特定的神经疾病，如癫痫，因此睡眠时相序列图也是一个有效的诊断工具。

为什么利用机器学习。手工画睡眠时相序列图是缓慢而枯燥的工作，对于高素质的专家也要花费 3 ~ 5h，而且专家也不是随时可以找到的。这就是为什么我们一直在努力开发能够实现从观测数据中识别各睡眠阶段的计算机程序，甚至希望能画出睡眠时相序列图。

为了实现这个功能，计算机需要对每个阶段有一个描述。然而这是很难实现的，医学专家也是通过长期观察获得经验的，他们所使用的特征和描述十分主观，无法转换为计算机程序。

这促使我们尝试通过预分类数据去训练分类器。具体而言，数据采集过程将 8h 的睡眠时期分为许多 30s 的片段，每个片段都视为一个单独的训练样本。几个小时的睡眠可以提供数以百计的训练样本。然后不同的设备对着 30s 的样本进行描述。

属性和类别。与以往一样，第一步仍然是剔除可能无关或冗余的属性。在前面溢油识别的应用程序中，属性选择的目的是增强 k-NN 分类器（由于该分类器对此类特征非常敏感）的性能。在睡眠分类中，还有一个需要考虑的原因：医生希望附加在被试者身上的设备尽可能的少。设备的存在不仅会让被试者感到不舒服，而且还打扰睡眠，会影响睡眠分析的结果。

至于类别的标示，在这个实验中可能比在漏油检测问题中的类标更不可信。“相邻的”（相似的）睡眠阶段之间的差异是非常不明显的，任意两个专家同时认同某一样本分类结果的概率很少能达到 70% ~ 80% 的，因此训练样本包含大量的类标噪声也不是什么奇怪的事了。低质量的数据使得我们要给分类器一个最小错误率的限制，这在基于数据构造的分类器上都可以实现。

分类器和性能。在这一特殊案例中，我们使用的是决策树和神经网络相结合的分类器方法，其具体细节在这里并不重要。这里需要说明的是，分类器在独立数据集上的精度确实达到了 70% ~ 80%，这意味着已达到了人类专家观察的性能极限。值得注意的是，单纯决策树的方法与两者相结合的方式相比，精确度要弱几个百分点。

但重要的是，分类器的行为并不能单用分类精度来描述（这与 8.1 节中 OCR 的应用相似）。第一，分类器大部分时间能够正确识别这 7 类中的一部分，但剩下的部分不能识别，分类器在快速眼动睡眠阶段的识别情况更是令人堪忧。该案例中分类精度范围在 90% ~95% 之间，乍一看结果足够好了。然而，仔细查看训练数据发现，只有不到 10% 的样本属于快速眼动睡眠类。这意味着即便一个分类器获得了好分类精度，也可以说“测试数据中没有一个样本属于快速眼动睡眠阶段”——这不是我们希望得到的结果。

我们意识到这种性能度量标准是具有局限性的，所以必须寻找其他的标准。而且这也确实存在，将在第 11 章进行讨论。

通过后期处理改善分类性能。睡眠时相序列图的性能可以根据领域知识进行改善。事实上，一些经验规律可以用在这里。举例来说，图 8.3 中，深度睡眠（阶段 3/4）不太可能发生在 REM 阶段之后，并且阶段 2 不会发生在“移动阶段”之后。同时，睡眠时相序列图可以通过移除任何只持续一个 30s 周期的阶段来完成平滑。在后期处理的过程中这些规则的使用可以提高睡眠时相序列图的价值。

本节的案例是非常有价值的。在该应用中样本是时序数据，每个样本的类标与其前后样本之间是相关的。在这种情况下，后期处理可以提高分类器的性能。

正确使用分类器。最初的想法是从几个被测试者中获取样本训练分类器，然后使用这个分类器识别新的数据，以代替昂贵的“人工”分类。然而，这个想法是不现实的。实践经验表明，睡眠数据没有“通用的分类器”可以通过这种方式获得：用一个被测试者的数据训练得到的分类器很难在性能没有严重退化的情况下应用于另一个人的睡眠时相序列图。

但这并不意味着机器学习在这一领域是完全不合适的。用户只需要改变一下对分类器的期望：不再追求一个通用的分类器，而是针对每个被试者分别训练一个分类器。这也能大大降低专家的工作量。具体步骤如下：

1）专家确定为样本的一个子集进行分类，从而创建一个训练样本。

2）用训练样本训练分类器。

3）利用分类器识别剩余样本，从而节省专家的时间。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在这个案例中，为什么最小化属性数量是重要的？如何识别相关属性？
- 睡眠阶段有 7 个类别。技术人员对分类器性能进行评估的原因是什么？
- 在睡眠时相序列图中，样本按时间顺序排列。在数据后期处理中如何利用这种情况？
- 在什么情况下，机器学习可以帮助人类减轻数据分类的负担？

8.4 脑机界面

控制身体的肌肉的命令是由大脑运动皮层的相关区域下达的。所以完全瘫痪的病人也能从大脑中产生这些命令，但不幸的是，对于瘫痪的病人这些信息无法到达肌肉。事实上，这些信号与电极接触就会被检测到，所以人们产生了一个奇妙的想法：这些信号能否被正确记录再加以解释？是否可以用来控制计算机屏幕上的光标？如果能，那么瘫痪患者就有了和外界交流的方式，尽管他们本身不能说话也不能移动。

运动信号本身是十分复杂的，我们无法用数学公式来表示，但是我们可以记录信号，通过属性描述他们，再标注出具体的运动指令。这样我们就可以用一个训练样本集，通过一个机器学习程序来获得分类器。

实验样本。本节将描述一个仅包含两个类别的特殊例子：左和右。训练和测试样本都是通过下文的实验来获得的。一个被测试者坐在计算机显示器前，前方是一个有两个按钮的木板。他将根据显示器上的指令，一个用左手食指按压，一个用右手食指按压（见图 8.4）。

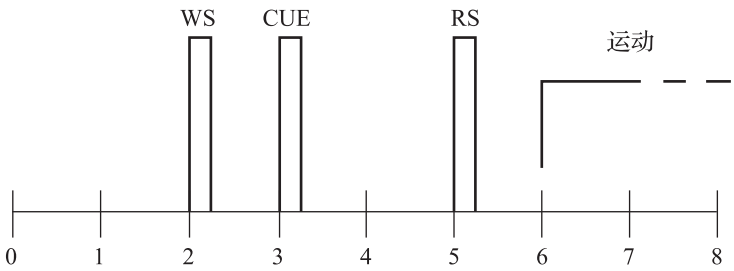


图 8.4　实验方式：准备（WS）信号后跟随着 CUE 信号（左或右）。当 RS 出现时，被试者暂停 1s，然后按下 CUE 按钮

整个场景按照图 8.4 所示的时间线进行，贴在头皮上的接触电极（见图 8.5）记录按压按钮前一段时间内的神经信号的强度，各电极产生的信号用 5 个数字属性描述，每个数字代表着一个特定的时间间隔内信号的强弱。

如图 8.5 所示，只有部分（11 个）电极被使用。对于这 11 个电极，每个用 5 个属性描述，总共 55 个属性。总之，被试者通过操作产生了包含两个类的少量样本。由于我们知道是哪个按钮实际上被按压，因此训练样本也不存在类标噪声。但实验者可能会怀疑这些属性中的很多都是不相关的。

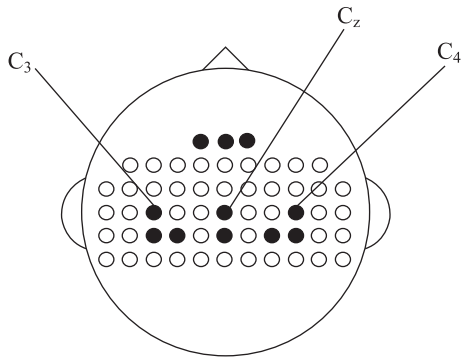


图 8.5　被测试者接触电极的贴放（仅有高亮的几个电极是有用的）

分类器和其局限性。包括多层感知机和最近邻分类器在内的一些机器学习技术在实验的时候，通常利用规范化属性的取值范围避免尺度相关问题。类似于前面章节的做法，利用决策树选择相关属性。

基于前面的测试数据，这些分类器的错误率稳定在 20% ~30% 之间，这个结果也取决于具体的被测试者。由于实验中电极提供的信息不够，因此分类器很难达到更好的精度了。

更重要的是在睡眠分类案例中，通过一个被测试者的样本数据获得的分类器仅适用于被测试者自己。所有创建通用分类器的尝试都以失败告终，其错误率高

到让分类过程看起来像是随机的。

检验分类器的实用性。跟之前的应用案例一样，错误率不能反映分类器的全部。分类器的实用性取决于它在实际应用中的方案。在这个例子中，关键问题在于：分类器是否能把光标移动到正确的位置。

图 8.6 所示的实验回答了这个问题，实验者仍坐在计算机显示器前，头皮上贴着电极，然而有两个按钮的木板已经移除。在显示器上，有两个矩形，一个在左边，一个在右边，中间是一个十字的形状。

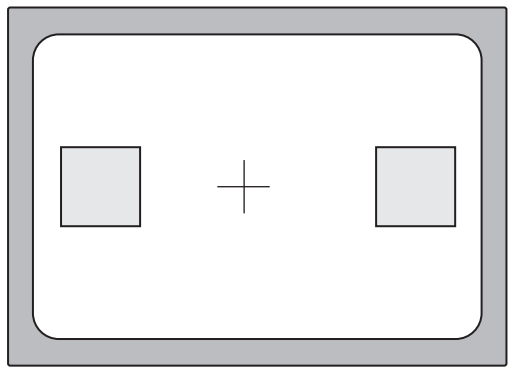


图 8.6　根据被试者思想让光标左右移动的分类任务

当听到指示要移动光标向右时，对象仅想象他按了右边的按钮。电极记录神经信号，并将它们传送到分类器。分类器根据这些信号选择光标的移动方向（类别：左和右）。移动过程非常缓慢从而有机会纠正错误的方向，为了做到这一点，对神经信号进行重复采样，每个采样都标记为一个样本送到分类器中。假设被测试者的意图是将光标移动到右侧，有可能第一个样本被判为左，使光标移动到了错误方向，但是如果接下来的样本能够正确地使光标移动到右侧，光标最终会停放在正确的矩形内。

什么样的错误率是可以接受的？在刚刚描述的场景中，30% 的错误率（看起来相当高）是完全可以接受的，即使光标开始在错误的方向，我们仍然有机会把错误的光标移动到正确的方向，结果是光标几乎都落在了正确的矩形里，即使在移动到正确方向前有些犹豫。

一个公正的观察员，在不了解分类器是如何被实际应用时，很少能注意到移

动过程中的错误，偶然错误（光标前后晃动）有时候是被测试者的犹豫不定导致的，并不是分类器的错。

只有当错误率高于30%才能使光标经常移错位置，这个分类器的结果才有点儿让人失望。对分类器的性能来说，移动光标的平均时间或许是一个更好的衡量标准。

历史简评。分类器的表现因人而异。基于多个被测试者在通用分类器上的训练，分类器产生的错误率高得几乎令人望而却步。它的一个缺点在于该分类器只能针对于同一患者。另一点，两个类别并不够用，至少要再增加上和下两个类，但这样得到的实验结果并不好。因此，要想实现和病人的交流，我们还需寻找其他方法。所以，整个实验是一个成功的机器学习案例，但它并不满足医学领域的需要，无法实际应用。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 讨论该案例中的现象：用一个被测试者的样本数据获得的分类器无法用于其他被测试者。
- 这一领域中，30%的错误率仍被人们认为是可以接受范围，解释其原因。有其他度量分类器性能的方法吗？在这个应用中你有什么建议？

8.5 医疗诊断

医生找出病人病因的过程就有点儿像一个分类器：基于某些属性（病人的症状、实验室测试的结果）给出诊断结果和治疗方案。在早期的机器学习研究中，因为这一原因人们认为机器学习天生就适用于医学诊断，几乎将其直接地应用于医学实践中。

一些结果。人们之所以对机器学习应用于医疗诊断抱有乐观的态度，是因为早期的研究结果，见表8.1。表中的分类准确率是分别用贝叶斯分类器、决策树

以及医生针对相同测试样本获得的，测试样本用属性向量来描述病人。4 种疾病的不同之处在于类的数量和任务的困难程度（例如，数据中的噪声、可用信息的相关性和可靠性）。

表 8.1　在 4 种疾病中，两种分类器的准确率和医生准确率的比较

	贝叶斯分类器	决策树	职业医师
原发性肿瘤	0.49	0.44	0.42
乳腺癌	0.78	0.77	0.64
甲状腺癌	0.70	0.73	0.64
风湿病	0.67	0.61	0.56

实验数据似乎证实了机器学习产生的分类效果能够与人类专家相匹敌的结论；甚至看起来要胜过人类专家的分类效果。在多个类别的原发性肿瘤领域的优势也许还不起眼，但是在其他领域中，机器学习的分类能力更好。但遗憾的是，这个数据表格并没有告诉我们分类结果的差异。也许标准差可以帮助我们更好地分析结果。当然，还有其他的方法可以确定结果统计的可靠性——在第 11 章和第 12 章将会介绍更多可以使用的方法。

结果是令人鼓舞的吗？结果似乎令人印象深刻，但我们不能过早下结论。第一个问题是比较是否公平？当样本被表示成属性向量，对于机器学习程序和分类器都是可以使用的，但是很难被医生同样地利用。就像医生可以利用主观的信息，但机器却不能使用一样。在这方面，人类是有某些优势的，这使得机器学习结果更令人印象深刻。

另一方面，该研究的作者承认参与实验的医生不一定是该领域的顶级专家，即机器学习只比一般的医师强。另一个方面值得我们注意的是，这项研究是在 20 世纪 80 年代进行的，那时的诊断工具并没有现代使用的那么复杂。从现代实验室测试技术中很难说谁受益更多：人类专家或机器学习？

除了分类，解释也是必要的。对于医疗诊断，仅仅是分类准确率是不够的；更准确地说，它不能够说明诊断是正确的。如果仅有“机器说是”这一诊断支持，则病人很难同意手术。也就是说“机器的准确率比医生平均高几个百分点”这个陈述不可能说服病人。

一个理智的病人需要的是一个令人信服的解释，为什么手术优于保守治疗？表 8.1 中医生通常能逐条记录什么证据支持具体的诊断和原因，以及在这种特殊情况下建议的治疗方案是什么、为什么。不幸的是，基本的机器学习技术并不擅长解释它们的决定（也许除了决策树以外）。这还需要软件进行自动推理，当然这超出了这本书的范围，就不在这里赘述了。

度量置信度的需求。还有一个需要注意的问题。在前面章节提到的大部分分类器只是给一个最后的预测，例如，“样本属于类别 77”。有时医生（或病人）需要知道更多的信息。例如，“给出的判断绝对准确吗？”或“只是比其他类稍微像一些？”如果是这样，哪些其他类还应该考虑？

换句话说，医学领域通常要求分类器能够告诉我们对于分类结果的置信度有多少。例如，这些分类器应该说“这个样本属于 C1 的概率为 0.8，属于 C5 的概率为 0.7”。概率可以通过贝叶斯分类器准确地计算出来，从神经网络也可以获得类似的信息。然而，这两种方法仍然无法提供解释。另一方面，决策树可以提供某种解释。至于确信度，只能增加额外的功能，这里我们就不再研究了。

文化障碍。机器学习在医学诊断中得到的令人鼓舞的结果没能打动医生，这里还有其他原因。这个原因就是一些小小的歧视，我们也可以称它为文化障碍。医生当然不喜欢被告知他们是多么容易被计算机所取代，因此在某种程度上我们可以理解医生不赞同的行为。这就是为什么在早期的机器学习应用中，许多医生并不急于开发必要的软件。

当然这是一种误解，机器学习的目的并不是替换人类专家，它只是提供建议，最后诊断决策仍然（即使在法律条款中）取决于病人的主治医生。不过，这些建议都应当很有价值。例如，分类器可以提醒医生一些额外的、没有想到的并发症状（病人往往同时遭受多种问题），它甚至可以指出是否需要进行一些特殊的额外实验检验。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 关于本节中给出的实验结果，为什么不能总结出分类器超越了人类专家的结论？
- 除了分类准确度，医学诊断还需要什么？
- 讨论基于机器的诊断的局限性，提出一个使机器学习在医学领域中得到应用的合理建议。

8.6 文本分类

假设有一个量很大的文本文档，读者需要判断哪些包含自己感兴趣的主题。如果文本量真的非常多，那么是没有办法进行手工分类的。然而，受以上应用案例的启发，可以先选择文档的一部分，阅读它们并分配类标签（正例和负例），然后训练一个分类器用于自动分类。

属性。一种常见的用来描述文档的方法是利用它所包含词语的频率。每个属性代表一个词，它的值是它在文本中出现的频率。例如，在一个文档中一共有 983 个词语，其中“分类器”（“classifier”）可能出现了 5 次，那么它的频率就是 5/983 = 0.005。

通常词汇表包含成千上万的词语，所以属性向量会很长。然而，由于只有一小部分可能会包含在文档中，因此大多数属性的值为零。简单的使用方法是只使用词汇表的一个子集，如 1000 个最常用的单词。

类标签。训练样本的类别确定是比较难的。即使我们只想要一些非常容易被归为计算机科学类别的文档，但可能存在一些文档相比于其他类别与这一类更相关的情况，因此对于一个文档是否属于该类别并不是那么容易确定。例如，一个非常明确的情况是：一个处理算法分析的科学论文；另一个是不那么相关的文档，只会提到一些算法比较的研究；最后一个文档就是流行杂志中的一篇文章。

处理这种情况的一种方法是“评分”类标签。例如，如果文档确定是相关的，则这个类评分为 2；如果只是有点相关，则为 1；仅当它是完全无关的，这个类评分为 0。通过这种方法用户可以自己决定在给定的应用中什么级别的相关性

是必需的。

关键点评。大量的属性使学习的计算代价非常昂贵。特别是使用决策树时，要分别计算每个属性的信息增益，这意味着成千上万的属性需要在每个测试中都进行分析。当样本量极大时，这些计算需要花费大量的时间。基于类似考虑，贝叶斯分类器和神经网络的计算代价也被看作是非常昂贵的。

或许有人会说使用简单一些的 k-NN 分类器或线性分类器就可以降低成本。然而，我们又要面对另一个困难：对于每个给定主题，绝大多数的属性都是无关的，这种情况降低了这两种分类器的实用性。

多标签样本问题。在前面章节讨论的案例中都假定每个样本都只对应一个类标。然而，在这个案例中每个文档可以同时属于两个或两个以上的（有时更多）类。

处理这种情况最常用的方法是对每个类生成不同的分类器。每当一个新的文档需要被分类时，那么将文档的属性向量同时提交到所有分类器，它们中的一些将返回“1”（即文档属于相应的类），其他则返回“0”。

但是这使得计算成本更加昂贵，因为这需要训练数以百计的分类器。由于每个分类器的学习过程都是很昂贵的，因此最终完成整个学习过程的代价是不可估量的。

另外，类之间并不是相互独立的。至少其中存在着子类或其他类，这样就可以形成一个类的层次结构。这意味着相应的分类器也许不应该以完全独立于其他分类器的方法去训练。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 在文本分类案例中如何描述样本？为什么说属性向量会非常长？
- 本节中“等级标签”是什么意思？
- 是什么让训练文本分类器变得非常昂贵？
- 讨论多标签样本的问题。

8.7 总结和历史简评

- 能用于学习的样本数量是由应用本身决定的。在某些领域样本十分丰富，如那些可以从数据库中自动提取的应用。某些领域样本是稀有和昂贵的，就像石油泄漏案例中的数据。
- 我们可能完全不知道哪些属性是重要属性。例如，在石油泄漏案例中，通过图像处理技术可以获取无数的属性用来描述形状和其他特征。
- 在某些领域，真正重要的属性是模糊的或者无法以合理的成本获得。所以训练仅能使用那些可以获得的属性，就算分类器的性能可能也因此被削弱也不得不这么做。
- 在脑机接口案例中，只要大多数的判断正确就足够了：在这种情况下光标就能落在预期的矩形中。我们可以看到的最大化分类准确率可能不是必需的。
- 在多类应用领域，错误率是无法完全描述分类器行为的。通常一些类几乎可以被完全识别，而有些类则出现了问题。因此，除了要知道分类器的平均性能，重要的是还要知道分类器在每个类上的性能。
- 在某些领域，我们还希望能够对最后的结论进行解释。高的分类准确率并不能满足我们的需要，如医疗诊断中涉及的内容。
- 假阳性错误的代价不同于假阴性错误的代价。在石油泄漏问题中，为海洋溢油付出的代价难以用金钱来描述。
- 许多应用都致力于最小化错误率，但是有时错误率不能解释一切。
- 另一个重要的需求是：用户要有可能通过参数来权衡假阳性和假阴性的比率（在石油泄漏领域问题中提及的）。

历史简评。表 8.1 的结果是来自于科诺年科[40]（Kononenko）等人的一个项目，他们参考了一个很久以前他们自己的项目。石油泄漏案例是库巴特[43]（Kubat）等人完成的。睡眠分类任务[45]和脑机接口[44]的机器学习应用也都是库巴特等人完成的。文字识别这个问题在计算机视觉领域已经进行了几十年的研究，

莫里[60]（Mori）等人第一次从机器学习的角度开发文字识别系统。文本分类任务是路易斯（Lewis）和盖尔[46]（Gale）最先开始研究的。

这6个应用程序，本书的作者参加了两个。介绍作者的工作并不是为了展示其让人惊叹的学识和在该领域不可估量的贡献。相反地，事实是，如果你在项目中工作很长一段时间，除了珍惜这些工作外也会有更深的认识，介绍这些案例远比那些阅读文献的方法更适合于教学。

8.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

思考题

1. 假设你知道训练集中的某些类标的正确性是不确定的。你会建议将这些“不可靠”的样本移出训练集吗？你在考虑的时候，不要忘记一些预分类样本将被用作测试样本来评估分类器的性能。
2. 讨论在某些领域用户为什么不愿接受机器学习工具。提出解决的方法，或至少能够降低他们疑虑和担忧的方案。
3. 8.2节提到了一种 k-NN 分类器控制两种错误（假阴性和假阳性）的简单机制。提出一种适用于贝叶斯分类器和神经网络的类似机制。你将如何在一个线性分类器中实现这个机制？
4. 在本章中讨论的多个案例中，减少无关属性和冗余属性的数量是必要的，其主要是通过决策树来完成的。请从前面章节介绍的技术中建议另一种方法，并讨论其优点和缺点。
5. 假设要实现一个程序，在一个一字棋游戏中，判断一个给定的位置是否可以获得全局胜利（见图8.7），你会用什么作为属性？如何收集训练样本？对收集数据中的噪声有什么评价？你会使用什么分类器？可能面对的困难

是什么？

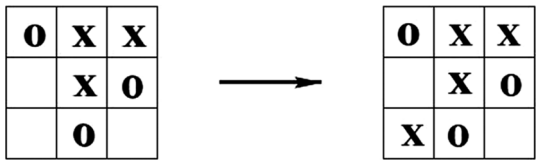


图 8.7　一字棋游戏：一行或一列或对角连成一条线

上机实验

1. 一些用于机器学习实验的数据集可以在国家标准与技术研究院（NIST）的网站上找到，找到这个网站并尝试一些新的应用案例。
2. 从互联网上找到关于美国 50 个州人口统计的网站。设置一个输出变量，如果这变量的值高于美国平均水平就视为正的，否则为负的，从而将每个州构成一个样本。根据网站提供的信息，确定用来描述样本的属性，并用这些样本数据训练分类器，利用分类器预测变量的输出值。

第 9 章　投票组合简介

面对困难的问题，人们常常会组织一次头脑风暴会议来找寻解决方法。在这种会议上，来自不同领域的专家们会共享他们的知识，从各自的视角提出可以互补的观点，进而启发彼此，得到具有原创性的解决方案。同样地，在机器学习问题中，也可以进行类似的活动。我们可以创建一组分类器，使它们中的每一个都与其他的有点儿不同。当需要对目标类进行投票时，所有分类器的“集体智慧”能够弥补单个分类器出现的缺陷。

本章介绍了用数据归纳出这一系列分类器的机制。“分类器集合”能有如此高性能的表现是靠着一种被称为“装袋（Bagging）”的简单方法。在此基础上，我们也可以采取更加复杂的“Boosting”算法及其相关变种算法，如“层叠（Stacking）”等方法。

9.1 “装袋”方法（Bagging）

为了简化算法描述过程，我们首先对限制条件进行设置：只进行两类分类，并规定每个样本只能为正或为负，最终所有的分类器都是通过一组训练数据集获得的。

基本原理。“装袋”方法通过引入一组分类器来完成学习。给出一个样本，所有的分类器都并行地工作，并对这一个样本各自标定类别。一个“主管分类器”负责收集这些分类信息，选择出现次数最多的类标来标记此样本。

我们假定每一个参与类标投票的分类器分别从不同的方向或角度完成了识别任务，这个分类器组（也称为“投票组合”）的结果一定比一个单独的分类器得到的结果要好。

投票组合的介绍。表 9.1 列出了 Bagging 算法的工作原理。从原始的训练集 T 中创建一定数量、相同大小的训练子集，T_1，T_2，…，T_n。当训练子集 T_i 被创建后，Bagging 算法会为其分配一个分类器 C_i。关于 C_i，前面章节中提到的任何一种学习算法都可以。我们需要注意的是，最开始的 Bagging 算法是假设使用了相同的分类器（即使用了相同的指定参数的决策树）。

表 9.1　Bagging 算法

输入：训练集 T 和用户选择的分类器。

1. 使用随机有放回的抽样方法，从数据集 T 中创建多个训练子集 T_1，T_2，…，T_n，其中每个训练子集中样本的数量是均等的。
2. 对每个 T_i 引入一个分类器 C_i（其中 $i=1$，…，n）。
3. 当出现一个新样本时，将它同时交给所有的 C_i，并且，让每一个分类器 C_i 都对这个样本的类别进行投票。
4. 一个“主管分类器”来标记其中哪个类获得了最多的投票。

Bootstrapping 方法。现在，我们再了解一下训练集 T_i 的选择方法。首先要保证每个样本都有同等的机会被选择。一旦某个样本被选择到训练集 T_i 中，它会被“反送（returned）”回 T，从而该样本将会与其他样本以同等的概率参与到训练集 T_{i+1}的样本选择中。对于一个包含 N 个样本的训练集 T，这个过程将重复 n 次，该过程被称作“Bootstrapping”。

可以看到，一个样本可能在 T_i 中出现多次，也可能不出现在 T_i 中。这意味着每一个 T_i 包含 N 个样本（可能存在重复样本），但各个 T_i 是不同的。从而使得每个分类器将针对学习问题的不同方面进行研究。

它为什么有效？ 图 9.1 解释了投票机制是如何降低错误率的。样例中一共使用了 3 个分类器。假设分类器的错误率较低，每一个分类器可能会在检测不同的样本时犯错，在这次试验中，一个样本最多被错分一次，另外两个分类器给出的类标是正确的。那么结果就是：一个分类器偶然犯的错误将会被其他的分类器纠正。

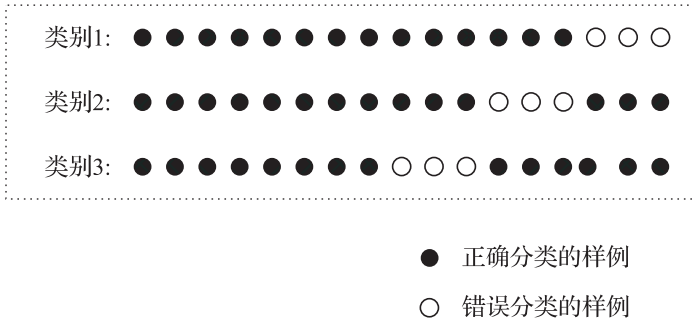


图 9.1　用 3 个分类器对 17 个样例进行分类。每个分类器都有 3 个各不相同的样例分错了类，这些错误可以通过投票的方式消除

当然了，实际状况并没有这么好。如果有一个样本被 3 个分类器中的两个错误地分类，那么结果会将这个样本分到错误的类。但是，增加分类器的数目，就能减少这种分类错误的出现概率。要注意的是，当各个分类器在互不相同的样例上出错时，Bagging 算法往往能够取得好的结果。

一些见解。实际经验显示，Bagging 技术在单个分类器的错误率小的情况下能取得更好的结果。见表 9.1，只要有足够多的分类器，某个分类器所犯的错误被其他分类器纠正的可能性就会很大。

但是，在单个分类器错误率很大的情况下，Bagging 技术就不能有很好的结果了。当分类器的数目多到某一特定值之后，单个分类器的错误率即便依然很高，这些错误也能被纠正。根据大数定律，潜在问题的每个方面都会表现出来。该方法唯一的缺陷是为了实现高准确率的识别，要引入的分类器数目太多。

从概念上讲，这种使用了大量的分类器的方法会有很大的随机性。图 9.1 让我们意识到，要是引入的分类器在性能上不是完全彼此独立，而是可以相互补充的话，就可以得到更好的结果。这种工作机制将在后面的章节详细叙述。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 你认为是什么原因使得一组分类器的投票结果比单独的分类器要好？在什么状况下这种投票机制会失效？
- 单个分类器是如何被引入到装袋方法中的？
- “Bootstrapping” 指的是什么？

9.2 夏皮尔提升（Schapire’s Boosting）

尽管 Bagging 方法有非常好的结果，但它依然存在一个严重缺陷：参与投票的分类器之间是彼此独立的，使用的数据都是随机选择的。我们可以设计一个更聪明，也更加有效的机制，使分类器之间呈现互补状态。例如，构建投票分类器组时，每一个分类器都选择不易被其他分类器感知的样例进行训练。Schapire’s

Boosting 方法便是以这种思想为基础创建的。

3 个互补分类器介绍。假设有一个随机子集 $T_1 \in T$（T_1 含 m 个样本），用它来训练出第一个分类器 C_1。当在整个训练集 T 上面测试这个分类器的时候，它肯定会错分某些样本。

假设我们现在又创建了另一个包含 m 个样本的训练子集 $T_2 \in T$，这 m 个样本中有 50% 来自于被分类器 C_1 正确分类的样本，另外 50% 来自于被错误分类的样本。这就意味着用分类器 C_1 来识别 T_2 中的样本得到的结果不是很好，甚至强行用分类器 C_1 进行分类的话，得到的结果像是随机分类的结果。因此，我们用训练集 T_2 训练出一个新的分类器 C_2。

由于 C_1 和 C_2 这两个分类器是用不同的样本训练而来的，因此它们标记特定样本的倾向性也是不一致的，但我们依然需要一个新的决策器。这时，我们要创建第 3 个训练子集 T_3，T_3 的样本只从 C_1，C_2 两个分类器给出的判断结果不一样的样本中选取，然后用 T_3 训练第三个分类器 C_3。

该方法的原理在表 9.2 中用伪代码进行了总结。当得到一个新样本时，主管分类器采集这 3 个分类器推荐的类标，并且返回得到票数最多的那个类。

理想状况是，每一个训练集 T_i 都有相同的大小，都有 m 个样本。

表 9.2　Schapire's Boosting 算法

输入：训练集 T，用户选择的分类器。

1. 创建一个随机训练子集 T_1，用它训练分类器 C_1。
2. 创建训练子集 T_2，并确保 C_1 分类器在 T_2 上面有 50% 的正确率，用 T_2 训练分类器 C_2。
3. 创建 T_3，C_1 和 C_2 在 T_3 中的样本上给出了不同的识别结果，并用 T_3 训练 C_3。
4. 使用投票法，以少数服从多数的原则进行分类。

递归实现。刚才介绍的规则可以采用递归的方式实现，如图 9.2 所示。虚线矩形中的是得到的分类器三元组，将这个三元组看作一个独立的分类器。紧接着，创建一个新的训练集 T_4，要求三元组分类器对 T_4 进行分类的错误率为 50%，并用它训练一个新的分类器 C_4。最后，选取分类器 C_4 和三元组分类器给出不同类标的样本作为新的训练子集 T_5 来训练得到分类器 C_5。图 9.2 展示了这种投票方法的层次结构。

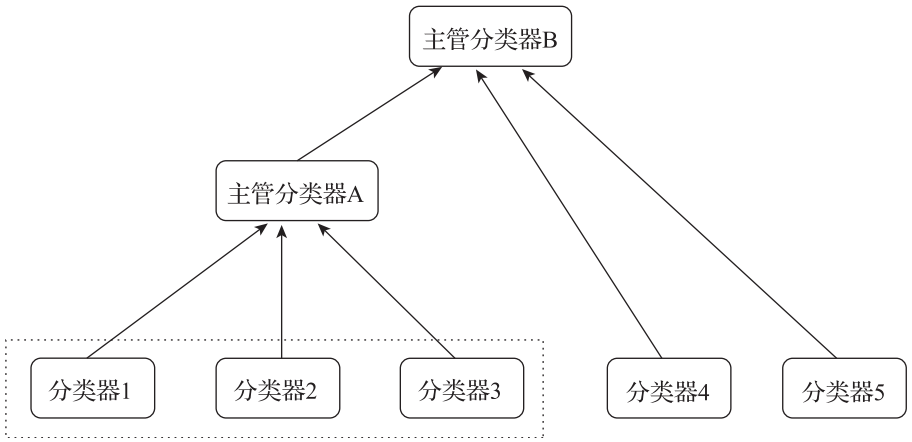


图 9.2 Schapire's Boosting 的递归应用，主管分类器 A 组合了分类器 1 ~ 3 的投票；然后，主管分类器 B 组合了 A 与分类器 4 和 5 的投票

理解投票机制的工作原理对于 Boosting 的学习是非常关键的。例如，如果分类器 1 ~ 3 都返回 0，那么 A 也会返回 0。然而，如果分类器 4 与 5 返回 1，那么主 B 就会否决 A 的结果，从而 B 的结果就会是 1。尽管 5 个分类器中的 3 个返回 0，整个结构的结果仍将是 1。

需要注意，使用单层递归时，分类器的总数是 $3+2=5$。当然，可以重复使用这个规则，重复一次，则用到的分类器个数变为 $5+2=7$。以此类推，假设递归了 N_R 次，那么参与的分类器总数就会是 $2N_R+3$。

性能考虑。假设每一个生成的分类器的错误率都低于 ε。数学界已经证明投票三元组的错误率将会低于 $3\varepsilon^2-2\varepsilon^3$，这个数值总是小于 ε 的。例如，$\varepsilon=0.2$，那么 $3\varepsilon^2-2\varepsilon^3=3\cdot 0.04-2\cdot 0.008=0.104$；如果 $\varepsilon=0.1$，那么 $3\varepsilon^2-2\varepsilon^3=0.03-0.002=0.028$。

换言之，Schapire's Boosting 方法提升了单个分类器的性能。但是当第一个投票三元组的错误率为（见上文）$3\varepsilon^2-2\varepsilon^3=0.104$ 时，其他分类器（4 和 5）就很难得到同样低的错误率了。解决这个问题的一个方案是将每一个分类器（如分类器 4 和 5）都做成一个三元组，如图 9.3 所示。在这种情况下，迭代 N_R 次后参与投票的分类器个数将会是 3^{N_R}。

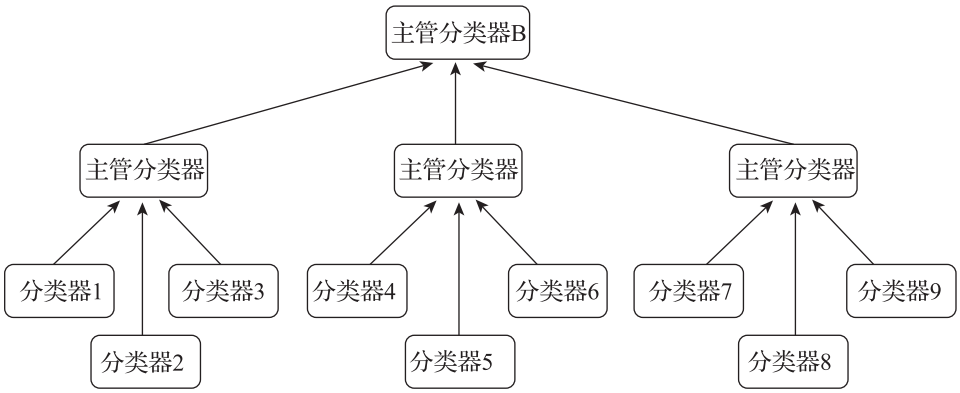


图 9.3　Schapire's Boosting 的另一种递归应用实现

在这里我们要明确一点，虽然每增加一层迭代都会降低错误率，理论上可以将错误率降到 0，但是实际上是无法实现的，下文我们将会讲到相关原因。

制约条件。如何找到后续的训练子集中的样本是最主要的难题。首先，这些样本必须满足限制条件。如上文所述，为了产生第二个分类器 C_2，我们要创建的训练子集 T_2 必须满足：使前一个分类器 C_1 的正确率仅为 50%。

这一点说起来容易，实现却很难。例如，如果整个训练集 T 有 100 个样本，第一个分类器的错误率是 10%，那么意味着我们仅有 10 个被错分的样本，这 10 个样本再加上 10 个被正确分类的样本可以用来产生 T_2。这就意味着 T_2 不能超过 20 个样本。同理，对于第三个训练子集 T_3 来说，它应该只包含 C_1 和 C_2 给出的类标不一致的样本，那么找到 m 个符合条件的样本是不可能的。

最后，即便我们成功地创建了这样的 3 个训练子集，也会发现相同的递归规则无法再使用下去。因为如果继续下去，我们需要的样本集将会大到无法想象（基本上不可能获得）。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- Schapire's Boosting 方法中用来产生每个分类器的训练子集是如何获得的?

- 请解释递归应用这个规则的两种方式。如果使用了 N_R 次递归，则需要生成多少个分类器？阐述取得最终分类结果的投票机制。
- 与单个分类器的错误率相比较，最终分类结果的错误率会怎样？
- 阐述实际应用中 Schapire's Boosting 方法的制约因素，为什么不能找到足够的样本？

9.3 Adaboost——Boosting 的实用版本

Bagging 算法的主要缺陷是它的随机性：算法中的每一个分类器在生成时，都是完全独立于其他分类器的。在 Schapire's Boosting 方法中，这种随机性大大降低，但是我们注意到另一个问题：在现实中，寻找满足第二个和第三个分类器要求的相当数目的训练样本也是不可行的。目前最有可能实现的 Boosting 算法是一种被称为 Adaboost 的方法，它根据样本的概率分布来训练样本，这种概率分布能够使整个“集合（assembly）”逐渐聚焦于那些难以处理的方面上。

一般性方法。与 Schapire 的方法相似，Adaboost 也是逐个创建分类器，每一个分类器都由不同的训练子集产生，而训练子集的组成依赖于前一个分类器的情况。这两种方法的主要区别在于产生训练子集的方法不同。Schapire's Boosting 根据某种精确定义的条件来选择样本，而 Adaboost 却是根据概率分布选取样本。每一个样本被选择的概率是特定的，所有样本被选择的概率相加必然等于 1。Adaboost 使用了一种完全不同的机制，旨在确保被前面分类器多次误分的样本成为下一个训练集中一员的可能性更高。另一个不同点是分类器的数目，与 Schapire 的三元组不同，Adaboost 方法更依赖于大量的分类器。准确来说，Adaboost 方法中的最终识别结果不是用简单的 Bagging 投票实现的，而是使用一种基于权重的多数投票机制。

训练样本的选择概率。我们提到，对于各个训练子集 T_i，它们是从原始样本集 T 中按照一定的概率分布选取的。这里先介绍一下这个法则的实现方法，假设第 i 个样本被选择的概率是 $p(\boldsymbol{x}_i)=0.1$，使用随机数产生器生成一个 0.0 ~ 1.0 之间的数字，如果返回的数字在 0.0 ~ 0.1 之间，那么此样本就被选择，否则不

选择。

创建第一个训练子集 T_1 时，每个样本都有相同的被选择概率，即如果 T 包含 m 个样本，每个样本被选择的概率都是 $p=1/m$。对于训练子集 T_i，每个样本被选择的概率会根据前一个分类器 C_{i-1} 的表现进行修改。主要目的是确保被前一个分类器 C_{i-1} 错分的样本出现在 T_i 中的可能性更大，以便新分类器 C_i 能够更加关注前一个分类器犯错的地方。

修改第 i 个训练集的概率。开始时，m 个样本都具有相同的概率，即被选择的概率为 $p=1/m$。根据这种概率 p，生成了第一个训练集 T_1，并训练出了第一个分类器 C_1。然后，训练样本被选择的概率会根据分类器 C_1 的结果进行修改。具体而言，被分类器 C_1 正确分类的样本再被选择的概率要降低，而被 C_1 错分的样本再次被选择的概率会增加。

表 9.3 给出了根据第 i 个分类器的分类结果修改样本概率的过程。首先，Adaboost 计算了第 i 个分类器的整体误差 ε_i，作为整个原始训练集的加权误差和，这可以通过对错分样本概率求和得到。加权误差和可用于减小那些已经被 C_i 正确分类的样本概率。具体来说，就是每一个样本的概率乘上概率 β_i，$\beta_i=\varepsilon_i/(1-\varepsilon_i)$。

表 9.3　Adaboost 算法

输入：包含 m 个样本的训练集 T，以及用户选择的分类器。

1. 令 $i=1$，对于每个 $\boldsymbol{x}_j \in T$，令 $p_1(\boldsymbol{x}_j)=1/m$。
2. 根据给定的概率分布，创建包含 m 个样本的子集 T_i，并训练分类器 C_i。
3. 将 T 中样本 $\boldsymbol{x}_j \in T$ 在分类器 C_i 上测试。如果 C_i 错分了 $\boldsymbol{x}_j$，则令 $e_i(\boldsymbol{x}_j)=1$；否则，令 $e_i(\boldsymbol{x}_j)=0$。
   1）计算 $\varepsilon_i = \sum_{j=1}^{m} p_i(\boldsymbol{x}_j)\ e_i(\boldsymbol{x}_j)$。
   2）计算 $\beta_i=\varepsilon_i/(1-\varepsilon_i)$。
4. 修改正确分类样本的概率 $p_{i+1}(\boldsymbol{x}_j)=p_i(\boldsymbol{x}_j)\cdot\beta_i$。
5. 归一化概率以确保 $\sum_{j=1}^{m} p_{i+1}(\boldsymbol{x}_j)=1$。
6. 令 $i=i+1$，返回第 2 步，除非遇到终止条件。

概率归一化。整个算法中很重要的一步是将概率归一化，以确保它们的和等于 1。概率归一化最简单的方法是用概率的和除以每一个概率。例如，已知如下

概率的情况：

$$p_1 = 0.4,\ p_2 = 0.2,\ p_3 = 0.2$$

这些概率的总和为：$0.4 + 0.2 + 0.2 = 0.8$。用 0.8 分别除每一个概率，将获得下面的归一化值：

$$p_1 = \frac{0.4}{0.8} = 0.5,\ p_2 = \frac{0.2}{0.8} = 0.25,\ p_3 = \frac{0.2}{0.8} = 0.25$$

可以非常清晰地看到概率之和为 1：

$$p_1 + p_2 + p_3 = 0.5 + 0.25 + 0.25 = 1.0$$

一个示例。表 9.4 演示了 Adaboost 修改样本概率的过程。首先，每一个样本以相同的概率被选择成为训练集 T_1 中的样本，并用它训练第一个分类器 C_1。训练完成后用 C_1 测试原始训练集 T 中的每一个样本。

表 9.4　一个说明 Adaboost 修改样本概率的案例

假设训练集 T 共包含 10 个样本，即 $\boldsymbol{x}_1$，…，$\boldsymbol{x}_{10}$。样本的总数 $m = 10$，所有的样本概率都设为 $p(\boldsymbol{x}_i) = 1/m = 0.1$：

$p_1(\boldsymbol{x}_1)$	$p_1(\boldsymbol{x}_2)$	$p_1(\boldsymbol{x}_3)$	$p_1(\boldsymbol{x}_4)$	$p_1(\boldsymbol{x}_5)$	$p_1(\boldsymbol{x}_6)$	$p_1(\boldsymbol{x}_7)$	$p_1(\boldsymbol{x}_8)$	$p_1(\boldsymbol{x}_9)$	$p_1(\boldsymbol{x}_{10})$
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

根据概率分布选择训练子集 T_1，并用 T_1 生成分类器 C_1。

假设在 T 上实验时，分类器 C_1 正确将 $\boldsymbol{x}_1 \sim \boldsymbol{x}_7$ 分类（其中 $e_1(\boldsymbol{x}_j) = 0$），错误分类的样本为 $\boldsymbol{x}_8 \sim \boldsymbol{x}_{10}$（且 $e_1(\boldsymbol{x}_j) = 1$）。

加权误差通过下式得出：

$$\varepsilon_1 = \sum_{j=1}^{10} p_1(\boldsymbol{x}_j) \cdot e_1(\boldsymbol{x}_j) = 0.3$$

概率修改的乘数可以通过下式计算：

$$\beta_1 = \varepsilon_1 / (1 - \varepsilon_1) = 0.43$$

概率修改公式为：

$$p(\boldsymbol{x}_j) = p(\boldsymbol{x}_j) \cdot \beta_1$$

计算得到新值（未归一化）：

$p_2(\boldsymbol{x}_1)$	$p_2(\boldsymbol{x}_2)$	$p_2(\boldsymbol{x}_3)$	$p_2(\boldsymbol{x}_4)$	$p_2(\boldsymbol{x}_5)$	$p_2(\boldsymbol{x}_6)$	$p_2(\boldsymbol{x}_7)$	$p_2(\boldsymbol{x}_8)$	$p_2(\boldsymbol{x}_9)$	$p_2(\boldsymbol{x}_{10})$
0.043	0.043	0.043	0.043	0.043	0.043	0.043	0.1	0.1	0.1

归一化之后，其值为：

$p_2(\boldsymbol{x}_1)$	$p_2(\boldsymbol{x}_2)$	$p_2(\boldsymbol{x}_3)$	$p_2(\boldsymbol{x}_4)$	$p_2(\boldsymbol{x}_5)$	$p_2(\boldsymbol{x}_6)$	$p_2(\boldsymbol{x}_7)$	$p_2(\boldsymbol{x}_8)$	$p_2(\boldsymbol{x}_9)$	$p_2(\boldsymbol{x}_{10})$
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.17	0.17	0.17

注意，这 10 个概率的和为 1.0。进而根据以上概率分布从 T 中选择出新的子集 T_2，训练分类器 C_2。

那些被 C_1 正确分类样本的概率（如样本 $\boldsymbol{x}_1 \sim \boldsymbol{x}_7$）根据公式 $p_{i+1}(\boldsymbol{x}_j) = p_i(\boldsymbol{x}_j) \cdot \beta_i$ 计算，其中 $\varepsilon_i = \sum_{j=1}^{m} p_i(\boldsymbol{x}_j) e_i(\boldsymbol{x}_j)$、$\beta_i = \varepsilon_i / (1 - \varepsilon_i)$。然后将概率归一化处理以确保它们之和为 1。

然后，创建第二个训练子集 T_2，训练分类器 C_2，并用 C_2 测试 T。基于分类器 C_2 的结果，计算 ε_2 和 β_2 的值，并修改被正确分类的样本的概率。这个过程会一直继续下去，直到满足用户预定义的终止条件。这个条件可以是分类器达到某个最大允许的数目，也可以是整个“集合”的分类准确率达到了某个特定阈值，也可以是其他条件。

加权多数投票。一旦选好了分类器，分类器就会并行处理待分类的样本，根据分类器的分类结果对每一个分类器赋一个特定的权重，最终的分类结果会由加权多数投票的方式得出。最后的结果不再是用上一节介绍的简单投票方式计算，而是更进一步地，给每一个分类器赋不同的权重值，记作 w_i，使用一种非民主投票机制。

给定一个样本，每一个分类器都返回一个类标。最终结果通过比较给出正类投票的分类器的权重之和 W_{pos} 与负类投票的分类器的权重之和 W_{neg} 得出。

举一个例子，假设有 7 个分类器，其权重分别为［0.2，0.1，0.3，0.7，0.2，0.9，0.8］。如果前 4 个分类器返回值是 pos，后 3 个返回值为 neg。上一节描述的简单投票方式将会以 pos 作为这个样本的类标，因为 pos 类获得的投票较多。但是，加权多数投票法将把给出正类类标的分类器的权重相加，得到 $W_{\text{pos}} = 0.2 + 0.1 + 0.3 + 0.7 = 1.3$。然后再把给出负类类标的分类器的权重相加，得到 $W_{\text{neg}} = 0.2 + 09 + 0.8 = 1.9$。因为 $W_{\text{neg}} > W_{\text{pos}}$，所以主程序会将这个样本标记为负。

如何获得每个分类器的权重。分类器的权重是根据它的性能赋予的：可靠性越高，权重就越大。从另一个角度讲，如果分类器的错误率高于准确率，则权重甚至可以是负的。在实际操作中准确地赋予分类器权重是很重要的，因此有必要了解一下权重的确定方法。

其实确定权重可以有非常多的方案，Adaboost 的作者给出了用数学方法分析这些方案的具体准则。其实我们也可以使用第 4 章介绍的简单的感知器的学习方法。刚开始的时候给每个分类器相等的权重，然后不断地用一个接一个的样本去

训练这个系统。每一次系统错分一个样本，我们就根据系统的假设 $h(\boldsymbol{x})$ 和训练样本的真实值 $c(\boldsymbol{x})$ 之间的关系来增加或减小分类器的权重。

当然也可以使用其他方法，如窗口法（WINDOW），此方法可以很好地找出那些对整个系统的性能贡献不大的分类器（如那些使用不相关属性描述样本的分类器）。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 描述 Adaboost 选择训练子集 T_i 的样本时使用的机制。
- 阐述 Adaboost 在产生第 i 个分类器后，是怎样修改每个样本入选第 T_{i+1} 个训练子集的概率的。
- 阐述在决定一个指定样本的类别时，Adaboost 所使用的加权多数投票的原则。
- 如何确定每个分类器的权重?

9.4 Boosting 方法的变种

Boosting 的本质是结合几个并不完美的分类器以使它们彼此互补。Schapire 首次提出了生成分类器的具体方案，Bagging 和 Adaboost 是目前最常用的手段。

其实 Boosting 的实现方法是非常多样的，为了更好地了解 Boosting 方法，本节将介绍 Boosting 方法中几个重要的变种。

随机化属性集。除了用不同的训练子集训练每个分类器（正如之前 Boosting 介绍中所示），我们还可以考虑使用相同的训练样本，但使用不同的属性子集来描述。也就是训练样本集 T 作为训练样本，集合 A 是用于描述训练样本的属性集合。代替随机选择样本训练集的方法，我们随机选择了包含 N 个属性的集合 $A_i \in A$。第 i 个分类器（$i \in [1, N]$）即从样本集 T 中用属性集 A_i 来训练产生。与之前的方法相似，分类器的输出通过加权多数投票获得，这个权重通过感知器学习获得。

这个方法在具有高维属性的应用中是非常有效的，特别是当这些属性中的绝大多数都是冗余或无关的情况下。显而易见，使用没有价值的属性集产生的分类器的性能将会很差，这些分类器将获得很小的（甚至是负的）权重值。

这个方法可以与经典的 Bagging 方法结合使用，要做到每一个分类器同时使用不同的样本集和不同的属性集。

异质 Boosting（Non-homogeneous boosting）。到目前为止介绍的所有 Boosting 方法都是假设分类器是用不同的数据训练的，但分类器是相同的。其实我们并不一定非要这样做，异质 Boosting 方法就反其道而行，每一个分类器使用相同的数据，但使用的机器学习方法不同。这些分类器的输出再结合加权多数投票方法给出最终结果。

该方法的主要优点是可以减少误差。我们在第 10 章将会学到，任何一个分类器的误差都可以分为两大类。第一类误差是由数据引起的，使用不同的训练样本集得到的分类器模型会存在一定差异，这会导致各种错误。第二类是由于分类器的偏好性，当正负样本分离面是高度非线性时，一个线性分类器必然会错分部分样本。

值得关注的是，异质 Boosting 可以同时减少这两类误差：数据误差（所有的 Boosting 算法都可以降低该误差）和分类器偏好误差（异质 Boosting 特有的优势）。

层叠。异质 Boosting 的思想是得到一组分类器的输出，然后使用加权多数投票获得最终的分类结果。这里包括两层：底层是基本分类器，上层是主管分类器组合底层的输出。需要注意的是，上层管理者本身也是需要训练的分类器，因为这一层要解决的问题实质上是一个线性分类问题，因此可以用简单的感知器学习方法。

这个原理可以被泛化成一种称为层叠（Stacking）的方法。如前文所述，底层使用一组不同的分类器，上层使用的分类器不一定非要用线性分类器。事实上，上层分类器可以是任意范式的，如贝叶斯、最近邻、决策树或神经网络。在异质 Boosting 中使用的线性分类器只是众多可能中的一个。

同理，基层分类器可能来自于不同的范式，甚至各个分类器仅仅在选择具体参数值时略有不同。例如，几个决策树在修剪范围上有所不同。无论如何，每一个底层分类器都应当在样本分类的方式上体现出不同。

表 9.5 简述了该方法的使用，每一行代表一个分类器，这 6 个分类器分别使

用了不同的机器学习技术，每一列代表一个训练样本。其中，如果第 i 个分类器标记第 j 个样本为正的话，则第 i 行 j 列的值为 1；否则值为 0。这样，每一列就可以看作一个用来重新描述对应样本的二值向量。这个新的训练样本集使用一个机器学习程序处理，并使用这些样本产生主管分类器。

表 9.5　6 个基本分类器给出的类标被用作重新描述样本的属性，每一列代表一个样本训练主管分类器

	x_1	x_2	x_3	…	x_m
分类器 1	1	1	0	…	0
分类器 2	0	0	1	…	1
分类器 3	1	1	0	…	1
分类器 4	1	1	0	…	1
分类器 5	0	1	0	…	1
分类器 6	0	0	0	…	1
实际类别	1	1	0	…	1

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释“随机化属性集”的含义。这个方法的主要优点是什么？它是如何与经典 Bagging 结合的？
- 解释异质 Boosting 的原理。从分类器误差的角度看，它的主要优点是什么？
- 解释层叠（Stacking）方法的两层原理。在什么情况下，我们认为层叠是泛化的异质 Boosting？

9.5　Boosting 方法的计算优势

在一些机器学习方法中，计算代价会随着训练集大小的增加而迅速增长。实验中我们发现，使用一半数据进行训练的代价占使用全部数据训练时的代价的很

小一部分。同样地，学习方法的训练代价也会随着属性数目的增长而迅速增加。

一个例子。图9.4给出了计算代价和样本数量之间的关系曲线。假定学习算法从N个样本中训练出一个分类器所需的时间为T，那么，从半数样本中获得一个分类器所需的时间仅是T的1/5，即$0.2T$。基于这一点，如果我们使用两个分类器，其中一个使用前一半样本，另一个使用后一半样本，那么最终会花费$2 \cdot 0.2T = 0.4T$的计算时间，大约节省了总体功耗的60%。

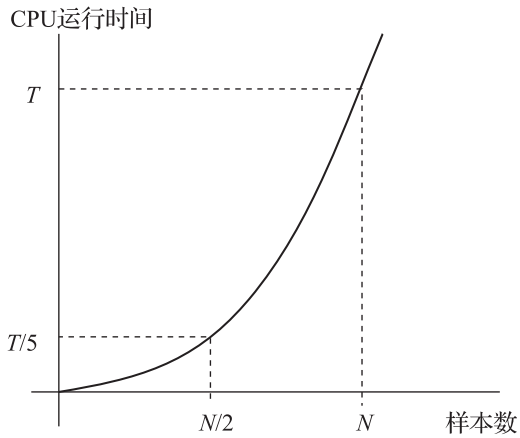


图9.4　在一些机器学习技术中，计算代价会随着训练集规模的增大而快速增长。通常，一半样本所花费的计算性能，只占所有样本所耗性能的一小部分

扩展到k个分类器。同理，我们可以考虑引入k个分类器，每个分类器使用不同的样本子集T_i训练，每个子集的规模为m（m是由用户设置的），m远小于N。使用这些子集获得的分类器进行投票，就像在Bagging方法中一样。在很多案例中，每个分类器的训练只占原时间的很小一部分，然而“投票组合”的分类性能和由整个样本集训练的分类器的性能差距并不大。相似的现象也可以在与Schapire's Boosting和Adaboost的比较中观察到。

总之，Boosting方式不仅提高了分类器的性能，它还起到了节约训练的计算代价这一作用，这一点在高维度大样本数据学习中尤其重要。高维度大数据学习的代价是非常大的，因此任何能够减少计算代价的方法都是极其受欢迎的。

在实际应用中的评价。在传统的Bagging方法中，T_i中样本数目与原始训练集中样本数目是一样的，因此如图9.4所示不会减小任何的计算代价。另一方面，

我们对 T_i 的规模进行假设是为了确保 Bagging 作为一种 Bootstrapping 技术的可行性。所以在实际应用中，这种假设并不必要，可以放松该限制。也就是说，和 Adaboost 方法一样，训练集 T_i 中 m 的大小可以是用户设置的常数。

我们可以回想一下 T_i 的大小在 Schapire's Boosting 方法和异质 Boosting 方法中都是作为参数使用的，这一规则也可应用于 Stacking 方法中。

样本选择的代价。在考虑减小计算代价时，我们不能忽略那些必要的开销。例如，每次选择下一个样本子集 T_i 时需要付出的额外代价。

在传统的 Bagging 的应用中，这种代价是非常小的；在 Adaboost 中花费更多一些，但还在承受范围内；但是 Schapire's Boosting 方法就不行了，寻找满足 T_2 和 T_3 训练集的代价巨大，特别是当有大量的训练样本可以使用，但满足条件的样本却仅占一小部分的时候。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 如何理解“使用小样本子集训练并产生投票组合的方法可以大大降低计算代价”这句话。
- 讨论在哪些状况下 Boosting 方法可以降低计算代价。
- 每一种 Boosting 技术在生成训练子集 T_i 时的计算代价是多少?

9.6 总结和历史简评

- 本章介绍了一种通用的机器学习方法，该方法需要一组分类器从不同的训练样本中获得学习模型。然后每个分类器分别对原始样本中每一个样本给出一个分类。最后，主管分类器根据投票结果给出最终的结论。
- 该方法最简单的实现方式是 Bagging。产生每一个分类器的样本子集是使用 Bootstrapping 机制从原始样本集 T 中选择的。假设 T 包含 m 个样本，那么该机制采用随机置换方式创建样本集 T_i。一些样本可能在 T_i 中不止一次出

现，但是有一些却总不出现。

- Schapire's Boosting 方法使用了 3 个分类器：C_1、C_2 和 C_3，并使它们在性能上尽可能地互补。这种互补性是通过创建训练子集的方式获得的：T_1 是随机创建的；T_2 用 C_1 对其进行检测的错误率为 50%；T_3 是用 C_1 与 C_2 分类结果不一致的样本组成的，但是每个子集包含的样本数目是相同的。
- 相比之下，Adaboost 方法按照样本概率分布构建样本子集，要求每次创建 T_i 时，那些前面分类器分类出错的样本被再次选择的概率应尽可能大。另一个区别是最终类别结果的获得是用加权多数投票的方式。
- 这个机制还有几个衍生的方法。第一个是随机选择属性集合：从相同的训练样本中训练分类器，但每次使用的属性子集都不同。另一个，异质 Boosting 方法，总是使用相同的训练集（以及相同的属性），但却使用不同的学习方法产生分类器，最终的类别结果也由加权多数投票的方式获得。最后一种是层叠方法，类似于异质 Boosting 方法，但是输出是由主管分类器决定的，而这个主管分类器是从底层分类器的分类结果训练得到的。

历史简评。Boosting 的思想是由夏皮尔（Schapire）提出的。他指出这种方法正如它的名字一样，即便是很弱的分类范式用这种方法处理后，其性能也会有很大提升。后来，弗罗因德（Freund）和夏皮尔（Schapire）又共同提出了一个更加实用的 Adaboost 方法（见参考文献［26］）。而布莱曼（Breiman）提出了 Bagging 技术（见参考文献［7］），布莱曼将这种技术用在了决策树中，提出了随机森林的方法（见参考文献［8］）。沃尔珀特（Wolpert）提出了层叠（即 Stacking）原则（称为泛化层叠（Stacking Generalization））。

9.7 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 假设用于 Adaboost 的样本数据的概率分布见表 9.6。通过该表可以创建 T_i，并且将 T_i 用于训练分类器 C_i。假设 C_i 错分样本 $\boldsymbol{x}_2$ 和 $\boldsymbol{x}_9$，请演示所有训练样本的概率分布更新和归一化的过程。

表 9.6　训练样本概率分布

$p(\boldsymbol{x}_1)$	$p(\boldsymbol{x}_2)$	$p(\boldsymbol{x}_3)$	$p(\boldsymbol{x}_4)$	$p(\boldsymbol{x}_5)$	$p(\boldsymbol{x}_6)$	$p(\boldsymbol{x}_7)$	$p(\boldsymbol{x}_8)$	$p(\boldsymbol{x}_9)$	$p(\boldsymbol{x}_{10})$
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.17	0.17	0.17

2. 假设 8 个分类器已经标好了一个样本。令返回 pos 标记的分类器权重分别为 [0.1, 0.8, 0.2]，并令返回 neg 类标的分类器的权重为 [-0.1, 0.3, 0.3, 0.4, 0.9]。那么，使用加权多数投票的主管分类器会返回什么类标？
3. 回到表 9.5，它记录了 6 个不同的分类器给样本中的 m 个样本进行分类的结果。假设使用一个 3-NN 分类器的主管分类器对样本做最终标记，且 3 个最近邻邻居（表 9.5 的列）分别是 $\boldsymbol{x}_1$、$\boldsymbol{x}_2$ 和 $\boldsymbol{x}_3$，那么最终的结果是什么？

思考题

1. 回忆 Schapire's Boosting 方法是怎样生成训练集 T_2 和 T_3 中的样本的。讨论在哪些情况下不能为这些样本集找到足够的样本。假设可以获得足够的样本，什么情况下其计算代价过大导致其不实用？相反地，假设能够获得足够的样本，在哪种状况下即便使用递归也能够使得计算代价在可承受的范围内？
2. 关于层叠方法，思考一种使得该方法失效的情况。相反地，再设计一种能让该方法优于异质 Boosting 的情况。

上机实验

1. 实现 Adaboost 基本算法，用户设置投票分类器的数目。同时，每个训练集

T_i 所含的样本的数目也作为该方法的输入参数。各个分类器的权重利用感知器学习获得。

2. 将上机实验第 1 题中的程序在一些 UCI 库上进行测试。仔细观察程序在不同数据集上的性能表现。对每一个数据集，画出显示分类器在这一数据集上的整体准确率与子分类器数目关系的曲线图。同时，观察训练数据集的错误率和测试数据集的错误率是怎样随着分类器数目的增长而慢慢减小的。
3. 完成堆叠算法在不同的底层学习算法和不同种类的主管分类器上的实现。将这些实现后的程序在多个基准数据集上训练，并观察它们的表现情况。

第 10 章　了解一些实践知识

也许你已发现，为了简明扼要，目前的介绍并没有包括机器学习的基本技术。本章的任务就是展示其中一些最重要的内容。

首先要考虑的是偏好（bias）。为了能够学习，学习器要根据当前问题建立一些假设，从而缩小搜索空间。下一个重要问题是，如果大多数的训练样例都只属于一个类，那么训练集规模的增加实际上对学习器有害。然后，我们讨论对那些随着语境和时间的变化，定义也会发生变化的类，以及如何去处理它们。最后的部分我们关注一些更平常的问题，如未知属性值、最有用属性集的选择，以及多标签样例的问题。

10.1 学习器的偏好

第7章中我们大胆地声明“不存在没有偏好的学习”。这种观点的意思是，没有限制（因此范围极广）的假设空间会包括许多假设，仅仅靠运气在训练集上正确分类，在以后分类时会产生很多错误。例如，在实际生活中找东西时，你要知道从哪儿开始找起，藏东西的地方越小，找到它的机会越大。

一个简单的例子。假设我们要找出接下来这组整数共有的属性：{2，3，10，20，12，21，22，28}。用机器学习的语言来说，这些数字构成了正例。与此同时也给出了一些负例：{1，4，5，11}。在这里，并**没有**给出属性。

学生常常研究数字理论里的概念，以试图寻找答案，如素数、奇数、超过一定临界值的整数、算术运算后的结果等。经过一番努力，找到了一些满足训练样例的属性。但通常来说，找出的规则都极其复杂，至少可以说是不合适的。

然而，一个简单的解法学生几乎没用过。其实，我们要找的属性并不是来自于算术方面。所有正例拥有的共同点是（负例没有）它们都以字母 t 打头：two，three，……，一直到28。相反，负例集中没有一个整数是以 t 打头的。

这个简单的解决方法之所以这么难找是因为大多数人找错了方向（算术）。用机器学习的语言来说，它们都依靠了错误的偏好。一旦他们得到了正确的答案，他们就会把这次的经验用到将来的分类上。如果再给一个相似的问题，他们就会

下意识地不仅想到算术，同时也会想到英文单词——会把这个新偏好考虑在内。

表示偏好 vs 过程偏好。就机器学习而言，偏好分为很多种形式。一种所谓的表示偏好（representational bias）取决于我们想把分类器用哪种语言形式化。例如，当属性为连续值时，一个可能的表示偏好会倾向选择线性分类器，另外一个可能喜欢选用多项式，而再一个则对神经网络感兴趣。如果所有属性都是离散值，那么工程师大概喜欢属性值的连接，甚至使用决策树方法。当然，所有这些偏好都各有利弊。

除此以外，工程师通常还会有某种过程偏好（procedural bias）。也就是说，我们对查找答案的某种方法有偏好，喜欢选一种特定的机器学习过程。例如，有这样的偏好依赖于一种假设，即修剪会改进决策树在未来数据上的分类性能。还有一个过程偏好是在神经网络中对参数集合的选择。然而在线性分类器的问题域中，工程师会用感知器学习而不用 WINNOW，或者相反。

偏好的强度 vs 偏好的正确性。假设工程师现在希望决定使用线性分类器还是神经网络来处理一个现成的机器学习问题。如果正负例都是线性可分的，那么线性分类器很明显就是更好的选择。当两种范式都能寻找解决办法时，神经网络更倾向于过度拟合训练集，因此难以推广到将来的数据中。另一方面，如果用于分开两个类的边界线高度非线性，那么线性分类器会缺乏必要的灵活性，而使用神经网络来处理可能很简单。读者现在开始理解每一种偏好都有两个关键的方面：强度和正确性。

如果一种偏好仅仅定义了一个小规模的分类器类，则它就是强偏好。从这个意义上来说，线性分类器的偏好远远强于神经网络的偏好：前者仅允许线性的决策面，而后者几乎能拟合任何决策面。

如果一种偏好能够解决当前任务，那么它就是正确的。例如，只有当正例和负例在一个域中线性可分时，线性分类器的偏好才是正确的。只有当一个类确实能被用布尔属性的组合描述时，这个布尔属性的组合才是正确的。当然了，正确偏好的反义词，不正确偏好，并不是一个精确的概念。它包含许多等级，有些偏好只是稍微不正确，而另外一些则很显著。

有用的经验法则：奥卡姆剃刀。理想情况下，工程师想要使用一种正确的偏

好（表示偏好或是过程偏好）。我们在第 7 章里学到：如果要在两个或两个以上的正确偏好中做出选择，那么工程师会偏向于选择更强烈的偏好，因为它更有可能获得成功。第 7 章中建议选择名为奥卡姆剃刀的最简单的方法。

不幸的是，我们很少能提前知道各种偏好正确还是不正确。有根据的推测已经是能希望的最好结果了。在一些范式中，如高阶多项式，偏好很弱，导致分类器很可能在整个训练集上分类的错误率为 0；然而因为 PAC 学习能力的问题，它在未来数据上的表现则不确定。偏好的增强（如通过减少一个多项式的阶数）会减少 VC 维，增加在未来数据上的机会——但只有当偏好仍保持正确的时候才有用。在某个特定时刻，加强偏好弊大于利，因为极有可能这种偏好变得不正确。

我们需要记住，现实中存在无法回避的权衡：一个不太正确但强的偏好比一个正确但很弱的偏好要好。但是，术语“不太正确的偏好”意味着要基于工程师的经验或额外的实验，以便在具体的应用中做决定（参见第 11 章）。

“终身学习”。在一些应用里，机器学习软件是要学习一系列概念或类，所有的这些都被期望在同类特定的偏好范围内找到解决方法。在这种情况下，在两个层次里组织学习过程就说得通了。在较低的层次里，任务是要分辨出最合适的偏好；在较高的层次里，软件通过使用这个偏好来归纳分类器。这一策略的名称是“终身学习”，提醒我们人类学习过程中遇到的典型困难：需要在特定领域内“学会如何学习”。

分类器错误的两个来源。到目前为止，我们观察到分类器错误有两个主要来源。首先是训练样例的变化。问题是，用以准确归纳分类器的数据并不能表示所属类的所有方面，部分原因是生成训练集的方式造成的。在一些应用里，训练集是随机生成的。在其他问题域中，它是由给定时刻可获得的样例构成的，有很大的随机性。然而在另外一些情况下，训练集是由专家生成的，他们把自己认为最有代表性的给定类的样例选了出来。最后一种情况难免主观，因此和前两种情况一样不可靠。纵观全部例子，很容易想象到从同一个问题域里可以生成不同的训练集。问题就在这儿，从不同的训练集里，会归纳出多少有些不同的分类器，而且不同的分类器会在未来数据上产生多少有些不同的错误。这就是为什么我们会说训练数据的变化是错误的重要来源。如果我们使用非常大的训练集，则它的负

面影响往往都能被减小。

错误的第二种来源是与偏好有关的。如果正（pos）和负（neg）两种类都不是线性可分离的，那么任何线性分类器都必然会分错某个最小比例的未来样例。与偏好相关的错误不可能减少到低于某个极限值，因为它们是所选类型的分类器固有的性质。

仔细在两种来源间权衡是有益的。如果我们用偏好较弱的机器学习方法，与偏好相关的错误就能减少。很不幸，一个意外的结果是这种做法方差（variance）较大。反过来，不一致性从理论上能通过加强偏好来减小——这么做如果不正确，就会增加与偏好相关的错误的频率。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 请解释表示偏好和过程偏好的不同，使用例子说明。
- 请解释强偏好和弱偏好之间的差别、正确偏好和错误偏好之间的差别。讨论两个二分类法的相互关系。
- 本节指出分类器在新的数据中表现不佳的两个典型因素是什么?

10.2 不平衡训练集

当讨论到漏油事件问题时，8.2 节指出的看上去特别明显的溢油图像相对罕见。确实，这个项目只能依赖于几十个正例，而负例有很多。两个类之间的不平衡对机器学习的影响非常严重。这一小节会解释出现困难的原因，然后继续解释一些减少它们负面影响的简单方法。

一个简单的实验。假设我们有一个训练集。这个训练集很小，仅仅包括 50 个正例和 50 个负例。对这个训练集进行 5 次交叉验证[㊀]：我们将其平均分为 5 块；然后，执行 5 次不同的实验，每次移走其中一块，在剩余 4 块组成的集合中归纳

㊀ 在 11.6 节中将介绍此评价方法。

出一个分类器，然后在被移除的那块上测试这个分类器。这样，我们能够消除或至少减少在选择具体的训练集时随机性的影响。最后，我们记下各次测试的平均结果：正例的分类精度，负例的分类精度，以及两种分类精度的几何均值。

假设现在可供我们支配的负例远远超过我们的预期。想要知道这一新情况是如何影响学习的，我们在原始训练集的基础上再加入 50 个负例（正例保持不变），重复实验过程，然后记下新的结果。同样地，我们在保证原始的 50 个正例的情况下，往训练集中再加入 50 个负例。

观察。如果绘制以上一系列实验的结果，我们很可能得到一张和图 10. 1 一样的图，其中使用到 1-NN 分类器。读者可以看到，随着多数类样例的增加，归纳分类器就会变得偏向这一类，负例（多数类）的分类精度逐渐聚拢向 100%，而正例（少数类）的分类精度降到 20% 以下。两个值的几何平均值也会下降。

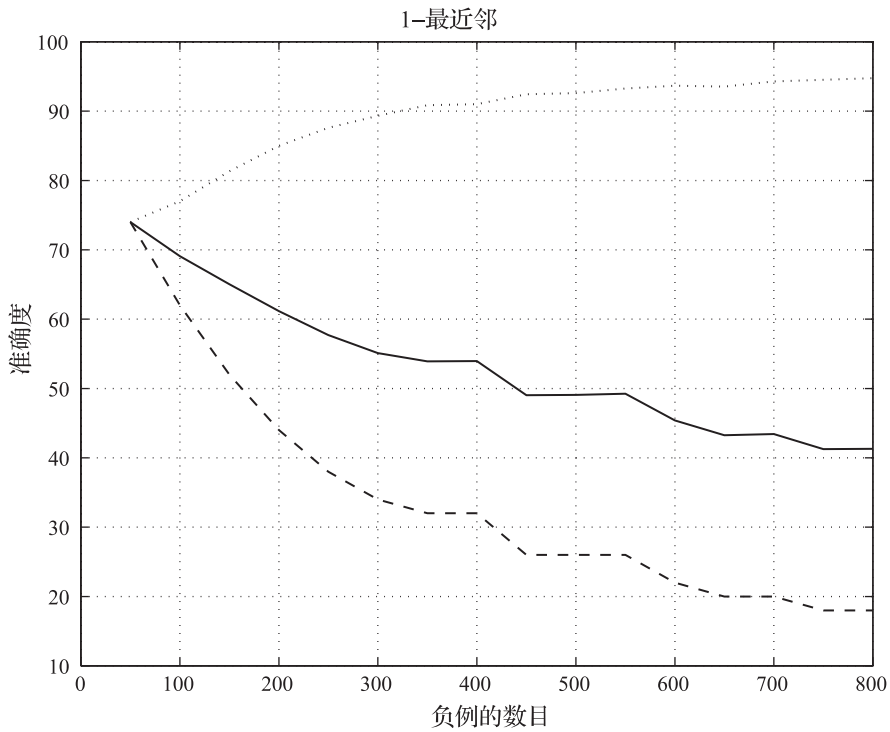


图 10. 1　点线：负类的分类精度；短画线：正类的分类精度；实线：两种分类精度的几何均值

在一定程度上，观测结果可能是违反直觉的。确实，如果有更多的训练样例可用，归纳分类器应该变得更加强大，即便增加的样例碰巧全都属于同一类？不过，前面描述的意外表现在机器学习方法中很典型。工程师经常称之为不平衡类表示问题。

多数类欠抽样（机械法）。实验证明，通过加入更多的多数类的样例可能造成分类器在少数类的表现上降级。这可能会成为一个严重的缺陷。在溢油问题域中，少数类代表溢油，是机器学习的主要目标。在医疗诊断中，典型的少数类是我们想识别出的任何紊乱。同样的道理，一些软件的任务是在公司产品被不当使用时发出警示（例如，电话卡的错误使用，或信用卡诈骗）。在这种类型的问题域里，吸引我们的是少数类。我们现在知道盲目地往训练集里添加更多的多数类的样例很可能弊大于利。

假设提供给我们一个严重不平衡的训练集，其中 9/10 的样例都是负例。在这种情况下，移除一些负例是有益的。最简单可行的做法是随机地移除，例如，每一个负例有 50% 的概率被移出训练集。正如上文提到的那样，从这种减少的训练集中归纳出来的分类器将优于从整个训练集中归纳出来的分类器。

辨识原因。前文指出的机械方法很难让思维缜密的工程师满意。他们想弄明白为什么移除数据有效；或者反过来，为什么增加多数类样例会产生不好的结果。

假设 1-NN 分类器使用了一个训练集，其中大多数都是负例，只有很少是正例。而且，数据被相当多的数据标签噪声干扰。让我们把数据限制在一个容易想象的两个类的域里，图 10.2a 展示了一个这样的训练集。读者很可能注意到了一点：由于噪声的缘故，几乎所有正例的最近邻都是负例。实际上，这些邻居可能是正例，它们被标为负例是因为在生成训练集的过程中产生了错误。不管怎样，1-NN 分类器错误地将这些正例进行了分类，这就是为什么会有这么多假的负例和这么少的（如果有的话）假的正例的原因。

当然，并不是所有的机器学习方法都会经历与 1-NN 分类器一样的情形。但它们大多数会在一定程度上经历，并且现在我们知道了原因。

可靠的解决方法：单边选择。知道问题的根源后，我们准备好了一个补救方法，即令我们头痛的是在正例的区域中出现了很多带有类噪声的样例，如果我们

主动移除这种样例，则情况将会改善（而不是用上文提到的机械法中的随机选择）。

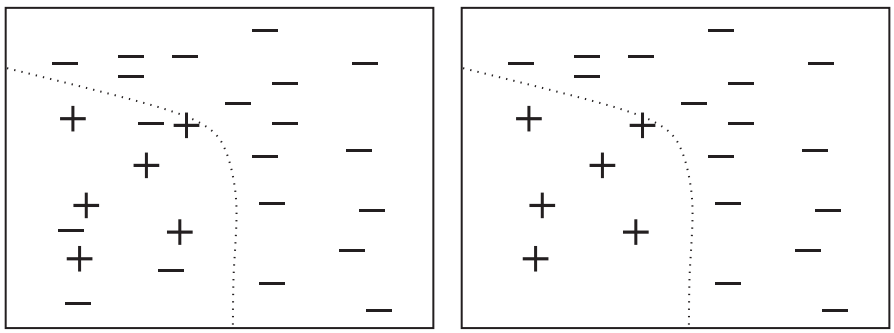


图 10.2　在负例远远多于正例并且有噪声的域中，移除托梅克连接（Tomek links）中的负例能增强分类器的性能

在第 3 章我们接触过一个简单算法：托梅克连接，能够识别“可疑”的样例。读者还记得有两个样例（$\boldsymbol{x}$，$\boldsymbol{y}$），如果满足 3 个条件，就可使用托梅克连接：①两个样例的类标签不同；②$\boldsymbol{x}$ 的最近邻是 $\boldsymbol{y}$；③$\boldsymbol{y}$ 的最近邻是 $\boldsymbol{x}$。像图 10.2a 中描述的那样，许多噪声样例确实都满足托梅克连接。这表明如果我们删除训练集中托梅克连接对中的负例，就可能改善分类器的表现。这个原理被称为单边选择，因为在训练集中只有托梅克连接的一侧被选中。

如果将这个技术用到图 10.2a 所示的训练集中，我们能得到图 10.2b 所示的小一些的训练集。不难看出，假的负例的频率变低了。这个方法的效率通常高于随机机械移除法。

相反的解决方法：少数类的过抽样。不过，在一些问题域中，训练集太小，以至于用任何欠抽样方法减少规模都不可行。即便是多数类的样例也很稀少，这时移除它们中的任何一个都可能删除学习任务中的一些关键信息，从而损害分类器的性能。

在这种情况下，相反的方法有时更好。与其移除多数类的样例，不如我们添加代表少数类的样例。由于我们没有这一类真实的样例，因此必须人工创建它们。

这可以通过两种方法做到：

1. 对来自少数类的每个样例，生成一个副本并且把这个副本加到训练集中；或者为每个样例创建两个或更多的副本并添加。

2. 对来自少数类的每个样例，生成一个稍微修改过的版本，并把它添加到训练集中。这个修改是通过对连续属性做微小（随机的）改动得到的；对离散型的属性值，这么做不太有用，但仍然是可能的。

图 10. 2a 帮助我们理解为什么这能行得通。对于一些“被破坏”的正例（那些最近邻因为噪声而变成反例）的邻居，在少数类过采样中插入额外的正例，结果，1-NN 分类器不会再被误导。这个原则也有助于改善其他类型分类器的表现。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 什么是不平衡训练集？请解释为什么从不平衡训练集中做归纳经常得出令人失望的结果。
- 多数类欠采样的本质是什么？请解释机械的方法，然后回答使用托姆克连接单边选择的动机和原则。
- 请解释少数类过抽样的原则。说明向训练集中添加新样例的两种方式。

10. 3 语境相关域

直到现在，我们都一直假设一个给定类是固定不变的，也就是说，一个分类器一旦被归纳出来，在所有的环境下会表现出相同的（或至少相似的）行为。但是，情况并不总是这样。

语境相关的类。一些类会随着环境改变它们的内涵。稍加思索，你就会发现在日常生活中有很多这样的例子。例如，“时装”的含义会随时间变化；不同的文化对着装的要求也不同；100 多年前“最先进科技”和现在的也完全不同；即使是“民主”或“正义”这样的词，其含义也依赖于政治背景和历史环境。如果

你想要更专业的例子，那么想一想语音识别软件中遇到的问题：每个人都知道同一个词的英式发音和美式发音不同，但是软件要“理解”来自两种背景的说话者。

语境相关的特性。从本书的需要出发，语境被定义为一种特性，在独立的情况下对类没有影响，但当它和其他特性结合在一起时会影响类。

例如，假设你希望引进一个分类器，这个分类器能够依据一系列的症状识别X来帮助医疗诊断。一些属性特征没有任何预测功能，如性别。男性病人并不意味着他更容易得前列腺癌，但是属性值“性别 = 女性”就是一个明确的暗示，这个类不是更容易得前列腺癌。当然，这是一个极端的例子。在另一个诊断中，性别的影响会受到某些测试的关键值的限制，如关键临界值对于男性病人来说是$p = 0.5$，而对女性病人来说是$p = 0.7$。另外，先验概率会有影响，乳腺癌对女性来说是更典型的疾病，尽管男性也有可能得乳腺癌。

在语境相关域中归纳。假设你想要归纳一个语音识别系统，而且你有一套英国人和美国人的语音训练样本。假设属性向量描述每一个例子包含“语境”属性，即说话者的地区。另一个属性捕捉具体的数字信号。每一个类标签都代表一个不同的音素。

在归纳分类器时，对决定代表哪个音素的属性向量，工程师能采用两种不同的策略。第一种利用语境属性，把训练样本分为两个子集，一个是英式英语的，一个是美式英语的。然后从每种训练子集中分别归纳出各自的分类器。第二种策略是把所有的样本混合成一个大的训练集，然后归纳出一个“通用”分类器。

实践经验表明，在这类应用中，如果那些实时系统知道选择哪一个分类器，则第一个策略表现较好。通过附加一个二值分类器，它经过训练后能分辨出英式英语和美式英语，可以帮助做分类器的选择。

概念漂移。有时候，语境会随时间改变。之前提到的“时装”例子就属于这一类型，政治词汇也是这样。在这种情况下，机器学习专家会谈到所谓的概念漂移。他们的意思是，随着时间的推移，一个类的本质含义会从一个语境漂移到另一个语境。

漂移有许多方面。其中一种是语境在一定程度上改变了类的意义。极少数情况下改变非常大，导致分类器变得完全不能用，必须重新归纳一个新的分类器。不过更典型的情况是变化不太大，仅仅导致分类器性能的轻微下降。也许经过一些微调，旧的分类器仍然可以使用。

另一个值得考虑的特性是漂移的“速度”，极端情况是剧变。在某个时刻，一个语境直接被另一个代替。不过更典型的是渐进变化。一个语境代替另一个语境时会有一个“过渡”期。在这种情况下，工程师会问这个过渡有多快，以及概念漂移是否（或什么时候）需要采取特殊措施。

时变类的归纳。概念漂移中所遇到最简单的情景如图 10.3 所示。这里，一个分类器会面对一个样例流，一次到达一个，或按有规律或无规律的间隔到达。每次样例到达时，分类器会给它们贴上类标签。这时也许有或没有一个反馈告诉系统（立即或有些延迟）分类是否正确。如果不是，那么正确的类标签应该是什么。

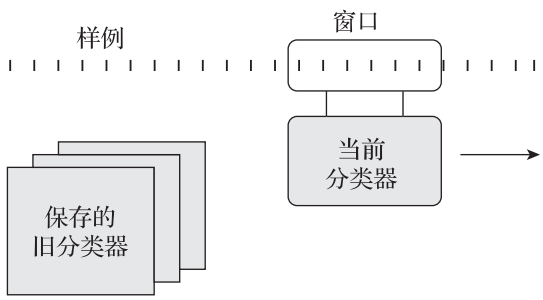


图 10.3　一个窗口经过一系列样例；“当前分类器”定期更新以反映所属类的变化。如果相关语境再现，则系统会检索出以前的分类器

如果我们有理由怀疑一个偶然概念漂移的概率，那么利用图 10.3 中展示出来的滑动窗口大概是个好办法。这样，仅仅从“在窗口中看到的”样例中归纳出一个分类器。每当有新样例到达时，就会添加至窗口中。到了一个适当的时候，旧样例会被单独或成组移除，如“最旧的 25% 的样例”。删除的动机很简单：工程师认为旧的样例大概属于过时的语境，而他只想让窗口中包含最近的样例。

正如上文提到的那样，分类器本应只反映包含在窗口中的样例。在最简单的实现中，每次窗口中的内容改变了，分类器都会被重新归纳。另一种情况，窗口内容的改变只修正或适应已有的分类器。

然而图 10.3 展示了这个学习范式的另一个方面：有时一个旧的语境可能会重新出现（例如，因为某种“季节性”）。出于这个原因，保存之前归纳的分类器的版本可能是一个好主意，以备它们在未来会被用到。

滑窗方法中的工程问题。在实施滑动窗口方法的时候，有些重要的问题是工程师必须要考虑的。首先是窗口大小的问题。如果太小，那么包括的样例就不足以进行成功的学习。如果太大，就可能包含了过时语境中的样例。那么，理想的状态是只要语境没有改变，这个窗口就会变大（没有旧的样例被删除）。当一个改变被监测到时，一定量的最旧的样例就将被删除，因为它们不再可信。

这就引出了下一个重要的问题：如何分辨出一个语境已经改变了？一个简单的解决措施是依靠分类器行为的反馈，发现分类器性能有突然的下降就能识别出语境发生了变化。

最后的问题是，应该从窗口中删除的最旧的样例是多少呢？答案要看语境是如何逐步变化的，以及变化的程度。极端情况下，一次剧烈的相当大的变化可能会删掉所有的样例。另一个极端是，两个极其相似语境的一次缓慢过渡只会删掉很少一部分最旧的样例。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 请列举一些类，它们的含义会由于时间或地理位置的不同而变化。请列举出那些之前含义会适时重现的域。
- 请描述基于时间顺序的一串样例构成的场景。请解释滑动窗方法用于归纳时变域的原则。
- 简要讨论在滑动窗方法中遇到的基本工程问题。

10.4 未知属性值

许多的域中都会有某个未知的属性值。病人拒绝透露他的年龄、测量装置不成功，以及一些信息的丢失或因为其他一些原因得不到数据。因此，我们就有了如表 10.1 那样不完全的训练集。表中的一些数值被问号代替。在一些域中，问号代替了属性值值域的大部分，这会使学习任务变得复杂。工程师需要理解这些未知值会带来哪些损害，以及有哪些解决措施。

负面影响。在朴素 k-NN 分类器中，只有构成向量的所有值已知，两个向量的距离才能通过计算得到。诚然，距离测度能够被修改，以便能量化像红色和未知之间的距离。但是以这种方式测量出来的距离会相当特别。

现在的情况不比多项式分类器和线性分类器好多少。不知道所有属性值，就不可能计算加权和，即 $\sum w_i x_i$。它的符号告诉分类器要选择哪个类标签。同样地，未知的属性值会使贝叶斯分类器和神经网络的使用复杂化。

从某种意义上说，决策树更灵活。当要分类一个样例时，很有可能一个未知属性值不是必须测试的（从根结点到终端结点的路径上没有它）。

填补缺失值的方法。在一个有充足的仅包含极少问号的大训练集的域中，要移除所有未知属性值的样例并无大碍。然而，这在有很多问号的域中就行不通了。在这样的域中，如果移除所有受影响的样例则会破坏大部分的训练集，丢失有价值的信息。

在这种情况下，我们尝试着用一些数值来代替问号，即使这些值是不正确的。这很容易操作。当属性值是离散的时候，那么我们就会用出现最频繁的属性值来代替问号。因此在表 10.1 中，样例 ex8，未知的外壳—颜色的值就会被白色替代，因为这是在特定训练集中最常见的。在连续属性值的情况下，可以使用平均值。在 ex6 和 ex10 中，权重（Weight）的值是不知道的。在 10 个已知样例中，权重的均值是 5.1，所以我们在 ex 6 和 ex10 中就用这个数值。

表 10.1 有缺失属性值的训练样例

样例	Shape	Crust		Filling		Weight	类别
		Size	Shade	Size	Shade		
ex1	Circle	Thick	Gray	Thick	Dark	7	pos
ex2	Circle	Thick	White	Thick	Dark	2	pos
ex3	Triangle	Thick	Dark	Thick	Gray	2	pos
ex4	Circle	Thin	White	?	Dark	3	pos
ex5	Square	Thick	Dark	?	White	4	pos
ex6	Circle	Thick	White	Thin	Dark	?	pos
ex7	Circle	Thick	Gray	Thick	White	6	neg
ex8	Square	Thick	?	Thick	Gray	5	neg
ex9	Triangle	Thin	Gray	Thin	Dark	5	neg
ex10	Circle	Thick	Dark	Thick	?	?	neg
ex11	Square	Thick	White	Thick	Dark	9	neg
ex12	Triangle	Thick	White	Thick	Gray	8	neg

这么做的时候需要小心。读者必须牢记，使用常见值或平均值补全样例不太可靠，甚至是值得怀疑的。因此，这个方案应该少量使用。当很多值都缺失时，应该使用更精细的方法（见下文）。

学会填补缺失值。有时候，最常见值或平均值的使用会误导学习过程。一个更好的填补空位置的方法是使用互相依赖的属性值。例如，一个人越高，他的体重越重。如果某个身高为 6.5ft（1ft = 0.3048m）的人的体重未知，这时使用整个人口的平均重量来计算就不是一个聪明的做法，毕竟，比普通人高的人肯定会比平常人重。因此，使用身高大于 6ft 的人来计算均值会更合适。

对于相互依赖属性就不赘述了。但通常情况下，相互关系要复杂得多，经常包括 3 个或多个属性。在这种情况下，一个简单的预测未知值的方法依赖于决策树归纳法。表 10.2 给出了这种技术的伪代码。

表 10.2 一个决定未知属性值的算法

令 T 为原始训练集，令 at 为未知值的属性。

1. 生成一个新的训练集 T'，在这个集里，at 是类标签；样例被所有剩余属性描述，前一个类标签（例如，正例或负例）被当作另一个属性。

（续）

2. 从 T' 中移除所有未知 at 值的样例。
3. 从这个最终版本的 T' 中归纳出一个决策树。
4. 利用决策树决定样例中未知的 at 值。

这个想法很简单。假设 at 是训练集中一个属性，有许多问号。希望用最可能的数值代替问号，可以用决策树的方法来做。我们把原来的训练集 T 转变为 T'，在这个训练集里原来的类标记（例如，正例或负例）都变成其中一个属性，与之相反，at 会被当作一个类标记。从这个训练集中，我们移除了所有 at 未知值的样例。在剩下的样例中，我们归纳决策树，然后使用决策树方法来填补缺失的数值。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 因未知属性值而出现的主要困难是什么？它们给归纳分类器造成的典型结果是什么？
- 请描述处理包含未知属性值样例时用到的方法。讨论它们的局限性。
- 请解释决策树归纳法是如何被用于确定未知属性值的。

10.5 属性选择

在许多问题域里，描述训练样例会用到许多属性：成千上万个，甚至更多。从这类数据源中学习的成本极高。而且，我们需要面对学习能力的问题，同时还有不相关属性或多余属性的问题。工程师需要熟练选择最适合属性的方法。

不相关以及多余的属性。不是所有属性都是等同的。有一些属性的值对样例所属的类没有影响，另一些则是多余的，它们的值能够通过其他属性的数值来获得，如年龄能够通过出生日期得到。这种类型的属性会误导一些归纳技术。例如，不相关的属性（或多余属性）扭曲了使用 k-NN 分类器时矢量间距离的计算。其

他的如决策树则没那么脆弱，但是它们可能需要过多的计算开销。

超长属性向量。一些问题域，如自动文本分类，被成千上万的属性所标记，这经常会出现问题。其中一个困难就是降低了学习能力：归纳分类器倾向于过渡拟合训练数据，在未来样例上测试时常让用户失望。计算成本也会变得不切实际，特别是当一个多层神经系统被使用的时候，每一个额外属性都会增加一定数量的需要训练的权重，因此会增加计算量。

而且，用数以千计的属性来描述的样例往往是非常稀疏的，这会误导许多机器学习方法。例如，第 3 章所解释的 k-NN 分类器中的稀疏问题。

众所周知，不管出于什么目的，这些属性中大多数是没用的，应该被处理掉。

选择属性的过滤器方法。选择属性最简单的方法可能是基于所谓的“过滤”。关键是要计算现有分类任务中每个属性的“效用”，然后把它们排序。选择前 N% 的属性，截至点 N 的选择。通常需要经过反复试验得出。

如果这些属性是离散的，那么给这些属性排序就会用到 6.3 节中的信息。由于二值化机制的存在（如 6.4 节），即使当属性是连续值时，也可使用信息增益。然而，人们通常更喜欢用统计方法进行相关性的测量。

批评属性过滤方法的一个原因是它忽视了属性间存在的关系。结果很难或几乎不可能去识别冗余属性。我们知道，相比其他属性，一个冗余属性不会带来任何额外信息，它的信息增益还很高。

要克服这些弱点有一个相对简单的方法。我们归纳出一个决策树，然后仅仅使用那些在测试决策树内部结点时遇到的属性。细心的读者会想起在第 8 章讨论的一些简单应用中，这个方法曾被使用过。

选择属性的包装方法。这个用于属性选择的所谓包装器（wrapper）方法更强大，计算也更昂贵。基本原理是，假设我们想比较两个属性集 A_1 和 A_2 的质量，在原始训练集 T 中，我们产生两个训练集 T_1 和 T_2，在这两个训练集里，所有的样例都有和 T 一样的类标签。然而，T_1 使用 A_1 描述样例，T_2 使用 A_2 描述。从两个新产生的训练子集中，归纳出两个分类器，并在一些独立的评估集 T_E 上评价，能产生更高性能的属性集被认为更好。

这是使用基于搜索的算法的原理，其伪代码见表 10.3。输入由训练集 T 和一

系列属性 A 组成。输出是最有用的属性子集 $S \in A$。开始时，S 是空集。每一步都从 A 中选择一个最好的属性添加至 S 中，“最好”是由 S 的属性描述的样例归纳的分类器的性能（在一个独立测试集里）决定的。如果 S 没有造成分类性能的改善，或 S 中没更多属性的加入，那么这个算法就会终止。

表 10.3　连续属性选择的包装器方法

将预分类样例的可用集分为两部分：T_T 和 T_E。设 A 为属性值集合。建立一个空集 S。

1. 对于每一个属性值，$at_i \in A$。
   1）将 at_i 添加到 S 中，使 T_T 和 T_E 中的所有样例都由 S 中的属性描述。
   2）从 T_T 中归纳出一个分类器，然后用 T_E 评价它的性能；性能用 P_i 表示。
   3）从 S 中移除 at_i。
2. 在 P_i 中找出最大值，把它从 A 中移除，然后添加到 S 中。
3. 如果 $A = \varnothing$，则结束运算；如果最后的运算性能不再增加，那么将最后的值从 S 中移除，结束运算。无论是两种结果的哪一种，S 都是最终属性的集合。
4. 返回第 1 步。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 从哪种意义上我们说一些属性会比其他的更没用？为什么工程师经常不能选择正确的属性来描述样例？
- 请解释和讨论以过滤为基础的选择属性的方法。
- 描述选择属性的包装方法的原理。请解释如何使用上述的原则，在简单的基于搜索的方法中选择属性。

10.6　杂项

在导论课本中，一些问题值得了解，但不需要单列一节。但是它们也不能被忽略，在这里我们对它们做一个简要概括。

数据缺乏规律性。假设你被要求从随机数发生器产生的训练样例中归纳一个分类器，其中所有属性值都是随机的，类标签也是。显然，这些数据毫无规律——然而机器学习技术能够以零错误率从训练集中归纳出一个分类器。当然，在训练集中完美的表现并不会在未来的样例中相似地表现。

这个情况建议我们用一个简单的机制来测量数据规律性的程度。这个想法将数据简单地分为两个子集：一个是训练集，一个是测验集。分类器从训练集中归纳出来，然后被应用到测试集上。在随机数据的情况下，我们观察到在训练样例中只有一个小的（如果有）错误率，但在测试样例上的结果会很差。相反地，数据越有规律，测试集中的结果就越好。

多标签域。在一些域里，一个样例可以同时属于两个或更多的类。例如，一个文档文本可以被说成代表营养、饮食、竞技、科学，可能还有许多其他分类。实际上，像表 10.4 上半部分所示的训练集，属性向量就用一些样例名所代替。注意，那里大多数的样例都会被标注两个或更多个类标签。

对于这种域，最常见的方法是为每个类归纳一个单独的二元分类器。更确切地说，第 i 个类的分类器是从训练集 T_i 中归纳出来的，这个分类器会包括与新标签相同的样例。如果原始训练集标记为 C_i，则在 T_i 中样例类是“1”，如果相反，则是“0”。

表 10.4 下半部分所示证明了这一点。因为原始训练集包含 5 个类标签，即 C_1，…，C_5，创建 5 个新训练集，即 T_1，…，T_5。从 T_i 开始，归纳了 C_i 类分类器。要注意一些新的训练集 T_i 会表现为不平衡的类，就如 10.2 节讨论过的那样。在这个具体的例子里，T_4 和 T_5 就是这样。

表 10.4　多标签域的训练集被转化成 5 个二元训练集，每个集合对应一个类标签

	类
ex_1	C_1，C_2
ex_2	C_2
ex_3	C_1，C_3，C_5
ex_4	C_2，C_3
ex_5	C_2，C_4

（续）

T_1	
ex_1	1
ex_2	0
ex_3	1
ex_4	0
ex_5	0

T_2	
ex_1	1
ex_2	1
ex_3	0
ex_4	1
ex_5	1

T_3	
ex_1	0
ex_2	0
ex_3	1
ex_4	1
ex_5	0

T_4	
ex_1	0
ex_2	0
ex_3	0
ex_4	0
ex_5	1

T_5	
ex_1	0
ex_2	0
ex_3	1
ex_4	0
ex_5	0

工程师需要决定是否要解决多数类欠抽样或是少数类过抽样的问题。

多标签样例的问题。一些域里的多标签样例非常多。因此在文本分类里，一个人很容易遇到成百上千的样例，每一个都有数以千计的属性来描述，以及有数以千计的类做标签。要从训练集里归纳那些数以千计的二进制分类器，计算会非常昂贵。的确，如此昂贵以至于要采取特别的措施来使学习变得可行。分类器的性能评估也并不简单。每一个被归纳出来的分类器可以有不同的性能，因此平均很有必要。在 11.4 节中会对此有更详细的说明。

分层组织类的说明。在一些域里（经常在多标签域中，前文有解释），这些类并不完全相互独立。一个完全分层可能存在，如图 10.4 中展示出来的一小部分，每一个这些类的归纳都需要相应地排列。例如，首先归纳的是最上面的类；然后，归纳了归入它的类，同样也最有效地利用父类的标签。

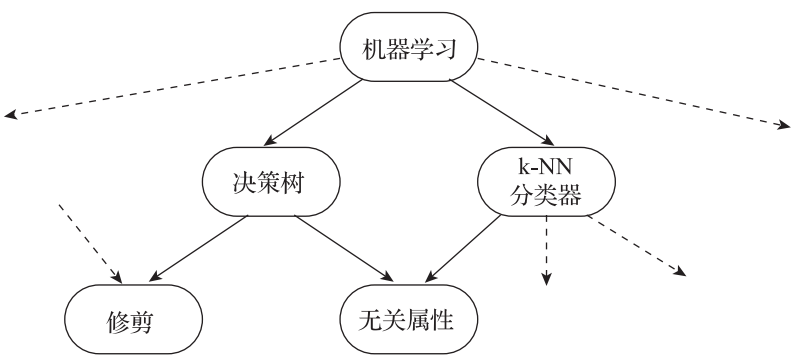


图 10.4　有时类是分层组织的，这种分层可能会提前知道，或被归纳程序发现

归纳过程并不像看起来那么简单。一两个特别的问题使任务更复杂。它们中的其中一个就是所谓的误差传播：如果一个样例被分到一个错误的域，则分类器会尝试用这个信息把它放在一个子集里，因此会把错误扩大到下一个层次。另一个很复杂的情况就是，几乎所有时候，当训练集和层次里单个结点联系在一起时，就会非常不平衡。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 请解释当要测量数据中规律性的程度时，要如何使用机器学习。请列举这类规律性比较低的域。
- 什么是多标签域？在这些域中归纳的最简单的方法是什么？工程师会遇到的典型问题是什么？
- 解释当域用层次组织的类标记时，应考虑的原则及基本问题。

10.7 总结和历史简评

- 第 7 章提供了一些数学讨论以支持关于“没有偏好就没有学习”的言论。某些实际问题使我们相信事情就是这样的。
- 有时类的含义依赖具体的语境；这个语境会随时间改变，这种情况下我们会面临时变类的问题。
- 本书前面章节中的经典机器学习技术假定训练集中的两个（或全部）类都能被充分表示。但这个要求常常不能被满足，工程师要处理不平衡训练集所带来的困难。
- 对于不平衡训练集问题的最典型的方法就是多数类欠采样以及少数类过采样。
- 在许多训练集中，一些属性值是未知的，一个可能的解决方案是用所给属

性中最常见值或平均值来代替未知属性。

- 工程师常常需要选择最合适的属性集。这里可以使用两个基本的方法：过滤器方法和包装器方法。
- 在某些领域（如文本分类），每一个样例可同时被标记为多个类别。这就是所谓的多标签域。通常来说，通过对每个类归纳一个单独的二元分类器，问题就能解决，然后平行使用所有这些分类器。
- 有时候，类是按层次排序的，这在多标签域中很常见。经常可提前知道层次。

历史简评。在机器学习中，对不同偏好加以区分是由戈登（Gordon）和德斯贾尔丁[29]（desJardin）最早想到的。终身学习的原理是斯伦（Thrun）和米切尔[75]（Mitchell）首先提出的。而特尼[77]（Turney）和卡扎特[35]（Katzetal）则是最早发表终身学习论文并产生一定影响的人。库巴特[41]（Kubat）介绍了时变概念的归纳，而威德默（Widmer）和库巴特[84]（Kubat）共同完成了一些早期算法。据本书作者所知，麦卡勒姆[49]（McCallum）发表了最早的多标签类的论文。

10.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 考虑表 10.5 中出现的训练集，如何用最常见值代替缺失的数值（问号）？为了达到这个目的如何使用决策树？
2. 一旦用确切的数值来代替表 10.5 中的问号，识别出两个属性以得到最高的信息增益。

表 10.5 “未知值”的简单练习

样例	Shape	Crust		Filling		类别
		Size	Shade	Size	Shade	
ex1	Circle	Thick	Gray	Thick	Dark	pos
ex2	Circle	Thick	White	Thick	Dark	pos
ex3	Triangle	Thick	Dark	?	Gray	pos
ex4	Circle	Thin	White	Thin	?	pos
ex5	Square	Thick	Dark	Thin	White	pos
ex6	Circle	Thick	White	Thin	Dark	pos
ex7	Circle	Thick	Gray	Thick	White	neg
ex8	Square	Thick	White	Thick	Gray	neg
ex9	Triangle	Thin	Gray	Thin	Dark	neg
ex10	Circle	Thick	Dark	Thick	White	neg
ex11	Square	Thick	White	Thick	Dark	neg
ex12	Triangle	?	White	Thick	Gray	neg

思考题

1. 课本中强调了两种基本类型错误的不同：一些由错误的偏好导致（表示的或过程的），另一种由于训练数据的变化而导致。给出一个实验过程，使工程师知道从这两种不同来源造成给定域上的整体错误率是多少。
2. 提升（Boosting）算法对基于变化的错误表现比较健壮，请解释为什么会这样。进一步地，在 9.8 节中不均匀地提升意味着减少基于偏好的错误。同样，请给予解释。
3. 假设一个贝叶斯分类器要被用于不平衡的两类域里，一个类中的样例数量远远超过另一个类。这个分类器是否会和最近邻方法一样对这个情况很敏感？请用确切的论据来支持你的观点并用一个实验验证。
4. 本节中，不平衡训练集的问题仅仅在两类域的框架中进行了探索，其中每一个样例要么是正例，要么是负例。如何将同一个问题概括到拥有超过两

种类的域里？提出一些具体的情形，使其中不平衡类在这样的多类域中是一个问题，或者不是问题。

上机实验

1. 写一个计算机程序，能够接受一个训练集作为输入，在这个训练集中有许多缺失的属性值，把改善后的训练集作为输出。这个训练集中离散属性的缺失值用最频繁值来代替，连续属性的缺失值用平均值代替。完成一个计算机程序，通过实验来确定缺失值被替换后，对从这些数据中归纳出的决策树的性能是有帮助还是有害。
2. 选择一些公共域数据，如从 UCI 资料库[㊀]中。确保这个域中至少有一个二进制值属性，本练习建议将这个二进制属性代表一个语境。把训练数据分为两个子集，每个代表不同的语境（不同的二进制属性的值）。然后从每个子集中归纳出对应的语境依赖的分类器。假设每次都清楚要使用哪个分类器，则这两个分类器和从原始训练集中归纳出来的“通用”分类器相比，性能会好多少？

㊀ 网址是 www. ics. uci. edu/ ~ mlearn/MLRepository. html。

第 11 章　性能评估

前面的章节假设在机器学习中的性能评估是一件很简单的事情。工程师只需要将归纳得到的分类器应用到已知类别的样例集合，然后计算出分类器出现的错误即可。但是实际上，事情并不是那么简单。错误率很少能全面地反映情况，而且在一些情况中它可能会起误导作用。这就是为什么工程师希望熟悉其他标准来测试分类器性能的原因。这一章的知识能够帮助工程师们选择出最佳效益行为。

对分类器的评价如此之多。这些机器学习技术在给定问题域中的适用性略有不同？将预分类样例随机分为两个子集并不是最好的方法（一个用于归纳，一个用于测试），尤其是当训练集很小的时候，随机划分的子集不能准确地代表给定的问题域。为了获得更可靠的结果，有必要重复做这些随机划分的过程。

本章着重解决了这些问题，解释了量化分类器性能的两个标准，并且探讨了一些常用机器学习算法的实验评估策略。不过对结果的统计评估问题将会在下一章中讨论。

11.1 基本性能标准

首先，我们会给出错误率和分类准确率的定义。在这之后，如果支持正确分类的证据很弱，我们将会研究拒绝这种分类的后果。

正确和错误的分类。让我们首先定义在这一章中将会使用的 4 个基本量。当用已知类别的样例来测试一个分类器时，我们仅仅需要考虑以下 4 个不同的结果：①样例是正类的，分类器也正确地将其识别为“真正类”（True Positive）；②样例是负类的，分类器也正确地将其识别为“真负类”（True Negaive）；③样例是正类的，但是分类器将其标识为“假负类”（False Negative）；④样例是负类的，但是分类器将其标识为“假正类”（False Positive）。

在将分类器应用于一个完整的样例集（真实类别已知）时，这 4 个结果中的每一个都会以不同的数目出现，而且这些数值接下来会被用在性能标准中。表示 4 种结果的符号列在表 11.1 中。具体来说，N_{TP}是真正类的数目，N_{TN}是真负类的

数目，N_{FP}是假正类的数目，N_{FN}是假负类的数目。在整个样例集合 T 中，只可能有这 4 种分类。因此，集合的大小 $|T|$ 等于它们的和，即 $|T| = N_{FP} + N_{FN} + N_{TP} + N_{TN}$。

表 11.1　性能标准定义中的基本量。例如，N_{FP}是假正类的数目，即负类样例被分类器错误地分类为正类的样例

	分类器返回的类标签	
	正　类	负　类
真实类标签：正类	N_{TP}	N_{FN}
负类	N_{FP}	N_{TN}

这里需要提醒读者注意的是，正确分类的数目等于真正类的数目加上真负类的数目，即 $N_{TP} + N_{TN}$；而错误分类的数目等于假正类的数目加上假负类的数目，即 $N_{FP} + N_{FN}$。

错误率和分类准确率。分类器的错误率 E，就是分类器在一个给定样例集上发生错误的频率。这是由不同的错误数目，即 $N_{FP} + N_{FN}$，除以样例总数，即 $N_{TP} + N_{TN} + N_{FP} + N_{FN}$计算而来。

$$E = \frac{N_{FP} + N_{FN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}} \tag{11.1}$$

有时候，工程师更倾向于用相反的数量工作，即分类准确率 Acc：分类器在一个给定样例集中正确分类的频率。分类准确率由正确分类的数目，即 $N_{TP} + N_{TN}$，除以样例的总数目计算得到。注意，$Acc = 1 - E$。

$$Acc = \frac{N_{TP} + N_{TN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}} \tag{11.2}$$

拒绝一个样例。正如本书第 8 章第 1 小节所介绍的，当讨论光学字符辨识时，如果支持正确分类的证据不够充分，那么分类器应该可以被允许拒绝分类。这个动机很简单：在某些域中，错误分类的代价比完全不分类的代价要更高。

关于拒绝分类的例子并不难找。一个分类器拒绝返回精确邮政编码的结果就是要人工操作决定信件应该发往何地。需要明确的是，这种人工处理比自动处理

更加昂贵。但是另一方面，分类器返回一个错误的邮政编码会导致信件被送到一个错误的地址，进而导致信件交付的严重延迟。后者的成本经常比“手动”处理更高。类似地，一个错误的医学诊断往往比压根不给出诊断的代价更加昂贵；缺乏的信息能通过额外的检验来弥补，但是一个错误的判断会导致选择一个弊大于利的治疗。

这就是为什么当所有类别的证据都不充分时，分类器应该偶尔拒绝对样例进行分类的原因。在一些机器学习的算法里，证据不足一词很容易定义。例如，假设在一个 7-*NN* 分类器中，4 个近邻是正类的，剩余的 3 个是负类的。最终的结果是 4:3，两种类别的数目似乎“不相上下”。更常见的是，工程师往往会为胜出类和另外一类的票选最小差异值设定一个临界值。

在贝叶斯分类器中，这个技术很容易被实现。如果样例中两个最可能类别的概率差异小于用户设置的最小差异值，那么这个样例就会因为太模糊以至于不能分类而被拒绝。在神经网络中也有类似的事情发生：通过比较相应输出神经元的返回信息——如果没有明显的胜出方，那么就要拒绝分类。

在其他的分类器中，如决策树，拒绝机制的实现则不是这样直接，而是只有实施了“额外的技巧”才有可能完成。

拒绝分类的优点和缺点。拒绝对一个样例进行分类当然会更少出错。毫无疑问这个分类器的错误率会更低。确实，拒绝分类的样例越多，错误率就越低。但是这种谨慎性不应该被夸大。错误率得以降低，几乎接近 0 似乎是一件好事。但如果这个低错误率是拒绝对几乎所有样例进行分类而得到的，那么这个分类器就变得不实用了。这两方面（低错误率和几乎不分类）中的哪一个更重要，将取决于给定应用的实际情况。

图 11. 1 形象地展示出了这类决策权衡的本质。横轴代表了一个参数调整拒绝率的能力。自左向右，拒绝率增大，然而错误率不断降低，直到极限情况，即所有样例被拒绝。从这个角度看，零错误率意味着我们有一个毫无用处的分类器。由此获得的经验是明确的。尽管偶尔拒绝不明确的样例很有意义，但是需要慎用这条原则。

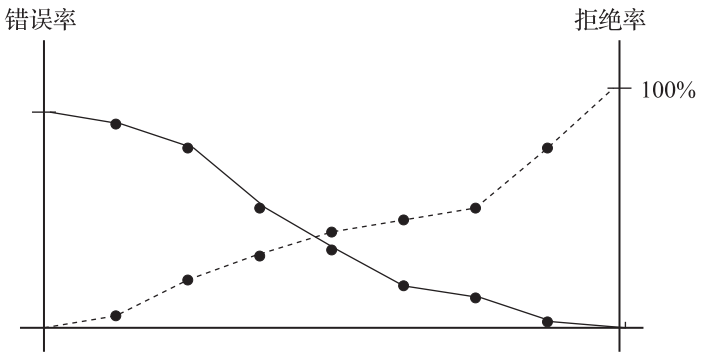


图 11.1　证据不足时允许分类器拒绝对一个样例分类，从而使错误率减小。极端情况下，所有样例被拒绝时错误率会降至 0

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 请给出假负类、假正类、真假类和真正类的定义。
- 请写出分类准确率和错误率的公式，并解释这两个准则是如何相互联系的?
- 分类器在什么时候会拒绝一个样例（如什么时候会拒绝分类)? 如何在不同类型的分类器中实现这个操作?
- 拒绝率和错误率间有怎样的联系? 两者间的平衡是什么? 在什么情况下拒绝分类是有用的? 在什么情况下拒绝分类反而有坏处?

11.2　精度和查全率

在一些应用中，负类样例的数量远远多于正类样例。当这种情况发生时，错误率给出的是对分类器性能误导性的判断。为了一探究竟，考虑所有样例中只有 2% 的样例是正的，剩下 98% 的样例都是负的的情况。一个分类器若把该集合中的任何样例都分类为负类，此时它的正确率将达到 98%——这可能看起

来是一个不凡的成绩。然而，这样一个永远不能辨识出正类样例的分类器是一无是处的。

重审不平衡类别。由于不平衡类的情况很常见，因此这种观察是值得的。在第 8 章和第 10 章我们就碰到了这样的情况，并且在其他的应用中我们也可以找到不平衡类。例如，在自动信息检索方面，用户可能想找到一篇与之相关的科学文档，这就类似是“分类器的性能评估。”对这个特别的用户来说，涉及此题材的文章仅仅占了成百上千万的电子图书馆文件中的一小部分。同样地，患某一种特殊疾病的病人在整个人口中也是相对罕见的。对于很少出现拖欠按揭付款或信用卡的欺诈使用情况来说也是一样的。一个经验丰富的工程师会认为，现实中的大部分应用在某种程度上都存在不平衡类别的现象。

在这种情况下，错误率和分类准确率几乎无法告诉我们分类器的实际效用。我们需要的是一个针对仅由很少样例代表但很重要的类别标准，而不是针对两个（或所有）类别的平均性能。下面将介绍一些标准。

精度。精度是真正类 N_{TP}，与被分类器标识为正类的所有样例 $N_{\mathrm{TP}}+N_{\mathrm{FP}}$ 的百分比。这个比值能通过式（11.3）计算得到：

$$Pr=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FP}}} \tag{11.3}$$

换句话说，精度就是分类器把一个样例标识为正类并正确的概率。

查全率。查全率是指正类样例被（分类器）正确辨认为正类的概率。因此这个概率值能由真正类的数目 N_{TP}，除以给定集合的正类样例的总数 $N_{\mathrm{TP}}+N_{\mathrm{FN}}$ 得到。

$$Re=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FN}}} \tag{11.4}$$

注意，式（11.3）和式（11.4）仅在分母上有差异。这很合理，因为精度代表真正类样例在分类器辨识出的正类样例中出现的频率，而查全率是真正类样例在样例集所有正类样例中出现的频率。

两种标准的说明。表 11.2 说明在一个包含了正类和负类两个不平衡类别的简单域中这两个标准的行为。归纳分类器在表现出高分类准确率的同时也会得到差的精度和查全率。具体来说，精度 $Pr=0.40$ 意味着在 50 个被分类器标识为正类

的样例中，仅仅有 20 个是真正类，其余的 30 个是假正类。对于查全率，情况甚至更糟糕：在测试集的 70 个正类样例中，只有 20 个被分类器正确辨识。

表 11.2　精度和查全率的说明

假设已归纳出一个分类器。对测试集的评估结果如下表所示：

	分类器返回的类标签	
	正类	负类
真实类标签：正类	20	50
负类	30	900

根据表中的数据，可以得到精度、查全率和准确率的值如下：

$$\text{精度} = \frac{20}{50} = 0.40 \quad \text{查全率} = \frac{20}{70} = 0.29 \quad \text{准确率} = \frac{920}{1000} = 0.92$$

假定修改分类器中的参数以增强它在正类样本上的表现。修改后，对测试集的评估结果如下表所示。

	分类器返回的类标签	
	正类	负类
真实类标签：正类	30	70
负类	20	880

根据表中的数据，可以得到精度、查全率和准确率的值如下：

$$\text{精度} = \frac{30}{50} = 0.60 \quad \text{查全率} = \frac{30}{100} = 0.30 \quad \text{准确率} = \frac{910}{1000} = 0.91$$

读者可以发现，精度被大大改善了，然而查全率几乎不受影响。需要注意的是，分类器的准确率也没有提高。

假定工程师决定通过修改分类器的一些内部参数来改善这种情况，并且假设这些修改能够增加真正类样例的数量（从 $N_{\text{TP}} = 20$ 到 $N_{\text{TP}} = 30$），减少假正类样例的数量（从 $N_{\text{FP}} = 30$ 到 $N_{\text{FP}} = 20$）。另一方面，假负类样例的数量也会上升：从 $N_{\text{FN}} = 50$ 到 $N_{\text{FN}} = 70$。表 11.2 中的计算表明查全率是几乎不受影响的，但是精度得以提高，从 $Pr = 0.40$ 增加到 $Pr = 0.60$。实际上，尽管精度有所提高，但分类准确率几乎保持不变，甚至会下降一些。

高精度何时重要。在某些问题域中，精度比查全率更为重要。例如，当你在一家网店买东西时，他们的推荐系统经常会回馈如“买了 X 的顾客同时也买了 Y”这样的信息。该信息的意图明显是想诱导你也购买 Y。

推荐系统有时是通过机器学习技术对公司历史数据进行分析完成的㊀。在评估它们的性能时，工程师希望得到较高的精度。这样，顾客也许会对推荐机制感到更加满意，否则很可能忽略推荐内容。

在这些问题域中，查全率的价值是不重要的。网店推荐清单的规模只能是有限的，因此即使系统只能辨认出一小部分顾客喜欢的产品也没太大关系。

高查全率何时重要。查全率同精度恰恰相反，在其他问题域中，查全率更为重要。这种情况在医药诊断中比较常见。例如，一个患 X 疾病的病人被确诊患 X 疾病，即真正类。患 X 疾病的病人没被诊断出患 X 疾病，即假负类。这是医生想要避免的一种情况——这意味着 N_{FN}应该要小。在查全率的定义 $Re=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FN}}}$中，假负类的数量在分母上，因此小的 N_{FN}意味着高的查全率。

ROC 曲线。在许多分类器中，通过微调一定的参数能修改 N_{FP}和 N_{FN}的值，因而（至少在一定程度上）会影响分类器的表现。例如，查全率的提高是以精度的降低为代价的，反之亦然。在某些问题域中，当用户需要决定这两个量哪个更重要时，这个曲线就派上用场了。

读者会发现“微调”有很多种方式。因此，在 k-NN 分类器中，工程师会将样例标识为负类，除非有充足的证据证明该样例为正类。例如，如果 7 个最近邻中有 4 个是正类，则分类器依然被指示返回负类（尽管推荐正类的样例稍多）。这样一来，假正类的数目得以减少，可是这经常意味着假负类的数目会增加。通过将任何样例都主张为负类来实现假正类数目 N_{FP}的大程度减少（假负类的数目 N_{FN}会增加），除非 7 个最近邻里有 5（或 6）个样例是正类。

相似的情况在贝叶斯分类器和神经网络中很容易实现。其中的理念是一样的：除非有充足的证据证明决策是错误的，否则样例都会被标识成首选类别。

不同参数值下分类器的表现能够通过 ROC 曲线直观展示。如图 11.2 所示，横轴代表负类样例的错误率，纵轴代表正类样例的分类准确率。图 11.2 展示了两个分类器的 ROC 曲线，c_1 和 c_2。理想情况下，我们希望曲线达到左上角，即负类

㊀ 所用到的具体技术超出本书讨论的范围，在这里不做介绍。

样例的错误率为零，而正类样例的准确率为 100%。但是由于各种各样的原因，这种情况鲜有发生。

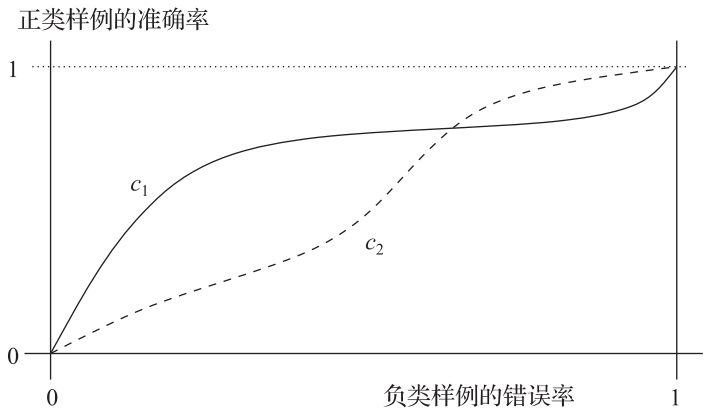


图 11.2　两种分类器 ROC 曲线的例子，c_1 和 c_2。分类器参数可用于修改假正类和假负类样例的数量

一个重要的问题是，c_1 和 c_2 哪条曲线更好。答案只能基于实际应用中的特定需求来回答。根据图像我们可以得出以下结论：就正类而言，在负类错误率低的区域，c_1 的表现优于 c_2。随着负类样例错误率的增加，c_2 在正类样例上的表现优于 c_1。再者，分类器表现的好坏取决于用户的标准。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 对什么类型的数据我们会选择使用精度和查全率来评估一个分类器的性能(而不是用错误率)?
- 计算这些标准的数值要使用什么公式？两个公式间有什么不同?
- 在什么情况下我们更喜欢高精度，而在什么情况下我们更喜欢高查全率?
- 请解释 ROC 曲线的本质。通过曲线我们还能知道有关分类器表现的哪些信息？ROC 曲线如何帮助用户对两个分类器做出选择?

11.3 测量性能的其他方法

除了错误率，分类准确率、精度和查全率，还有其他标准可用于测量性能，每一种标准反映分类器性能的不同方面。下面我们简单介绍一下其中最重要的几个方面。

精度和查全率的综合 F_β。使用两个不同的（有时是相反的）性能标准可能是不合适的，人们的初衷是想用一个数字来量化事物，特别是对于精度和查全率。工程师尝试把两个准则结合为一个量。最著名的是公式 F_β，定义如下：

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re} \tag{11.5}$$

式子中的参数 $\beta \in [0, \infty)$，用于衡量两个标准的相对重要性。如果 $\beta > 1$，那么查全率将获得更多的权重。如果 $\beta < 1$，那么精度将获得更多的权重。显而易见，当 $\beta \to \infty$ 时，F_β 能收敛于查全率；当 $\beta = 0$ 时，F_β 收敛于精度。

在多数情况下，工程师并不知道精度和查全率哪个更重要或究竟有多重要。如果出现这种情况，就会偏向于采用中立的参数值 $\beta = 1$：

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{11.6}$$

一个数值例子。假定在一个测试集上对分类器进行评估的结果如表 11.2 上部分所示，即精度和查全率的值分别为 $Pr = 0.40$ 和 $Re = 0.29$。利用这些数值，我们将会在参数 $\beta = 0.2$、$\beta = 1$ 和 $\beta = 5$ 的具体设置下计算 F_β 的值。

$$F_{0.2} = \frac{(0.2^2 + 1) \times 0.4 \times 0.29}{0.2^2 \times 0.4 + 0.29} = \frac{0.121}{0.306} = 0.39$$

$$F_1 = \frac{(1^2 + 1) \times 0.4 \times 0.29}{0.4 + 0.29} = \frac{0.232}{0.330} = 0.70$$

$$F_5 = \frac{(5^2 + 1) \times 0.4 \times 0.29}{5^2 \times 0.4 + 0.29} = \frac{3.02}{10.29} = 0.29$$

敏感性与特异性。在选择具体准则的时候经常会受到特定应用领域的影响——因为不同领域有着不可忽略的特定需求和根深蒂固的传统。因此，医学界

已经习惯通过敏感性和特异性来评价他们分类器的性能（不一定是由机器学习开发得到的）。其实，这些量只不过是分别在正类样例和负类样例上测定的查全率。让我们具体看一下：

敏感性即在正类样例上测定的查全率：

$$Se = \frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FN}}} \tag{11.7}$$

特异性即在负类样例上测定的查全率：

$$S_P = \frac{N_{\mathrm{TN}}}{N_{\mathrm{TN}} + N_{\mathrm{FP}}} \tag{11.8}$$

在机器学习的文献中，关于敏感性和特异性的介绍十分罕见，但是读者依然需要了解这两个标准。毕竟，在将机器学习应用于医疗数据时我们可能会被要求使用这两个标准，并且顾客往往就是那个最终发号施令的人。

几何平均值。当从一个不平衡类代表的域中归纳一个分类器时，工程师有时想要在正类和负类上都能有相似的性能。在这种情况下，就会使用到两个分类准确率（正类样例的和负类样例的）的几何平均值，即 *gmean*：

$$gmean = \sqrt{Acc_{\mathrm{pos}} \times Acc_{\mathrm{neg}}} = \sqrt{\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FN}}} \times \frac{N_{\mathrm{TN}}}{N_{\mathrm{TN}} + N_{\mathrm{FP}}}} \tag{11.9}$$

这里需要注意的是，几何平均值实际上是正类样例查全率和负类样例查全率的乘积，或者换句话说，就是敏感性和特异性的乘积。

几何平均值中最重要的方面大概就是：它不仅仅依赖于根号下 Acc_{pos} 和 Acc_{neg} 的值，还依赖于两个数值之间有多接近。这一点通过一个简单的数值例子就可以证明。

例如，0.75 和 0.75 的算术平均值是 $(0.75 + 0.75)/2 = 0.75$；同时，0.55 和 0.95 的算术平均值是 $(0.55 + 0.95)/2 = 0.75$。但是，前者的几何平均值是 $\sqrt{0.75 \times 0.75} = 0.75$，而后者的几何平均值是 $\sqrt{0.55 \times 0.95} = 0.72$，稍小于前者。由此可得，当两个数值不同时，几何平均值比较小；并且数值彼此间的差异越大，几何平均值的数值越小。

另一个数值例子。再者，假定分类器在测试集上的评估结果如表 11.2 上半部

分所示。敏感性、特异性以及几何平均值的计算方法如下：

$$Se = \frac{20}{50 + 20} = 0.29$$

$$S_P = \frac{900}{900 + 30} = 0.97$$

$$gmean = \sqrt{\frac{20}{50 + 20} \times \frac{900}{900 + 30}} = \sqrt{0.29 \times 0.97} = 0.53$$

成本函数。在机器学习的应用中，并不是所有的错误都受到相同的惩罚。假正类的惩罚可能比假负类更重一些，或者相反。在某些域中，两个分类器的比较会因为实际情况变得更加复杂，因为这两类错误的成本不能用同一个（至少是可比的）单元表示。

回想一下我们在第 8 章中讨论过的溢油识别问题。一个假正类的样例在这里意味着，一个“看起来像油溢出”的样例被错误地当作是油溢出（一个假正类）。当这样的情况发生时，派遣一架飞机到指定地点核实这一事件则是多余的。这次错误所引起的费用与这次飞行相关。与之相反，假负类意味着潜在的环境危险并没有被侦查出来，这时的代价就很难用金钱衡量了。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 请解释 F_β 是如何将精度和查全率联系起来的。请写出公式并讨论不同的 β 值给予精度或查全率更高权重的差异。β 取何值时会对两者给出相同的权重?
- 写出敏感性和特异性的计算公式。解释它们的本质并给出它们与查全率的关系。什么时候会用到这两个准则?
- 写出几何平均值的计算公式。解释它的本质并给出它与查全率的关系。什么时候会用到这一准则?
- 在什么情况下假正类的成本和假负类的成本有差异? 列举一个无法将它们做比较的情况。

11.4 多标签域内的性能

我们已经知道，在一些问题域中，每个样例能同时属于两个或多个分类。一个很好的例子是文本分类，其中的每个文件都可以被贴上两个或多个类标签。工程师需要知道如何在这些环境中测量归纳出分类器的性能。

宏平均。一般而言，经常采用的方法是性能准则的平均值。然而，有两种不同的方法求平均值。在宏（macro）平均中，首先计算每个分类的性能，然后求这些结果的平均值。在这种方法中，每个分类获得相同的权重。下面是精度、查全率和 F_1 的公式：

$$\begin{aligned} Pr^M &= \frac{1}{L}\sum_{i=1}^{L} pr_i \\ Re^M &= \frac{1}{L}\sum_{i=1}^{L} re_i \\ F_1^M &= \frac{1}{L}\sum_{i=1}^{L} f1_i \end{aligned} \tag{11.10}$$

微平均。另一个不同的方法由微平均提出。在这里，每个分类根据其在样例中出现的频率得到相应的权重。在这个方法中，性能是对所有例子的平均，就好像它们全属于一个类。下面是精度、查全率和 F_1 的公式：

$$\begin{aligned} Pr^{\mu} &= \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FP_i} \\ Re^{\mu} &= \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FN_i} \\ F_1^{\mu} &= \frac{2 \times pr^{\mu} \times re^{\mu}}{pr^{\mu} + re^{\mu}} \end{aligned} \tag{11.11}$$

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 列举一个多标签域的例子。
- 给出精度、查全率和 $F1$ 的宏平均公式。
- 给出精度、查全率和 $F1$ 的微平均公式。

11.5 学习曲线和计算开销

本章的前 4 个小节解决了由归纳得到的分类器的性能评价问题。本小节我们将把注意力转向对学习算法本身的评价。给定的归纳技术在计算上的有效性如何？归纳出的分类器有多好？如果我们选择其他的机器学习框架，能否得到更优的结果？

在这一小节，我们将研究机器学习的成本，成功归纳所需样例的数量和需要花费的计算时间。另一方面，下一节将介绍具有较高分类性能工具的能力。

学习曲线。当评估一个人学习如何解决某个具体问题的能力时，心理学家会依靠学习曲线（learning curve）来完成，机器学习也借用了这个概念来达成目标。

从我们的角度来看，学习曲线仅仅表明了训练集的大小如何影响归纳分类器的分类性能。图 11.3 展示了两条这样的曲线，横轴代表训练样例的数量；纵轴代表这些样例所归纳出的分类器的分类准确率。通常，我们评估的是独立测试样例的分类准确率，但也并不总是如此。

在大多数情况下，训练集越大意味着更高的分类性能——至少可以改进到不能继续优化为止。然而在理想情况下，我们更想从最小的合理训练集中得到最大的性能。这是因为，一方面，获得训练样例的成本可能十分高昂，有时可能无论付出多大的代价都得不到。另一方面，我们使用的样例越多，归纳所需的计算开销就越大。

比较具有不同学习曲线的学习器。图 11.3 所示是两个学习器的学习曲线，l_1

和 l_2。读者可以看出前一条曲线 l_1 上升迅速，在快速达到某一数值后基本不再上升，达到平衡——这个限制可能因为一个错误的偏见被强化（见 10.1 节）。比较而言，第二个学习器的学习曲线 l_2 增长较缓，但最后达到了比 l_1 更高的准确率。哪条学习曲线代表了更胜一筹的学习器取决于所给定应用的实际情况。当训练样例的资源有限时，显然第一个学习器更适合。如果样例充足，我们更青睐于另一个学习器，当然在这种情况下，计算开销是很大的。

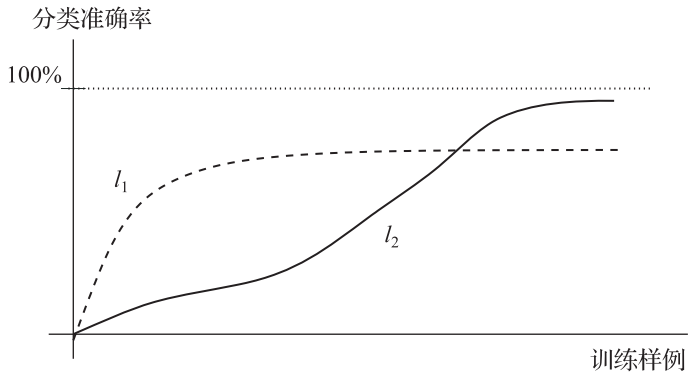


图 11.3　学习曲线展示了分类准确率如何依赖于学习过程中可用样例的数量

计算开销。计算开销包括两个方面。第一，从可用数据中归纳出分类器所需要的时间。第二，归纳出分类器对一系列样例进行分类所需要的时间。按照这些原则，本书所介绍的技术涵盖范围很广。关于归纳成本，最低廉的是 k – NN 分类器的基本版本：唯一包含的“计算”是用来储存训练样例的㊀。但是另一方面，k – NN 分类器的分类成本高昂。

例如，如果我们有一百万个训练样例，每个样例都有一万种属性描述。我们不得不进行数十亿次的算数运算来分类单个样例。当要求分类上百万个样例时，即使是一个非常快的计算机也要花费很长一段时间。

在决策树中的情况有些不同。分类样例的成本低廉，通常只需要适量的单属性检验。然而，如果可用的训练样例被多种属性所描述，那么决策树归纳依然需

㊀ 如果要去除噪声样例和多余样例，则一定量的计算十分重要。

要花费很多时间。

每个分类器的归纳成本和分类成本各不相同，工程师在为给定的应用选择最合适的机器学习范例时，会很深思熟虑地考虑这些成本。同样重要的是，工程师也需要透彻理解这些成本是如何被训练样例的数目和描述样例的属性数量所影响的，有时成本还会依赖于用户要求的准确率（如神经网络）。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 给定的机器学习范例主要有哪两方面的成本?
- 学习曲线告诉了我们机器学习算法的哪些行为？在理想情况下，我们想要得到什么形状的学习曲线？实际上我们又该期待什么形状呢?
- 在什么环境中，一个具有较低最大值的陡峭学习曲线会优于一个具有较高最大值的平缓曲线?

11.6 实验评估的方法

我们都知道不同的问题域受益于不同的归纳技术。通常情况下，选择分类器并不困难，并且一些可用的训练数据的知识，常常可以帮助我们选择最合适的范例。例如，若无关属性的比例很高，那么决策树可能比最近邻分类器的效果更好。然而，一个给定技术的成功也取决于各种参数的值。尽管某些已经经过时间考验的经验法则能有所帮助，但最好的参数设置通常是通过实验发现的。

基准方法及其局限性。基本情况十分简单。将预分类的样例集合划分为两个子集，一个用于训练，另一个用于测试。使用不同的参数设置，不断重复训练－测试过程，把其中性能最高的那一个选出来。然而，这只有在可用的预分类样例非常多的情况下才能实现。在缺乏样例的问题域中，或当获取样例的成本高昂时，随机地将样例划分为一对训练集和测试集将缺乏客观性。偶然情况下，两个子集中的任何一个都可能足以歪曲给定的问题域。统计学家告诉我们，训练集和测试

集都应该或多或少具有分布相同的样例。但是在小数据集中，这是无法保证的。

随机子采样。当预分类样例集很小时，工程师通常乐意多次重复训练 – 测试过程。在每次运行中，样例集被随机分成两部分，一部分用于训练，另一部分用于测试。每次运行都记录分类器表现出的性能，并且求其平均数。这里需要注意的是，单个的数据分割彼此相互独立。一旦过程重复 N 次（通常，$N = 10$ 或 $N = 5$），将会以平均分类准确率和标准差的形式给出结果，如 84.2 ± 0.6。关于平均值和标准差的计算，我们使用第 2 章的公式：平均数 μ，用式（2.12）计算；标准差 σ，是方差 σ^2 的平方根，用式（2.13）计算。

***N* 折交叉验证**。对于更高级的统计评估，经验丰富的工程师常常青睐于 N 折交叉验证，原理如图 11.4 所示。首先，预分类样例集被分为 N 个规模相同（或近乎等分）的子集—机器学习术语（并不完全准确），有时称之为“折”。

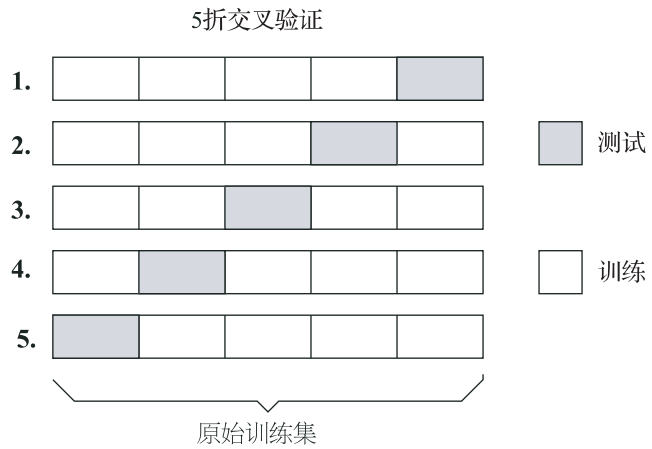


图 11.4　*N* 折交叉验证将训练集划分成 *N* 个规模相同的子集。N 次实验操作的任何一次，都会保留一个不同的子集进行测试，在剩余的 N – 1 个子集的集合中进行训练

N 折交叉验证进行 N 次实验。每次试验中，都会移除 N 个子集中的一个，其仅用于测试（这保证了每次运行中都会使用一个不同的测试集）。然后在剩余的 $N - 1$ 个子集的集合中进行训练。此外，结果要取平均值，并且计算标准差。

比起随机子抽样，N 折交叉验证的优势是测试集是分离的（不折叠的），这在某些统计评估（见第 12 章）中被认为是一大优势。

分层法。假设你要应对的是一个包含 60 个正例，940 个负例的域。如果你用

$N=10$ 的 N 折交叉验证，每一折可能包含不同数量的正例。如果按照平均值，则每折中将有 6 个正例，但在实际操作中具体的数目可能不同。在一个域中，当一个类的样例相当少时，有些“折”甚至根本不包括一个正例。这种情况在有严重不平衡类存在的域里是经常出现的。

在这种情况下，经验丰富的工程师更喜欢分层（stratified）法，而不是 N 折交叉验证。这个想法是为了确保 N 折的每一折都包含各个分类的（大致）相同样例代表。例如，在 60 个正例和 940 个负例构成的域中，当采用 5 折交叉验证时，每折都应包含 200 个样例，且其中有 12 个正例。

相同的原则经常在随机子抽样法中使用，但是在分层法中不再完全随机。此外，还需要确保每一个训练集和测试集中均包含每个分类的相同代表。

5×2 交叉验证（5×2 cv）。另一个机器学习技术的实验评估方法是 5×2 交叉验证，有时简写为 5×2cv。这可能是最受欢迎的机器学习方法。该方法的原理是建立在随机抽样组合和 2 折交叉验证上的。

具体来说，5×2 cv 将预分类样例集分成规模相同的两部分，T_1 和 T_2。接下来，用 T_1 训练、用 T_2 测试，然后互相交换再进行一次。将这个过程不断重复 5 次，每次都以不同的随机方法划分出两个子集。最终，我们得到了 10 个训练－测试组合。表 11.3 中的伪代码总结了其原理。

表 11.3　5×2 交叉验证算法（5×2 CV）

令 T 为预分类的原始集合。

1. 将 T 随机分成两个规模相同的子集，按同样的方法划分 5 次，最后我们得到 5 对子集，标记为 T_{i1} 和 T_{i2}（$i=1, 2, \cdots, 5$）。
2. 每一对子集，用 T_{i1} 训练、用 T_{i2} 测试，再交换角色，用 T_{i1} 测试、用 T_{i2} 训练。
3. 因此我们得到了 10 个训练－测试组合，计算所选性能标准的平均值和标准差。

许多工程师更喜欢使用这种方法的分层版本，以确保实验中的每部分都含有每个类别的相同代表。

没有免费的午餐定理。期待某些机器学习技术在所有的环境中都被优先选择是可笑的。不会有这样的事情存在。读者现在了解了每一个模型都有自己的优点，使得能够成功用于一些问题域，但也都也有一些缺点，使得在其他问题域中常常

失败。只有通过系统的实验，工程师才能知道应该选择哪种分类器和哪个归纳算法来完成当下的任务。事实是，没有任何一种机器学习方法在任何环境中都优于其他方法。

数学家已经用严谨的论证证明了这种观点的有效性，并将结果命名为（略显奇特）“没有免费的午餐定理”。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- N 折交叉验证和随机子抽样的区别是什么？为什么我们有时更喜欢使用这些方法的分层版本?
- 解释 5×2 交叉验证（5×2cv）和其分层版本的原理。
- 我们从没有免费的午餐定理中学到了什么?

11.7 总结和历史简评

- 衡量分类性能的基本标准是错误率 E，其定义是在给定集合中被错误分类的样例的百分比。与它互补的量为分类准确率，$Acc=1-E$。
- 当任何一个分类的证据都不够有力时，分类器最好拒绝该样例以避免昂贵的错误分类成本。此时，拒绝率成为另一种评估分类性能的重要标准。拒绝率越高往往意味着错误率越低；然而，一旦超过某一点，分类器的效用就会降低。
- 评估分类性能可以通过对真正类、真负类、假正类、假负类（表示为 NTP，NTN、NFP、NFN）分别计数来定义。
- 在存在不平衡类的问题域中，错误率可能是一个误导性的标准。这时我们使用精度$\left(Pr=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FP}}}\right)$和查全率$\left(Re=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FN}}}\right)$的效果更佳。
- 有时，将精度和查全率结合成一个单准则 F_β，定义如下：

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re}$$

由用户设定的参数 β 的数值决定了精度（$\beta > 1$）和查全率（$\beta < 1$）的相对重要性。当认为二者同样重要时，我们令 $\beta = 1$，即：

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$$

- 分类性能有少量共同的标准，包括敏感性（sensitivity）、特异性（specificity）和几何平均值（gmean）。
- 当一个样例可能同时属于多个分类的域时，常用每个类别性能的平均值来评估分类器的性能。有两种求平均值的方法：微平均和宏平均。
- 机器学习技术另一个重要的方面是，达到某一分类性能需要多少个训练样例。有时借助学习曲线可以得到直观的答案。值得注意的是，与归纳和分类有关的计算成本。
- 在预分类样例数量有限的问题域中，工程师在比较可选的机器学习技术时，应当使用随机子抽样、N 折交叉验证和 5×2 交叉验证方法。这些技术的分层版本确保了每个训练集（和测试集）中每个分类的样例比例是相同的。

历史简评。本章所讨论的大部分性能标准在统计学文献中早已出现并使用了很久，很难追根溯源。一个例外是几何平均数（gmean），它相对较新，由库巴特（Kubat）、霍尔泰（Holte）和马特温[42]（Matwin）提出。

赫尔曼[31]（Hellman）提出：如果在一个问题域中 k – NN 分类器不能根据大多数样例做决定，则拒绝分类。对此路易祖（Louizou）和梅班克（Maybank）进行了后续分析。迪特里奇[18]（Dietterich）提出并通过实验研究了 5×2 交叉验证的原理。沃伯特[87]（Wolpert）提出了没有免费的午餐定理。

11.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者

从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 假定在测试集上对分类器评估的计数结果如下表所示：

	分类器返回的类标签	
	正　类	负　类
真实类标签：正类	50	50
负类	40	850

计算精度、查全率、敏感性、特异性、几何平均值。

2. 使用第 1 题中的数据计算在不同的参数值（$\beta=0.5$、$\beta=1$ 和 $\beta=2$）下 F_β 的值。

3. 假定使用 5 折交叉验证评估一种机器学习技术，在测试集中测量得到的错误率结果如下：

$$E_{11}=0.14\ ;\ E_{12}=0.16;\ E_{13}=0.10;\ E_{14}=0.15;\ E_{15}=0.18$$

$$E_{21}=0.17\ ;\ E_{22}=0.15\ ;\ E_{23}=0.12;\ E_{24}=0.13;\ E_{25}=0.20$$

使用第 2 章的公式，计算错误率的平均值和标准差 σ（不要忘记标准差是方差 σ^2 的平方根）。

思考题

1. 列举出一个精度比查全率更重要的问题域；相反地，列举出一个查全率比精度更为重要的问题域。

2. 成对的敏感性和特异性反映了给定问题域的哪些方面？举例说明敏感性和特异性对于分类器性能的衡量要优于精度和查全率的情况。

3. 假定在某个问题域中，你要归纳出两个分类器：一个是精度很高，另一个是查全率很高。如果结合这两种分类器，我们能获得什么利益？如何实现这种组合？在什么环境中这种结合不能奏效？

4. 与 N 折交叉验证相比，思考一下随机子抽样的潜在优缺点。

上机实验

1. 假设一些机器学习实验得出了一张结果表，其中每一行代表了一个测试样例。第一列是样例类标签（“1”或“0”分别代表了正类样例和负类样例），第二列是分类器返回的类标签。

   1）编写一个程序，计算精度、查全率和用户设定的β值下的F_β。

   2）编写一个程序计算其他性能标准的值。

2. 假定有一个矩阵形式的训练集，其中每一行代表一个样例，每一列代表一个属性，最右边一列是类标签。编写一个程序，按照5×2交叉验证技术的要求，将这个集合随机分成规模相同的5对子集。然后编写一个程序使用分层方法创建子集，并且要求产生的每个子集中的每个分类均有大致相同的样例。

3. 编写一个程序，该程序可以接受两种输入：①多标签测试样例的一系列类标签；②多标签分类器分配给这些样例的标签。输出结果由精度和查全率的微平均和宏平均组成。

4. 编写一个程序，实现以下功能：输入一个训练集，输出可用于N折交叉验证的N个子集，并且保证这个过程使用的是分层法。如果决定使用5×2交叉验证而非一般的N折交叉验证，则需要如何修改程序？

第 12 章　统计显著性

假设你要在一个独立测试集上评估分类器的性能。在多大程度上可以信任取得的结果？当抛 10 次硬币有 8 次是头像时，任何人都会认为是巧合，觉得再抛 10 次结果会更接近真实情况。在度量分类器性能时的情况类似。只在一个测试集上评估分类器的精度是不够的，需要应用概率知识使得测量值是分类器真正表现的可靠估计。

数学统计可完成这个工作。为使学生熟悉必备的技术和过程，本章介绍一些基本概念，如标准误差、置信区间、假设检验，并从机器学习任务的角度加以解释和讨论。

12.1 总体抽样

如果我们在多个不同的测试集上测试分类器，它们的误差率会不一样，但也不是完全任意的：测量到的值的分布不会脱离统计规律。恰当理解这些规律有助于我们估计这些测量结果的代表性究竟如何。

一次观察。表 12.1 包括了 100 个 0 和 1，它们由随机数发生器生成，随机数发生器的参数设为输出中 0 占 20%，1 占 80%。当然实际的输出结果和设定的有轻微差别。本次输出中，有 82 个 1，18 个 0。

表 12.1 随机数发生器生成的一组二进制数，结果中的 1 占 80%；实际有 82 个 1，18 个 0；在行和列的末端是对应的汇总

0	0	1	0	1	1	1	0	1	1	6
1	1	0	1	1	1	1	1	1	1	9
1	1	1	0	1	1	1	1	1	1	9
1	1	1	1	1	1	0	0	1	1	8
1	1	1	0	1	0	1	0	1	1	7
1	1	1	1	1	1	1	1	1	1	10
1	1	1	1	1	1	1	1	0	1	9
1	1	1	0	1	1	1	0	1	1	8
1	1	1	0	1	0	1	1	1	1	8

（续）

1	0	1	1	1	1	1	0	1	1	8
9	8	9	5	10	8	9	5	9	10	82

表格右侧和底部的数字告诉我们每行和每列各有多少个 1。根据这些信息，可以发现在前两行中 1 所占的比例分别是 0.6 和 0.9，因为每行有 10 个数字。类似地，前两列中 1 所占的比例为 0.9 和 0.8。这 4 个比例的平均值为（0.6 + 0.9 + 0.9 + 0.8）/ 4 = 0.80，标准差是 0.08[㊀]。

对统计学家来说，每行和每列代表了对总体的一个**抽样（sample）**。所有样本有相同的尺寸：$n = 10$。现在假设增加这个值，如 $n = 30$。那么比例会怎样分布？

回到表中，前 3 行共包括 24（6 + 9 + 9）个 1，接下来 3 行包括 25（8 + 7 + 10）个 1，前 3 列包括 26（9 + 8 + 9）个 1，接下来 3 列包括 23（5 + 10 + 8）个 1。把这些数除以 $n = 30$，可得到如下的比例：$\frac{24}{30} = 0.80$，$\frac{25}{30} = 0.83$，$\frac{26}{30} = 0.87$，$\frac{23}{30} = 0.77$。计算这 4 个值的平均值和标准差可得 0.82 ± 0.02。

比较 $n = 10$ 和 $n = 30$ 的结果，有两点值得注意。第一，大样本的均值（0.82）和小样本的均值（0.80）有微小的差异。第二，大样本的标准差明显要小：$n = 30$ 时为 0.02，$n = 10$ 时为 0.08。这些情况是偶然造成的，还是某些内在规律产生的结果？

基于随机样本的估计。答案来自于这样的理论，即样本尺寸 n 越大，估计的精度就越高。更进一步，样本尺寸越大，样本估计的方差就越小。

另一个所谓“中心极限定理”的理论，说明单个估计的分布可以用第 2 章中学过的高斯正态分布来近似，读者可以回忆那个像钟一样的形状。不过，这个近似只有在比例 p 和样本尺寸 n 满足下面两个条件时才是相当精确的：

$$np \geqslant 10 \tag{12.1}$$

$$n(1-p) \geqslant 10 \tag{12.2}$$

如果条件不满足（如果至少 1 个乘积小于 10），则从样本中得到的估计的分

㊀ 回忆一下，标准差是方差的平方根，可根据第 2 章式（2.13）计算。

布将不能在没有精度损失的情况下用正态分布来近似。

12.2 节和 12.3 节将详细说明如何用正态分布的近似来帮助我们建立置信度，以测量分类器的性能。

一个例子。我们看一看，在表 12.1 中所示的样本情况下，这些条件是如何被满足的。我们知道，原始总体中 1 的比例是由用户设定的随机数发生器的参数决定的：$p=0.8$。我们从样本尺寸 $n=10$ 开始。结果显示两个条件都不满足，因为 $np=10\cdot 0.8=8<10$，并且 $n(1-p)=10\cdot 0.2=2<10$。在这么小的样本中观察到的比例，其分布不能用正态分布来近似。

第二次尝试中，样本尺寸增加到 $n=30$。得到 $np=30\cdot 0.8=24>10$，这时条件（式（12.1））满足了。然而，因为 $n(1-p)=30\cdot 0.2=6<10$，条件（式（12.2））不满足。于是，即便这样，正态分布仍不能提供足够的精度近似。

当样本尺寸增加到 $n=60$ 时情况将发生变化。计算可得 $np=60\cdot 0.8=48\geqslant 10$，同样 $n(1-p)=60\cdot 0.2=12\geqslant 10$。于是我们可以认为，当样本尺寸 $n=60$ 时，1 的比例的分布可以用正态分布去近似，不会有明显的精度损失。

***p* 的影响**。注意正态分布的适用性如何受到 p 的影响，这里 p 是 1 在总体中的分布。容易发现对于不同的 p 值，需要不同的样本尺寸以便满足两个条件。当 $p=0.5$ 时，较小的样本就够了；但是，比例值向 $p=0.5$ 的两侧离开得越远，所需的样本尺寸越大。

为了更好地理解实际中的含义，回忆一下，当 $p=0.8$ 时，样本尺寸 $n=60$ 就够了。那么，如果样本尺寸是 60，但是比例更高，如 $p=0.95$，则会怎么样？这时我们会发现，$n(1-p)=60\cdot 0.05=3<10$，意味着条件（式（12.2））不满足。这个样本中比例的分布不能用正态分布来近似。这时为了满足条件，要求样本尺寸至少为 $n=200$。因为 $200\cdot 0.05=10$，刚刚达到。另外要注意，因为条件（式（12.1））和（式（12.2））是对称的，所以对 $p=0.05$ 和 $p=0.95$ 两种情况，所需的最小样本尺寸是一样的，为 $n=200$。

分布的参数。再回到曾经讨论过的问题，根据样本来估计 1 的比例。我们现在知道，如果样本足够多，则在不同样本上得到的估计的分布可以用正态分布来近似，其均值等于整个总体上的 1 的（理论）比例，如果可以完成这

样的实验。

分布的另一个参数是标准差（standard deviation）。在我们的语境中，统计学家更喜欢用术语：标准误差（standard error），这两个术语的意思有微妙的差别：标准差指**任意变量**的分布（如体重、年龄或温度），而标准误差用于从一个抽样到另一个抽样所得到的估计的变化。这也是我们讨论比例时感兴趣的地方。

用 S_E 表示标准误差，数学家已经给出了根据样本尺寸 n，理论比例 p 来计算它的公式：

$$S_E = \sqrt{\frac{p\ (1-p)}{n}} \tag{12.3}$$

例如，如果 $n=50$，$p=0.8$，则标准误差为：

$$S_E = \sqrt{\frac{0.80 \cdot 0.20}{50}} = 0.06$$

当用日常语言来表达时，人们会说标准误差是6%。

***n* 的影响；收益递减。**注意，标准误差的值和样本尺寸 n 的变化相反。具体来说，样本尺寸越大，标准误差就越小，反之亦然。因此当 $n=50$ 且 $p=0.8$ 时，$S_E=0.06$。如果用更大的样本，如 $n=100$，则标准误差将会降到 $S_E=\sqrt{\frac{0.8 \cdot 0.2}{100}}=0.04$。正态分布曲线于是变得更窄，不同样本中 1 的比例将更加趋近 p。

这就是说，我们应发现增加样本尺寸会引起收益递减（diminishing return）。让我们用一个简单的例子来说明这句话。前一段的计算结果告诉我们，当从 $n=50$ 变到 $n=100$ 时（样本加倍），我们使 S_E 减少了两个百分点，即从6%降到4%。而如果用 $n=1000$ 做同样的计算，可得 $S_E=0.013$，而 $n=2000$ 时 $S_E=0.009$。换句话说，把样本尺寸从1000加倍到2000，只能让标准误差从1.3%降到0.9%，也就是说，加倍样本只能产生微不足道的0.4%的变化。

值得记住最后这个例子，因为它很有用。在很多问题中，预分类的例子获得起来很难或非常昂贵。读者回忆一下，8.2 节中讨论的漏油问题就是这种情况。如果用较小的测试集就可以得到可以接受的估计，那么就不用再费力去找其他的例子了，得到的那一点好处也许抵不上额外的花销。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 如果从大量随机样本中得到的比例的分布符合正态分布，则比例应满足一定条件，写出定义这些条件的公式。
- 解释比例 p 如何影响样本尺寸 n，使得在不同样本上的比例满足正态分布。
- 基于不同样本的一组估计值的均值是什么？另外，写出计算标准误差（standard error）的公式。
- 解释这句话的含义："增加样本的尺寸导致收益递减"。

12.2 从正态分布中获益

上一节研究了来自某总体的样本中 1 的比例。样本尺寸用 n 表示，在整个总体中 1 的理论比例表示为 p。这个理论值未知，我们只能通过对样本的观察来估计它。另外我们知道，尽管每个样本的比例值是不同的，但这些值的分布通常可以用正态分布来近似——当满足条件（式（12.1））和（式（12.2））时这个近似是相当精确的。

正态分布可以帮助我们判断在一个训练集上得到的分类器的精度（或其他的性能指标）有多么可信。为此，我们简单了解一下如何计算所谓的置信度（confidence values）。

分类器性能的重新表示。在包含 100 个实例的测试集上评估分类器的性能（100 是表 12.1 中项的个数），假设 1 和 0 代表正确和不正确的分类。这里 1 的比例给出了分类器的精度，0 的比例定义了错误率。

在不同的测试集上评估分类器当然会得到不同的分类精度或错误率。但当测试集的数量很大时，精度值的分布，如我们所见，在某种程度上大致服从正态分布。

正态分布的性质。图 12.1 显示了正态分布的基本形状。纵轴代表第 2 章中提

到的概率密度函数，水平轴表示分类精度。均值用 p 表示，是理论分类精度，用问题域中所有的实例对分类器进行评估时可得到，显然它是未知的，这也是为什么我们想基于具体的样本（即可用的测试集）对它进行估计。

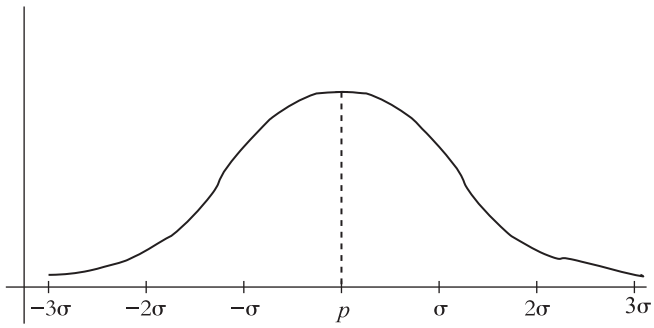


图 12.1　均值为 p 的高斯（正态）分布

类似钟形的密度函数说明在大多数测试集上得到的分类精度相当接近均值 p。离 p 越远的值，从一个随机测试集中得到它的概率越小。也注意到沿图形水平轴标出了距 p 的几个特定的距离：分布的标准差 σ 的倍数，或是基于样本的估计的标准误差的倍数。

正态分布的公式在 2.5 节中已介绍过，它被称为高斯“钟形”函数。根据这个公式，可以知道在特定区间［a，b］内的值的百分比。曲线下（从负无穷到正无穷）全部的面积是 1。因此如果区间［a，b］之上的面积是 0.8，则称在该区间里包括了 80% 的估计值。

标出感兴趣的区间。不是所有的区间都同等重要。为了评估分类器，我们对围绕均值 p 的那些区间感兴趣。例如，人们希望知道在［$p-\sigma$，$p+\sigma$］内的值占多少百分比。反过来，人们也许想知道包括了 95% 的全部值的区间（还是围绕 p）有多大。

严格来说，可通过数学分析来解答这类问题。幸运的是，不需要我们算，其他人已经算过了，用他们的结果即可。一些最常用的结果列在表 12.2 中。其中左边列中的百分比是置信水平（confidence level）；每行右边给出了组成这些百分比值的区间。注意到区间的宽度用 z^* 来标记，即两边到 p 的标准差的个数。更正式一些，可定义这些区间为［$p-z^*\sigma$，$p+z^*\sigma$］。

表 12.2 对均值为 p，标准差为 σ 的正态分布，左列给出了区间 $[p-z*\sigma, p+z*\sigma]$ 中的值所占的百分比

置信水平（%）	z^*
68	1.00
90	1.65
95	1.96
98	2.33
99	2.58

这里说明一下如何在实际中使用这个表。假设我们想知道包含了所有值中95%的区间大小。这个百分比值在第三行。右边的数字是1.96，意思是有95%的值落在区间 $[p-1.96\cdot\sigma, p+1.96\cdot\sigma]$ 内。类似地，68%的值在区间 $[p-\sigma, p+\sigma]$ 内，这从表中第一行可得到。

基于抽样估计的标准误差。我们看一下评估分类器精度时怎样使用这些知识。假设测试集都有相同的大小 n，并且这个 n 满足条件（式（12.1））和（式(12.2)），于是可以用正态分布。已知在独立测试集上得到的分类器的精度的均值覆盖了理论精度，也就是说，在所有实例上测试分类器时出现过理论精度。

标准误差[㊀]可用式（12.3）计算。例如，如果理论分类器的精度是 $p=0.70$，测试集的尺寸 $n=100$，那么从大量不同测试集中得到的分类器精度的标准误差可计算如下：

$$S_{Acc}=\sqrt{\frac{p\ (1-p)}{n}}=\sqrt{\frac{0.7(1-0.7)}{100}}=0.046 \tag{12.4}$$

四舍五入后，我们说分类器的精度为70% ±5%。再次提醒，如果用大的测试集，则标准误差会降低。也就是说，测试集越大，评估越彻底，因此得到的置信度就越高。

现在我们问，如果在另外同样大小的测试集上评估分类器，会得到什么样的

㊀ 如12.1节所述，对从不同的样本中获得的结果的分布，我们更喜欢用术语标准误差而不用更一般的标准差。

值呢？再一次我们用表 12.2 来回答这个问题。首先，找到代表 95% 的行，这行右边的列给出值 $z^* = 1.96$，意思是 95% 的结果落在区间 $[p - 1.96 \cdot S_{Acc}, p + 1.96 \cdot S_{Acc}] = [0.80 - 1.96 \cdot 0.46, 0.80 + 1.96 \cdot 0.46] = [0.61, 0.79]$ 之内。

不要忘记，这些只有在测试集的大小相同，即 $n = 100$ 时成立。对于不同的 n，式（12.4）将给出不同的标准误差 S_{Acc}，于是有不同的区间。

两个重要的提醒。一个提醒是正态分布这个假定究竟有什么好处。特别是，如果分布是正态的，则从表 12.2 中我们就可以知道包含了给定百分比值的区间，它的尺寸是多大（以 p 为中心）。

另一方面，标准误差公式（式（12.3））总的来说是有效的，即使分布不是正态的。计算标准误差时，两个条件（式（12.1））和（式（12.2））不是必须满足的。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 怎样将前一章中的内容用于评估导出分类器的性能？
- 从表 12.2 中我们能够得到什么类型的信息？从大小为 n 的测试集中得到分类精度，如何用这个表得到该精度的置信度？
- 如何计算一个给定测试集上的估计的标准误差？这个标准误差如何依赖测试集的尺寸？

12.3 置信区间

现在我们看一看，用前两章所学的知识，对分类器在指定测试集中表现出的性能，我们可以有多少信心。

置信区间：例子。现在我们理解，在不同的测试集中，分类精度是以某种分布存在的。现在想知道的是，在一个具体的测试集上测到的值和理论值之间接近程度如何，对这个结论我们有多大的信心。

假设测试集的尺寸为 $n=100$，令 $Acc=0.85$ 为在这个测试集上测得的分类精度，对这样的大小的测试集，标准误差为：

$$S_{Acc}=\sqrt{\frac{0.85\cdot 0.15}{100}}=0.036 \tag{12.5}$$

检查正态分布条件我们发现，它们都符合，因为 $100\cdot 0.85\geqslant 10$ 并且 $100\cdot 0.15\geqslant 10$。这意味着我们可以用表 12.2 中的 z^* 值。从表中我们可以很容易地找到 95% 的值在 $[Acc-1.96\cdot S_{Acc},\ Acc-1.96\cdot S_{Acc}]$ 区间内。对 $Acc=0.85$ 及 $S_{Acc}=0.036$，我们发现对应的区间是 $[0.85-0.07,\ 0.85+0.07]=[0.78,\ 0.92]$。

这个结果是说，在给定的测试集上，分类精度的真值落在 $[0.78,\ 0.92]$ 区间中的某处的可信度是 95%。

两个新术语：置信水平和误差界限。置信区间反映了特定的**置信水平**。它们定义为百分比的值，列在表 12.2 左侧。在我们特定的例子中，置信水平是 95%。

每个置信水平定义了不同的置信区间。这个区间可以被写为 $p\pm M$，其中 p 是均值，M 是所谓的**误差界限**。例如，区间 $[0.78,\ 0.92]$ 的均值为 $p=0.85$，误差界限为 $M=z^*S_{Acc}=1.96\cdot 0.036=0.07$。

选择置信水平。在上面的例子中，置信水平 95% 是相当常见的选择。对于其他的置信水平，可得到另外的置信区间。如果是 99%，表 12.2 给出 $z^*=2.8$，置信区间为 $[0.85-2.58\cdot S_{Acc},\ 0.85+2.58\cdot S_{Acc}]=[0.76,\ 0.94]$。注意这个置信区间比 95% 的要长。可以预料，分类精度的理论值落在一个长的区间中的概率更高。反过来，理论值不太可能落在一个窄的区间中。对置信水平 68% 也是这样（标准误差近似为 $S_{Acc}=0.04$），置信区间是 $[0.85-0.04,\ 0.85+0.04]=[0.81,\ 0.89]$。

要记住的很重要的一点是，即使在置信水平为 99% 的情况下，也不能绝对肯定地说理论值一定会落在相应的区间中。仍有 1% 的可能，测到的值落在区间的外面。

另一个参数：样本尺寸。现在读者理解了置信区间的长度依赖于标准误差，而标准误差接下来又依赖于测试集的尺寸 n（见式（12.3））。本质上，测试集越大，支持测量值的证据越强，于是置信区间越窄。这就是为什么我们说误差界限

和测试集的尺寸是相反的关系：测试集尺寸增大，误差界限减小。

前面我们提到高的置信水平导致更长的置信区间。如果我们觉得这个区间太大，则可以通过使用大的测试集来缩小它，也就是更大的 n 值（它可减少测量值的标准误差）。

如果我们想限定误差界限到某个最大值，有一个方法可以决定测试集应该多大。如下是一个计算误差界限的公式：

$$M = z^* S_{Acc} = z^* \sqrt{\frac{p\ (1-p)}{n}} \tag{12.6}$$

解出公式中的 n（指定 M、p、z^* 的值）可给出所需的测试集尺寸。

结束语。用最简单的性能准则及分类精度，解释了对给定置信水平确定置信区间的方法。不过这个方法的适用范围更宽：代表比例的任何变量的不确定性都可被量化。在机器学习上下文中，可用同样的方法设计第 11 章中的性能标准的置信水平，可以是精度、召回率或其他的量。

但必须小心才能做得对。一定不要忘了给定量的值的分布只在条件（式（12.1））和（式（12.2））被满足时才能用正态分布来近似。其次，当用式（12.3）计算标准误差时，要确保理解 n 的含义。例如，读者还记得用公式$\frac{N_{TP}}{N_{TP}+N_{FP}}$计算精度：所有样本中真阳性的百分比被分类器标为阳性。这意味着在大小 $n = N_{TP} + N_{FP}$的样本中，我们处理的是真阳性样本的比例值。对召回率（recall）和其他性能标准的处理也类似。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释术语置信区间。什么是误差界限?
- 置信区间（误差界限）的大小如何依赖用户指定的置信水平?它如何依赖测试集的大小?
- 讨论其他性能评价标准，如**精度**和**召回率**的置信区间的计算。

12.4 一个分类器的统计评价

通过实验的方法，将分类器作用于一个完成预分类的样本集，能够确认或拒绝关于该分类器性能的假设。对依照表 12.3 中的算法对结果进行统计评估。下面，通过一个简单的例子给出详细解释。

表 12.3 统计评价分类器性能的算法

1. 给定一个样本规模为 n 的测试集，一个分类准确率理想值为 Acc 的分类器，检验正态分布的条件是否得以满足：
$$n \cdot Acc \geqslant 10,\ n \cdot (1 - Acc) \geqslant 10$$
2. 使用下列公式计算标准误差：
$$S_{Acc} = \sqrt{\frac{Acc\ (1 - Acc)}{n}}$$
3. 假定服从正态分布，从表 12.2 中找到要求的置信水平对应的z^*值。计算相应的置信区间为 $[Acc - z^* \cdot S_{Acc},\ Acc + z^* \cdot S_{Acc}]$。
4. 若测试集上的测量值位于区间外，则拒绝准确率为 Acc 的原假设。否则，认为拒绝原假设的证据不充分。

一个简单实例。假定一位机器学习专家告诉你某分类器的分类准确率为 $Acc = 0.78$。俗话说得好，“尽管信任，仍要验证”，因此你决定探究一下专家的声明是否正确。为了完成检验，你准备了一个包含了 100 个已知分类的实例集，并且将其作为衡量分类器性能的测试集。

最终，实验得到的结果是分类器的分类准确率为 0.75，小于给出的 0.78。此时，你可能会问：这个可观察到的差异是否位于合理范围内呢？换句话说，有没有可能专家的结论是正确的，只不过由于测试数据具有随机性才得到了较低的性能测量值？毕竟，不同的测试集会得出不同的分类准确率。

正态分布的条件检验。这里需要考虑的第一个问题是，实验得到的分类性能的分布是否近似于正态分布。如果答案是肯定的，那我们便可以根据表 12.2 的数值进行统计评价。

确认条件（式（12.1））和（式（12.2））很容易。鉴于 $np = 100 \cdot 0.75 = 75$

$\geqslant 10$，并且 $n(1-p)=100\cdot 0.25=25\geqslant 10$，因此我们知道条件得到满足，可以采用正态分布的假设。

寻找置信水平为 95% 的置信区间。假定如果分类器对任意测试集分类时得到的准确率是 0.75 的概率达到 95 %，你就准备接受专家的主张（$Acc=0.78$）。也就是说，0.75 的准确率要落在以 0.78 为中心的与 95 % 的置信水平对应的置信区间内。下面看一下实际情况是否如此。通过表 12.2 可知，当置信水平为 95 % 时，$z^*=1.96$。这就意味着，由随机测试集得到的准确率有 95 % 的概率位于区间 $[Acc-1.96\cdot S_{Acc},\ Acc+1.96\cdot S_{Acc}]$ 中，其中，$Acc=0.78$ 是原假设，S_{Acc}则是在规模为 n 的测试集上的统计标准误差。

在具体的训练集里，样本规模 $n=100$。标准差的计算如下：

$$S_{Acc}=\sqrt{\frac{Acc\ (1-Acc)}{n}}=\sqrt{\frac{0.75\cdot 0.25}{100}}=0.043 \tag{12.7}$$

我们推断置信区间为 $[0.78+1.96\cdot 0.043,\ 0.78+1.96\cdot 0.043]$，四舍五入后的结果是 $[0.70,\ 0.86]$。

一个关于专家声明的结论。对此处训练集得到的 $Acc=0.75$ 的分类准确率进行评估，其评估值恰好处于 95 % 的置信水平对应的置信区间内。

这个结论令人欢欣鼓舞。对于 $Acc=0.78$ 的原假设，我们发现在随机测试集上获得的分类准确率位于区间 $[0.70,\ 0.86]$ 内的概率达到 95 %。这确实在当前的情境中发生了。因此，尽管我们的实验结果是 $Acc=0.75$，略微低于专家所说，但是不得不承认我们没有令人信服的证据可以拒绝该声明。正是由于缺乏此类证据，因此我们认为专家的声明是有效的。

类型 I 统计评估错误：以真为假。现在，你已经理解了统计评估的基本原则。某些人还对性能做了陈述。基于测试集的规模（假定服从正态分布），我们计算包含了给定的所有值的 95 % 的区间规模。也就是说，在原假设正确的前提下，测试结果只有 5 % 的概率落到区间之外。这就是我们为什么会拒绝测试结果位于这不足 5 % 的区域内的任何假设。因为我们简单地假定这在现实中几乎是不可能发生的。

然而，这种情况依然有 5 % 的概率会出现。我们必须承认把分类器作用在一

个随机测试集上存在一定的风险使得结果处于给定的置信区间之外。在上述例子里，拒绝专家的声明显然是不公平的。统计学家称之为类型 I 错误：错误地拒绝了正确的声明，这种拒绝往往基于某些结果是非典型的事实。

如果我们想要避免这种错误，则增加要求的置信水平即可。如果我们选择 99 % 的置信水平而不是 95 % 的置信水平，则假警报的错误将减少。但是这种错误率的降低是需要付出代价的，下个段落会给出解释。

类型 II 统计评估错误：以假为真。与上面截然相反的情况也是可能出现的，即原假设是假的，然而由测试结果得到的分类准确率却处在给定的置信区间内。当这种情形发生时，我们必须得出结论：实验无法提供有效的证据拒绝原假设，因此不得不接受原假设。

你或许觉得这是不幸的，但这的确会偶尔发生。一个不正确的主张没有被拒绝，统计学家称之为类型 II 错误。要求的置信水平高时容易出现这个问题，因为相应的置信区间范围很大以至于测试结果基本不会位于区间以外。因此，实验结果几乎不会拒绝原假设。

需要注意的是，要均衡两类错误。增加置信水平可以减少类型 I 错误，但是却要付出增加类型 II 错误的代价，反之亦然。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释如何统计评价某分类器在一个测试集上的实验测量值。
- 术语类型 I 错误（假警报）是什么意思？我们怎样才能减少该类错误？
- 术语类型 II 错误（没检出）是什么意思？我们怎样才能减少该类错误？

12.5 另外一种统计评价

现在，读者会理解对实验数据统计处理的本质，并且学会如何使用这些信息评价分类器的性能，但是，借助统计能够实现的远不止这些。

两个测试集是否代表了两种不同的语境？第 10 章里提到了这样一种情况：有时不同的语境下也许会学习归纳出不同的分类器，就像英语中的英式发音和美式发音。这里介绍的就是统计是如何帮助我们在数据领域里辨别此类问题的。

假定我们在两个不同的测试集上测试两个分类器。第一次测试得到的分类准确率为$\widehat{p_1}$，第二次测试得到的数值为$\widehat{p_2}$（字母 p 代表正确结果的比例）。两个测试集的尺寸分别用n_1 和n_2 表示。最后，用$\widehat{p}$代表两个集合中被正确分类的样本比例的平均值。

统计量用式（12.8）定义：

$$z = \frac{\widehat{p_1} - \widehat{p_2}}{\sqrt{\widehat{p}(1-\widehat{p})\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}} \tag{12.8}$$

将计算得到的结果与表 12.2 给出的特定置信水平的临界值进行比较。

一个实例。假定在规模$n_1 = 100$ 和$n_2 = 200$ 的两个测试集上评价分类器。由此得到的两个分类准确率分别为$\widehat{p_1} = 0.82$，$\widehat{p_2} = 0.74$，因此两个测试集上的平均分类准确率为$\widehat{p} = 0.77$。容易证明已经满足正态分布的条件。

将数值代入式（12.8），得到：

$$z = \frac{0.82 - 0.74}{\sqrt{0.77 \times (1 - 0.77) \times \left(\frac{1}{100}+\frac{1}{200}\right)}} = 1.6 \tag{12.9}$$

因为这个值低于表格 12.2 给出的 95 % 的置信水平给定的数值，我们认为结果位于相应的置信区间内，所以这两个结果在统计上是不可区分的。

你学到了什么？

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 为什么我们要关心分类器被应用到错误数据上的情况？这里使用的公式是什么？如何进行评估？

12.6 机器学习技术的比较

有时，我们需要知道在可选的两种机器学习技术中，哪种更适合给定的分类问题。这里提出的一般方法与前面几节内容有些不同，尽管它们基于相同的原理。抛开冗长的理论，这里我们用一个包含具体数据的简单例子介绍该评估方法。

实验方法。第11章讨论了一种使用5×2交叉验证比较两种机器学习算法的方法。读者可以回想一下，该方法将预分类的一组数据划分成同样大小的两个子集T_{11}和T_{12}。首先，两种机器学习技术基于子集T_{11}归纳出各自的分类器，然后将这两个分类器在T_{12}上进行测试；接下来，再从T_{12}上归纳出两个分类器，在T_{11}上完成测试。将上述过程重复5遍，每遍都将数据集随机划分成两个子集，T_{i1}和T_{i2}。

最终，我们得到了10对测试集分类准确率（或错误率、精度、召回率，或者其他性能指标）。现在，我们可以确切地提出一个问题："这10对结果的差异是否是统计显著的?"

实验结果的例子：成对比较。我们分别用T_{i1}和T_{i2}表示第i对子集。假定要比较的两种机器学习算法为ML1和ML2，我们会在上述的10次实验操作中评价它们。

假定实验结果见表12.4。在该表中，每列是以与具体实验操作次序对应的测试集的名字标识。表格内容则是两种归纳技术归纳得出的分类器作用于相应测试集得到的分类准确率（以百分比的形式）。最后一行给出了两个分类准确率的差异。需要注意的是，这里的差异有正也有负。

对这些结果进行计算，我们知道差异均值$\bar{d}=2.0$，并且差异标准差$s_d=4.63$。

表12.4 两种机器学习技术ML1和ML2比较的实验结果。前两行的数值为分类准确率（以百分比的形式），最后一行给出的是差值d

	T_{11}	T_{12}	T_{21}	T_{22}	T_{31}	T_{32}	T_{41}	T_{42}	T_{51}	T_{52}
ML1	78	82	99	85	80	95	87	57	69	73
ML2	72	79	95	80	80	88	90	50	73	78
d	6	3	4	5	0	7	3	7	4	5

配对差异的统计评价原理。观察差异值 $\bar{d}$（标准差为$s_{\bar{d}}$）的均值，我们要问：此处的差异是否具有统计显著性？换言之，该差异是否在先前我们说的置信区间内，如置信水平为 95 %？注意，置信区间的中点为 $\hat{d}=0$。

与前面讨论的方法相比，我们必须指出两点主要的不同。第一点，现在我们处理均值 d，而不是比例。第二点，由于统计评估使用数值的数目 n 很小，并且标准差的估计也只是基于给定的 10 个观察值（总体未知），因此不再要求样本服从正态分布。在这种环境中，我们必须采取其他的理论分布，即所说的 t 分布。它的形状与正态分布（“钟形”）相似，但是更平滑一些。此外，它的“平滑”或“陡峭”程度依赖于自由度的数目。在包含 10 个测试集时，有 $10-1=9$ 个自由度。表 12.5[㊀]展示了自由度为 9 时一些典型的 t 值。

表 12.5　自由度为 9 时的概率 t 值

自由度	置　信　水　平			
	0.10%	0.05%	0.02%	0.01%
9	1.83	2.26	2.81	3.35

计算成对测试量的 t 值。采用 t 测试的统计评价在本质上与服从正态分布时相同。对于存在一定差异的均值 $\bar{d}$ 和标准差$s_{\bar{d}}$，t_9（下标是自由度编号）的计算公式如下，其中 n 为测试量的数目。

$$t_9=\frac{\bar{d}-0}{s_{\bar{d}}/\sqrt{n}} \tag{12.10}$$

将计算得出的t_9和表 12.5 列出的与具体置信水平相关的临界阈值进行比较。例如，依据表 12.4 的结果样本，我们可以得到：

$$t_9=\frac{2-0}{4.63/\sqrt{10}}=1.35 \tag{12.11}$$

因为得到的数值低于 2.26，即表中列出的置信水平为 95% 时的 t_9 值，我们得

㊀ 随着自由度的增加，曲线会越来越接近正态分布，当自由度大于等于 30 时，形状与正态分布基本相同。

出结论：无法拒绝（在给定的置信水平上）这两种技术产生可比较的分类准确度的假设。也就是说，我们接受这个主张。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释能够得到 10 对结果的 5×2 交叉验证技术的原理。
- 在前面小节中，为什么我们不能采用正态分布？这里使用了什么其他的分布？
- 写出计算 t 值的公式。解释如何计算公式中每个变量的值。

12.7 总结和历史简评

- 统计评价的本质是基于对相对较小样本的观察，得出总体所具有的行为的结论。
- 尽管不同的样本得出不同的结果，但是这些结果的分布一定遵循某些统计规律。掌握这些统计规律能够帮助机器学习工程师计算出他们在具体测试集上得到的分类准确率的信任程度。
- 最典型的“抽样结果”的分布量服从高斯正态分布。但是该分布的使用需要满足两个必要条件。给定训练集规模 n 和平均值 p 时，条件为 $np\geqslant10$ 和 $n(1-p)\geqslant10$。
- 不管是否服从正态分布，基于不同测试集得到的精度的标准误差 Acc 的计算公式如下：

$$s_{Acc}=\sqrt{\frac{Acc\ (1-Acc)}{n}}=\sqrt{\frac{0.75\times0.25}{100}}=0.043$$

- 对于每个置信水平，正态分布的假设都给出了一个特定的 z^* 值（见表 12.2）。在计算出标准误差并选定了一个置信水平后，我们可以按照下列公式计算置信区间：

$$[Acc - z^* S_{Acc},\ Acc + z^* S_{Acc}]$$

其中，$z^* S_{Acc}$被称为误差界限。不同的度量，采用相似的计算公式。

- 假定我们要测试关于某分类器准确率 *Acc* 的声明。如果实验估计值落入选定置信水平对应的置信区间内，我们则假定没有充足的证据可以拒绝该 *Acc* 声明。但是，若结果落在置信区间外，则拒绝这个声明。
- 我们经常借助 *t* 检验的结果以及 5 ×2 交叉验证技术来比较两种机器学习技术。其中，*t* 检验依赖于 *t* 分布而不是正态分布。*t* 分布在不同的自由度上形状有些许不同。

历史简评。本章介绍的统计方法历史悠久、体系完备，统计学教科书往往不再关心这些方法的发明者是谁。然而，站在机器学习的角度，我们必须指出，从 5 ×2 交叉验证得到实验结果并进行 *t* 检验的做法是迪特里奇[18]（Dietterich）提出的。

12.8 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

1. 假定我们要评价一个分类准确率 $p = 0.80$ 的某分类器的性能。如果我们想要依据正态分布和表 12.2 的信息进行评价，那么需要的测试集规模 n 的数值为多少？
2. 假定存在一个分类准确率 $p = 0.9$ 的分类器。将其应用于规模 $n = 400$ 的测试集时，得到的标准误差s_E 是多少？并计算置信水平为 95 % 时对应的置信区间。注意，不要忘记验证此时条件（式（12.1））和（式（12.2））的有效性。

3. 假定提供了一个声称理论准确率为 $p = 0.85$ 的分类器。当将该分类器应用于包含了 200 个实例的测试集时得到的准确率为 0.81，这个数值显然低于希望值。请判断，该分类器准确率 $p = 0.85$ 的声明是正确的概率能否至少为 95%？那 99% 呢？

思考题

假定开始你用 95% 的置信水平所对应的置信区间来测试一个分类器的性能，后来你改变了想法，决定改用置信水平为 99% 对应的置信区间，则该做法是否可行？这种增加置信区间的做法需要付出的代价是什么？

上机实验

假设读者已经实现了把数据分成 5 “折”的程序，以使用 5×2cv 方法进行所需的性能验证，并且已经实现了至少两个类别归纳的程序。

编写一个程序对两种采用了 5×2 交叉验证方法的分类器归纳技术进行比较，并使用 t 检验评估性能。

第 13 章　遗传算法

机器学习的本质是对问题最优解进行搜索，即找出一个在训练样本和新样本上都尽可能正确的分类器。第 1 章解释了一种最流行的基于人工智能搜索技术的方法，即爬山算法，并说明如何在分类器的导出中应用。

另外一种搜索方法——遗传算法（Genetic Algorithm），灵感来自于达尔文的进化论。读者需要了解这种算法，因为它在处理很多机器学习问题时非常有用。本章给出该算法的基本版本，并以 k-NN 分类器问题来说明它的用法。

13.1 基本遗传算法

首先我们简要介绍遗传算法的基本原理，具体实现细节下一节讲解。

基本原理。本节中分类器被编码成染色体的形式。染色体有时也称作“基因”，常用二进制字符串的形式表示。遗传算法在染色体的种群上进行操作，每条染色体描述一个个体（分类器）。每个这样的个体有一个适应函数（fitness function）值，这个值根据分类器的性能确定，适应函数的角色类似于启发式搜索中的进化函数[㊀]。

遗传算法循环。遗传算法在图 13.1 所示的无限循环中执行。任意时刻，种群中的每个个体都有一个适应函数值。这个值决定了该个体在“幸运轮”（wheel of fortune）中的片断长度，这个长度决定存活概率。理解这个过程的概率性质很重要。尽管拥有较长片断的个体享有更高的存活概率，但也无法保证，因为生存游戏存在不确定性。在现实世界中也是一样的，一个有着优秀基因的个体可能在一次事故中消失，而较差的基因却可能仅因为运气好而活下来。但是，如果时间很长、种群很大，概率规律还是会偏向让适应性高的那些基因存活下来。

存活的样本接下来选择“配对体（mating partners）。”配对过程中，染色体个体进行重组（recombined），产生一对新染色体。这些新染色体接下来接

㊀ 本章使用的术语“进化函数”“存活函数”和“适应函数”可互换。

受变异（mutation），即在基因串（string）中加入噪声（noise）。整个原理的伪代码见表 13.1。

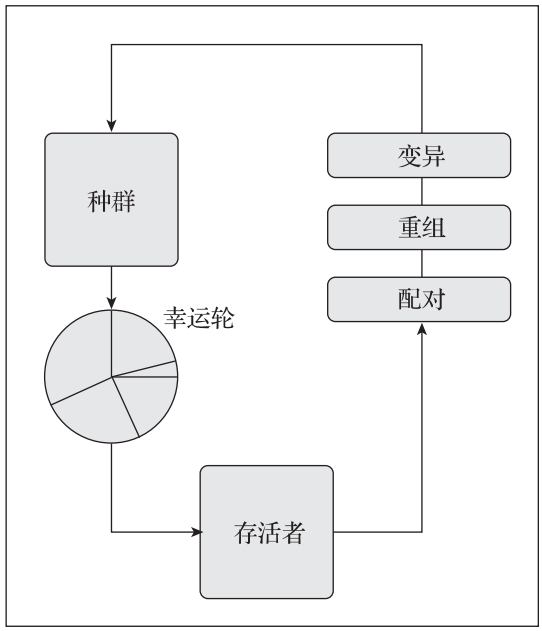


图 13.1　遗传算法的无尽循环。种群中的每个染色体都有自己的存活几率，与配对体的基因重组后产生新的染色体，并可能产生变异

表 13.1　遗传算法原理

初始状态：染色体个体种群。
1. 评估每个个体的适应度。根据适应度值，随机选择存活（survival）的个体。
2. 存活个体选择配对体。
3. 存活个体和配对体进行染色体重组（recombination），产生新个体。
4. 用随机变异（mutation）来改变染色体。
5. 算法返回第 1 步，除非满足终止条件。

无限循环如何工作。一旦产生出新种群，则将进入一个新的循环，个体将经过同样的幸运轮，然后配对、重组、变异、一直下去，直到满足结束标准时停止。需要注意偶然出现的错误是如何被概率特性消除的。低质量染色体可能侥幸经过幸运轮，但如果它的后代适应度仍很低，则会在随后的世代更替中被淘汰。另一

种可能是低质量染色体的基因经过重组，结合了其他染色体的基因而变得优秀。通过这种偶然性的机会，使进化过程更加灵活，但是在固定设置下，这些情况是不可能出现的。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释遗传算法的主要原理。染色体个体是如何被描述的?“存活机会(survival chance)”的含义是什么?
- 总结遗传算法的基本循环。
- 与确定性存活相比，根据概率的存活有哪些优点?

13.2 单个模块的实现

我们仔细看一看怎样用计算机程序来实现遗传算法的基本特征：存活游戏，配对（配偶选择），染色体重组，以及变异。我们从最简单的解决方案开始，更先进的技术放到后续章节中讨论。

为简化起见，假设染色体用二进制位串表示，如“1 1 0 1 1 0 0 1”，其中每一位代表一个属性，属性存在，该位为1，属性不存在，该位为0。因此在简化版的“派”问题中，第一个位可表示外壳（crust）是不是厚的、第二位可表示馅料(filling）是不是黑色的等。

初始种群。产生初始种群最简单的方法是使用随机数发生器。有时，设计者可依靠已有的信息产生更好的初始种群。在“派”问题中，关于正例的描述就是这种信息。然而，无论什么方法都需要保证初始种群足够大，并且有很好的多样性。

存活游戏。遗传算法假定有一种方法来计算每个样本的存活机会。在某些应用中，个体样本在实际实验的竞争中获得存活机会。在其他一些领域，可通过用户指定的评估函数计算适应度，适应度值取决于染色体的属性。如果染色体代表分类器，则适应度函数可用该分类器对训练样本做出正确分类的百分比来表示。

个体的存活由概率决定。看一下如何用程序实现“幸运轮”。令 F_i 表示第 i 个样本的适应度，令 $F = \sum_i F_i$ 是所有样本适应度之和，所有适应度分布在区间 $(0, F]$。存活率用随机数生成器从区间 $(0, F]$ 随机生成一个值 r 来建模：被 r 命中的连续子域指向存活者。图 13.2 给出了包括 4 个样本的小规模种群，一个随机数落在第 3 个区间上，于是 3 号个体被选中。如果最终希望 20 个样本存活，则要生成 20 个随机数落在 $(0, F]$ 区间上来标出存活者。

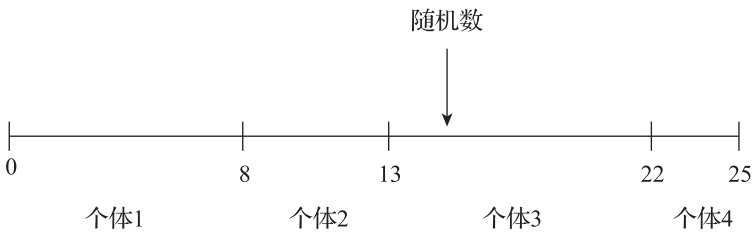


图 13.2　轴代表种群中的 4 个个体。适应度分别为 8、5、9 和 3。因为随机产生的数字 15 落在第 3 个区间，所以第 3 个个体被选中

鉴于适应度小的样本可能被淘汰，那些适应度大的样本能够在存活池中出现不止一次。生物学家不会想到这种“克隆”，但在计算机程序世界中，相同的个体可以存活两次，三次，甚至更多次。

配对算子。生存游戏接下来进行配对。自然界中，个体根据力量、速度或利齿判断配偶是否合适。在计算机中，是通过适应度函数来判断的。两者的区别在于：性别通常被忽略，而染色体可以和任何其他染色体配对。

最常见的策略是个体任意配对，如产生区间 $[1, N_s]$ 里随机的整数对，其中 N_s 是样本在种群中的代号。但这种做法没有考虑适应度高的个体比其他个体更有吸引力。一种简单的改进策略是把所有个体按适应度从大到小排序，让相邻的两个配对。

另一种策略是让概率决定。首先选出等级最高的个体，然后用图 13.2 所示的生存游戏的机制选出它的配偶。对等级第二高的个体也做同样处理，然后是第三个，依次类推，直到新种群达到所需的规模。这种策略下，“较好的”个体可能（尽管不保证）和其他强壮的个体配对，有时可能配对的值很低（因为根据概率选择），但这增加了多样性。这些低质量样本中的优秀染色体片段，系统有机会把它们保留下来。

长寿和永生的个体。这种算法的缺点是很好的有机体可能被低值的后代取代，优秀基因最终消失。为避免发生这种情况，计算机程序把最好的样本直接复制到新生代中，和后代放在一起。例如，程序直接在新一代中插入 20% 最好的样本，再对 95% 较好的个体应用重组、变异，生成其余的 80%，最差的 5% 完全忽略。用这种方法，不仅最好的样本会活得更久（甚至变成“永生”），而且那些偶然存活的特别弱的样本也会被去掉。

染色体重组：单点交叉。实现染色体重组最简单的办法是单点交叉（one-point crossor），一种把父辈染色体信息进行部分交换的操作。原理很简单：假设每个染色体由一条 n 位的串构成，一个随机数发生器返回一个整数 $i \in [1, n]$。于是，第一条染色体的最后 i 个位（长度为 i 个位的尾）和第二条染色体的最后 i 个位互换。具体操作中允许 $i = n$，这种情况下两个后代只复制它们的父辈。在下面的例子中，随机整数 $i = 4$，表示 4 位长的尾被交换（交叉点用空格标出）。

$$\begin{array}{ll} 1101\ 1001 & \\ & \Rightarrow \\ 0010\ 0111 & \end{array} \qquad \begin{array}{l} 1101\ 0111 \\ \\ 0010\ 1001 \end{array}$$

读者能够看到后代和它们的父辈很像，特别是当尾部很短时。当 $i = n - 1$ 时，父辈和后代间距离达到最大。

在很多应用中，重组算子只用于一定比例的个体。例如，如果 50 对选作配对，用户设重组概率为 80%，那么只有 40 对重组，其余 10 对只是复制进入下一代。

变异算子。变异的任务是改变继承的基因信息。说得具体一点，就是一个很小百分比的位翻转了一下，如一个位的值由 0 变成 1，或相反。具体的百分比的数值（变异的频率）是用户设定的参数。假设这个参数要求 $p = 0.001$ 的位要受影响。相应的计算机程序要为每一个位产生一个 1 ~ 1000 之间的随机整数。如果这个整数等于 1，则该位的值改变，否则不变。

想一想我们需要的变异是什么频率。在一种极端情况下，如果变异极少，则几乎不会有什么影响。另一种极端是，变异频繁发生，结果破坏了太多的染色体，损害了遗传搜索。如果频率接近 50%，则新的染色体会变得像一个随机生成的位串；遗传算法就退化成为随机数发生器。

变异算子和交叉算子实现的目的不同。单点交叉没有产生新信息，只是将现

有的子串做交换。变异引入了过去种群中不存在的一些新的样式。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 生存游戏的主要任务是什么？如何用计算机程序来实现它？
- 描述选择配对体的简单机制。简述重组算子和变异算子。

13.3 为什么能起作用

我们直观地解释一下遗传算法的性能。

函数最大化。表 13.2 中所示简单问题的目标是找到 x 的值，使函数 $f(x) = x^2 - x$ 的值最大。表 13.2 的上半部分的表的第二列里的每一条染色体都表示为一个二进制码的整数，对应的十进制值列在第三列中。第四列给出了对应的函数值 $f(x)$，函数值概率在第五列里，该值决定了个体的存活概率。例如，第一个样本的 $f(x) = 12^2 - 12 = 132$，相对存活概率（在特定种群中）是 12%，因为 $132/(132 + 600 + 56 + 342) = 0.12$。最右列告诉我们每个个体被选择进入下一代中的次数。

表 13.2 遗传算法说明

假设希望遗传算法找到 $f(x) = x^2 - x$ 的最大值。令 x 为以二进制串表示的整数。初始种群由 4 个位串组成，在表中每个串给出了整数值 x、对应的 $f(x)$、生存概率（$f(x)$ 的比例），以及该样本在下一代中被选中的次数。

No.	初始种群	x	$x^2 - x$	存活机会	选中次数
1	01100	12	132	0.12	1
2	11001	25	600	0.50	2
3	01000	8	56	0.05	0
4	10011	19	342	0.31	1
平均值			282		
最大值			600		

（续）

上述相邻的样本进行配对，分别交换 1 位长的尾和 3 位长的尾，尾的长度随机产生（空格代表交叉点），没有用到变异。最后两列给出了后代的 x 和 $f(x)$ 的值。

初始种群	配对	尾长度	新种群	x	$x^2 - x$
0110 0	2	1	01101	13	156
1100 1	1	1	11000	24	552
11 001	4	3	11011	27	702
10 011	3	3	10001	17	289
平均值					425
最大值					702

读者可以看到最好样本的值以及整个种群的平均值都增加了。

下一步，存活者找到它们的配偶。假定我们把相邻样本进行配对：第一个和第二个，第三个和第四个。接下来，交叉点随机决定第一对交换 1 位长的尾，第二对交换 3 位长的尾，没有用到变异。结果显示在表 13.2 下半部分的表中，其中最后 3 列分别是新的二进制串、对应的十进制数和 $f(x)$ 的值。注意到适应度函数的均值和最大值都增加了。

后代必须要优于它们的父辈吗？让我们看看是什么引起了这种改进。在图 13.3 中给出了直观的回答。图中显示了两个父辈的位置及存活函数 $f(x)$ 的值（垂直虚线）。当两条染色体交换 4 位长的尾时，产生了两个后代，和父辈相当接近。每个后代所在位置的 $f(x)$ 均比父辈高的情况回避了一个问题：后代是否总要比父辈更适应？事实远非如此。这些都取决于交换尾部的长度以及适应度函数的形状。想象一下，到了下一代，两个后代再次配对，随机生成的交叉点也在原来的位置。于是这两个后代的后代就会成为最初的两个串（他们的祖父），这意味着存活几率减小回最初的值。有时两个后代都优于父辈；也有时都弱于父辈；更常见的是混合情况。重要的是，在一个大的种群中，多数更好的样本存活下来，因为进化过程喜欢选择适应度 $f(x)$ 更强的个体。不适合的样本偶尔会幸存，但在长期过程中它们还是容易被淘汰的。

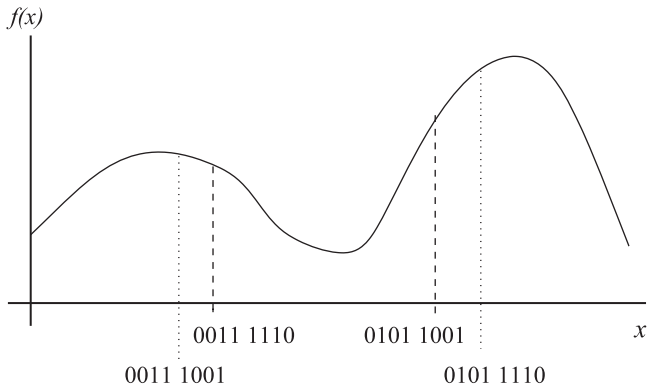


图 13.3　交换 4 位的尾后，两条父染色体（上面的串）产生两个后代（下面的串）。至少一个后代将有机会优于两个父辈

如果交换的尾串很短，则后代和父辈的染色体将很接近；如果很长，则后代和父辈间的相似之处就少。至于变异，后代和父辈间的相似性取决于哪些位变异了。如果是最左边的位，则变异将引起染色体的值沿水平轴有很大的跳跃。如果是最右边的位，则跳跃就短。两种情况下变异都是对重组的补充。不过后者是探索父辈染色体附近的空间，前者是探索其他空间。

适应度函数的形状。在适应度函数的定义中有一些固有的潜在陷阱，如图 13.4 所示，左侧的函数几乎是平的，不同的个体几乎有相同的存活机会，这达不到存活游戏的目的。当存活者从近乎一致的分布中选出的时候，这些个体的质量不会有明显的竞争优势。这个缺点可通过使 $f(x)$ 不要那么平坦来改善。有无数种方法可以做到，如用 $[f(x)]^2$ 来替代 $f(x)$。

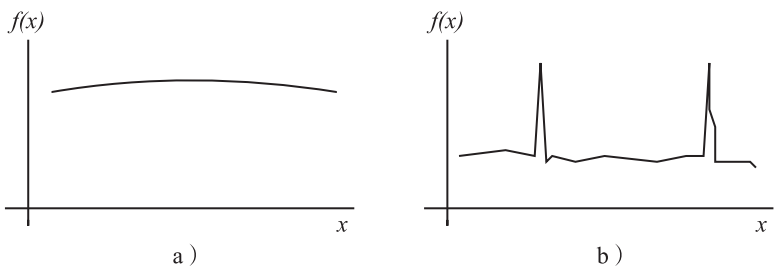


图 13.4　难以指导遗传搜索的两个适应度函数的例子。为了有用，存活函数不应太平坦，也不应包括独立的尖峰

图 13.4b 显示了另外一种陷阱：独立的尖峰。在和尖峰的宽度比较时，后代会觉得它们和父辈隔得很远。例如，如果父辈刚好躺在一个山脚下，后代可能在相对的一侧，这种情况下，尖峰将不会被注意到。这种问题比前面的问题更难预防。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释交叉点的位置是如何决定后代和父辈间有多大差别的。
- 解释重组和变异之间的相互作用是如何影响存活机会的。说明它们又是如何依赖存活函数的形状以及父辈位置的。

13.4 过早退化的危险

如果遗传算法取得的值不是经过一系列改善得到的，那么就意味着搜索是失败的。这种稳定的状态可用其他情况来解释。

过早退化。简单的遗传算法达到预定的代数后将停止。一个更精巧的版本是跟踪迄今为止得到的最高的适应度值，当这个值不再提高时停止搜索。

然而这里有个问题。适应度达到稳定水平（plateau）并不能保证问题找到了解。更准确地说，搜索过程可能是到达了所谓过早退化（premature degeneration）的阶段。假设表 13.2 中的搜索得到如下一个种群：

0 1 0 0 0

0 1 0 0 1

0 1 0 0 0

0 1 0 0 0

改进这个种群的机会是什么？重组不会改变什么。如果最后两个染色体配对了，后代只是复制了父辈。如果前两个配对，则 1 点交叉只能交换最右边的位，也不会产生新的染色体。能引起改变的唯一办法是变异。通过改变适当的位，变

异可以再次引发搜索。例如，当变异发生在第一条染色体的第3位（从左数）和最后一条的第4位时。不幸的是，变异极少见，等它出现不现实。事实上，过早退化意味着搜索卡住了。

防止过早退化。过早退化和种群的多样性（diversity）息息相关。不好的种群里所有染色体的位串都完全一样，这需要避免。计算机实现时需要用一个模块来监视种群的多样性，当下降到某个水平时将采取措施。识别这种情况的简单做法是计算染色体对之间的平均相似度，可以用计算每个串中相同位的个数来实现。例如，串“0 0 1 0 0”和“0 1 1 0 0”的相似度是4（4个位相同），“0 1 0 1 0”和“1 0 1 0 1”的相似度是0。

一旦检测到染色体间的平均相似度出现大的下降，则系统必须做出反应。这还不会引发警报。例如，在函数最大化的例子中，当大多数样本接近最大值时，先进的代将会被种群标记出来。这种“退化”当然不会被当成“过早”。然而，如果最好的染色体和问题的解有很大差别，则形势就不同了。这种情况下，我们必须增加多样性。

增加多样性。有多种策略。最简单的办法可在当前种群里插入一两条随机产生的新个体。更好的做法是让遗传算法并行运行在两个种群上，彼此独立。于是要么经过随机时间间隔，要么在怀疑发生过早退化时，一个种群的样本允许从另一个种群中选择配偶。当使用这种技术时，我们需要考虑后代放到哪个种群里。

种群大小的影响。种群大小需要引起设计者特别的注意。在大多数情况下，整个基因搜索过程中种群大小是一个常数。种群中个体数量由具体应用决定。根据经验，较小的种群要经历很多代才可以达到较好的解，除非它们出现过早退化。很大的种群有较强的健壮性防止退化，但可能引起过大的计算开销。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 遗传算法是如何依靠适应度定义取得成功的？两个主要的陷阱是什么？如

何应对?

- 你推荐哪种终止遗传搜索的标准?这些标准各有什么优缺点?
- 什么是过早退化(premature degeneration)?如何检测和改正?为什么种群需要多样性?
- 讨论种群大小的影响。

13.5 其他遗传算子

我们已经介绍了一种很简单的遗传算法及算子。大家已经理解了其原理,接下来我们介绍其他的算子。

两点交叉。上面介绍的单点交叉只是两点交叉(two - point crossover)更一般的特例。现在,随机数发生器要产生两个整数用于确定位串中的两个位置。父辈将在这两个位置进行子串交换,过程如下(交叉点用空格表示):

$$\begin{array}{l} 110\ 110\ 01 \\ 001\ 001\ 11 \end{array} \Rightarrow \begin{array}{l} 110\ 001\ 01 \\ 001\ 110\ 11 \end{array}$$

各染色体的两个交叉点可以不同。这时父辈将用如下方式“交换(trade)”子串:

$$\begin{array}{l} 1\ 101\ 1001 \\ 001\ 001\ 11 \end{array} \Rightarrow \begin{array}{l} 1\ 001\ 1001 \\ 001\ 101\ 11 \end{array}$$

随机串交换。在染色体重组模式中还有一种所谓的“随机位交换(random bit exchange)”。随机数产生器选择用户指定数字的位置,然后交换这些位,如下所示:

$$\begin{array}{l} 1\,1\,0\,1\,1\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1\,1\,1 \end{array} \Rightarrow \begin{array}{l} 1\,0\,0\,1\,1\,1\,0\,1 \\ 0\,1\,1\,0\,0\,0\,1\,1 \end{array}$$

这里第2位和第6位(从左数)被交换了。注意到如果交换最右位不会有任何变化,因为在两个染色体中这两个位的值是相同的。交换的二进制位的个数可以变化,但大多数应用中交换的位数比染色体长度要小得多。

现实中经常把两种或多种重组算子相结合。例如,选出的一对父辈有50%的

概率用两点交叉，有 30% 的概率用随机位交换，有 20% 的概率可能是根本不用重组。

逆序变异。重组算子作用于成对的染色体，其他一些算子作用于单个样本。变异是一种，逆序变异（inversion）是另一种。在典型实现中，随机数发生器返回两个整数，定义了位串中的两个位置（类似两点交叉）。于是两个位置间的子串被反转如下：

$$110\ 110\ 01 \Rightarrow 110\ 011\ 01$$

注意到在第 3 位到第 7 位间（从左数）的子串中，0 和 1 的次序颠倒了（即第 4、5、6 位旋转变成了 6、5、4）。两点的位置决定了反转对染色体的影响。如果两个整数彼此接近，如 4 和 7，那么只有一小部分染色体受影响。

在高级实现中，逆序变异是对变异的补充。例如，一个染色体指定位发生变异的概率可设为 0.2%，而随机子串逆序变异发生的概率是 0.7%。与变异相似，要非常小心以确保逆序算子极少被使用，使用过度会毁坏重组的积极贡献。

逆序变异和过早退化。在防止过早退化上，逆序变异比变异的表现要好得多。想知道为什么，看一看下面这个退化种群：

0 1 0 0 0

0 1 0 0 1

0 1 0 0 0

0 1 0 0 0

逆序第一个染色体的中间 3 位和第 2 个染色体的最后 3 位，可得到下面的种群：

0 0 0 1 0

0 1 1 0 0

0 1 0 0 0

0 1 0 0 0

读者可以发现多样性的确增加了。可以用这种简单的方法处理过早退化：不断增加（暂时的）逆序和变异的频率。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释单点交叉、两点交叉以及随机位交换的区别。
- 哪些具体方面使得重组算子与变异及逆序算子不同?
- 逆序变异如何影响遗传搜索?

13.6 高级版本

遗传算法是一个通用的一般框架，有无数可能的变化。本节将介绍两种有趣的技术。

拉马克（Lamarckian）选择的说明。计算机程序不像生物学那样会受到限制。设计者常抛弃这些限制，就像早期飞行员抛弃了羽毛翅膀的想法那样。在创造“永生”样本时我们就违反过限制，把样本直接复制进后代从而免受到重组和变异的破坏。接下来再看看另一种情况。

在基准遗传算法中，新的子串只能在重组和变异的随机过程中产生。在这之后，遗传信息在样本整个生命周期中都将保持不变。一个比达尔文更早的生物学家让-巴蒂斯特·拉马克（Jean-Baptiste Larmarck）的想法更加灵活：他认为生物体的进化是受需求驱动的。长颈鹿为了能吃到高处的树叶而不停地伸脖子，于是脖子变长，长脖子这一进化又遗传给后代。尽管拉马克的假说在生物学里被认为是不成立的，但是在其他领域并不是完全没有用。例如，研究人员可通过发表科学论文把知识传递给后来人。

相比经典的达尔文进化过程，拉马克的进化过程要快得多，这就是为什么我们要在遗传算法中实现它的原因。把这个概念并入到图 13.1 所示的循环中的最简单的办法是把“拉马克”算子放在“幸运轮”和重组算子之间。拉马克算子的任务是通过适应性来改善染色体。例如，我们可以设问，如果某一位产生了变异会发生什么？因为变异是不可逆的，所以我们可以在变异过程中更灵活一些，先测

试当第 i 位翻转后将发生什么，然后选择较好的版本。

多种群搜索。多种群搜索要和遗传算法依靠的多参数一起进行。大多数时候要靠个人的经验，另外也可以在相同的初始种群上并行地运行多个遗传算法，每个都有各自的变异频率、是否逆序变异、不同的重组算子的组合或修改过的适应度函数。在这些情况中，有一些搜索能更快地找到解。

回忆一下在讨论过早退化那一节曾提到的多种群搜索。在那里建议让两个或多个种群在相对隔绝的情况下独立进化，偶尔允许异种杂交。而如果种群使用了本文前面提到的不同的染色体定义，则这种杂交可能不容易实现。这时编程人员需要使用专门的程序实现一种编码向另一种编码的转换。

数据串，符号串。染色体编码不需要必须是二进制位串，也可以是数字串或字符串。前面提到的重组算子都可以用于这两种编码形式，但变异算子需要做些改变。在数字串中最常见的变异是用“噪声”叠加到部分（或全部）染色体的“基因”上。例如，如果所有的位置包含区间［0，100］的数字，于是噪声就可以建模为区间［$-a$，$+a$］的随机数，其中 a 是用户指定的参数，类似于前面二进制串中的变异频率。它的工作原理见下表：

变异前	10	22	17	42	16
“噪声”		-3	1	-2	
变异后	10	19	18	40	16

如果染色体是字符串的形式，则情况会略微不同。这时变异可以用随机数产生器选出的符号来代替染色体中随机选出的字符。例如，对染色体“d s r d w k l”，变异可以把左边第 3 位的 r 变为 s，最终染色体成为“d s s d w k l”。

“混合”染色体也是可能的。这时有些位置是二进制位，一些是数字，另一些是字符。这里变异通常是一些单独方法的组合的实现。例如，程序选择染色体中一个随机的位置，首先判断该位置是二进制位、数字，还是字符，然后应用适当的变异方法。

树结构形式染色体。在一些应用中，二进制位串、数字串、字符串都不适合，树结构可能更加灵活。当分类器以逻辑表达式的形式出现时，如图 13.5 所示的例

子，表达式用树状的染色体表示。

$$(\text{shape} = \text{circle} \wedge \text{crust} - \text{size} = \text{thick}) \vee \neg\ \text{crust} - \text{shade} = \text{grey}$$

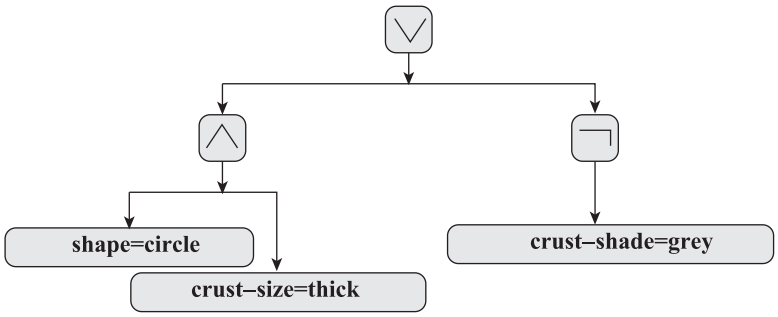


图 13.5 “派”问题候选表达式的树形表示

表达式由属性、属性的值、与、或、非逻辑算符构成。可以看到表示成树形结构是很自然的。内部结点代表逻辑算子，叶子包含属性 - 值对，重组交换子树。变异影响叶子或者是属性的名称，或者是属性的值，或两者都是。另一种可能的变异是偶尔用∨代替∧，或者相反。

初始种群的产生方式应引起特别的注意。编程人员应确保种群中已包含了一些有意义的表达式。一种可能的方式是产生一组随机表达式，在其中插入一些有意义的样本。存活函数（将取最大值）可以被定义为训练集上的分类精度。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 达尔文和拉马克的进化过程有什么不同？哪一种更快？
- 多种群遗传算法弥补了什么不足？多种群以什么方式解决了这个问题？
- 如果染色体是一个二进制位、数字、字符的混合串，那么如何实现变异操作？
- 当染色体是树形结构时，如何实现重组和变异操作？

13.7 k-NN 分类器的选择

我们看一下遗传算法在一个机器学习领域实际问题上的应用。

属性选择与样例选择。我们知道一个成功的 k-NN 分类器依赖于样例的质量以及对描述这些样例的属性的选择。选择正确的样例和属性的问题很容易转换为搜索问题。例如，初态可以定义为完整的样例以及完整的属性；搜索算子就是去掉样例和/或属性；评估函数（将取最小值）被定义为 1-NN 规则在一个独立的测试集上达到的误差率。

另一种可能是采用遗传算法。关键是需要确定如何把问题表示成染色体，如何定义适应函数、重组以及变异算子。然后，需要清楚如何解释（或利用）搜索结果。

染色体编码问题。一个很简单的办法可以把二进制编码的染色体分成两部分：第一部分的每个位置对应一个训练样例，第二部分的每个位置对应一个属性。如果某位是 0，则对应的样例和属性被忽略，否则保留。适应度函数被设计用来找出最小化 1 的数目。

当训练集中包含太多的样例时，这种解决方案将会导致很长的不切实际的染色体，如果训练集包含 1 万个样例，则需要 1 万个二进制位。更好的解决办法是选择更灵活的编码形式，染色体中的元素包含一个指向训练样例或属性的整数。染色体的长度是相关属性加上代表性样例的数目。这种机制被称为值编码（value encoding），如图 13.6 所示。

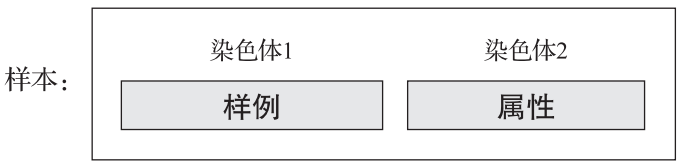


图 13.6　每个样本用两个染色体来描述，一个表示样例，一个表示属性。重组独立地应用在两者上面

解码染色体。我们必须保证正确地解码染色体对。例如，样本“3，14，39”，“2，4”表示由3、14和39三个训练样例组成的训练子集，用第2个和第4个属性来描述。当这种样本用作分类器时，系统选出第一个染色体所决定的样例和第二个染色体所决定的属性去描述它们。向量 $x=(x_1, \cdots, x_n)$ 和 $y=(y_1, \cdots, y_n)$ 间的距离可用式（13.1）计算：

$$D(x,y)=\sqrt{\sum_{i=1}^{n} d(x_i,y_i)} \tag{13.1}$$

其中，$d(x_i, y_i)$ 是第 i 维的贡献。对数字属性，这个贡献可用欧几里得距离公式来计算，即 $d(x_i, y_i)=(x_i-y_i)^2$；对二进制位属性和离散属性，可以定义当 $x_i=y_i$ 时，$d(x_i, y_i)=0$；当 $x_i \neq y_i$ 时，$d(x_i, y_i)=1$。

适应度函数。下一个问题是如何量化个体的存活几率。我们想要在不损失分类器精度的条件下减少样例和属性的数量。但这些需求是矛盾的，因为在无噪声的情况下，全部训练集比缩减的训练集有更高的分类性能。同样地，如果属性能提供相关信息，则去掉它们很难说对分类是有积极影响的。

应该让用户有机会通过设定适应度函数的参数进行权衡。让适应度函数要么最大化分类精度，要么最小化训练样例和属性的数量。这可以用式（13.2）来达到，其中 E_R 是在给定的样本上分类错误的训练样例的数目，N_E 是保留样例的数目，N_A 是保留属性的数目。

$$f=1/(c_1 \times E_R + c_2 \times N_E + c_3 \times N_A) \tag{13.2}$$

注意到如果样本的错误率低，保留样例集小，并且很多属性被去掉，那么样本的适应度就高。函数由用户设定的3个参数 c_1、c_2、c_3 来控制，用户根据偏好设置它们的权重。例如，如果 c_1 高，则分类精度被加强。如果 c_2、c_3 高，则重点就分别放在最小化保留样例和属性的数量上。

本应用中的遗传算子。父辈是按概率选择的。式（13.3）用于计算样本 S' 被选择的概率。

$$Prob(S')=\frac{f(S')}{\sum f(S)} \tag{13.3}$$

这里，$f(S')$ 是样本 S 按式（13.2）计算的适应度。分母是种群中所有样本

的适应度函数之和，这使得概率总和等于 1。

一旦一对父辈被选出，它们的染色体将通过两点交叉进行重组。因为每个样本用一对染色体来定义，每条有不同的含义，我们对它们分别进行重组操作。令 N_1 表示一条父染色体的长度，N_2 表示另一条染色体的长度。使用统一分布，算法从闭区间［1，N_1］里选择一对整数，从闭区间［1，N_2］里选择另一对整数。这两对分别定义了染色体中的子串（第一和最后的位置包括在子串中）。交叉算子接下来交换来自两个父染色体的子串。注意，因为各子串可能尺寸不同，所以后代的长度可能和父辈的长度不同。

图形示例。原理如图 13.7 所示，A 和 B 染色体的中间部分被交换。注意，这里 A 和 B 的长度受到了影响。需要决定是否允许交换片断的长度为 0，或另一个极端：一个片段能否是整个父染色体。

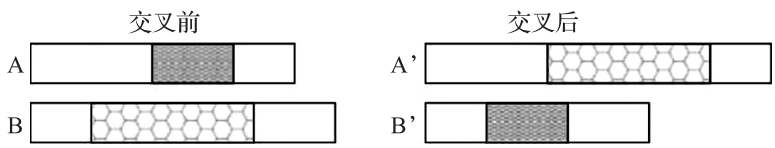


图 13.7　两点交叉算子通过交换在父辈染色体中随机选择的子串来创建子染色体

变异（mutation）算子要防止种群过早退化并保证种群在搜索空间中有代表性。一种可能的做法是，在新产生的种群中随机选定一定比例的位置，在这些位置中放入独立产生的随机整数。结果取样例/属性数目的模（mod）。令样例/属性的初始数目是 100，选出来的包含变异的位置的数是 95。如果随机产生的整数是 22，则变异后的值是（95 + 22）mod 100 = 17。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 选择最好的属性及最有代表性的样例可做到什么?
- 用两条而不是一条染色体有什么优点?

- 适应度函数的选择如何反映小属性/样例集和高分类精度两者间的竞争性需求？
- 为什么要用重组来交换不同长度的子串？变异是如何实现的？

13.8 总结和历史简评

- 受达尔文的进化论启发产生的遗传算法是对传统人工智能搜索技术的一种补充。最简单的实现是使用二进制编码的形式。
- 算法通过 3 种基本操作作用于种群中的个体：基于存活的适应度函数，成对染色体的重组，以及变异。有时也使用子串的逆序变异。
- 实际应用遗传算法时常遇到的一个问题是种群的过早退化。检测的方法是查看种群中染色体的多样性。解决办法是向种群中加入人工创建的染色体。逆序算子也有用。
- 遗传算法还可用于数字串，符号，混合字符串，甚至是树结构的编码形式。
- 本章通过一个简单的最近邻分类器问题示范了遗传算法的实际使用。

历史简评。根据生物进化理论产生遗传算法的想法归功于赫兰德[32]（Holland），不过其他人在此之前也提出过类似的想法。这些人中，需要提到的是瑞查博格[67]（Rechenberg），而福格尔（Fogel）、欧文斯（Owens）和沃尔什[25]（Walsh）是遗传算法程序实现的先行者。把遗传算法具体应用到 k-分类器的是瑞兹托克（Rizstoak）和库巴特[69]（Kubat）。

13.9 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

如表 13.2 所列那样，用纸和笔手工模拟实现一个遗传算法。自己选择适应度函数、不同的初始种群和随机的单点交叉点。再用两点交叉重复这个练习。

思考题

1. 解释不同的种群规模如何影响达到一个优质解所需的世代更替数目。仔细说明种群规模和过早退化问题之间的关系。再讨论一下适应度函数形状的影响。
2. 在哪些类型的搜索问题上遗传算法比传统搜索算法可能更有效？
3. 找出一些适合应用遗传算法的具体问题（不同于前面本书中提到的），使问题中的染色体最适合用二进制位或数字串来表示，或更适合用树结构表示。
4. 说一下自然进化和计算机模型间的区别。思考能否从自然界得到更多的灵感。你认为和生物进化相比，计算机程序的优势在哪里？

上机实验

1. 实现使用二进制编码染色体的基本遗传算法。确保存活函数，幸运轮，重组，变异都是单独的模块，并且这些模块都很通用以便修改。
2. 为第 1 章中的“派”和“圈”问题创建一个初始的种群，作为上机实验第 1 题中程序的输入。注意，在“派”问题中，你要考虑对原始程序做少量修改以使它能处理数字串编码的染色体。
3. 对你选择的问题，实现几种不同的配对策略。在实际完成中确定哪个策略可最快找到解。找最优解的速度可以用经过适应度评估的染色体的数量来度量。
4. 对选出的问题，用另一种“鸡尾酒”重组算子做实验，使用不同的重组、变异、逆序频率，画图显示搜索的速度（用上机实验第 3 题中所说的方法度量）是如何依赖于参数的具体设置的。

机器学习导论

第 14 章　增强学习

本书要解决的基本问题是如何导出一个分类器，并确定一个对象所属的类。目前已经有一些关于这方面的研究。但是在增强学习（reinforcement learning）中，完成的方式有所不同。智能体（agent）不是根据已经预先分类的实例集去归纳，而是在系统上“实验”，系统对实验做出奖励或惩罚。然后智能体根据系统做出的反应来优化自己的行为，使奖励最大、惩罚最小。

这种选择模式不同于分类器导出型任务所用的模式，于是有人认为应把它归入其他书里，如这本书的续集，关于这一方面的资料很多，也足够这么做。不过作者认为，如果不在本书中介绍一些增强学习的基本概念，本书的结构会显得不够完整，于是有了这最后一章。

14.1 如何选出最高奖励的动作

为了引入有关术语并对增强学习有初步的理解，我们从介绍一个简化任务开始。

***N* 臂老虎机**。图 14.1 显示了 5 台老虎机，每台老虎机的平均回报各不相同，但是我们不知道平均回报具体是多少。如果想最大化收益，我们需要找出每台老虎机的平均回报大小，并且一直使用那台回报最大的机器。这就是机器学习中 *N* 臂老虎机（*N*-armed bandit）问题的本质。顺便说一句，老虎机因会夺走人们的钱而臭名昭著。

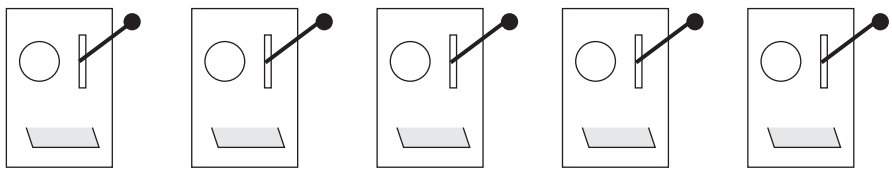


图 14.1 问题：哪台老虎机的平均回报最大

从理论上讲，这应该很简单。为什么不直接在各机器上反复尝试，以观察回报，然后选择回报最高的机器呢？但是在现实中，这不是一个好主意。在对哪台是最好的机器做出可靠的判断之前，太多的硬币可能会被浪费掉。

一个简单策略。考虑到花费，实际操作中工程师将限制试验次数，并根据几次试验做出初步选择。因为知道这个选择不太可靠，所以不会一直坚持。偶尔会测试其他机器：也许它们更好？如果收益更好，理所当然要用这个新的换掉“先前最好的”那个。这个策略非常自然，不是机器学习专家也能想到。

这就是增强学习（reinforcement learning）要模仿的行为。在图 14.1 的例子中，有 5 个动作可选。上面介绍的原理把利用（exploitation）当前最好的机器和探索（exploration）其他可选机器的方法结合了起来。“利用”占绝大多数，“探索”则非常少。在最简单的实现中，探索步骤的频率由用户设定的参数 ϵ 来控制。例如，$\epsilon = 0.1$ 表示“最好的”机器（在前面实验中表现最好的）被选中的次数占 90%；其余机器被选中的次数占 10%。

保存奖励的全部记录。“最好的动作”定义为产生平均回报最高的那一个[㊀]。对每一个动作，学习器会记录对应的回报；这些回报的平均值被当成这个动作的质量（quality）。例如，我们用整数 1，2，3，4，5 来标记图 14.1 中的各机器。动作a_i 表示选择第 i 台机器。假设最左边的机器被选中 3 次，产生的回报为：$r_1 = 0$，$r_2 = 9$ 以及$r_3 = 3$，则这次选择的质量为 $Q(a_1) = (r_1 + r_2 + r_3)/3 = (0 + 9 + 3)/3 = 4$。

由于保存前面所有动作的奖励比较烦琐，工程师可以利用式（14.1）来解决这一问题。其中 $Q_k(a)$ 是动作 a 从 k 次奖励中算出的质量，r_{k+1}是第（$k+1$）次奖励。

$$Q_{k+1}(a) = Q_k(a) + \frac{1}{k+1}[r_{k+1} - Q_k(a)] \tag{14.1}$$

通过式（14.1），可以只靠 k 和Q_k（a）的值就能“记住”每个动作，这就是全部所需要的，通过结合最新的回报，可用于在第（$k+1$）步更新动作的值。

这个过程有时被称作 ϵ-贪心（ϵ-greedy）策略。表 14.1 给出了算法的伪代码。

㊀ 在此说明，回报可以是负数，即相当于“惩罚”。

表 14.1 ϵ-贪心（ϵ-greedy）增强学习算法

输入：用户指定参数 ϵ，如 $\epsilon = 0.1$；一系列动作由 a_i 表示，初始估计值为 $Q_0(a_i)$；
对每一个动作 a_i，令 $k_i = 0$（动作已经执行的次数）。

1. 从均匀分布中生成一个随机数，$p \in (0, 1)$。
2. 如果 $p \geqslant \epsilon$，选择有最大值的动作（利用）；否则，随机选择一个其他动作（探索）。
3. 前一步选择的动作记作 a_i，观察奖励记作 r_i。
4. 用下式更新 a_i 的值：

$$Q(a_i) = Q(a_i) + \frac{1}{k_i + 1}[r_i - Q(a_i)]$$

5. 使 $k_i = k_i + 1$，返回第 1 步。

初始化过程。为了使用式（14.1），我们需要从某处开始，为每个动作赋初值 $Q_0(a_i)$。一种可能是选择一个值高于任何现实的回报。例如，如果已知所有回报在区间［0，10］中，则一个合理的初值是 $Q_0(a_i) = 50$。

在任意时刻，系统均按（$1 - \epsilon$）的概率选择质量最高的动作。在系统开始阶段，所有动作被选择的机会相等。假设 a_i 被选中，接下来得到奖励，根据式（14.1）这个动作的质量将会降低。因此，当要选下一个动作时，很有可能会选到其他的动作（如果是利用模式），其质量也将被降低。长话短说，在游戏的早期阶段，由于所有动作的质量都被初始化为同样的大数，因此它们都会被有组织地选中进行实验。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

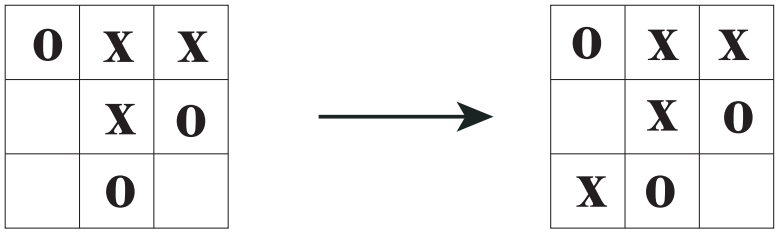
- 描述在 N-臂老虎机问题中搜索最好的机器用到的 ϵ-贪心策略算法。解释动作以及它们的质量的含义。利用和探索指的是什么？
- 描述维护平均回报的简单机制。它是如何更新动作的质量的？
- 为什么本节中建议把所有动作的初始值 $Q_0(a_i)$ 设为比典型奖励大几倍的值？

14.2 游戏的状态和动作

这个简单的老虎机例子可用于解释基本的术语。不足之处是选出一个适当的动作时每台机器只有一个状态。

现实中情况要复杂得多。通常，状态有很多，对应动作也很多。下面用一字棋（tic-tac-toe）游戏来说明这个实质。

一字棋游戏。基本的 3 ×3 棋盘如图 14.2 所示。两人对弈时，一人画叉，一人画圈。只要首先把 3 个叉或圈连成一条线就算赢，无论是连成一行、一列或对角线都可以。如果出现图 14.2a 所示的情况，且轮到画叉的人走，他在左下角画叉就可以赢。如果轮到他的对手走，就会在左下角位置画圈来阻止。



a）　　b）

图 14.2　一字棋，两人对弈，画叉或画圈。把 3 子连成一线则胜（水平、垂直或对角均可）

状态和动作。棋盘上每一个棋面代表一个状态。对每一个状态，棋手将选择一个具体的动作。对图 14.2a 所示的状态，有 3 个空位，因此有 3 个动作可选（其中一个动作可以胜利）。所有的走法可以用一张查找表来描述，表里的每个状态 – 动作都对应着一个确定的值 $Q(s, a)$。根据这个值，ϵ – 贪心策略决定根据当前的状态执行哪个动作。该动作执行后会得到一个奖励 r，这个奖励可根据式（14.1）更新状态 – 动作对应的值。

实现这个学习场景最典型的做法是让程序自己玩这个游戏很多次，从某个随机的选择开始（只基于初始值 $Q(s, a)$），逐渐改进，最终达到很高

的值。

实现这个做法最主要的问题是如何决定一个具体动作的奖励。理论上有 3 个方法可以考虑。

情节模式（Episodic formulation）。这可能是处理奖励分配问题最简单的办法。当一局结束，如果赢了，在整个游戏过程中学习智能体经历的所有状态 - 动作都可以得到奖励 1；如果输了，则得到奖励 -1。

这个办法的最大缺点是并非每一步都对最终结果的贡献相等。一个棋手可能走出很多步好棋，但是只要有一步走错，就会导致输棋。在这种情况下，惩罚好的动作不公平也不切实际。反过来也一样，较差的行为因为对手意外的失误而胜利，较差的行为都得到奖励。有些人可能说，在长时间的实验中，这些小的“不公平”会被平均掉，但总的说来能赢棋的行为还是要更好一点。情节模式的优点是简单。

连续模式（Continuing formulation）。如果能在一个动作完成后立刻知道奖励是多少，那么前面情节模式的问题（因为一步错误而惩罚一连串好的行为）就能避免。有时这的确是能做到的，即便在不可能的领域，也至少可以做些估计。

然而，大多数时候，在游戏结束前试图确定某一动作的奖励带有投机性，并且会对结果产生误导。所以这种方法很少使用。

折中：打折回报（discounting return）。其本质是对情景模式的改进，根据游戏的步数来决定奖励。例如，一字棋赢一局的时间越长，奖励就越小。这其中的逻辑是，动作越强，赢得越快。实施这个策略的方法是通过在胜利前所采取的步骤的数目来折扣最后的奖励。

这个想法的公式化表示：令r_k 为第 k 步实验得到的奖励，令 $\gamma \in (0, 1)$ 为用户设定的折扣系数常量。于是折扣回报（discount return）R 可用式（14.2）计算：

$$R = \sum_{k=1}^{\infty} \gamma^k r_k \tag{14.2}$$

注意，随着 k 的增加，与r_k 相乘的系数将减小。如果在第 10 步得到最终的奖励，并且取得胜利的奖励是“1”，那么对于 $\gamma = 0.9$ 的奖励折扣是 $R = 0.9^{10} \cdot 1 = 0.35$。

示例：杆平衡。图 14.3 所示的杆平衡问题很好地说明了回报折扣是个好策

略。这里每个状态被一些属性所描述，如小车的位置、小车的速度、杆的角度，以及杆的角度变化时的速度等。这个任务本质上有两个基本动作可选：①从左向右施加力；②从右向左施加力，并且还可以允许使用不同大小的力。这个任务最简化的版本是按固定的时间间隔执行动作，如 0.2s。

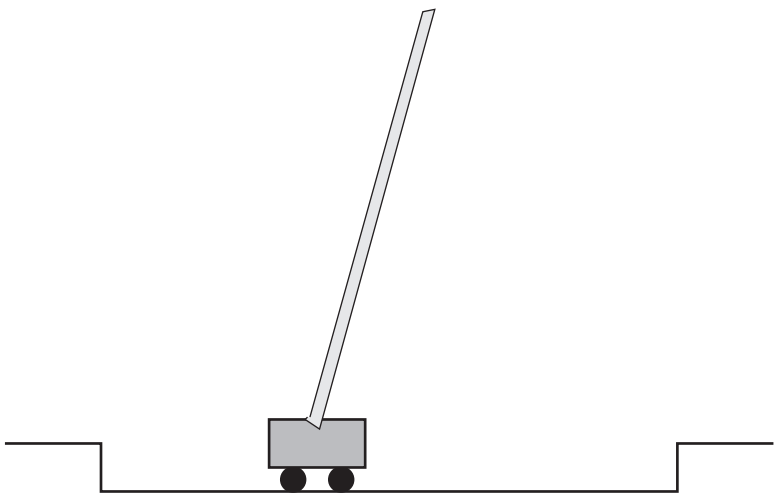


图 14.3　任务：左右移动小车使杆保持持续竖直

在这个游戏里，在杆倒下前坚持的时间越长，赢得越大。这也是为什么时间长要比时间短得到的奖励多。一个简单的办法来实现这一情况是，游戏中每个状态的动作奖励为 0，最后杆倒下时 $r = -10$。于是回报折扣为 $R = -10\,\gamma^N$，其中 N 为杆倒下前移动的次数。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中有任何困难，请回顾本节的相关内容。

- 解释状态和动作的区别。“状态 – 动作的值”的含义是什么?
- 分配奖励时，情节模式和连续模式有什么不同?
- 讨论折扣回报这个想法背后的动机。给出精确的公式，并且用杆平衡游戏来说明它。

14.3 SARSA 方法

前两节只介绍了一个很简单的机制，用来处理增强学习的问题。在这里我们简要地介绍一种更为流行的 SARSA 方法。算法的伪代码见表 14.2。

表 14.2 SARSA 算法——使用 ϵ-贪心策略及任务情节模式

输入：用户指定参数 ϵ，α，γ；所有动作 – 值对的初始值为 Q_0（s_i，a_i）；对每一个状态 – 动作对，s_i，a_i，初始值 $k_{ij}=0$。

1. 选择一个初始状态 s。
2. 从表 14.1 中使用 ϵ-贪心策略选择动作 a。
3. 执行动作 a。进入状态 s'，并得到奖励 r。
4. 在达到状态 s'后，使用 ϵ-贪心策略选择动作 a'。然后更新 $Q(s, a)=Q(s, a)+\alpha[r+\gamma Q(s', a')-Q(s, a)]$。
5. 令 $s=s'$且 $a=a'$。若 s 是终止状态，则转到第 1 步开始一个新的情景；否则转到第 3 步。

本质上，这里使用了带折扣的情节模式。这一方法始于选择一个初始状态 s（在有些应用领域，初始状态是随机产生的）。在一系列连续的步骤中，根据 ϵ-贪心策略来选择动作。每个动作产生一个新状态 s'，并且得到奖励 r。同样的 ϵ-贪心策略又被用来选择下一个动作 a'（在状态 s'时选择）。之后状态 – 动作对的质量 $Q(s, a)$按式（14.3）更新：

$$Q(s, a)=Q(s, a)+\alpha[r+\gamma Q(s', a')-Q(s, a)] \tag{14.3}$$

式中，α 是用户设定的常量；γ 是折扣因子。

这里需要注意的是，状态 – 动作对质量的更新基于 5 元组（s，a，r，s'，a'）。这也是方法名称的由来。

你学到了什么?

为确保你已经理解了本节的内容，请试着回答以下问题。如果在解答过程中

有任何困难，请回顾本节的相关内容。

- 描述增强学习 SARSA 方法的原理。这个方法的名字是怎么来的？

14.4 总结和历史简评

- 不同于前面章节介绍的分类器归纳问题，增强学习假定智能体在它要控制的系统中直接进行实验来学习和分类。
- 在非常简化的 N-臂老虎机问题中，智能体寻求识别最有希望的动作，即平均回报最高的那个。实际中最简单的实现基于所谓的 ϵ-贪心策略。
- 在更真实的任务实现中，假定存在一个状态集合，对每个状态，智能体从动作集合中做选择。这个选择可以通过基于状态 – 动作对 $Q(s, a)$ 的质量的 ϵ-贪心策略完成。
- 对于状态 – 动作对分配奖励的问题可以通过情节模式、连续模式或情节模式加上折扣的方式来解决。
- 作为增强学习的更高级方法，本章简要地介绍了 SARSA 方法。

小结。贝尔曼[2]（Bellman）第一个提出了“老虎机”问题的系统化处理方法，并完成了一些早期的工作。更为重要的是，他后来发展了动态规划（dynamic programming）的原理，被认为是增强学习的先驱[3]。但是，增强学习的基本理论及发展应该归功于萨顿[74]（Sutton）。

14.5 巩固你的知识

下面的练习用于进一步巩固所学的知识。本书提供的思考实验可以帮助读者从不同的角度理解本章所提及的思想并提高读者独立思考的能力。上机实验可以帮助读者注意到那些容易被忽视的细节。

练习

计算图 14.2 中一字棋状态 – 动作对的个数。

思考题

1. 本章全部的内容都建立在使用 ϵ-贪心策略上。你认为这个策略有什么局限性？有什么办法克服吗？
2. 本章用简单的游戏解释了增强学习的原理。你能否想出现实世界中其他有趣的应用案例？主要难点在于如何把具体问题映射成增强学习的形式。
3. 对于图 14.2 中的一字棋，可能需要多少个情景来解决这个问题？

上机实验

1. 用程序实现图 14.1 所示的 *N*-臂老虎机。
2. 思考图 14.4 所示的迷宫问题（maze-problem），要求找出从起点（S）到终点（G）最近的路。利用计算机，根据增强学习的原理并基于大量的走步训练来学习最短路径。请给出状态和动作的数据结构，并写出基于情节模式和 ϵ-贪心策略的程序。

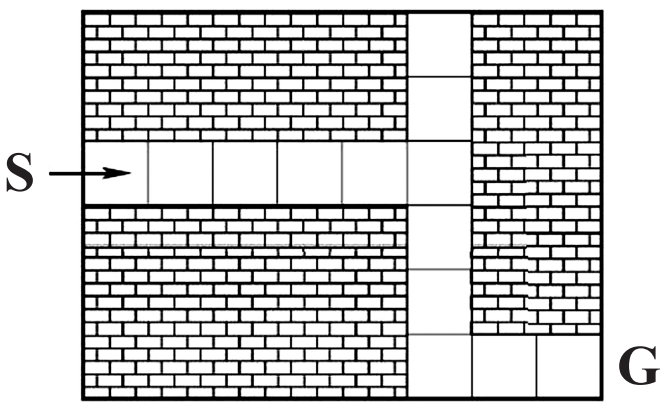


图 14.4　智能体从 S 出发，任务是找出到达终点 G 最近的路径

参 考 文 献

[1] Ash, T. (1989). Dynamic node creation in backpropagation neural networks. *Connection Science: Journal of Neural Computing, Artificial Intelligence, and Cognitive Research*, *1*, 365-375.

[2] Bellman, R. E. (1956). A problem in the sequential design of experiments. *Sankhya*, *16*, 221-229.

[3] Bellman, R. E. (1957). *Dynamic programming*. Princeton: Princeton University Press.

[4] Blake, C. L. & Merz, C. J. (1998). Repository of machine learning databases. Department of Information and Computer Science, University of California at Irvine. www. ics. uci. edu/~mlearn/MLRepository. html.

[5] Blumer, W., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journals of the ACM*, *36*, 929-965.

[6] Bower, G. H., & Hilgard, E. R. (1981). *Theories of learning*. Englewood Cliffs: Prentice-Hall.

[7] Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*, 123-140.

[8] Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.

[9] Breiman, L., Friedman, J., Olshen, R., & Stone, C. J. (1984). *Classification and regression trees*. Belmont: Wadsworth International Group.

[10] Broomhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, *2*, 321-355.

[11] Bryson, A. E., & Ho, Y.-C. (1969). *Applied optimal control*. New York: Blaisdell.

[12] Chow, C. K. (1957). An optimum character recognition system using decision functions. *IRE Transactions on Computers*, *EC-6*, 247-254.

[13] Coppin, B. (2004). *Artificial intelligence illuminated*. Sudbury: Jones and Bartlett.

[14] Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, *EC-14*, 326-334.

[15] Cover, T. M. (1968). Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, *IT-14*, 50-55.

[16] Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *IT-13*, 21-27.

[17] Dasarathy, B. V. (1991). *Nearest-neighbor classification techniques*. Los Alomitos: IEEE Computer Society Press.

[18] Dietterich, T. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*, 1895-1923.

[19] Dudani, S. A. (1975). The distance-weighted *k*-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-6*, 325-327.

[20] Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, *8*, 87-102.

[21] Fisher, R. A. (1936). The use of multiple measurement in taxonomic problems. *Annals of Eugenics*, *7*, 111-132.

[22] Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139-172.

[23] Fix, E., & Hodges, J. L. (1951). Discriminatory analysis, non-parametric discrimination. USAF School of Aviation Medicine, Randolph Field, TX, Project 21-49-004, Report 4, Contract AF41 (128) -3.

[24] Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3* (3), 209-226.

[25] Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. New York: Wiley.

[26] Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on machine learning*, Bari (pp. 148-156).

[27] Gennari, J. H., Langley, P., & Fisher, D. (1990). Models of incremental concept formation. *Artificial Intelligence*, *40*, 11-61.

[28] Good, I. J. (1965). *The estimation of probabilities: An essay on modern Bayesian methods*. Cambridge: MIT.

[29] Gordon, D. F., & desJardin, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, *20*, 5-22.

[30] Hart, P. E. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, *IT-14*, 515-516.

[31] Hellman, M. E. (1970). The nearest neighbor classification rule with the reject option. *IEEE Transactions on Systems Science and Cybernetics*, *6*, 179-185.

[32] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

[33] Holte, R. C. (1993). Very simple classification rules perform well on most commonly used databases. *Machine Learning*, *11*, 63-90.

[34] Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction*. New York: Academic.

[35] Katz, A. J., Gately, M. T., & Collins, D. R. (1990). Robust classifiers without robust

features. *Neural Computation*, *2*, 472-479.

[36] Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, MA: MIT.

[37] Kodratoff, Y. (1988). *Introduction to machine learning*. London: Pitman.

[38] Kodratoff, Y., & Michalski, R. S. (1990). *Machine learning: An artificial intelligence approach* (Vol. 3). San Mateo: Morgan Kaufmann.

[39] Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78* (9), 1464-1480.

[40] Kononenko, I., Bratko, I., & Kukar, M. (1998). Application of machine learning to medical diagnosis. In R. Michalski, I. Bratko, & M. Kubat (Eds.), *Machine learning and data mining: Methods and applications*. Chichester: Wiley.

[41] Kubat, M. (1989). Floating approximation in time-varying knowledge bases. *Pattern Recognition Letters*, *10*, 223-227.

[42] Kubat, M., Holte, R., & Matwin, S. (1997). Learning when negative examples abound. In *Proceedings of the European conference on machine learning* (*ECML' 97*), Apr 1997, Prague (pp. 146-153).

[43] Kubat, M., Holte, R., & Matwin, S. (1998). Detection of oil-spills in radar images of sea surface. *Machine Learning*, *30*, 195-215.

[44] Kubat, M., Koprinska, I., & Pfurtscheller, G. (1998). Learning to classify medical signals. In R. Michalski, I. Bratko, & M. Kubat (Eds.), *Machine learning and data mining: Methods and applications*. Chichester: Wiley.

[45] Kubat, M., Pfurtscheller, G., & Flotzinger D. (1994). AI-based approach to automatic sleep classification. *Biological Cybernetics*, *79*, 443-448.

[46] Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval* (*SIGIR' 94*), Dublin (pp. 3-12).

[47] Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, *2*, 285-318.

[48] Louizou, G., & Maybank, S. J. (1987). The nearest neighbor and the bayes error rates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *9*, 254-262.

[49] McCallum, A. Multi-label text classification with a mixture model trained by EM. In *Proceedings of the workshop on text learning* (*AAAI' 99*), Orlando, Florida (pp. 1-7).

[50] Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem. In *Proceedings of the 5th international symposium on information processing* (*FCIP' 69*), Bled, Yugoslavia (Vol. A3, pp. 125-128).

[51] Michalski, R. S., Bratko, I., & Kubat, M. (1998). *Machine learning and data mining: Methods and applications*. New York: Wiley.

[52] Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (1983). *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga Publishing Company.

[53] Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (1986). *Machine learning: An artificial intelligence approach* (Vol. 2). Palo Alto: Tioga Publishing Company.

[54] Michalski, R. S., & Tecuci, G. (1994). *Machine learning: A multistrategy approach*. Palo Alto: Morgan Kaufmann.

[55] Mill, J. S. (1865). *A system of logic*. London: Longmans.

[56] Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT.

[57] Michell, M. (1998). *An introduction to genetic algorithm*. Cambridge, MA: MIT.

[58] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203-226.

[59] Mitchell, T. M. (1997). *Machine learning*. New York: McGraw-Hill.

[60] Mori, S, Suen, C. Y., & Yamamoto, K. (1992). Historical overview of OCR research and development. *Proceedings of IEEE*, *80*, 1029-1058.

[61] Neyman, J., & Pearson E. S. (1928). On the use and interpretation of certain test criteria for purposes of statistical inference. *Biometrica*, *20A*, 175-240.

[62] Ogden, C. K., & Richards, I. A. (1923). *The meaning of meaning*. New York: Harcourt, Brace, and World. Eighth edition 1946.

[63] Quinlan, J. R. (1979). Discovering rules by induction from large colections of examples. In D. Michie (Ed.), *Expert systems in the micro electronic age*. Edinburgh: Edinburgh University Press.

[64] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81-106.

[65] Quinlan, J. R. (1993). *C4. 5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

[66] Parzen E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, *33*, 1065-1076.

[67] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer systeme nach principien der biologischen evolution*. Stuttgart: Frommann-Holzboog.

[68] Rosenblatt, M. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386-408.

[69] Rozsypal, A., & Kubat, M. (2001). Using the genetic algorithm to reduce the size of a nearestneighbor classifier and to select relevant attributes. In *Proceedings of the 18th international conference on machine learning*, Williamstown (pp. 449-456).

[70] Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*. Cambridge: MIT Bradford Press.

[71] Russell, S., & Norvig, P. (2003). *Artificial intelligence, a modern approach* (2nd ed.). Englewood Cliffs: Prentice Hall.

[72] Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5, 197-227.

[73] Shawe-Taylor, J., Anthony, M., & Biggs, N. (1993). Bounding sample size with the Vapnik-Chervonenkis dimension. *Discrete Applied Mathematics*, *42* (1), 65-73.

[74] Sutton, R. S. (1984). Temporal credit assignment in reinforcement learning. PhD Dissertation, University of Massachusetts, Amherst.

[75] Thrun, S. B., & Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, *15*, pp. 24-46.

[76] Tomek, I. (1976). Two modifications of CNN. *IEEE Transactions on Systems, Man and Communications*, *SMC-6*, 769-772.

[77] Turney, P. D. (1993). Robust classification with context-sensitive features. *Proceedings of the sixth international conference of industrial and engineering applications of artificial intelligence and expert systems*, Edinburgh (pp. 268-276).

[78] Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134-1142.

[79] Vapnik, V. N. (1992). *Estimation of dependences based on empirical data*. New York: Springer.

[80] Vapnik, V. N. (1995). *The nature of statistical learning theory*. New York: Springer.

[81] Vapnik, V. N., & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, *16*, 264-280.

[82] Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University.

[83] Whewel, W. (1858). *History of scientific ideas*. London: J. W. Parker.

[84] Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*, 69-101.

[85] Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON convention record*, New York (pp. 96-104).

[86] Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*, 241-259.

[87] Wolpert, D. (1996). The lack of a priori distinctions between learning algorithms. *Neural Computation*, *8*, 1341-1390.

译后记

目前，人类已经进入了智能信息时代。以机器学习为核心的人工智能技术正在推动着各个领域的发展，比如机器人、智能手机、物联网、数据挖掘等。机器学习这一学科融合了计算机科学、统计学、最优化理论和神经计算学等多个学科的精华，以学习数据内在结构和其中蕴藏的规律性知识为目标。如果说数据是矿藏资源，那机器学习无疑就是发掘资源价值的生产工具。而现在，几乎每个学科，包括社会科学领域，都在不停地产生数据，这也就是机器学习得以快速发展和普及的缘由。

本书翻译自米罗斯拉夫·库巴特所撰写的一本入门级的机器学习教材。库巴特是美国迈阿密大学电子与计算机工程系机器学习和人工智能方面的专家。本书几乎覆盖了机器学习所有的基本议题，比如贝叶斯分类器、决策树、神经网络、计算学习理论、集成学习、遗传算法和增强学习等，而且本书特别强调学以致用，不仅仅希望学生能够掌握基本的机器学习技术，而且希望他们作为未来的工程师了解各种方法的优势与劣势，以适应工业界的需求。和其他机器学习图书相比，本书最大的特点是浅显易懂，能很好地满足公众想了解机器学习基本原理的需求。另外，本书的组织和选材适合作为本科生机器学习课程的教材。当然，本书也可以作为一本参考书来使用。

全书共14章。由王勇、仲国强和孙鑫三位老师共同翻译完成。王勇负责第1、10～14章，仲国强负责第2～6章，孙鑫负责第7～9章。张伊丹、谭碧君、王天之、王丽丽、李逸、刘雯、凌霄、王海珍、魏洪旭、张康、王丽娜等人都对本书的出版做出了贡献。

在本书翻译过程中，得到了华东政法大学高宏老师的大力支持，在此表示衷心感谢。译者也感谢机械工业出版社的编辑们，正是他们的努力，本书才得以与读者见面。

本书涉及的内容广泛，对一些没有标准译法的术语，尽量采用当前流行的译法，对可能产生歧义的和较为重要的术语均附上了英文原文。翻译时力求忠于原著，并符合中文的表达习惯，为了翻译得简洁准确，进行了大量讨论。尽管如此，

译者简介

王勇，先后就读于西安电子科技大学、西安交通大学，获计算机软件与理论博士学位，2011 至 2012 年间在美国南佛罗里达大学作访问学者。现任中国海洋大学信息学院计算机系副教授，硕士生导师，软件工程教研室主任，海大惠普软件实验室主任。主要研究方向包括机器学习和软件工程。近年来主持或参与国家自然科学基金等研究课题多项，在包括《计算机学报》、*IEEE Trans* 等在内的学术期刊和国际会议发表多篇论文。现为中国计算机学会会员、IEEE 会员。

仲国强，先后就读于河北师范大学、北京工业大学和中国科学院自动化研究所，获工学博士学位。2011 年到 2013 年间在加拿大魁北克大学从事博士后研究。现任中国海洋大学信息学院副教授，硕士生导师。主要研究方向包括机器学习、模式识别等。近年来主持国家自然科学基金等研究课题多项，在 *Neural Computation*、*Pattern Recognition*、AAAI、ICDM、IJCAI 等国际学术期刊和会议上发表论文多篇。现为中国人工智能学会模式识别专业委员会委员、ACM 会员、IEEE 会员。

孙鑫，分别于 2013 年、2010 年和 2007 年获吉林大学博士、硕士和学士学位，并于 2016 年至 2017 年间在德国慕尼黑大学从事博士后研究。现任中国海洋大学信息学院计算机系副教授，硕士生导师。主要研究兴趣包括机器学习、模式识别和复杂网络等。近年来主持或参与国家自然科学基金等研究课题多项，在包括《计算机学报》、*Pattern Recognition* 等在内的学术期刊和国际会议发表论文多篇。现为中国计算机学会会员、IEEE 会员。

［美］米罗斯拉夫·库巴特
（Miroslav Kubat）

美国迈阿密大学教授，从事机器学习教学和研究超过25年。他已发表100余篇经过同行评审的论文，与人合编了两本著作，是近60个会议和研讨会的程序委员会委员，并担任3本学术刊物的编委。他在两个方面的前沿研究上得到了广泛赞誉：时变概念的归纳学习和在非平衡训练集上的学习。此外，在多标签样例上的归纳学习、层次组织的类别上的归纳学习、遗传算法、神经网络的初始化等问题上，他也做出了很多贡献。

电话服务
服务咨询热线：010-88361066
读者购书热线：010-68326294
010-88379203
网络服务
机工官网：www.cmpbook.com
为中华崛起传播智慧

AN INTRODUCTION
TO MACHINE LEARNING

雅克·凯瑞特 |《计算评论》

本书组织完备，每一节都有一个“你学到了什么”的小结，每一章都有简明扼要的总结及历史简评，并提供了大量的练习。全书章节都有经过精心选择和设计的例子，以帮助读者理解每一个概念。通过阅读本书，我学到了很多机器学习的基本知识。

刘成林 | 中国科学院自动化研究所副所长、模式识别国家重点实验室主任

米罗斯拉夫·库巴特所著的这本《机器学习导论》更像是一本科普性质的读物，作者尽量避开复杂的数学公式，用生动形象的方式介绍机器学习算法，而且本书篇幅适当，又涵盖了几乎所有基本的机器学习方法，使得本书不仅适合作为本科学生机器学习课的教材，也适合想了解机器学习入门知识的普通读者。

- 本书系统全面，既可以自学又可以作为研究参考。
- 既道出了机器学习的前世今生，又展望了发展的未来，让道听途说的信息止于智者。
- 全面讨论机器学习方法和技术，层次合理、叙述清晰、难度适中。
- 涵盖了经典的机器学习算法和理论，同时补充了近年来新出现的机器学习方法。

ISBN 978-7-111-54868-3