深入浅出 Python 机器学习

段小手　著

清華大學出版社
北　京

内 容 简 介

机器学习正在迅速改变我们的世界。我们几乎每天都会读到机器学习如何改变日常的生活。如果你在淘宝或者京东这样的电子商务网站购买商品，或者在爱奇艺或是腾讯视频这样的视频网站观看节目，甚至只是进行一次百度搜索，就已经触碰到了机器学习的应用。使用这些服务的用户会产生数据，这些数据会被收集，在进行预处理之后用来训练模型，而模型会通过这些数据来提供更好的用户体验。此外，目前还有很多使用机器学习技术的产品或服务即将在我们的生活当中普及，如能够解放双手的无人驾驶汽车、聪明伶俐的智能家居产品、善解人意的导购机器人等。可以说要想深入机器学习的应用开发当中，现在就是一个非常理想的时机。

本书内容涵盖了有监督学习、无监督学习、模型优化、自然语言处理等机器学习领域所必须掌握的知识，从内容结构上非常注重知识的实用性和可操作性。全书采用由浅入深、循序渐进的讲授方式，完全遵循和尊重初学者对机器学习知识的认知规律。本书适合有一定程序设计语言和算法基础的读者学习使用。

图书在版编目(CIP)数据

深入浅出Python机器学习 / 段小手著. — 北京：清华大学出版社，2018
ISBN 978-7-302-50323-1

Ⅰ. ①深… Ⅱ. ①段… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 114982 号

责任编辑：刘 洋
封面设计：李召霞
版式设计：方加青
责任校对：宋玉莲
责任印制：沈 露

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 **邮 编：**100084
社 总 机：010-62770175 **邮 购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经 销：全国新华书店
开 本：187mm×235mm **印 张：**18.25 **字 数：**343 千字
版 次：2018 年 8 月第 1 版 **印 次：**2018 年 8 月第 1 次印刷
定 价：69.00 元

产品编号：079842-01

前　言

计算机是由程序驱动的，人工智能（AI）不过是一些特殊的算法。只要你有一些程序设计语言的基础，跟随本书，你也能进入人工智能的世界，成为一个 AI 应用的开发者。

人工智能，火了

仿佛就在一夜之间，人工智能火了，一跃成为 IT 业内最受关注的热点话题。如果读者关注互联网圈子的话，应该会听说在 2017 年 12 月乌镇举办的第四届互联网大会上，一众互联网大咖张口闭口都在谈人工智能，使得 AI 成了毫无疑问的最大风口。

例如，网易公司的 CEO 丁磊就表示，人工智能会成为每一个行业的标配。他认为，任何一个行业都可以用到人工智能，并且建议每个企业的领导者都不要忽略人工智能对自己所在领域的影响。

苹果公司的 CEO 蒂姆·库克也谈到人工智能可以让世界变得更美好。同时，他表示并不担心机器会像人一样思考，但强调必须为技术注入人性，赋予技术应有的价值。

把“All in AI”作为口号的百度，其 CEO 李彦宏更是极为推崇人工智能。他的观点是，人工智能不可能超越人类的能力，但是随着它的能力逐步逼近人类，就会开始一个行业一个行业地去颠覆了。

还有一位不能不提的人，就是阿里巴巴集团董事局主席马云。在他的演讲中提到，与其担心人工智能会带走很多就业机会，不如拥抱技术，解决新的问题。人工智能只会让人的工作更有价值，更有尊严。

当然，小米的 CEO 雷军更是不忘在大会上直接做了个硬广告。他告诉与会嘉宾，小米正在拥抱人工智能，在 2018 年小米手机将深度利用 AI 技术。

此外，大名鼎鼎的斯坦福大学计算机系教授、Coursera 创始人吴恩达，更是人工智能的坚定拥护者。他直言未来政府和企业会在人工智能领域扮演着越来越重要的角色，监管得力才能使 AI 发展得更好，企业领导者更应该将 AI 技术融入企业文化中，创造一个 AI 支持下的未来。

国内的顶级学者也高度重视 AI 的发展，中国工程院院士倪光南老先生强调，AI 是未来非常重要的一个发展方向，会产生什么我们很难预料，但一定会产生重大的影响；同时他指出，未来人和机器应该和谐相处，我们可以把重复性的劳动交给人工智能，人类去做更多有创造力的工作。

诸如此类，我们这里不一一列举了，但从上述这些大咖们的言论之中，人工智能的火爆程度，已然是可见一斑了。

置身事外，还是投身其中

既然各路大咖都如此看好人工智能的前景，那么我们应该怎样面对这一波浪潮呢？

前不久，我们的朋友圈几乎被同一种情绪刷屏，那就是对人工智能即将取代人类的强烈担忧。各路自媒体不惜笔墨地渲染人工智能将逐步蚕食人类的就业机会，并且最终取代人类统治世界，以此来博得大众的眼球，吸引粉丝的关注。

对于此，笔者的观点是：这简直是“咸吃萝卜淡操心”！这种通过夸大其词误导大众换取关注的方法是不可取的。纵观人类历史，发生过三次大的工业革命，而在这三次革命当中，确实发生过短暂的对于人类就业的冲击。但是人类自起源以来，依靠强大的适应能力一直存活到今天，并没有被任何机器或者别的物种所取代。相反地，我们的生活质量还在不断提高，享受着新兴技术给我们带来的高效与便捷。

不过，尽管我们不必担心新的技术取代人类，但还是要面对一个现实，那就是每次大的技术革命带来的阶层转换与固化。比如距离我们最近的这一次信息技术蓬勃发展，制造了一大批顶级富豪，如微软的比尔 • 盖茨、亚马逊的贝索斯，国内的李彦宏、马云、马化腾、张朝阳、丁磊等。

如果说，在上一次互联网带来巨大机遇的时候，咱们年龄还小——有可能还在上中学甚至小学，没能抓住这一波浪潮，那么这一波 AI 带来的机会就真的是“生逢其时”，估计本书的读者朋友年龄段会相对集中在“80 后”“90 后”这一代，正是青春好年华，体力和脑力都处在一个非常出色的阶段，可以说是完美地“遭遇”了这一千载难逢的好时机。

笔者一直认为，评价一门技术是否值得我们投入时间和精力去认真钻研，一个重要的标准就是它是否能及时地让我们的生活质量得到巨大的提升。更具体一点说，或许它可以让我们“富可敌国”，或许让我们“权倾朝野”，亦或许让我们“转角遇到爱”，总之是要解决具体的问题。当然有些读者朋友会觉得笔者有点过于“实用主义”，但人生苦短，成功需趁早。

前途光明，马上开始

自从 2017 年下半年开始，笔者强烈感觉到，各大互联网公司在疯狂地抢夺人工智能领域的人才，随手打开微信朋友圈都能看到众多猎头好友在发布人工智能工程师、机器学习工程师、算法工程师等职位，而且一挂就是很久，说明这些职位相当难招。这也导致这些职位的薪酬是水涨船高，随便打开一个招聘网站，搜索一下相关的岗位关键词，你都会发现类似的岗位招聘人数很多，薪酬也都很令人咂舌。

例如，我们打开猎聘，在职位搜索中输入“机器学习”这个关键词，会得到 10 000+ 个结果，我们随便挑一些来看一下，如图 0-1 所示。

图 0-1　“机器学习”搜索结果

从图 0-1 中我们可以看到，和机器学习相关的职位，即便没有经验的要求，年薪也达到了 24 万~48 万，而一个 2 年以上工作经验的算法工程师，薪酬更是达到了 36 万~72 万，绝对称得上是“钱途光明”。

看到这里，读者朋友可能已经跃跃欲试，希望能够尽快投身到人工智能的领域当中，但是如何迈出第一步呢？

要知道，人工智能是一个非常宽广的领域，它涵盖图像识别、自然语言处理、语音识别、数据挖掘等，究竟哪个方向的前景是最好的呢？

对于这个问题，笔者是这样考虑的：现在并不需要过分纠结未来深入研究的领域，现阶段最应该做的是开始打基础，进行入门知识的学习。在上述若干个领域中，不论应用的场景有多么大的区别，其背后的原理无外乎是使用机器学习算法对数据进行学习，并且得到分类、回归、聚类的结果，因此笔者强烈建议读者朋友从“机器学习”着手，然后向“深度学习”进发，再结合实际工作需求选择一个具体的应用方向进行深入的研究。

本书内容及体系结构

为了让读者的学习过程相对比较轻松愉快，本书写作力求语言生动，并且以实例来进行讲解。

本书第 1 章先用小 C 追求女神的故事阐述了什么是机器学习，然后用蝙蝠公司的业务单元展示了机器学习的应用场景，力图让读者对机器学习产生兴趣，并且参考本章给出的建议开始学习。

本章后面的内容会阐述机器学习的基本概念，包括有监督学习和无监督学习、分类与回归的基本概念、模型的泛化，以及什么是过拟合，什么是欠拟合。

本书第 2 章将会详细地指导读者配置机器学习的环境，这一章中读者将能够掌握 Python 的下载和安装，以及相关工具的安装和使用方法。

本书第 3 章 ~ 第 8 章将详细介绍机器学习中常见的一些算法，在这些章节中读者将会掌握常见算法的原理和使用方法。

在第 9 章和第 10 章中，读者将可以学习到如何对数据进行预处理和聚类，以便让复杂数据集中的关键点可以使人一目了然。

从第 11 章和第 12 章，读者将可以学习如何让算法能够有更良好的表现，以及如何找到模型的最优参数，还有怎样通过建立流水线让模型协同工作以便达到最好的效果。

本书第 14 章介绍了如何编写一个简单的爬虫来进行数据的获取，并且介绍了如何使用潜在狄利克雷分布进行文本数据的话题提取。

本书的最后——第 15 章，向读者简单介绍了目前“人工智能”领域的人才需求现状和未来的学习方向，以及如何使用常见的竞赛平台磨炼自己的技能。

注意：限于篇幅，本书不会详细介绍 Python 语言的基本语法和其在机器学习之外的应用。感兴趣的读者可以搜索 Python 的基础教程来进行学习。但是本书的学习并不要求读者对 Python 达到精通的水平。

本书特色

1. 内容实用实在、详略得当，讲授符合初学者的认知规律

本书内容涵盖了有监督学习、无监督学习、模型优化、自然语言处理等机器学习领域所必须掌握的知识，从内容结构上非常注重知识的实用性和可操作性。必须掌握的细节处绝不吝惜笔墨、手把手细致到每一次的鼠标点击；仅需要大致了解处绝不铺张浪费纸张、整体结构的描述提纲挈领。这样的安排注重了对初学阶段必备知识的深入了解，大致了解的知识也能够有所认识，这种由浅入深、循序渐进的讲授完全是遵循和尊重了初学者对机器学习知识的认知规律。

2. 行文幽默诙谐，以实例引导全程，特别适合初学者阅读

本书介绍的基本理论知识、用于分类的机器学习算法、用于回归的机器学习算法、数据预处理、数据表达与特征工程等，都是使用非常贴近生活场景的实例来引导的，这样就避免了知识讲述过于抽象，非常易于理解。同时，作者以幽默诙谐，贴近时代的语言对这些知识进行生动、通俗的一一讲解，犹如一位你的老朋友，帮助你缩短入门机器学习的时间。纵观全书，作者将大学生小 C 追求女神以及帮助他的朋友处理日常问题同机器学习的理论与操作进行对比介绍，这就使得整个学习过程变得简单、生动起来。

3. 配套的人才培养与引入计划，帮助读者将学习成果转化为真正的生产力

在笔者过去的工作当中，累积了数量可观的各大互联网公司招聘通道资源，以及诸多猎头资源，可以帮助学有所长的读者快速进入一个实际操作的场景中进一步提高自己的实操能力。除此之外，笔者和国内大部分相关的产业发展部门有着密切的联系，对于有志于在人工智能领域创业的创业者来说，也能够帮助其对接政策资源，帮助大家在创业过程中得到有关部门的支持，从而使得创业之路变得不那么坎坷。

本书读者对象

- 想要进入机器学习领域的初学者
- 企业中想要向机器学习的工程师、数据科学家转型的非开发岗人员
- 机器学习、人工智能方向的培训班学员
- 各计算机科学、非计算机科学专业的大中专院校学生
- 想要在人工智能领域创业的创业者
- 需要使用机器学习知识解决日常问题的人员
- 其他对机器学习、人工智能有兴趣爱好的各位自学者

最后，还要感谢各位编辑老师的辛勤劳动，使本书可以顺利出版。如果读者朋友在学习过程中有任何问题，或者单纯地想聊聊天，欢迎添加作者本人微信：dynhyper，谢谢大家！

作 者

2018 年 3 月

目 录

第 7 章　支持向量机 SVM——专治线性不可分

第 8 章　神经网络——曾入“冷宫”，如今得宠

第 9 章 数据预处理、降维、特征提取及聚类——快刀斩乱麻

第 10 章 数据表达与特征工程——锦上再添花

第 13 章 文本数据处理——亲，见字如“数”

第 14 章 从数据获取到话题提取——从“研究员”到“段子手”

第 15 章　人才需求现状与未来学习方向——你是不是下一个“大牛”

第1章 概 述

近年来，全球新一代信息技术创新浪潮迭起。作为全球信息领域产业竞争的新一轮焦点，人工智能的发展迎来了第三次浪潮，它正在推动工业发展进入新的阶段，掀起第四次工业革命的序幕。而作为人工智能的重要组成部分，机器学习也成了炙手可热的概念。本章将向读者介绍机器学习的基础知识，为后面的学习打好基础。

本章主要涉及的知识点有：

- 什么是机器学习
- 机器学习的主要应用场景
- 机器学习应该如何入门
- 有监督学习和无监督学习的概念
- 分类、回归、泛化、过拟合与欠拟合等概念

1.1 什么是机器学习——从一个小故事开始

要搞清楚什么是机器学习，我们可以从一个小故事开始。

小 C 是一个即将毕业的大学生、单身的小伙子，他一直在暗地里喜欢隔壁班的女神，可是又苦于没有机会接近她，于是在很长一段时间里，小 C 只能保持这种暗恋的状态。

突然有一天，在一个很偶然的机会下，小 C 得到了女神的微信号，并且添加了她。然后开始密切关注她的朋友圈，观察她的一举一动。

不久小 C 就有了重大发现，女神在朋友圈经常发三种类型的内容：书籍、电影和旅游。这可是个了不起的发现，对于小 C 来说，千载难逢的机会来了。

接下来，小 C 把女神喜欢的书名和特征（Features）保存在电脑上，做成一个数据集（Dataset），然后根据这个数据集用“算法（Algorithm）”建立了一个“模型（Model）”，并且通过这个模型预测出了女神会喜欢哪一本新书，之后小 C 买下了模型预测出来的书，作为礼物送给了女神。

收到新书的女神很开心，也对小 C 产生了好感。

后来小 C 又用同样的方法预测出了女神喜欢的电影，并买票请女神去看。不出所料，每次女神的观影体验都棒极了，两个人的关系也越来越近。

再后来，小 C 又预测了女神会喜欢的旅游地点，订好机票和酒店，对女神发出了邀请。当然，女神不会拒绝小 C 了，因为这次旅游的目的地可是她一直想去的地方呢！

整个旅途愉快极了，小 C 总能像手术刀一样精准地切到女神最感兴趣的话题上。女神觉得太不可思议了，她问小 C：“为什么你会这么了解我呢？”小 C 按捺住内心的喜悦，故作镇定地说道：“这是机器学习的力量。”

“什么是机器学习啊？”女神不解。

是时候让小 C 展现出扎实的学术底蕴了，他抬头 45° 仰望星空，深沉地说道：

“机器学习，最早是由一位人工智能领域的先驱，Arthur Samuel（见图 1-1），在 1959 年提出来的。本意指的是一种让计算机在不经过明显编程的情况下，对数据进行学习，并且做出预测的方法，属于计算机科学领域的一个子集。公认的世界上第一个自我学习项目，就是 Samuel 跳棋游戏。而我也是通过机器学习的方法，通过你在社交媒体的数据预测出你的喜好的。”

图 1-1　Arthur Samuel 和他的跳棋游戏

毫无悬念地，女神对小 C 产生了深深的崇拜感，并且芳心暗许。从此以后，两个人走在了一起，并过上了幸福的生活。

对于一部童话来说，故事到这里就可以结束了。可是对于一本机器学习的入门书来说，我们才刚刚开始。

有了女朋友的小 C 也要背负起自己的责任了，他需要一份工作，才能为两个人的生活提供经济来源。很幸运的是，他通过校园招聘进入了国内最大的互联网公司——蝙蝠公司，成为一名机器学习工程师，从此开始了他的职业生涯。

1.2　机器学习的一些应用场景——蝙蝠公司的业务单元

小 C 入职的蝙蝠公司，作为国内互联网行业的龙头企业，其业务覆盖面十分广泛，包括电子商务、社交网络、互联网金融以及新闻资讯等。每一个方向在内部都被称为一个 BU（业务单元）。每个 BU 相对独立运作，有自己完善的体系。但机器学习技术，在每个 BU 都有非常深入的应用。下面我们来大致了解一下。

1. 电子商务中的智能推荐

蝙蝠公司的电子商务 BU 是国内最大的在线零售平台，其用户接近 5 亿，每天在线商品数超过 8 亿，平均每分钟会售出 4.8 万件商品。正因此，电子商务 BU 拥有海量的用户和商品数据。当然，为了让平台的成交总额（Gross merchandise Volume，GMV）不断提高，电子商务 BU 必须精确地为用户提供商品优惠信息。和小 C 预测女神的喜好类似，电子商务 BU 要通过机器学习，来对用户的行为进行预测。但在如此海量的数据下，模型要比小 C 的模型复杂很多。

比如某个男性用户的浏览记录和购买记录中有大量的数码产品，而且系统识别出该

用户访问平台时使用的设备是 iPhone 7，则算法很有可能会给该用户推荐 iPhone X 的购买链接。而另外一个女性用户浏览和购买最多的是化妆品和奢侈品，那么机器就会把最新款的 Hermès 或者 Chanel 产品推荐给她。

2. 社交网络中的效果广告

蝙蝠公司旗下的社交网络平台目前有超过 9 亿的月活跃用户（Monthly Active Users，MAU），其主要盈利模式是通过在社交网络中投放效果广告，从而为商家提供精准营销的服务。在这种盈利模式下，该 BU 需要保证广告的投放尽可能精准地到达目标受众，并转化为销售。因此需要机器学习算法来预测用户可能感兴趣的广告内容，并且将符合要求的内容展示给用户。

比如用户经常转发或点赞和汽车有关的信息，那么系统就会把某品牌新车上市的广告展示给用户，而如果用户经常转发或点赞的是和时尚相关的信息，那么系统推荐的广告就会是新一季的服装搭配潮流等。

3. 互联网金融中的风控系统

蝙蝠公司旗下另外一个业务单元——互联网金融事业部，主要是为用户提供小额贷款和投资理财服务。目前该 BU 拥有 4.5 亿用户，具有每秒处理近 9 万笔支付的能力。而且它的资产损失比率仅有 0.001%，这是一个非常恐怖的数字！要知道全球最老牌的在线支付工具资产损失比率也要 0.2%。要达到如此低的资产损失比率，必须要有强大的风控系统作为支撑，而风控系统背后，就是机器学习算法的应用。

例如，在这个场景中，风控系统要能够收集已知的用户欺诈行为，并对欺诈者的行为数据进行分析，再建立模型，在类似的欺诈行为再度发生之前就把它们扼杀在摇篮里，从而降低平台的资产损失比率。

4. 新闻资讯中的内容审查

蝙蝠公司旗下的新闻资讯业务单元的表现也同样让人眼前一亮。这个 BU 的产品主要是新闻客户端 APP，据称其激活用户数已经超过 6 亿，而平均每个用户使用时长达到了 76 分钟。而令人咂舌的是，这个业务单元下据说没有编辑人员，所有的内容处理都是通过机器学习算法自动完成的。

比如该产品的“精准辟谣”功能，就是主要依赖机器学习的算法，对内容进行识别。如果判断为是虚假信息，则会提交给审核团队，审核属实之后，虚假信息就会被系统屏蔽，不会给用户进行推送。

当然，蝙蝠公司的业务单元远远不止上述这几个，同时机器学习在这些业务单元中的应用也远远不止上述这几个方面。限于篇幅，本书就不再一一罗列了。

5. 机器学习在蝙蝠公司之外的应用

蝙蝠公司代表的是互联网行业，然而在互联网行业之外，机器学习也被广泛的应用。例如在医疗行业中所使用的专家系统，典型的案例就是诞生于 20 世纪 70 年代的 MYCIN 系统，该系统由斯坦福大学研制，它可以用患者的病史、症状和化验结果等作为原始数据，运用医疗专家的知识进行逆向推理，找出导致感染的细菌。若是多种细菌，则用 0 到 1 的数字给出每种细菌的可能性，并在上述基础上给出针对这些可能的细菌的药方。

此外，还有诸如智能物流、智能家居、无人驾驶等领域。可以说机器学习，已经非常深切地融入了我们的生活与工作当中。

6. 一些炫酷的“黑科技”

除了上述我们提到的已经广泛应用的领域，还有一些代表着未来发展趋势的案例，例如：2016 年，Google 旗下的 AI 程序 AlphaGo 首次战胜了人类围棋世界冠军。2017 年，埃隆 • 马斯克创办的 OpenAI 公司开发的人工智能程序在电子竞技游戏 DOTA 中战胜了人类世界冠军 Dendi。除此之外，还有很多诸如 AI 写新闻稿、画插画、写诗词等消息充斥着各大新闻网站的首页，仿佛用不了多久，AI 就会在各个方面全面替代人类，进而统治世界了。

本书并不想争论 AI 究竟会让人类生活得更美好，还是会成为地球的主宰者奴役我们。在这里只想和大家一起探究一下这些案例背后的原理。不管是 AlphaGo 还是 OpenAI，其背后的原理都是机器学习中的深度学习，它们的崛起在全球范围内掀起了一阵人工智能和深度学习的热潮。

实际上，人工智能这个概念并不是最近几年才出现的。早在 20 世纪 60 年代，人工智能就被提出，并且分为诸多学派。其中联结学派就是神经网络，或者说深度学习的代表。但受限于当时的计算能力，人工智能的发展也出现了停滞。

而随着时代的发展，现在的芯片计算能力越来越强，同时用户的数据量也越来越大，为人工智能的进一步发展提供了必要的先决条件，而机器学习、深度学习、神经网络等概念也随之火爆起来。同时在人才市场上，机器学习工程师、算法工程师、数据分析师等职位也呈现出了供不应求的场面，因此有更多的人开始投身到机器学习的研究当中。

1.3　机器学习应该如何入门——世上无难事

相信在看了上面的内容之后，一些读者朋友也已经动心，想要加入机器学习的领域当中了。保不齐能像故事中的小 C 一样，既能抱得美人归，又能找到一份心仪的工作。

但另外，又会担心自己基础薄弱，不知道从何入手。

不用担心！只要你肯多动脑，勤动手，相信很快就可以入门的。下面是我们给大家的一点学习方面的建议。

1. 从一种编程语言开始

如果你之前完全没有编程的基础，那么我们建议先从一门编程语言开始。目前市面上常用的编程语言有很多种，如 C++、Java、Python、R 等。那么该选择哪一种呢？不必纠结，编程语言并没有绝对的“好”和“不好”的区别，只是它们各自有各自的特点而已。而且如果你掌握了其中的一种，再学习其他的编程语言时，上手会快得多。

本书使用的语言是 Python，主要原因是：在数据科学领域，Python 已经成为了一门通用的编程语言。它既有通用编程语言的强大能力，同时还具有诸如 MATLAB 或者 R 之类针对某个专门领域语言的易用性。同时丰富和强大的库，让 Python 在数据挖掘、数据可视化、图像处理、自然语言处理等领域都有非常不俗的表现。

Python 还被称为“胶水语言”，因为它能够把用其他语言编写的各种模块轻松连接在一起。而它简洁清晰的语法和强制缩进的特点，都让 Python 对初学者非常友好。此外，它还是完全开源的，用户完全不需要支付任何费用。

由于 Python 语言的简洁性、易读性以及可扩展性，在国外用 Python 做科学计算的研究机构日益增多，一些知名大学已经采用 Python 来教授程序设计课程。众多开源的科学计算软件包都提供了 Python 的调用接口，如著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK。Google 的深度学习框架 TensorFlow 兼容得最好的语言之一，也是 Python。

2017 年 7 月 20 日，IEEE 发布 2017 年编程语言排行榜：Python 高居首位。

还有一个重要的原因，对于用户来说，Python 的学习成本是非常低的。哪怕是完全零基础的读者，在一个月左右的努力学习之后，也可以大致掌握它的基本语法和主要的功能模块。

因此我们推荐读者使用 Python 进行机器学习方面的研究与开发，在后面的章节我们会带大家配置基于 Python 的开发环境。

2. 熟悉机器学习中的基本概念

在对编程语言有了基本的掌握之后，读者朋友需要熟悉机器学习中的一些基本概念，比如什么是“有监督学习”，什么是“无监督学习”，它们之间的区别是什么，在应用方面有什么不同。另外，对机器学习的“分类”和“回归”有基本认知，清楚在什么场景下使用分类算法，在什么场景下使用回归算法。最后理解模型的“泛化”，明白在什

么情况下模型会出现“过拟合”的现象，在什么情况下会出现“欠拟合”的现象。

3. 了解机器学习中最常见的算法

在了解了基本概念之后，读者朋友就可以开始了解机器学习中最常用的一些算法了。比如 K 最近邻算法、线性模型、朴素贝叶斯、决策树、随机森林、SVMs、神经网络等。

在这个过程中，读者需要了解每种算法的基本原理和用途，它们的特性分别是什么，在不同的数据集中表现如何，如何使用它们建模，模型的参数如何调整等。

4. 掌握对数据进行处理的技巧

读者朋友可根据前述内容，对小数据集进行建模并且做出一些预测。但是在真实世界中，数据往往比我们拿来实验的小数据集复杂很多倍。它们的特征变量会大很多，也就是说数据的维度会高很多，同时可能完全没有训练数据集供你使用，这时候读者就需要掌握一些数据处理的技能，比如如何对数据进行降维，或者聚类，从而让数据更容易被理解，并从中找到关键点，为建模奠定基础。

5. 学会让模型更好地工作

学会用算法建模和对数据进行处理之后，读者朋友要做的是如何让模型更好地工作。例如，怎样做可以让算法的效率更高，怎样找到最适合的模型，模型最优的参数是什么，以及如何打造一个流水线，让几个模型在其中共同协作，以解决你的问题等。

6. 动手，一定要动手操作

学习一门知识最好的办法就是使用它，因此建议读者朋友一定要自己动手实操。不要嫌麻烦，尽可能把本书中的代码全部自己敲一下这样才能对内容有更加深刻的理解。如果觉得不够过瘾，还可以到知名的 Kaggle 大赛平台，或者“天池”算法大赛平台上，使用那些来自真实世界的数据来磨炼自己的技能。

当然，还有个更好的方法，那就是去企业中寻找一个机器学习工程师或是算法工程师的职位，在工作中学习，效果是最好的了。

1.4 有监督学习与无监督学习

在机器学习领域，有监督学习和无监督学习是两种常用的方法。有监督学习是通过现有训练数据集进行建模，再用模型对新的数据样本进行分类或者回归分析的机器学习方法。在监督式学习中，训练数据集一般包含样本特征变量及分类标签，机器使用不同的算法通过这些数据推断出分类的方法，并用于新的样本中。目前有监督学习算法已经比较成熟，并且在很多领域都有很好的表现。

而无监督学习，或者说非监督式学习，则是在没有训练数据集的情况下，对没有标签的数据进行分析并建立合适的模型，以便给出问题解决方案的方法。在无监督学习当中，常见的两种任务类型是数据转换和聚类分析。

其中数据转换的目的是，把本来非常复杂的数据集通过非监督式学习算法进行转换，使其变得更容易理解。常见的数据转换方法之一便是数据降维，即通过对特征变量较多的数据集进行分析，将无关紧要的特征变量去除，保留关键特征变量（例如，把数据集降至二维，方便进行数据可视化处理）。

而聚类算法则是通过把样本划归到不同分组的算法，每个分组中的元素都具有比较接近的特征。目前，聚类算法主要应用在统计数据分析、图像分析、计算机视觉等领域。

1.5 机器学习中的分类与回归

分类和回归是有监督学习中两个最常见的方法。对于分类来说，机器学习的目标是对样本的类标签进行预测，判断样本属于哪一个分类，结果是离散的数值。而对于回归分析来说，其目标是要预测一个连续的数值或者是范围。

这样讲可能会有一点抽象，我们还是用小 C 的例子来理解一下这两个概念。

比如，小 C 在使用算法模型预测女神的电影喜好时，他可以将电影分为“女神喜欢的”和“女神不喜欢的”两种类型，这就是二元分类，如果他要把电影分为“女神特别喜欢的”“女神有点喜欢的”“女神不怎么喜欢的”以及“女神讨厌的”四种类型，那么这就属于多元分类。

但如果小 C 要使用算法模型预测女神对某部电影的评分，例如，女神会给“速度与激情 8”打多少分，从 0 到 100，分数越高说明女神越喜欢，最终模型预测女神会给这部电影打 88 分，这个过程就称为回归。小 C 需要将女神给其他电影的评分和相对应的电影特征作为训练数据集，通过建立回归模型，来给“速度与激情 8”打分。

1.6 模型的泛化、过拟合与欠拟合

在有监督学习中，我们会在训练数据集上建立一个模型，之后会把这个模型用于新的，之前从未见过的数据中，这个过程称为模型的泛化（generalization）。当然我们希望模型对于新数据的预测能够尽可能准确，这样才能说模型泛化的准确度比较高。

那么我们用什么样的标准来判断一个模型的泛化是比较好的，还是比较差的呢？

我们可以使用测试数据集对模型的表现进行评估。如果你在训练数据集上使用了一个非常复杂的模型，以至于这个模型在拟合训练数据集时表现非常好，但是在测试数据集的表现非常差，说明模型出现了过拟合（overfitting）的问题。

相反，如果模型过于简单，连训练数据集的特点都不能完全考虑到的话，那么这样的模型在训练数据集和测试数据集的得分都会非常差，这个时候我们说模型出现了欠拟合（underfitting）的问题。

而只有模型在训练数据集和测试数据集得分都比较高的情况下，我们才会认为模型对数据拟合的程度刚刚好，同时泛化的表现也会更出色。

1.7　小结

现在我们来对本章的内容进行一下总结。在本章开始的部分，我们通过一个小故事了解了机器学习的基本概念，之后又对机器学习的部分应用场景进行了初步的学习。

之所以说“部分”应用场景，是因为机器学习的应用范围实在太广，我们很难穷举，相信读者朋友日后还会接触到更多元化的案例。

当读者朋友对机器学习产生兴趣之后，还可以在本章中找到对于机器学习入门的步骤和一些建议。当然，每个人有自己独到的学习方法，本章所列的方法仅仅供读者朋友参考，你也完全可以根据自身的情况安排自己的学习计划。

同时，为了让读者朋友能够更加顺利地学习后面的章节，本章还初步介绍了一些机器学习领域的术语，如监督学习、无监督学习、过拟合和欠拟合等。但请读者朋友注意，这部分内容也并非是将所有的术语进行罗列，如半监督学习和强化学习的概念，本章还没有涉及。相信日后随着读者朋友学习进程的加深，还会解锁更多新的知识点。

在第 2 章，本书将用手把手的方式，和读者朋友一起搭建机器学习的开发环境，相信乐于动手的你会找到很多乐趣。

第2章　基于Python语言的环境配置

在看完第1章的内容后，相信有一部分读者朋友可能已经忍不住想要动手操作一下了。工欲善其事，必先利其器。本章将帮助读者朋友把实验环境配置好。

事实上，有一些第三方机构发行了一些已经集成好必要的库的Python开发工具，如Anaconda、Enthought Canopy、Python（x,y）等，它们的主要功能是用于进行科学计算和大规模数据处理。如果你不想自己去动手逐步配置环境，那么直接下载这些工具也是不错的选择。但是对于新手而言，我强烈建议下载Python的原始安装文件，并尝试自己安装这些库，这样可以充分锻炼自己的动手能力。

本章主要涉及的知识点有：

- Python的下载和安装
- Jupyter Notebook的安装及使用方法
- Numpy、Scipy、matplotlib、pandas、scikit-learn等库的安装和使用方法

2.1 Python 的下载和安装

首先，我们需要通过 Python 官方网站 http://www.python.org 下载 Python 安装包，目前最新的版本是 3.6.2。在官网首页的导航条上找到“Downloads”按钮，鼠标悬停在上面时会出现一个下拉菜单，如图 2-1 所示。

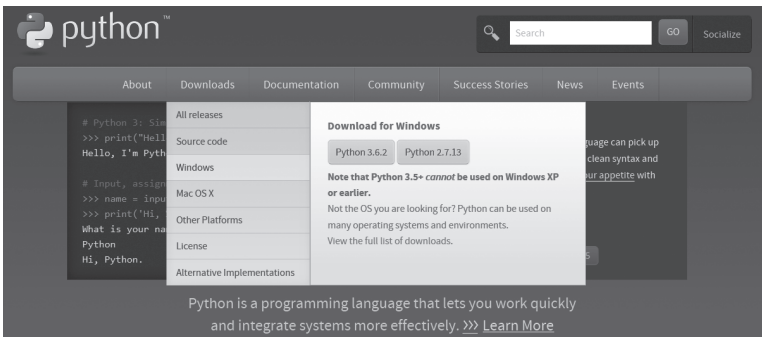

图 2-1　Python 官网下载入口

在下拉菜单中，根据自己的操作系统选择对应的 Python 版本，本书将以 Windows 为例进行讲解。

注意 **苹果的 MAC OS 自带了 Python 2.7.X，需要另行安装 Python 3.6.2。但是由于系统运行依赖于自带的 Python 2.7.X，因此请务必不要删除系统自带的版本。**

单击图 2-1 中所示的 Windows 按钮之后，将进入下载页面，在这里选择和自己系统匹配的安装文件。为了方便起见，我们选择 executable installer（可执行的安装程序），如果你的操作系统是 32 位的，请选择 Windows x86 executable installer；如果操作系统是 64 位的，请选择 Windows x86-64 executable installer，如图 2-2 所示。

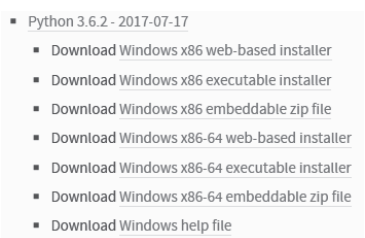

图 2-2　Python 3.6.2 不同版本下载链接

下载完成后，双击安装文件，在打开的软件安装界面中选择 Install Now 即可进行默

认安装，而选择 Customize installation 可以对安装目录和功能进行自定义。记得勾选 Add Python 3.6 to PATH 选项，以便把安装路径添加到 PATH 环境变量中，这样就可以在系统各种环境中直接运行 Python 了。

2.2　Jupyter Notebook 的安装与使用方法

在安装好 Python 后，使用其自带的 IDLE 编辑器就已经可以完成代码编写的功能了。但是自带的编辑器功能比较简单，所以可以考虑安装一款更强大的编辑器。本书推荐使用 Jupyter Notebook 作为开发工具。

Jupyter Notebook 是一款开源的 Web 应用，用户可以使用它编写代码、公式、解释性文本和绘图，并且可以把创建好的文档进行分享。目前，Jupyter Notebook 已经广泛应用于数据处理、数学模拟、统计建模、机器学习等重要领域。它支持四十余种编程语言，包括在数据科学领域非常流行的 Python、R、Julia 以及 Scala。用户还可以通过 E-mail、Dropbox、Github 等方式分享自己的作品。Jupyter Notebook 还有一个强悍之处在于，它可以实时运行代码并将运行结果显示在代码下方，给开发者提供了极大的便捷性。

时下最热门的 Kaggle 算法大赛中的文档几乎都是 Jupyter 格式，本书也使用了 Jupyter Notebook 进行创作。

下面我们来讲解一下 Jupyter Notebook 的安装和基本操作。

2.2.1　使用pip进行Jupyter Notebook的下载和安装

以管理员身份运行 Windows 系统自带的命令提示符，或者是 MAC OS X 的终端，输入下方的命令提示符如图 2-3 所示。

```
pip3 install jupyter
```

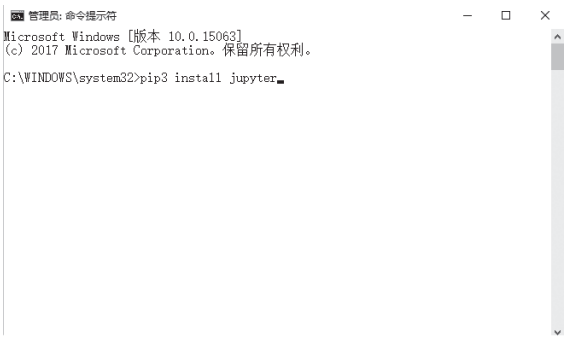

图 2-3　安装 Jupyter Notebook

稍等片刻，Jupyter Notebook 就会自动安装完成。在安装完成后，命令提示符会提示 Successfully installed jupyter-1.0.0，如图 2-4 中划线部分所示：

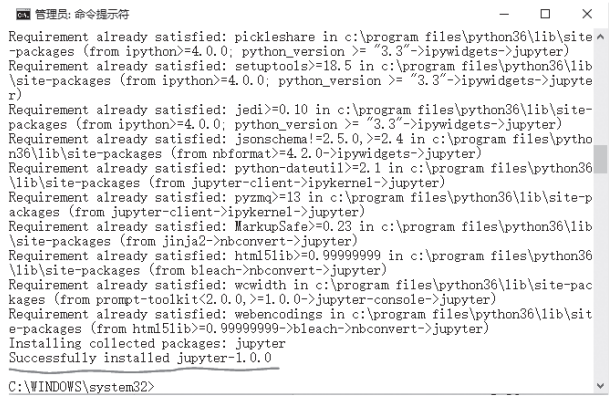

图 2-4　Windows 命令提示符提示 Jupyter Notebook 安装完成

2.2.2　运行Jupyter Notebook

在 Windows 的命令提示符或者是 MAC OS X 的终端中输入 jupyter notebook，就可以启动 Jupyter Notebook，如图 2-5 所示。

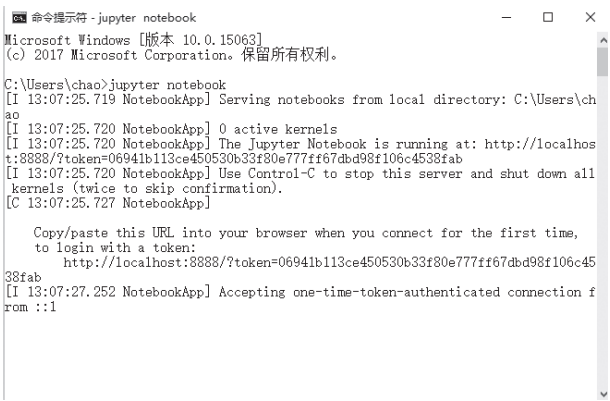

图 2-5　启动 Jupyter Notebook

这时电脑会自动打开默认的浏览器，并进入 Jupyter Notebook 的初始界面，如图 2-6 所示。

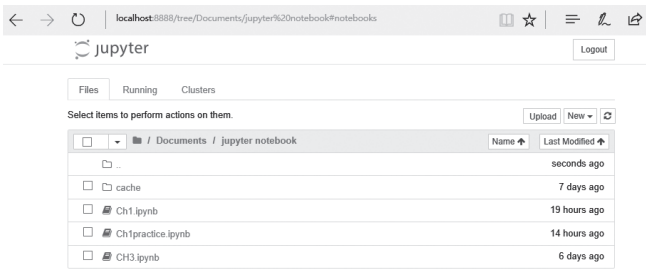

图 2-6　Jupyter Notebook 界面

2.2.3　Jupyter Notebook的使用方法

启动 Jupyter Notebook 之后，我们就可以使用它工作了。首先我们要先建立一个 notebook 文件，单击右上角的 New 按钮，在出现的菜单中选择 Python 3，如图 2-7 所示。

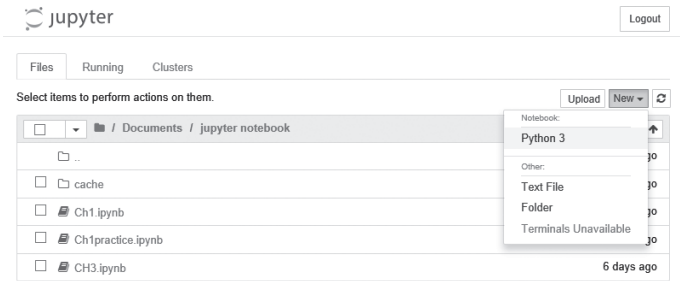

图 2-7　在 Jupyter Notebook 中新建一个文档

之后 Jupyter Notebook 会自动打开新建的文档，并出现一个空白的单元格（Cell）。现在我们试着在空白单元格中输入如下代码：

```
print('hello world')
```

按下 Shift + 回车键，你会发现 Jupyter Notebook 已经把代码的运行结果直接放在了单元格下方，并且在下面又新建了一个单元格，如图 2-8 所示。

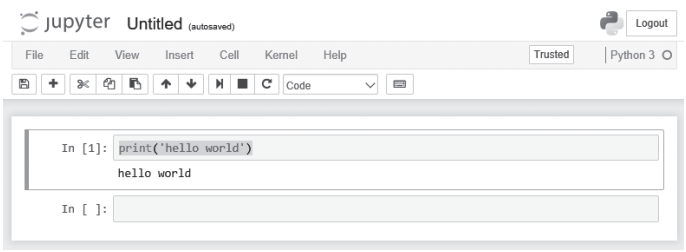

图 2-8　使用 Jupyter Notebook 打印“hello world”

注意 在 Jupyter Notebook 中，Shift + 回车表示运行代码并进入下一个单元格，而 Ctrl + 回车表示运行代码且不进入下一个单元格。

现在我们给这个文档重新命名为“hello world”，在 Jupyter Notebook 的 File 菜单中找到 Rename 选项，如图 2-9 所示。

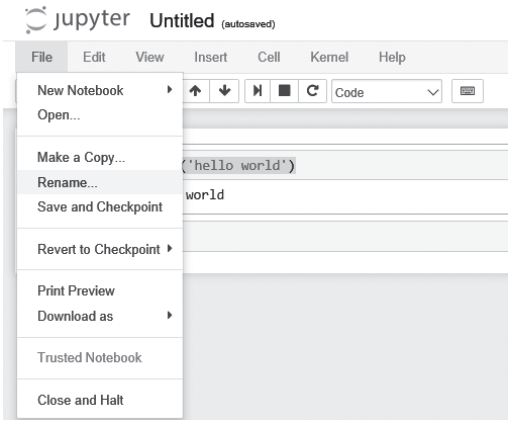

图 2-9　对文档进行重命名操作

之后在弹出的对话框中输入新名词“hello world”，单击 Rename 按钮确认，就完成了重命名操作。由于 Jupyter Notebook 会自动保存文档，此时我们已经可以在初始界面看到新建的“hello world.ipynb”文件了，如图 2-10 所示。

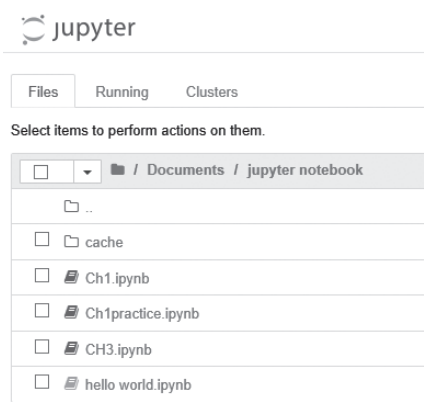

图 2-10　新建的 hello world 文档

Jupyter Notebook 还有很多奇妙的功能，我们留给读者慢慢探索。相信在熟悉之后，读者会对这个工具爱不释手的。

2.3　一些必需库的安装及功能简介

现在我们已经安装好了 Python 和 Jupyter Notebook，但是这还不够，我们还需要安装一些库，才能完成本书内容的学习与练习。这些库包括 Numpy、Scipy、matplotlib、pandas、IPython，以及非常核心的 scikit-learn。下面我们一起来安装这些库。

首先，如果你用的是 MAC OS X，那安装的过程会令人很舒服，你只需要在 MAC 的终端中输入一行命令：

```
sudo pip3 install numpy scipy matplotlib ipython pandas scikit-learn
```

然后安静地等待计算机把这些库逐一下载并安装好就可以了。

但如果是 Windows 系统，你可能会在安装 Scipy 这一步时遇到一些问题，解决方法是在下面这个链接中手动下载 Numpy + MKL 的安装文件和 Scipy 的安装文件。

http://www.lfd.uci.edu/ ～ gohlke/pythonlibs/

在这个链接的页面中分别找到和你的系统及 Python 版本相对应的 Numpy + MKL 安装文件和 Scipy 安装文件，并下载到本地计算机；然后以管理员身份运行 Windows 命令提示符，在命令提示符中进入两个安装文件所在的目录，输入命令如下：

```
pip install 安装文件全名
```

一定要先安装 Numpy + MKL 安装包，再安装 Scipy 才能成功。安装完成后，在 Python IDLE 中输入 import + 库名称来验证是否安装成功，例如，想知道 Scipy 是否安装成功，就在 IDLE 中输入如下代码：

```
import scipy
```

如果没有报错，则说明安装已经成功，可以使用了。现在我们一起来看一下这些库的主要功能。

注意 如果操作系统是 Windows 10，那么记得用管理员身份运行命令提示符，否则安装过程中可能会提示拒绝访问。

2.3.1　Numpy——基础科学计算库

Numpy 是一个 Python 中非常基础的用于进行科学计算的库，它的功能包括高维数组（array）计算、线性代数计算、傅里叶变换以及生产伪随机数等。Numpy 对于 scikit-learn 来说是至关重要的，因为 scikit-learn 使用 Numpy 数组形式的数据来进行处理，所以我们需要把数据都转化成 Numpy 数组的形式，而多维数组（n-dimensional array）也是

Numpy 的核心功能之一。为了让读者直观了解 Numpy 数组，下面我们在 Python 的 IDLE 中新建一个文件，然后输入几行代码来进行展示：

```
import numpy
#给变量i赋值为一个数组
i = numpy.array([[520,13,14],[25,9,178]])
#将i打印出来
print("i:\n{}".format(i))
```

将这三行代码保存成一个 py 文件，然后在编辑器窗口按 F5 运行，我们会得到如图 2-11 所示的结果。

```
代码运行结果：
i:
[[520  13  14]
 [ 25   9 178]]
```

图 2-11　一个简单的 numpy 数组

【结果分析】这里 i 就是一个典型的 Numpy 数组，在本书中，我们会大量用到 Numpy。后面我们会用“np 数组”来指代 Numpy 数组。

注意　对于零基础的读者来说，先不必纠结 Python 的 IDLE 编辑器用法，后面我们会主要使用 Jupyter Notebook 来进行代码的编写和运行。

2.3.2　Scipy——强大的科学计算工具集

Scipy 是一个 Python 中用于进行科学计算的工具集，它有很多功能，如计算统计学分布、信号处理、计算线性代数方程等。scikit-learn 需要使用 Scipy 来对算法进行执行，其中用得最多的就是 Scipy 中的 sparse 函数了。sparse 函数用来生成稀疏矩阵，而稀疏矩阵用来存储那些大部分数值为 0 的 np 数组，这种类型的数组在 scikit-learn 的实际应用中也非常常见。

下面我们用几行代码来展示一下 sparse 函数的用法：

```
import numpy as np
from scipy import sparse

matrix = np.eye(6)
#上面用numpy的eye函数生成一个6行6列的对角矩阵
#矩阵中对角线上的元素数值为1，其余都是0

sparse_matrix = sparse.csr_matrix(matrix)
```

```
#这一行把np数组转化为CSR格式的Scipy稀疏矩阵（sparse matrix）
#sparse函数只会存储非0元素

print("对角矩阵：\n{}".format(matrix))
#将生成的对角矩阵打印出来
print ("\nsparse存储的矩阵：\n{}".format(sparse_matrix))
#将sparse函数生成的矩阵打印出来进行对比
```

运行代码得到结果如图 2-12 所示。

```
对角矩阵：
[[ 1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  1.]]

sparse存储的矩阵：
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
  (4, 4)        1.0
  (5, 5)        1.0
```

图 2-12　对角矩阵和 sparse 存储的矩阵

【结果分析】在上面的代码中，我们使用了 numpy 的 eye 函数生成了一个 6 行 6 列的对角矩阵，所谓对角矩阵，即矩阵从左上角到右下角的对角线位置上的数值都是 1，而其他的位置都是 0。而用 sparse 进行转换后，我们可以看到矩阵的形式发生了一些变化。（0，0）表示矩阵的左上角，这个点对应的值是 1.0，而（1，1）代表矩阵的第 2 行第 2 列，这个点对应的数值也是 1.0，依此类推，直到右下角的点（5，5）。

从上面的代码和运行结果中，我们可以大致理解 sparse 函数的工作原理，在后面的内容中，我们还会接触到 Scipy 更多的功能。

2.3.3　pandas——数据分析的利器

pandas 是一个 Python 中用于进行数据分析的库，它可以生成类似 Excel 表格式的数据表，而且可以对数据表进行修改操作。pandas 还有个强大的功能，它可以从很多不同种类的数据库中提取数据，如 SQL 数据库、Excel 表格甚至 CSV 文件。pandas 还支持在不同的列中使用不同类型的数据，如整型数、浮点数，或是字符串。下面我们用一个例子来说明 pandas 的功能。在 Jupyter Notebook 中输入代码如下：

```
import pandas

#先创建一个同学个人信息的小数据集
```

```
data = {"Name":["小芋","小菡","小榆","小桔"],
        "City":["北京","上海","广州","深圳"],
        "Age":["18","20","22","24"],
        "Height":["162","161","165","166"]}
data_frame = pandas.DataFrame(data)
display(data_frame)
```

运行上述代码，会得到一个数据表如图 2-13 所示。

	Age	City	Height	Name
0	18	北京	162	小芋
1	20	上海	161	小菡
2	22	广州	165	小榆
3	24	深圳	166	小桔

图 2-13　pandas.Dataframe 生成的数据表

同时，我们还可以从数据表中进行查询操作，例如我们想把不在北京的同学信息显示出来，可以输入下面这一行代码：

```
display(data_frame[data_frame.City != "北京"])
#显示所有不在北京的同学信息
```

运行结果如图 2-14 所示。

	Age	City	Height	Name
1	20	上海	161	小菡
2	22	广州	165	小榆
3	24	深圳	166	小桔

图 2-14　显示所有不在北京的同学信息

现在我们对 pandas 有了一些初步的了解，在本书后面的内容中，我们还将深入讲解 pandas 的功能和用法。

2.3.4　matplotlib——画出优美的图形

matplotlib 是一个 Python 的绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形，它能够输出的图形包括折线图、散点图、直方图等。在数据可视化方面，matplotlib 拥有数量众多的忠实用户，其强悍的绘图能力能够帮我们对数据形成非常清晰直观的认知。下面我们来简单展示一下 matplotlib 的能力，在 Jupyter Notebook 中输入如下代码：

```
%matplotlib inline
#激活matplotlib
import matplotlib.pyplot as plt
#下面先生成一个从-20到20，元素数为10的等差数列
x = np.linspace(-20,20,10)
#再令 y = x^3 + 2x^2 + 6x + 5
y = x**3 + 2*x**2 + 6*x + 5
#下面画出这条函数的曲线
plt.plot(x,y, marker = "o")
```

运行代码可以得到结果如图 2-15 所示。

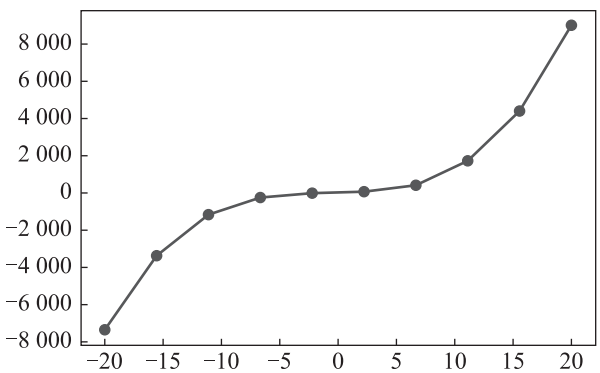

图 2-15　使用 matplotlib 绘制的图像

注意 **在代码开头的 %matplotlib inline 允许 Jupyter Notebook 进行内置实时绘图。如果不写这一行代码，则需要在最后加入 plt.show() 这一句，才能让图形显示出来。这两种方法在本书中都会涉及。**

2.4　scikit-learn——非常流行的 Python 机器学习库

scikit-learn 是如此重要，以至于我们需要单独对它进行一些介绍。scikit-learn 是一个建立在 Scipy 基础上的用于机器学习的 Python 模块。而在不同的应用的领域中，已经发展出为数众多的基于 Scipy 的工具包，它们被统一称为 Scikits。而在所有的分支版本中，scikit-learn 是最有名的。它是开源的，任何人都可以免费地使用它或者进行二次发行。

scikit-learn 包含众多顶级机器学习算法，它主要有六大类的基本功能，分别是分类、回归、聚类、数据降维、模型选择和数据预处理。scikit-learn 拥有非常活跃的用户社区，

基本上其所有的功能都有非常详尽的文档供用户查阅，我们也建议读者可以抽时间认真研究一下 scikit-learn 的用户指南以及文档，以便对其算法的使用有更充分了解。

2.5 小结

回顾一下，在这一章中，我们一起配置好了开发环境，并学习了 Numpy、Scipy、matplotlib、pandas、scikit-learn 等库的安装和基本使用方法，同时对 Jupyter Notebook 以及 Python 自带的 IDLE 编辑器也有了一定的了解。接下来在第 3 章中，我们将详细介绍 K 最近邻算法以及它的使用方法，希望读者能够从中学到对自己有用的知识。

第 3 章　K 最近邻算法——近朱者赤，近墨者黑

眼看没几天，就要到七夕佳节了，小 C 打算筹备个烛光晚餐，和女朋友浪漫一下。说到烛光晚餐，自然是得有瓶好酒助兴。可惜小 C 同学对酒实在没有什么研究，连最基本的酒的分类也说不清楚，看来又得求助机器学习了。

本章我们将介绍 K 最近邻算法（K-Nearest Neighbors，KNN）的原理和它的基本应用，并用它来帮助小 C 对酒进行分类。

本章主要涉及的知识点有：

- K最近邻算法的原理
- K最近邻算法在分类任务中的应用
- K最近邻算法在回归分析中的应用
- 使用K最近邻算法对酒的分类进行建模

3.1 K 最近邻算法的原理

K 最近邻算法的原理，正如我们本章标题所说——近朱者赤，近墨者黑。想象一下我们的数据集里面有一半是“朱”（图中浅色的点），另一半是“墨”（图中深色的点）。现在有了一个新的数据点，颜色未知，我们怎么判断它属于哪一个分类呢？如图 3-1 所示。

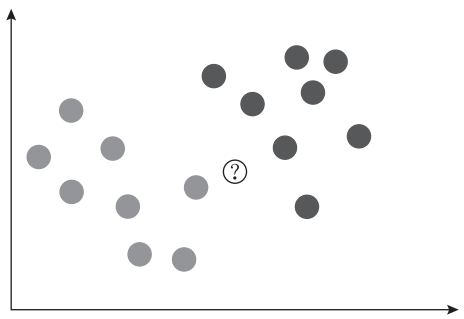

图 3-1　判断新数据点属于“朱”还是“墨”

对于 K 最近邻算法来说，这个问题就很简单：新数据点离谁最近，就和谁属于同一类，从图 3-1 中我们可以看出，新数据点距离它 8 点钟方向的浅色数据点最近，那么理所应当地，这个新数据点应该属于浅色分类了，如图 3-2 所示。

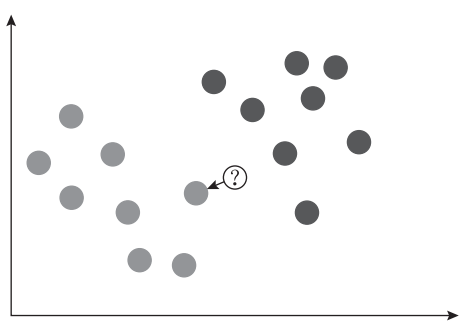

图 3-2　最近邻数等于 1 时的分类

看起来，K 最近邻算法真是够简单的，这么轻松就完成了分类的工作。别急，我们还没说完。刚才只是举的最简单的例子，选的最近邻数等于 1。但如果我们在模型训练过程中让最近邻数等于 1 的话，那么非常可能会犯了“一叶障目，不见泰山”的错误，试想一下，万一和新数据点最近的数据恰好是一个测定错误的点呢？

所以需要我们增加最近邻的数量，例如把最近邻数增加到 3，然后让新数据点的分

类和 3 个当中最多的数据点所处的分类保持一致，如图 3-3 所示。

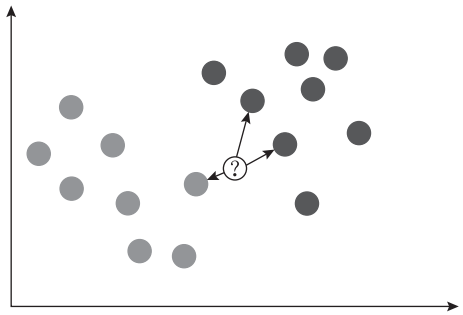

图 3-3　最近邻数等于 3 时的分类

从图 3-3 中我们看到，当我们令新数据点的最近邻数等于 3 的时候，也就是找出离新数据点最近的 3 个点，这时我们发现与新数据点距离最近的 3 个点中，有 2 个是深色，而只有 1 个是浅色。这样一来，K 最近邻算法就会把新数据点放进深色的分类当中。

以上就是 K 最近邻算法在分类任务中的基本原理，实际上 K 这个字母的含义就是最近邻的个数。在 scikit-learn 中，K 最近邻算法的 K 值是通过 n_neighbors 参数来调节的，默认值是 5。

注意 **K 最近算法也可以用于回归，原理和其用于分类是相同的。当我们使用 K 最近邻回归计算某个数据点的预测值时，模型会选择离该数据点最近的若干个训练数据集中的点，并且将它们的 y 值取平均值，并把该平均值作为新数据点的预测值。**

3.2　K 最近邻算法的用法

在这一小节中，我们会向大家展示 K 最近邻算法在分类和回归任务当中的应用，请大家准备好 Jupyter notebook，和我们一起进行实验吧。

3.2.1　K最近邻算法在分类任务中的应用

在 scikit-learn 中，内置了若干个玩具数据集（Toy Datasets），还有一些 API 让我们可以自己动手生成一些数据集。接下来我们会使用生成数据集的方式来进行展示，请大家在 Jupyter notebook 中输入代码如下：

```
#导入数据集生成器
```

```
from sklearn.datasets import make_blobs
#导入KNN分类器
from sklearn.neighbors import KNeighborsClassifier
#导入画图工具
import matplotlib.pyplot as plt
#导入数据集拆分工具
from sklearn.model_selection import train_test_split
#生成样本数为200，分类为2的数据集
data = make_blobs(n_samples=200, centers =2,random_state=8)
X, y = data
#将生成的数据集进行可视化
plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.spring, edgecolor='k')
plt.show()
```

在这段代码中，我们使用了 scikit-learn 的 make_blobs 函数来生成一个样本数量为 200，分类数量为 2 的数据集，并将其赋值给 X 和 y，然后我们用 matplotlib 将数据用图形表示出来，运行代码，会得到如图 3-4 所示的结果。

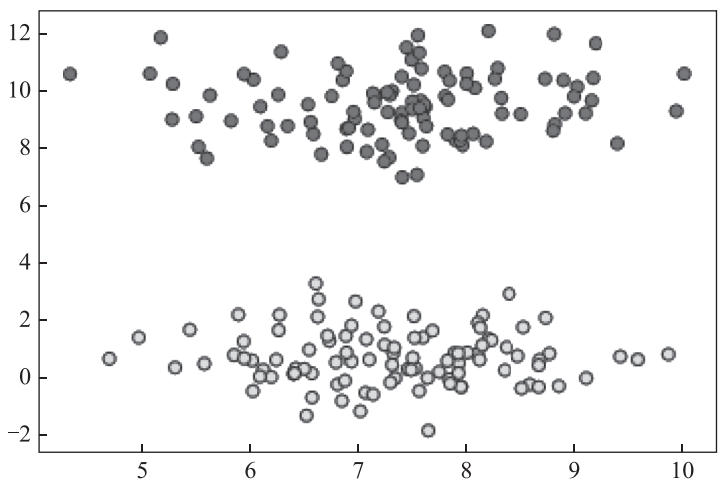

图 3-4　使用 make_blobs 生成的数据集

【结果分析】从图 3-4 中可以看出，make_blobs 生成的数据集一共有两类，其中一类用深色表示，而另外一类用浅色表示。读者朋友可能有点疑惑：这不是已经进行好分类了吗？我们还需要 K 最近邻算法做什么呢？

这确实是初学者非常容易提出的问题，答案是这样的——我们这里生成的数据集，可以看作机器学习的训练数据集，是已知的数据。我们就是基于这些数据用算法进行模型的训练，然后再对新的未知数据进行分类或者回归。

下面我们就使用 K 最近邻算法来拟合这些数据，输入代码如下：

```
import numpy as np
clf = KNeighborsClassifier()
clf.fit(X,y)

#下面的代码用于画图
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Pastel1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.spring, edgecolor='k')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Classifier:KNN")

plt.show()
```

运行代码，会得到如图 3-5 所示的结果。

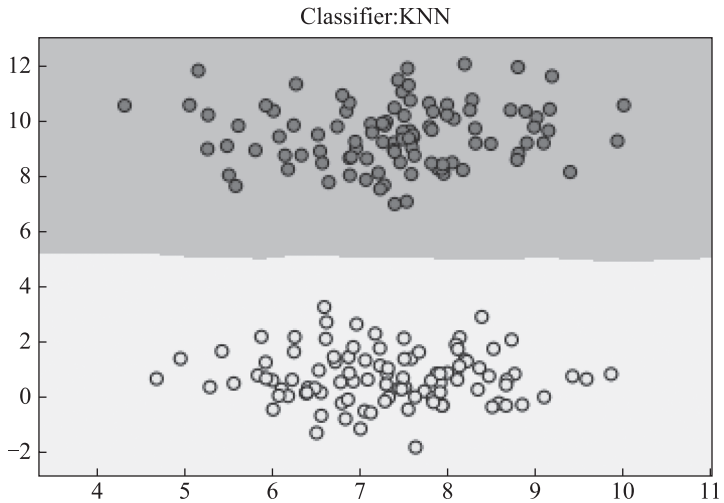

图 3-5　使用 K 最近邻算法创建的分类模型

【结果分析】从图 3-5 中我们可以看到，K 最近邻算法基于数据集创建了一个分类模型，就是图中粉色区域和灰色区域组成的部分。那么如果有新的数据输入的话，模型就会自动将新数据分到对应的分类中。

例如，我们假设有一个数据点，它的两个特征值分别是 6.75 和 4.82，我们来试验下模型能不能将它放到正确的分类中，首先我们可以在上面那段代码中，plt.show() 之前加一行代码如下：

```
#把新的数据点用五星表示出来
plt.scatter(6.75,4.82, marker='*', c='red', s=200)
```

再次运行代码，会得到结果如图 3-6 所示。

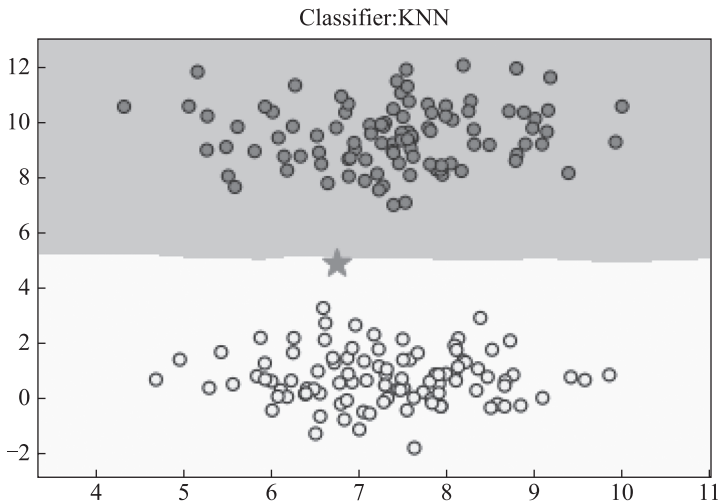

图 3-6　新的数据点所在的位置

【结果分析】图 3-6 中五角星就代表了新的数据点所在的位置，可以看到 K 最近邻算法将它放在了下方的区域，和浅色的数据点归为了一类。

下面我们再验证一下，输入代码如下：

```
#对新数据点分类进行判断
print('\n\n\n')#这一行代码主要是为了让截图好看一些
print('代码运行结果：')
print('================================')#打印分隔符让结果美观一些
print('新数据点的分类是：',clf.predict([[6.75,4.82]]))
print('================================')#同上
print('\n\n\n') #这一行代码主要是为了让截图好看一些
```

运行代码，我们将得到结果如图 3-7 所示。

```
代码运行结果：
================================
新数据点的分类是： [1]
================================
```

图 3-7　分类器对新数据点的分类判断

【结果分析】看起来，K 最近邻算法的工作成果还是很不错的，不过这可能是因为我们这次的任务有点太简单了。下面我们给它增加一点难度——处理多元分类任务。

3.2.2　K最近邻算法处理多元分类任务

接下来，我们要先生成多元分类任务所使用的数据集，为了让难度足够大，这次我们通过修改 make_blobs 的 centers 参数，把数据类型的数量增加到 5 个，同时修改 n_samlpes 参数，把样本量也增加到 500 个，输入代码如下：

```
#生成样本数为500，分类数为5的数据集
data2 = make_blobs(n_samples=500, centers=5,random_state=8)
X2,y2 = data2
#用散点图将数据集进行可视化
plt.scatter(X2[:,0],X2[:,1],c=y2, cmap=plt.cm.spring,edgecolor='k')
plt.show()
```

运行代码，会得到结果如图 3-8 所示的结果。

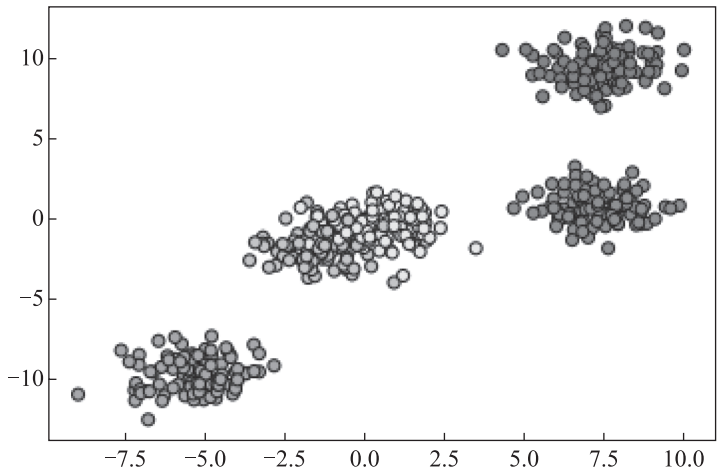

图 3-8　分类数量为 5 的数据集

【结果分析】从图 3-8 中我们可以看到，新的数据集的分类数量变成了 5 个，而其中有两类数据还有一些重合（图片中心位置的点），这下难度提高了不少。

让我们再次用 K 最近邻算法建立模型来拟合这些数据，输入代码如下：

```
clf = KNeighborsClassifier()
clf.fit(X2,y2)

#下面的代码用于画图
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Pastel1)
```

```
plt.scatter(X2[:, 0], X2[:, 1], c=y2, cmap=plt.cm.spring, edgecolor='k')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Classifier:KNN")
plt.show()
```

运行代码，将会得到如图 3-9 所示的结果。

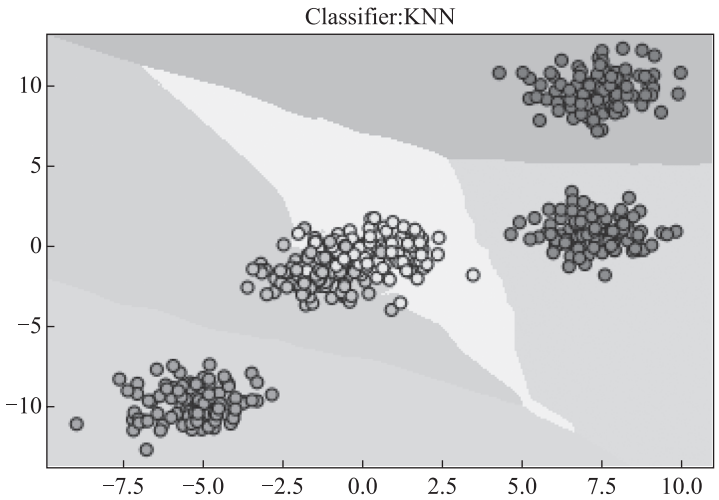

图 3-9　K 最近邻算法对 5 类数据进行的分类

【结果分析】从图 3-9 中我们可以看到，K 最近邻算法仍然可以把大部分数据点放置于正确的分类中，但有一小部分数据还是进入了错误的分类中，这些分类错误的数据点基本都是互相重合的位于图像中心位置的数据点。

那么模型的正确率究竟有多高呢？我们用下面的代码来进行一下评分：

```
#将模型的评分进行打印
print('\n\n\n')
print('代码运行结果：')
print('==============================')
print('模型正确率：{:.2f}'.format(clf.score(X2,y2)))
print('==============================')
print('\n\n\n')
```

运行代码，可以得到结果如图 3-10 所示。

```
代码运行结果：
==============================
模型正确率：0.96
==============================
```

图 3-10　模型在新数据集中的得分

【结果分析】看来虽然我们故意刁难了 K 最近邻算法一下，但它仍然能够将 96% 的数据点放进正确的分类中，这个结果可以说还是相当不错的。

接下来，我们再试试使用 K 最近邻算法来进行回归分析，看看结果如何。

3.2.3　K最近邻算法用于回归分析

在 scikit-learn 的数据集生成器中，有一个非常好的用于回归分析的数据集生成器，make_regression 函数，这里我们使用 make_regression 生成数据集来进行实验，演示 K 最近邻算法在回归分析中的表现。

首先我们还是先来生成数据集，输入代码如下：

```
#导入make_regression数据集生成器
from sklearn.datasets import make_regression
#生成特征数量为1，噪音为50的数据集
X, y = make_regression(n_features=1,n_informative=1, noise=50,random_state=8)
#用散点图将数据点进行可视化
plt.scatter(X,y,c='orange',edgecolor='k')
plt.show()
```

为了方便画图，我们选择样本的特征数量仅为 1 个，同时为了增加难度。我们添加标准差为 50 的 noise，运行代码，将会得到如图 3-11 所示的结果。

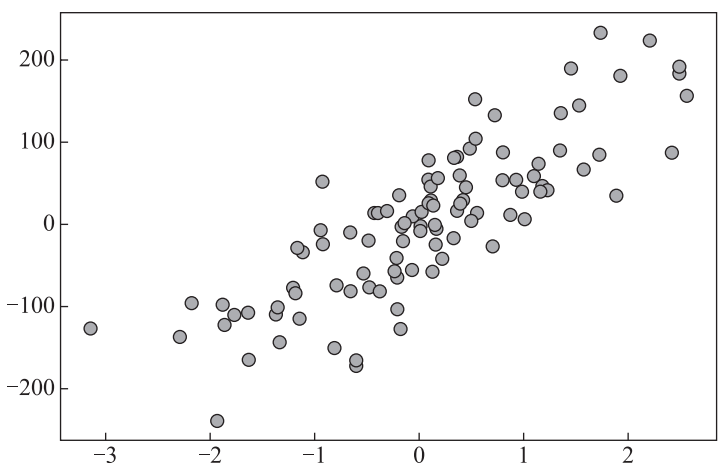

图 3-11　使用 make_regression 生成的数据集

【结果分析】从图 3-11 中我们可以看到，横轴代表的是样本特征的数值，范围大概在 −3 ～ 3；纵轴代表样本的测定值，范围大致在 −250 ～ 250。

下面我们使用 K 最近邻算法来进行回归分析，输入代码如下：

```
#导入用于回归分析的KNN模型
from sklearn.neighbors import KNeighborsRegressor
reg = KNeighborsRegressor()
#用KNN模型拟合数据
reg.fit(X,y)
#把预测结果用图像进行可视化
z = np.linspace(-3,3,200).reshape(-1,1)
plt.scatter(X,y,c='orange',edgecolor='k')
plt.plot(z, reg.predict(z),c='k',linewidth=3)
#向图像添加标题
plt.title('KNN Regressor')
plt.show()
```

运行代码，将会得到如图 3-12 所示的结果。

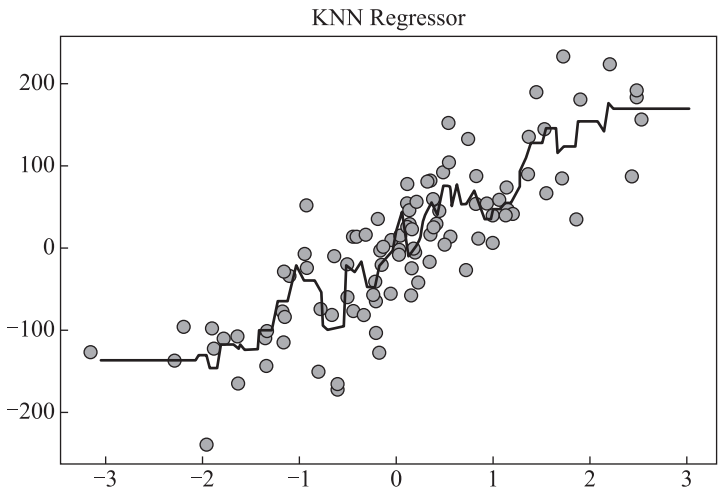

图 3-12　使用 K 最近邻算法对数据进行回归分析

【结果分析】图 3-12 中黑色的曲线代表的就是 K 最近邻算法拟合 make_regression 生成数据所进行的模型。直观来看，模型的拟合程度并不是很好，有大量的数据点都没有被模型覆盖到。

现在我们尝试给模型进行评分，看看结果如何，输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================')
print('模型评分：{:.2f}'.format(reg.score(X,y)))
print('==============================')
print('\n\n\n')
```

运行代码，会得到结果如图 3-13 所示。

```
代码运行结果：
==============================
模型评分：0.77
==============================
```

图 3-13　最近邻数为 5 时 KNN 回归模型的得分

【结果分析】模型的得分只有 0.77，这是一个差强人意的结果，和我们目测的情况基本一致，为了提高模型的分数，我们将 K 最近邻算法的近邻数进行调整。由于在默认的情况下，K 最近邻算法的 n_neighbors 为 5，我们尝试将它减少。

输入代码如下：

```
from sklearn.neighbors import KNeighborsRegressor
#减少模型的n_neighbors参数为2
reg2 = KNeighborsRegressor(n_neighbors=2)
reg2.fit(X,y)
#重新进行可视化
plt.scatter(X,y,c='orange',edgecolor='k')
plt.plot(z, reg2.predict(z),c='k',linewidth=3)
plt.title('KNN Regressor: n_neighbors=2')
plt.show()
```

在这段代码中，我们将 K 最近邻算法的 n_neighbors 参数降低为 2，再次运行代码，将会得到如图 3-14 所示的结果。

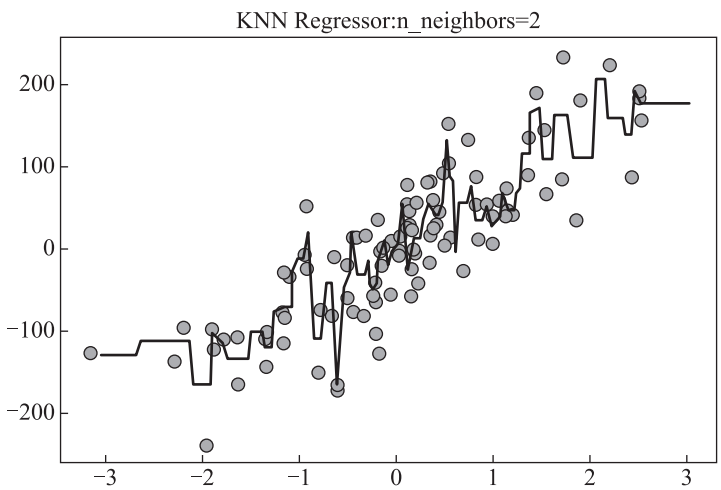

图 3-14　n_neighbors=2 时的模型

【结果分析】从图 3-14 中我们可以看到，相对于图 3-10 来说，黑色曲线更加积极地试图覆盖更多的数据点，也就是说，模型变得更复杂了。看起来比 n_neighbors 等于 5

的时候更加准确了，我们再次进行评分，看看分数是否有了提高。

输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================')
print('模型评分：{:.2f}'.format(reg2.score(X,y)))
print('==============================')
print('\n\n\n')
```

运行代码，会得到如图 3-15 所示的结果。

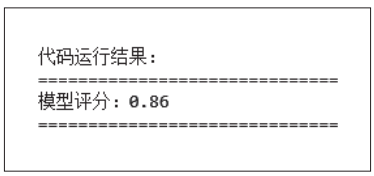

图 3-15　降低 n_neighbors 参数数值后的模型得分

【结果分析】和我们预料的一样，模型的评分从 0.77 提升到了 0.86，可以说是有显著的提升。不过以上都是基于我们虚构的数据所进行的实验，接下来我们用一个来自真实世界的数据集来进行 K 最近邻算法的实战。

3.3　K 最近邻算法项目实战——酒的分类

在看完上面的内容之后，我们和大家一起动手，用 K 最近邻算法帮助小 C 对酒的分类进行建模。这里建议读者一定要跟着书本的内容自己把代码敲一遍，这样可以对 Python 中几个用于机器学习的功能库有更加直观的体会。

3.3.1　对数据集进行分析

在本节中，我们将使用 scikit-learn 内置的酒数据集来进行实验，这个数据集也包含在 scikit-learn 的 datasets 模块当中。下面我们在 Jupyter Notebook 中新建一个 Python 3 的记事本，从头开始完成这个小项目。

首先，我们要把酒的数据集载入项目中，在 Jupyter Notebook 中输入代码如下：

```
from sklearn.datasets import load_wine
#从sklearn的datasets模块载入数据集
wine_dataset = load_wine()
```

现在读者朋友可能会比较好奇这个酒数据集中的数据都包含些什么。实际上，使用

load_wine 函数载入的酒数据集，是一种 Bunch 对象，它包括键（keys）和数值（values），下面我们来检查一下酒数据集都有哪些键，在 Jupyter Notebook 中输入代码如下：

```
#打印酒数据集中的键
print('\n\n\n')
print('代码运行结果：')
print('================================')
print("红酒数据集中的键:\n{}".format(wine_dataset.keys()))
print('================================')
print('\n\n\n')
```

运行代码，会得到结果如图 3-16 所示。

```
代码运行结果：
================================
红酒数据集中的键:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
================================
```

图 3-16　酒的数据集中的键

【结果分析】从结果中我们可以看出，酒数据集中包括数据“data”，目标分类“target”，目标分类名称“target_names”，数据描述“DESCR”，以及特征变量的名称“features_names”。

那么这个数据集中究竟有多少样本（samples），又有多少变量（features）呢？可以使用 .shape 语句来让 Python 告诉我们数据的大概轮廓，在 Jupyter Notebook 中输入代码如下：

```
#使用.shape来打印数据的概况
print('\n\n\n')
print('代码运行结果：')
print('================================')
print('数据概况：{}'.format(wine_dataset['data'].shape))
print('================================')
print('\n\n\n')
```

运行代码，会得到结果如图 3-17 所示。

```
代码运行结果：
================================
数据概况：(178, 13)
================================
```

图 3-17　酒的数据集的概况

【结果分析】从图 3-17 中我们可以看出，酒数据集中共有 178 个样本，每条数据有 13 个特征变量。

更细节的信息，我们可以通过打印 DESCR 键来获得，下面我们输入代码如下：

```
#打印酒的数据集中的简短描述
print(wine_dataset['DESCR'])
```

运行代码，我们将会看到一段很长的描述，如图 3-18 所示。

```
Wine Data Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - 1) Alcohol
                - 2) Malic acid
                - 3) Ash
                - 4) Alcalinity of ash
                - 5) Magnesium
                - 6) Total phenols
                - 7) Flavanoids
                - 8) Nonflavanoid phenols
                - 9) Proanthocyanins
                - 10)Color intensity
                - 11)Hue
                - 12)OD280/OD315 of diluted wines
                - 13)Proline
                - class:
                - class_0
                - class_1
                - class_2
```

图 3-18　酒数据集的部分简短描述

【结果分析】从结果中我们可以看出，酒数据集中的 178 个样本被归入 3 个类别中，分别是 class_0，class_1 和 class_2，其中 class_0 中包含 59 个样本，class_1 中包含 71 个样本，class_2 中包含 48 个样本。而从 1）至 13）分别是 13 个特征变量，包括酒精含量、苹果酸、镁含量、青花素含量、色彩饱和度等。我们先不用管每一个变量具体的含义，接下来先对数据进行一些处理。

3.3.2　生成训练数据集和测试数据集

在我们创建一个能够自动将酒进行分类的机器学习的算法模型之前，先要能够对模型的可信度进行评判，否则我们无法知道它对于新的酒所进行的分类是否准确。那么问题来了，如果我们用生成模型的数据去评估算法模型，那得分肯定是满分，这就好像我们按照一个体重 75kg 的人的身材数据缝制了一件衣服，那么这件衣服对于这个人肯定是百分之一百合身的，但如果换了一个体重 85kg 的人，这件衣服就不一定合适了。

所以我们现在要做的工作是，把数据集分为两个部分：一部分称为训练数据集；另一部分称为测试数据集。训练数据集就好比我们缝制衣服时所用到的模特的身材，而测

试数据集则是用来测试这件衣服，对于别人来说究竟有多合身的模特。

在 scikit-learn 中，有一个 train_test_split 函数，它是用来帮助用户把数据集拆分的工具。其工作原理是：train_test_split 函数将数据集进行随机排列，在默认情况下将其中 75% 的数据及所对应的标签划归到训练数据集，并将其余 25% 的数据和所对应的标签划归到测试数据集。

注意　我们一般用大写的 X 表示数据的特征，而用小写的 y 表示数据对应的标签。这是因为 X 是一个二维数组，也称为矩阵；而 y 是一个一维数组，或者说是一个向量。

接下来，我们使用 train_test_split 函数将酒的数据集中的数据分为训练数据集和测试数据集。在 Jupyter Notebook 中输入代码如下：

```
#导入数据集拆分工具
from sklearn.model_selection import train_test_split
#将数据集拆分为训练数据集和测试数据集
X_train, X_test, y_train, y_test = train_test_split(
wine_dataset['data'], wine_dataset['target'], random_state=0)
```

此时，我们已经对酒数据集完成了拆分。在上述代码中，我们看到了一个参数称为 random_state，并且我们将它指定为 0。这是因为 train_test_split 函数会生成一个伪随机数，并根据这个伪随机数对数据集进行拆分。而我们有时候需要在一个项目中，让多次生成的伪随机数相同，方法就是通过固定 random_state 参数的数值，相同的 random_state 参数会一直生成同样的伪随机数，但当这个值我们设为 0，或者保持缺省的时候，则每次生成的伪随机数均不同。

下面我们看一看 train_test_split 函数拆分后的数据集大概是什么情况，在 Jupyter Notebook 中输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印训练数据集中特征向量的形态
print('X_train shape:{}'.format(X_train.shape))
#打印测试数据集中的特征向量的形态
print('X_test shape:{}'.format(X_test.shape))
#打印训练数据集中目标的形态
print('y_train shape:{}'.format(y_train.shape))
#打印测试数据集中目标的形态
print('y_test shape:{}'.format(y_test.shape))
print('\n==============================')
print('\n\n\n')
```

运行代码，得到结果如图 3-19 所示。

```
代码运行结果：
==============================

X_train shape:(133, 13)
X_test shape:(45, 13)
y_train shape:(133,)
y_test shape:(45,)

==============================
```

图 3-19　经过拆分的训练集与测试集的数据形态

【结果分析】此刻我们可以看到在训练数据集中，样本 X 数量和其对应的标签 y 数量均为 133 个，约占样本总量的 74.7%，而测试数据集中的样本 X 数量和标签 y 数量均为 45 个，约占样本总数的 25.3%。同时，不论是在训练数据集中，还是在测试数据集中，特征变量都是 13 个。

3.3.3　使用K最近邻算法进行建模

在获得训练数据集和测试数据集之后，就可以机器学习的算法进行建模了。scikit-learn 中整合了众多的分类算法，究竟应该使用哪一种呢？这里选择了 K 最近邻算法，因为我们在接下来的一章当中会详细介绍 K 最近邻算法，所以提前给读者们展示一下它的用法。

K 最近邻算法根据训练数据集进行建模，在训练数据集中寻找和新输入的数据最近的数据点，然后把这个数据点的标签分配给新的数据点，以此对新的样本进行分类。现在我们在 Jupyter Notebook 中输入代码如下：

```
#导入KNN分类模型
from sklearn.neighbors import KNeighborsClassifier
#指定模型的n_neighbors参数值为1
knn = KNeighborsClassifier(n_neighbors = 1)
```

到这里读者可能会发现，我们给 KNeighborsClassifier 指定了一个参数，n_neighbors=1。正如我们在前文中所说，在 scikit-learn 中，机器学习模块都是在其固定的类中运行的，而 K 最近邻分类算法是在 neighbors 模块中的 KNeighborsClassifier 类中运行。而我们从一个类中创建对象的时候，就需要给模型指定一个参数。对于 KNeighborsClassifier 类来说，最关键的参数就是近邻的数量，也就是 n_neighbors。而 knn 则是我们在 KNeighborsClassifier 类中创建的一个对象。

接下来我们要使用这个叫作 knn 的对象中称为“拟合（fit）”的方法来进行建模，建模的依据就是训练数据集中的样本数据 X_train 和其对应的标签 y_train，所以我们输入

代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#用模型对数据进行拟合
knn.fit(X_train, y_train)
print(knn)
print('\n==============================')
print('\n\n\n')
knn.fit(X_train, y_train)
```

运行上述代码，可得到结果如图 3-20 所示。

```
代码运行结果：
==============================

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')

==============================
```

图 3-20　程序返回的模型参数

【结果分析】从图 3-20 中我们可以看到 knn 的拟合方法把自身作为结果返回给了我们。从结果中我们能够看到模型全部的参数设定，当然了，除了我们指定的 n_neighbors=1 之外，其余参数都保持默认值即可。

3.3.4　使用模型对新样本的分类进行预测

现在我们可以使用刚刚建好的模型对新的样本分类进行预测了，不过在这之前，可以先用测试数据集对模型进行打分，这就是我们创建测试数据集的目的。测试数据集并不参与建模，但是我们可以用模型对测试数据集进行分类，然后和测试数据集中的样本实际分类进行对比，看吻合度有多高。吻合度越高，模型的得分越高，说明模型的预测越准确，满分是 1.0。

下面开始评分，在 Jupyter Notebook 中输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型的得分
print('测试数据集得分：{:.2f}'.format(knn.score(X_test, y_test)))
print('\n==============================')
print('\n\n\n')
```

运行代码，得到评分如图 3-21 所示。

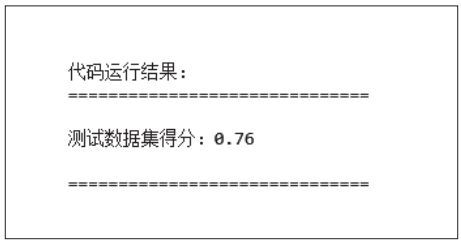

图 3-21　模型在测试数据集中的得分

【结果分析】我们看到，这个模型在预测测试数据集的样本分类上得分并不高，只有 0.76，也就是说，模型对于新的样本数据做出正确分类预测的概率是 76%。这个结果确实差强人意，不过我们只是用来演示 K 最近邻算法，所以可以先不用太纠结分数的问题。

下面假设我们得到了一瓶新的酒，它的特征变量值经测定如表 3-1 所列。

表 3-1　对新酒测定的特征变量

1） Alcohol	13.2
2） Malic acid	2.77
3） Ash	2.51
4） Alcalinity of ash	18.5
5） Magnesium	96.6
6） Total phenols	1.04
7） Flavanoids	2.55
8） Nonflavanoid phenols	0.57
9） Proanthocyanins	1.47
10）Color intensity	6.2
11）Hue	1.05
12）OD280/OD315 of diluted wines	3.33
13）Proline	820

现在我们用建好的模型对新酒做出分类预测，在 Jupyter Notebook 中输入代码如下：

```
import numpy as np
#输入新的数据点
X_new = np.array([[13.2,2.77,2.51,18.5,96.6,1.04,2.55,0.57,
                   1.47,6.2,1.05,3.33,820]])
#使用.predict进行预测
prediction = knn.predict(X_new)
print('\n\n\n')
print('代码运行结果：')
```

```
print('==============================\n')
print("预测新红酒的分类为：{}".format(wine_dataset['target_names'][prediction]))
print('\n==============================')
print('\n\n\n')
```

运行代码，得到结果如图 3-22 所示。

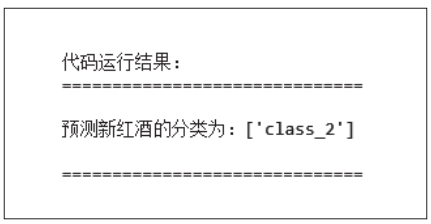

图 3-22　模型对于新酒数据进行的分类判断

【结果分析】模型把新酒的分类预测为 class_2，虽然准确率只有 76%，但对于我们的第一个机器学习的实战项目来说，还是相当不错的。

3.4　小结

在本章中，我们介绍了 K 最近邻算法的原理和它的使用方法，包括 K 最近邻分类和 K 最近邻回归，并且使用 K 最近邻算法帮助小 C 对酒的分类进行了分析。不过我们也看到，对于这个 13 维的数据集来说，K 最近邻算法的表现，并不能用优异来形容。这也确实是 K 最近邻算法的一大软肋。

K 最近邻算法可以说是一个非常经典而且原理十分容易理解的算法，作为第一个算法来进行学习是可以帮助大家在未来能够更好地理解其他的算法模型。不过 K 最近邻算法在实际使用当中会有很多问题，例如它需要对数据集认真地进行预处理、对规模超大的数据集拟合的时间较长、对高维数据集拟合欠佳，以及对于稀疏数据集束手无策等。所以在当前的各种常见的应用场景中，K 最近邻算法的使用并不多见。

接下来，我们会开始学习同样经典，而且在高维数据集中表现良好的算法——广义线性模型。

第 4 章　广义线性模型——“耿直”的算法模型

别看小 C 对女朋友体贴得无微不至，但朋友们都知道他其实是一个“直男”，说话直，做事直，连最常用的算法都很“直”——也就是我们在这一章要介绍的线性模型。

线性模型是一类广泛应用于机器学习领域的预测模型，在过去的几十年里有众多学者都对其进行了深入的研究。线性模型是使用输入数据集的特征的线性函数进行建模，并对结果进行预测的方法。在这一章中，我们会介绍几种常见的线性模型。

本章主要涉及的知识点有：

- 线性模型的基本概念
- 线性回归模型
- 岭回归模型
- 套索回归模型
- 二元分类器中的逻辑回归和线性SVC模型

4.1 线性模型的基本概念

线性模型原本是一个统计学中的术语，近年来越来越多地应用在机器学习领域。实际上线性模型并不是特指某一个模型，而是一类模型。在机器学习领域，常用的线性模型包括线性回归、岭回归、套索回归、逻辑回归和线性 SVC 等。下面我们先来研究一下线性模型的公式及特点。

4.1.1 线性模型的一般公式

在回归分析当中，线性模型的一般预测公式如下：

$$\hat{y} = w[0] \cdot x[0] + w[1] \cdot x[1] + \cdots + w[p] \cdot x[p] + b$$

式中：$x[0]$，$x[1]$，…，$x[p]$ 为数据集中特征变量的数量（这个公式表示数据集中的数据点一共有 p 个特征）；w 和 b 为模型的参数；$\hat{y}$ 为模型对于数据结果的预测值。对于只有一个特征变量的数据集，公式可以简化为

$$\hat{y} = w[0] \cdot x[0] + b$$

是不是觉得这个公式看上去像是一条直线的方程的解析式？没错，$w[0]$ 是直线的斜率，b 是 y 轴偏移量，也就是截距。如果数据的特征值增加的话，每个 w 值就会对应每个特征直线的斜率。如果换种方式来理解的话，那么模型给出的预测可以看作输入特征的加权和，而 w 参数就代表了每个特征的权重，当然，w 也可以是负数。

注意 **$\hat{y}$ 读作“y hat”，代表 y 的估计值。**

假设我们有一条直线，其方程是 $y = 0.5x+3$，我们可以使用 Jupyter Notebook 将它画出来，在 Jupyter Notebook 中输入代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
#令x为-5到5之间，元素数为100的等差数列
x = np.linspace(-5,5,100)
#输入直线方程
y = 0.5*x + 3
plt.plot(x,y,c='orange')
#图题设为"Straight Line"
plt.title('Straight Line')
plt.show()
```

运行代码，我们可以得到如图 4-1 所示的结果。

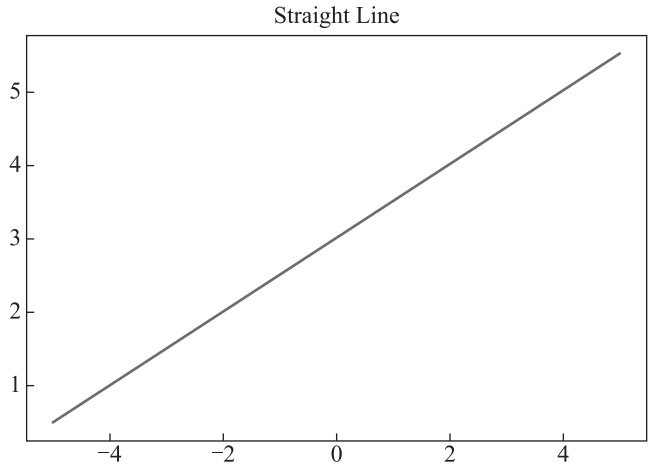

图 4-1　$y = 0.5x+3$ 的直线

【结果分析】图 4-1 中的直线，便是直线方程 $y = 0.5x+3$ 的图像，而线性模型正是通过训练数据集确定自身的系数（斜率）和截距的。

下面我们来看一下线性模型的工作原理。

4.1.2　线性模型的图形表示

大家肯定还记得，我们在初中数学（也可能是小学数学）中学过，两个点可以确定一条直线。假设有两个点，它们的坐标是（1，3）和（4，5），那么我们可以画一条直线来穿过这两个点，并且计算出这条直线的方程。下面我们在 Jupyter Notebook 中输入代码如下：

```
#导入线性回归模型
from sklearn.linear_model import LinearRegression
#输入两个点的横坐标
X = [[1],[4]]
#输入两个点的纵坐标
y = [3,5]
#用线性模型拟合这个两个点
lr = LinearRegression().fit(X,y)
#画出两个点和直线的图形
z = np.linspace(0,5,20)
plt.scatter(X,y,s=80)
plt.plot(z, lr.predict(z.reshape(-1,1)),c='k')
#设定图题为Straight Line
plt.title('Straight Line')
#将图片显示出来
plt.show()
```

运行代码，将会得到如图 4-2 所示的结果。

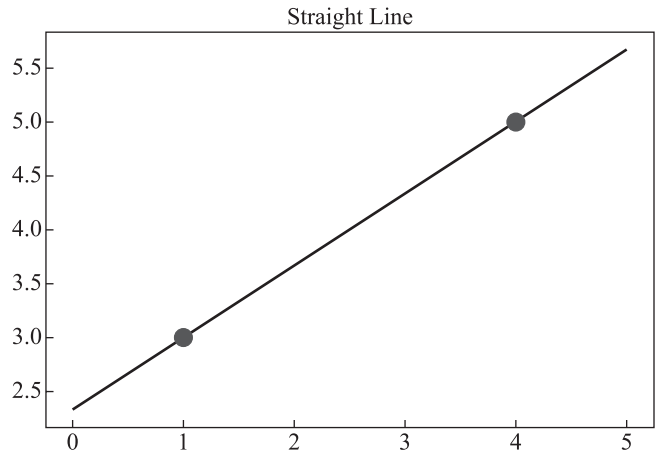

图 4-2　穿过点（1，3）和（4，5）的直线

【结果分析】图 4-2 表示的就是穿过上述两个数据点的直线，现在我们可以确定这条直线的方程。

在 Jupyter Notebook 中输入代码如下：

```
print('\n\n\n直线方程为：')
print('==========\n')
#打印直线方程
print('y = {:.3f}'.format(lr.coef_[0]),'x','+ {:.3f}'.format(lr.intercept_))
print('\n==========')
print('\n\n\n')
```

运行代码，将会得到如图 4-3 所示的结果。

```
直线方程为：
==========

y = 0.667 x + 2.333

==========
```

图 4-3　程序计算出的直线方程

【结果分析】通过程序的计算，我们很容易就可以得到这条直线的方程为

$$y = 0.667\,x + 2.333$$

这是数据中只有 2 个点的情况，那如果是 3 个点会是怎样的情况呢？我们来实验一下，

假设现在有第 3 个点，坐标是（3，3），我们把这个点添加进去，看会得到怎样的结果。输入代码如下：

```
#输入3个点的横坐标
X = [[1],[4],[3]]
#输入3个点的纵坐标
y = [3,5,3]
#用线性模型拟合这3个点
lr = LinearRegression().fit(X,y)
#画出2个点和直线的图形
z = np.linspace(0,5,20)
plt.scatter(X,y,s=80)
plt.plot(z, lr.predict(z.reshape(-1,1)),c='k')
#设定图题为Straight Line
plt.title('Straight Line')
#将图片显示出来
plt.show()
```

运行代码，会得到如图 4-4 所示的结果。

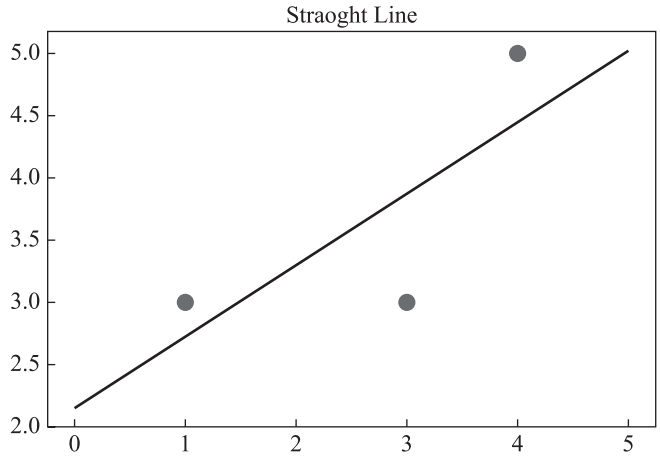

图 4-4　对 3 个点进行拟合的线性模型

【结果分析】从图 4-4 中我们可以看到，这次直线没有穿过任何一个点，而是位于一个和 3 个点的距离相加最小的位置。

下面我们可以在计算出这条直线的方程，输入代码如下：

```
print('\n\n\n新的直线方程为：')
print('==========\n')
#打印直线方程
print('y = {:.3f}'.format(lr.coef_[0]),'x','+ {:.3f}'.format(lr.intercept_))
print('\n==========')
print('\n\n\n')
```

运行代码，将会得到结果如图 4-5 所示。

```
新的直线方程为：
==========

y = 0.571 x + 2.143

==========
```

图 4-5　对 3 个点进行拟合的线性模型方程

【结果分析】从图 4-5 中我们可以看到，新的直线方程和只有 2 个数据点的直线方程已经发生了变化。线性模型让自己距离每个数据点的加和为最小值。这也就是线性回归模型的原理。

当然，在实际应用中，数据量要远远大于 2 个或是 3 个，下面我们就用数量更多的数据来进行实验。

现在我们以 scikit-klearn 生成的 make_regression 数据集为例，用 Python 语句绘制一条线性模型的预测线，更加清晰地反映出线性模型的原理。在 jupyter notebook 中输入代码如下：

```
from sklearn.datasets import make_regression
#生成用于回归分析的数据集
X, y = make_regression(n_samples=50, n_features=1, n_informative=1,
                       noise=50,random_state=1)
#使用线性模型对数据进行拟合
reg = LinearRegression()
reg.fit(X,y)
#z是我们生成的等差数列，用来画出线性模型的图形
z = np.linspace(-3,3,200).reshape(-1,1)
plt.scatter(X,y,c='b',s=60)
plt.plot(z, reg.predict(z),c='k')
plt.title('Linear Regression')
```

按下 shift+ 回车键后，会得到结果如图 4-6 所示的结果。

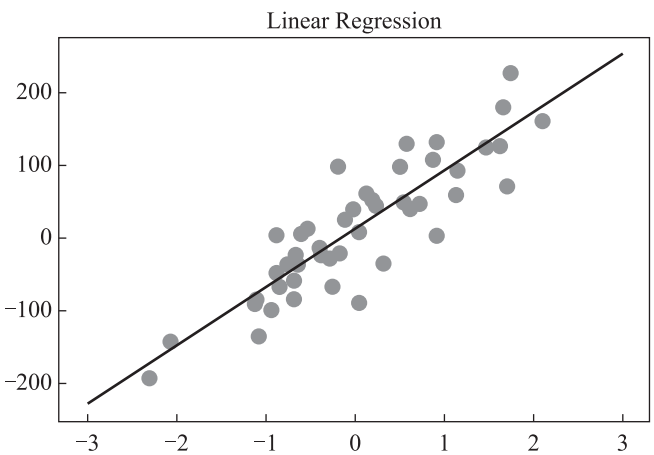

图 4-6　线性回归模型的预测线

【结果分析】从图 4-1 中我们可以看出，黑色直线是线性回归模型在 make_regression 数据集中生成的预测线。接下来我们来看一下这条直线所对应的斜率和截距。

输入代码如下：

```
print('\n\n\n代码运行结果：')
print('==========\n')
#打印直线的系数和截距
print('直线的系数是：{:.2f}'.format(reg.coef_[0]))
print('直线的截距是：{:.2f}'.format(reg.intercept_))
print('\n==========')
print('\n\n\n')
```

运行代码，会得到结果如图 4-7 所示。

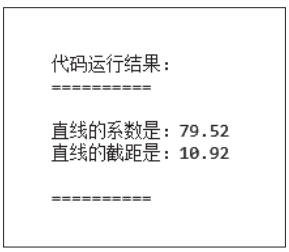

图 4-7　直线的系数和截距

【结果分析】从图 4-7 中我们可以看到，在我们手工生成的数据集中，线性模型的方程为

$$y = 79.52\,x + 10.92$$

而这条直线距离 50 个数据点的距离之和，是最小的。这便是一般线性模型的原理。

注意　细心的读者可能注意到 coef_ 和 intercept_ 这两个属性非常奇怪，它们都是以下划线 _ 结尾。这是 sciki-learn 的一个特点，它总是用下划线作为来自训练数据集的属性的结尾，以便将它们与由用户设置的参数区分开。

4.1.3　线性模型的特点

在上面的内容中，我们使用的都是特征数只有 1 个的数据集。用于回归分析的线性模型在特征数为 1 的数据集中，是使用一条直线来进行预测分析，而当数据的特征数量达到 2 个时则是一个平面，而对于更多特征数量的数据集来说，则是一个高维度的超平面。

如果和 K 最近邻模型生成的预测进行比较的话，你会发现线性模型的预测方法是非常有局限性的——很多数据都没有体现在这条直线上。从某种意义上说，这是一个问题。

因为使用线性模型的前提条件，是假设目标 y 是数据特征的线性组合。但需要特别注意的是，使用一维数据集进行验证会让我们有一点偏颇，而对于特征变量较多的数据集来说，线性模型就显得十分强大。尤其是，当训练数据集的特征变量大于数据点的数量的时候，线性模型可以对训练数据做出近乎完美的预测。

用于回归分析的线性模型也有很多种类。这些模型之间的区别在于如何从训练数据中确定模型参数 w 和 b，以及如何控制模型复杂度。下面的小节我们来看看几种回归分析中最流行的线性模型。

4.2 最基本的线性模型——线性回归

线性回归，也称为普通最小二乘法（OLS），是在回归分析中最简单也是最经典的线性模型。本节中我们将介绍线性回归的原理和在实践中的表现。

4.2.1 线性回归的基本原理

线性回归的原理是，找到当训练数据集中 y 的预测值和其真实值的平方差最小的时候，所对应的 w 值和 b 值。线性回归没有可供用户调节的参数，这是它的优势，但也代表我们无法控制模型的复杂性。接下来我们继续使用 make_regression 函数，生成一个样本数量为 100，特征数量为 2 的数据集，并且用 train_test_split 函数将数据集分割成训练数据集和测试数据集，再用线性回归模型计算出 w 值和 b 值。现在我们在 jupyter notebook 中输入代码如下：

```
#导入数据集拆分工具
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X, y = make_regression(n_samples=100,n_features=2,n_informative=2,random_
state=38)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=8)
lr = LinearRegression().fit(X_train, y_train)
```

在 4.1 节我们已经学过，方程的斜率 w，也被称为权重或者系数，被存储在 coef_ 属性中，而截距 b 被存储在 intercept_ 属性中，我们可以通过“print”函数将它们打印出来看一下：

```
print('\n\n\n代码运行结果：')
print('==========\n')
print("lr.coef_: {}".format(lr.coef_[:]))
print("lr.intercept_: {}".format(lr.intercept_))
print('\n==========')
```

```
print('\n\n\n')
```

按下 shift+ 回车键后，可以看到结果如图 4-8 所示。

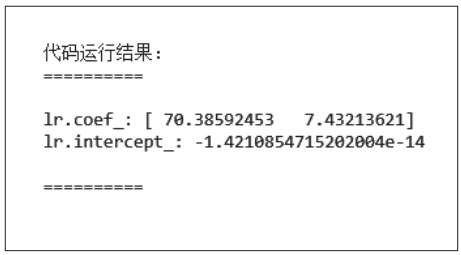

图 4-8　模型的系数和截距

【结果分析】intercept_ 属性一直是一个浮点数，而 coef_ 属性则是一个 NumPy 数组，其中每个特征对应数据中的一个数值，由于我们这次使用 make_regression 生成的数据集中数据点有 2 个特征，所以 lr.coef_ 是一个二维数组。也就是说，在本例中线性回归模型的方程可以表示为

$$y = 70.385\ 9 \times X_1 + 7.4321 \times X_2 - 1.42e^{-14}$$

4.2.2　线性回归的性能表现

下面我们来看看线性回归在 make_regression 生成的训练数据集和测试数据集中的性能如何，在 jupyter notebook 中输入如下代码：

```
print('\n\n\n代码运行结果：')
print('==========\n')
print("训练数据集得分：{:.2f}".format(lr.score(X_train, y_train)))
print("测试数据集得分：{:.2f}".format(lr.score(X_test, y_test)))
print('\n==========')
print('\n\n\n')
```

按下 shift+ 回车键，就会得到如图 4-9 所示的结果。

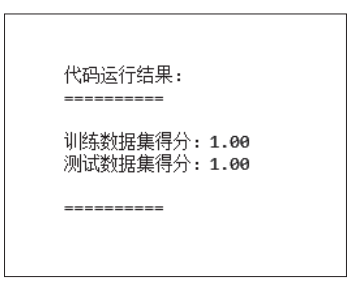

图 4-9　线性回归模型在训练集和测试集中的得分

【结果分析】这是一个令人振奋的分数，模型在训练集和测试中分别取得了满分，也就是 1.00 分的好成绩！不过不要高兴太早，这是因为我们这次没有向数据集添加 noise，所以分数自然会打到满分了。不过真实世界的数据集可就没有那么简单了。

真实世界的数据集，往往特征要多得多，而且 noise 也不少，这会给线性模型带来不少的困扰，下面我们就来生成一个来自真实世界的数据集——糖尿病情数据集，再来测试一下。在 jupyter notebook 中输入代码如下：

```
from sklearn.datasets import load_diabetes
#载入糖尿病情数据集
X, y = load_diabetes().data, load_diabetes().target
#将数据集拆分成训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=8)
使用线性回归模型进行拟合
lr = LinearRegression().fit(X_train, y_train)
```

然后我们来看下这个模型针对训练数据集和测试数据集的得分如何，在 jupyter notebook 中输入代码如下：

```
print('\n\n\n代码运行结果：')
print('==========\n')
print("训练数据集得分：{:.2f}".format(lr.score(X_train, y_train)))
print("测试数据集得分：{:.2f}".format(lr.score(X_test, y_test)))
print('\n==========')
print('\n\n\n')
```

按下 shift+ 回车键之后，得到结果如图 4-10 所示。

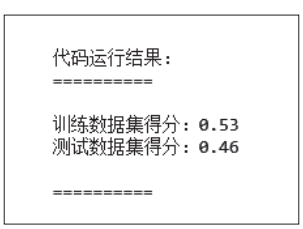

图 4-10　线性回归模型在糖尿病数据集中的得分

【结果分析】对比这两个分数，你会发现这次模型的分数降低了很多，模型在训练数据集中分数只有 0.53，而测试数据集的得分就只有 0.46 了。

由于真实世界的数据复杂程度要比我们手工合成的数据高得多，使得线性回归的表现大幅下降了。此外，由于线性回归自身的特点，非常容易出现过拟合的现象。在训练集的得分和测试集的得分之间存在的巨大差异是出现过拟合问题的一个明确信号，因此，我们应该找到一个模型，使我们能够控制模型的复杂度。标准线性回归最常用的替代模型之一是岭回归，我们将在下一小节中进行探讨。

4.3　使用 L2 正则化的线性模型——岭回归

岭回归也是回归分析中常用的线性模型，它实际上是一种改良的最小二乘法。本节中我们将介绍岭回归的原理及在实践中的性能表现。

4.3.1　岭回归的原理

从实用的角度来说，岭回归实际上是一种能够避免过拟合的线性模型。在岭回归中，模型会保留所有的特征变量，但是会减小特征变量的系数值，让特征变量对预测结果的影响变小，在岭回归中是通过改变其 alpha 参数来控制减小特征变量系数的程度。而这种通过保留全部特征变量，只是降低特征变量的系数值来避免过拟合的方法，我们称之为 L2 正则化。

岭回归在 scikit-learn 中是通过 linear_model.Ridge 函数来调用的，下面我们继续使用波士顿房价的扩展数据集为例，看看岭回归的表现如何。现在在 jupyter notebook 里输入代码如下：

```
#导入岭回归
from sklearn.linear_model import Ridge
#使用岭回归对数据进行拟合
rigde = Ridge().fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("岭回归的训练数据集得分：{:.2f}".format(ridge.score(X_train, y_train)))
print("岭回归的测试数据集得分：{:.2f}".format(ridge.score(X_test, y_test)))
print('\n==========')
print('\n\n\n')
```

按下 shift+ 回车键，将会得到结果如图 4-11 所示。

```
代码运行结果：
==========

岭回归的训练数据集得分：0.43
岭回归的测试数据集得分：0.43

==========
```

图 4-11　岭回归的模型评分

【结果分析】现在我们看到，使用岭回归后，训练数据集的得分比线性回归要稍微低一些，而测试数据集的得分却出人意料地和训练集的得分一致，这和我们的预期基本是一致的。在线性回归中，我们的模型出现了轻微的过拟合现象。但由于岭回归是一个

相对受限的模型，所以我们发生过拟合的可能性大大降低了。可以说，复杂度越低的模型，在训练数据集上的表现越差，但是其泛化的能力会更好。如果我们更在意模型在泛化方面的表现，那么我们就应该选择岭回归模型，而不是线性回归模型。

4.3.2 岭回归的参数调节

岭回归是在模型的简单性（使系数趋近于零）和它在训练集上的性能之间取得平衡的一种模型。用户可以使用 alpha 参数控制模型更加简单性还是在训练集上的性能更高。在上一个示例中，我们使用默认参数 alpha = 1。

注意 alpha 的取值并没有一定之规。alpha 的最佳设置取决于我们使用的特定数据集。增加 alpha 值会降低特征变量的系数，使其趋于零，从而降低在训练集的性能，但更有助于泛化。

下面我们再看一个例子，仍然使用糖尿病数据集，但是把正则项参数 alpha 设置为 10，在 jupyter notebook 中输入如下代码：

```
#修改alpha参数为10
rigde10 = Ridge(alpha=10).fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("训练数据集得分：{:.2f}".format(ridge10.score(X_train, y_train)))
print("测试数据集得分：{:.2f}".format(ridge10.score(X_test, y_test)))
print('\n==========')
print('\n\n\n')
```

按下 shift+ 回车键，得到结果如图 4-12 所示。

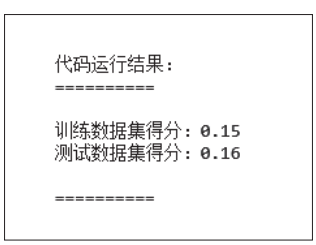

图 4-12　alpha 值等于 10 时模型的得分

【结果分析】提高了 alpha 值之后，我们看到模型的得分大幅降低了，然而有意思的是，模型在测试集的得分超过了在训练集的得分。这说明，如果我们的模型出现了过拟合的现象，那么我们可以提高 alpha 值来降低过拟合的程度。

同时，降低 alpha 值会让系数的限制变得不那么严格，如果我们用一个非常小的

alpha 值，那么系统的限制几乎可以忽略不计，得到的结果也会非常接近线性回归。比如下面这个例子：

在 jupyter notebook 中输入：

```
#修改alpha值为0.1
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("训练数据得分：{:.2f}".format(ridge01.score(X_train, y_train)))
print("测试数据得分：{:.2f}".format(ridge01.score(X_test, y_test)))
print('\n==========')
print('\n\n\n')
```

代码运行的结果如图 4-13 所示。

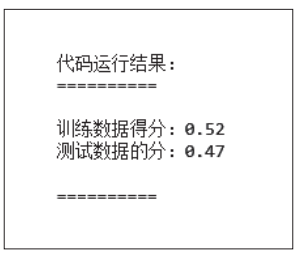

图 4-13　alpha 值等于 0.1 时的模型得分

【结果分析】现在我们看到，把参数 alpha 设置为 0.1 似乎让模型的在训练集的得分比线性回归模型略低，但在测试集的得分却有轻微的提升。我们还可以尝试不断降低 alpha 值来进一步改善模型的泛化表现。现在需要记住 alpha 值是如何影响模型的复杂性的。在后面的章节我们还会具体讨论设置参数的方法。

为了更清晰地看出 alpha 值对于模型的影响，我们用图像来观察不同 alpha 值对应的模型的 coef_ 属性。较高的 alpha 值代表模型的限制更加严格，所以我们认为在较高的 alpha 值下，coef_ 属性的数值会更小，反之 coef_ 属性的数值更大。下面用 jupyter notebook 把图形画出来。

在 jupyter notebook 中输入代码：

```
#绘制alpha=1时的模型系数
plt.plot(ridge.coef_, 's', label = 'Ridge alpha=1')
#绘制alpha=10时的模型系数
plt.plot(ridge10.coef_, '^', label = 'Ridge alpha=10')
#绘制alpha=0.1时的模型系数
plt.plot(ridge01.coef_, 'v', label = 'Ridge alpha=0.1')
#绘制线性回归的系数作为对比
plt.plot(lr.coef_, 'o', label = 'linear regression')
plt.xlabel("coefficient index")
```

```
plt.ylabel("coefficient magnitude")
plt.hlines(0,0, len(lr.coef_))
plt.legend()
```

按下 shift+ 回车键运行代码，jupyter notebook 会绘制一张图像如图 4-14 所示。

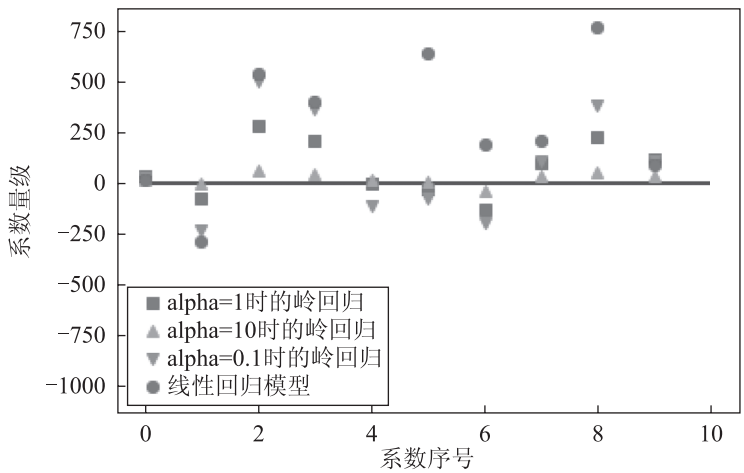

图 4-14　不同 alpha 值对应岭回归的参数大小及线性回归参数对比

【结果分析】在图 4-14 中，横轴代表的是 coef_ 属性：$x = 0$ 显示第一个特征变量的系数，$x = 1$ 显示的是第二个特征变量的系数，依此类推，直到 $x = 10$。纵轴显示特征变量的系数量级。从图中我们不难看出，当 alpha = 10 时，特征变量系数大多在 0 附近；而当 alpha = 1 时，岭模型的特征变量系数普遍增大了。而当 alpha = 0.1 时，特征变量的系数就更大了，甚至大部分与线性回归的点重合了，而线性回归模型由于没有经过任何正则化处理，其所对应的特征变量系数值就会非常大，其中有一些都快跑到图表之外了。

还有一个能够帮助我们更好理解正则化对模型影响的方法，那就是取一个固定的 alpha 值，然后改变训练数据集的数据量。比如我们在糖尿病数据集中采样，然后用这些采样的子集对线性回归模型和 alpha 值等于 1 的岭回归模型进行评估，并用 jupyter notebook 进行绘图，得到一个随数据集大小而不断改变的模型评分折线图，其中的折线我们也称之为学习曲线（learning curves）。下面我们来初步画一下两个模型在糖尿病数据集中的学习曲线，输入代码如下：

```
from sklearn.model_selection import learning_curve,KFold
#定义一个绘制学习曲线的函数
def plot_learning_curve(est, X, y):
#将数据进行20次拆分用来对模型进行评分
    training_set_size,train_scores, test_scores = learning_curve(
     est,X, y,train_sizes=np.linspace(.1, 1, 20), cv=KFold(20,shuffle=True,
```

```
                                                        random_state=1))
    estimator_name = est.__class__.__name__
    line = plt.plot(training_set_size, train_scores.mean(axis=1), '--',
                    label="training " + estimator_name)
    plt.plot(training_set_size, test_scores.mean(axis=1), '-',
             label="test " + estimator_name, c=line[0].get_color())
    plt.xlabel('Training set size')
    plt.ylabel('Score')
    plt.ylim(0, 1.1)

plot_learning_curve(Ridge(alpha=1), X, y)
plot_learning_curve(LinearRegression(), X, y)
plt.legend(loc=(0, 1.05), ncol=2, fontsize=11)
```

运行代码，会得到如图 4-15 所示的结果。

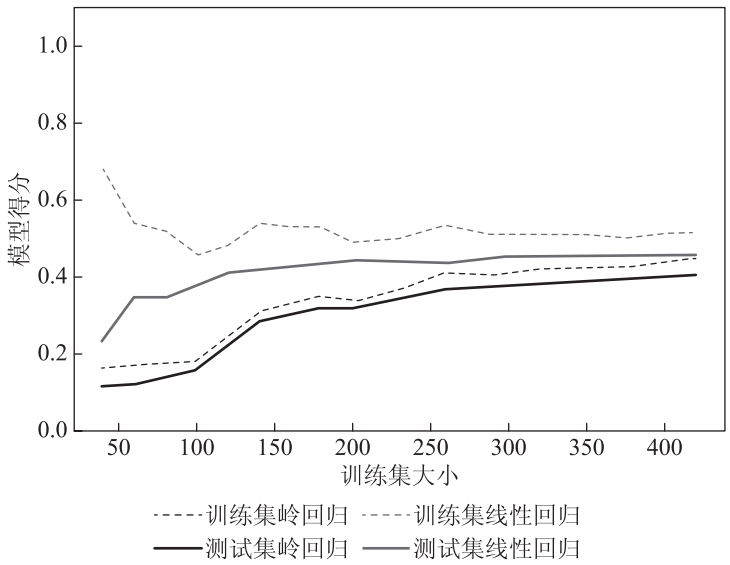

图 4-15　糖尿病数据集中岭回归与线性回归的学习曲线

【结果分析】毫无疑问，不论是在岭回归中还是在线性回归中，训练数据集的得分都比测试数据集的得分要高。而由于岭回归是经过正则化的模型，因此它在整个图像中训练数据集的得分要比线性回归的得分低。然而，岭回归在测试数据集的得分与训练数据集的得分差异就要小一些，尤其是在数据子集比较小的情况下。在数据量小于 50 条的情况下，线性回归几乎不能让机器学到任何东西。随着数据集的规模越来越大，两个模型的表现也越来越好，最后线性回归的得分赶上了岭回归的得分。不难看出，如果有足够多的数据，那么正则化就显得不是那么重要了，岭回归和线性回归的表现也相差无几。

注意 **读者可能会发现，随着数据量的增加，线性回归在训练数据集的得分是下降的，这说明随着数据增加，线性回归模型就越不容易产生过拟合的现象，或者说越难记住这些数据。**

4.4 使用 L1 正则化的线性模型——套索回归

除了岭回归之外，还有一个对线性回归进行正则化的模型，即套索回归（lasso）。本节我们重点探讨套索回归的原理及其在实际应用中的表现。

4.4.1 套索回归的原理

和岭回归一样，套索回归也会将系数限制在非常接近 0 的范围内，但它进行限制的方式稍微有一点不同，我们称之为 L1 正则化。与 L2 正则化不同的是，L1 正则化会导致在使用套索回归的时候，有一部分特征的系数会正好等于 0。也就是说，有一些特征会彻底被模型忽略掉，这也可以看成是模型对于特征进行自动选择的一种方式。把一部分系数变成 0 有助于让模型更容易理解，而且可以突出体现模型中最重要的那些特征。

让我们再用糖尿病数据集来验证一下套索回归，在 jupyter notebook 中输入代码如下：

```
#导入套索回归
from sklearn.linear_model import Lasso
#使用套索回归拟合数据
lasso = Lasso().fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("套索回归在训练数据集的得分：{:.2f}".format(lasso.score(X_train, y_train)))
print("套索回归在测试数据集的得分：{:.2f}".format(lasso.score(X_test, y_test)))
print("套索回归使用的特征数：{}".format(np.sum(lasso.coef_ != 0)))
```

代码运行的结果如图 4-16 所示。

```
代码运行结果：
==========

套索回归在训练数据集的得分：0.36
套索回归在测试数据集的得分：0.37
套索回归使用的特征数：3

==========
```

图 4-16　套索回归模型得分

【结果分析】这里我们看到，套索回归在训练数据集和测试数据集的得分都相当糟糕。

这意味着我们的模型发生了欠拟合的问题，而且你会发现，在 10 个特征里面，套索回归只用了 3 个。与岭回归类似，套索回归也有一个正则化参数 alpha，用来控制特征变量系数被约束到 0 的强度。

4.4.2　套索回归的参数调节

在上面的例子里，我们用了 alpha 的默认值 1.0。为了降低欠拟合的程度，我们可以试着降低 alpha 的值。与此同时，我们还需要增加最大迭代次数（max_iter）的默认设置。让我们来看下面的代码：

```
#增加最大迭代次数的默认设置
#否则模型会提示我们增加最大迭代次数
lasso01 = Lasso(alpha=0.1, max_iter=100000).fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("alpha=0.1时套索回归在训练数据集的得分：{:.2f}".format(lasso001.score(X_train,
y_train)))
print("alpha=0.1时套索回归在测试数据集的得分：{:.2f}".format(lasso001.score(X_test,
y_test)))
print("alpha=0.1时套索回归使用的特征数：{}".format(np.sum(lasso001.coef_ != 0)))
print('\n==========')
print('\n\n\n')
```

运行代码，得到结果如图 4-17 所示。

```
代码运行结果：
==========

alpha=0.1时套索回归在训练数据集的得分：0.52
alpha=0.1时套索回归在测试数据集的得分：0.48
alpha=0.1时套索回归使用的特征数：7

==========
```

图 4-17　alpha 等于 0.1 时的套索回归得分

【结果分析】从结果来看，降低 alpha 值可以拟合出更复杂的模型，从而在训练数据集和测试数据集都能获得良好的表现。相对岭回归，套索回归的表现还要稍好一点，而且它只用了 10 个特征中的 7 个，这一点也会使模型更容易被人理解。

但是，如果我们把 alpha 值设置得太低，就等于把正则化的效果去除了，那么模型就可能会像线性回归一样，出现过拟合的问题。比如我们把 alpha 值设为 0.000 1，试着运行下面的代码：

```
#修改alpha值为0.0001
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print('\n\n\n代码运行结果：')
print('==========\n')
print("alpha=0.0001时套索回归在训练数据集的得分：{:.2f}".format(
    lasso00001.score(X_train, y_train)))
print("alpha=0.0001时套索回归在测试数据集的得分：{:.2f}".format(
    lasso00001.score(X_test, y_test)))
print("alpha=0.0001时套索回归使用的特征数：{}".format(np.sum(
    lasso00001.coef_ != 0)))
print('\n==========')
print('\n\n\n')
```

代码运行的结果如图 4-18 所示。

```
代码运行结果：
==========

alpha=0.0001时套索回归在训练数据集的得分：0.53
alpha=0.0001时套索回归在测试数据集的得分：0.46
alpha=0.0001时套索回归使用的特征数：10

==========
```

图 4-18　alpha 等于 0.000 1 时套索回归的模型评分

【结果分析】从结果中我们看到，套索回归使用了全部的特征，而且在测试数据集中的得分稍微低于在 alpha 等于 0.1 时的得分，这说明降低 alpha 的数值会让模型倾向于出现过拟合的现象。

4.4.3　套索回归与岭回归的对比

接下来，我们继续用图像的方式来对不同 alpha 值的套索回归和岭回归进行系数对比，运行下面的代码：

```
#绘制alpha值等于1时的模型系数
plt.plot(lasso.coef_, 's', label="Lasso alpha=1")
#绘制alpha值等于0.11时的模型系数
plt.plot(lasso01.coef_, '^', label="Lasso alpha=0.1")
#绘制alpha值等于0.0001时的模型系数
plt.plot(lasso00001.coef_, 'v', label="Lasso alpha=0.0001")
#绘制alpha值等于0.1时的岭回归模型系数作为对比
plt.plot(ridge01.coef_, 'o', label="Ridge alpha=0.1")
plt.legend(ncol=2,loc=(0,1.05))
plt.ylim(-25,25)
plt.xlabel("Coefficient index")
```

```
plt.ylabel("Coefficient magnitude")
```

运行代码，我们会得到结果如图 4-19 所示。

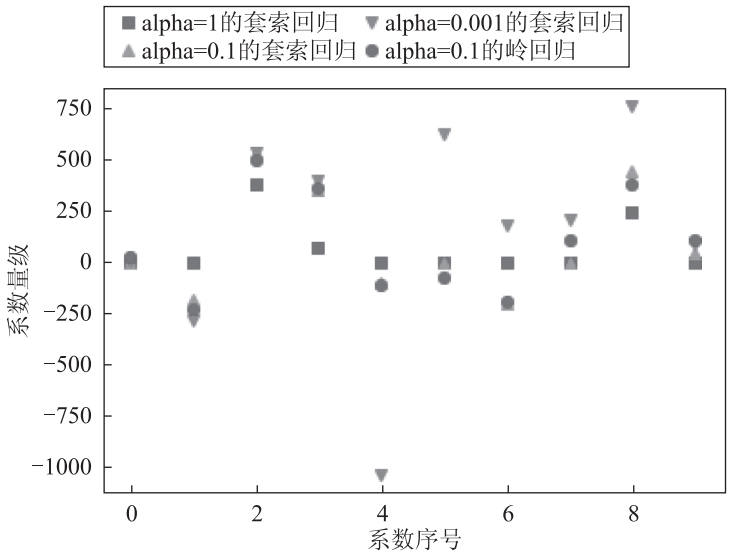

图 4-19　不同 alpha 值对应的套索回归系数值与岭回归系数值对比

【结果分析】从图中我们不难看出，当 alpha 值等于 1 的时候，不仅大部分系数为 0，而且仅存的几个非零系数数值也非常小。把 alpha 值降低到 0.01 时，如图中正三角形所示，大部分系数也是 0，但是等于 0 的系数已经比 alpha 等于 1 的时候少了很多。而当我们把 alpha 值降低到 0.000 1 的时候，整个模型变得几乎没有被正则化，大部分系数都是非零的，并且数值变得相当大。作为对比，我们能看到圆点代表的是岭回归中的系数值。alpha 值等于 0.1 的岭回归模型在预测能力方面基本与 alpha 值等于 0.1 的套索回归模型一致，但你会发现，使用岭回归模型的时候，几乎所有的系数都是不等于 0 的。

在实践当中，岭回归往往是这两个模型中的优选。但是如果你的数据特征过多，而且其中只有一小部分是真正重要的，那么套索回归就是更好的选择。同样如果你需要对模型进行解释的话，那么套索回归会让你的模型更容易被人理解，因为它只是使用了输入的特征值中的一部分。

注意 **scikit-learn 还提供了一种模型，称为弹性网模型（Elastic Net）。弹性网模型综合了套索回归和岭回归的惩罚因子。在实践中这两个模型的组合是效果最好的，然而代价是用户需要调整两个参数，一个是 L1 正则化参数，另一个是 L2 正则化参数。**

4.5 小结

在本章中，我们介绍了几种常用的线性模型，包括线性回归、岭回归和套索回归。实际上，线性模型并不止这三种，还有比较知名的逻辑斯谛回归（Logistic Regression）、线性支持向量机（Linear SVM）等，它们不仅仅可以用来进行回归分析，在分类任务中也相当常见。对于线性模型来说，最主要的参数就是正则化参数（Regularization Parameter）。在线性回归、岭回归和套索回归中，是通过 alpha 参数来进行调节的，而对于逻辑斯谛回归和线性支持向量机来说，则是通过调节参数 C 来实现的。当然在实际应用中，我们常常要先决定是使用 L1 正则化的模型还是 L2 正则化的模型。大体的原则是这样，如果你的数据集有很多特征，而这些特征中并不是每一个都对结果有重要的影响，那么就应该使用 L1 正则化的模型，如套索回归；但如果数据集中的特征本来就不多，而且每一个都有重要作用的话，那么就应该使用 L2 正则化的模型，如岭回归。

虽然线性模型是一个存在历史相当悠久的算法模型，但目前它们的应用依然非常普遍，这主要是因为线性模型的训练速度非常快，尤其是对于那些超大型数据集来讲。而且其过程非常容易被人理解——基本上学过初中数学的人都能明白线性模型的原理。但是它也有一定的局限性，当数据集的特征比较少的时候，线性模型的表现就会相对偏弱一些。

在第 5 章当中，我们将带大家一起学习另外一种非常流行的算法——朴素贝叶斯算法。这是一种基于概率理论的算法，它的效率比线性模型还要更高一些。请大家做好准备，和我们一起向下一站出发。

第 5 章　朴素贝叶斯——打雷啦，收衣服啊

朴素贝叶斯（Naïve Bayes）算法是一种基于贝叶斯理论的有监督学习算法。之所以说“朴素”，是因为这个算法是基于样本特征之间互相独立的“朴素”假设。正因为如此，由于不用考虑样本特征之间的关系，朴素贝叶斯分类器的效率是非常高的。这一章将向大家介绍朴素贝叶斯算法的基本概念和用法。

本章主要涉及的知识点有：

- 贝叶斯定理简介
- 朴素贝叶斯的简单应用
- 贝努利朴素贝叶斯、高斯朴素贝叶斯和多项式朴素贝叶斯
- 朴素贝叶斯实例——判断肿瘤是良性还是恶性

5.1 朴素贝叶斯基本概念

贝叶斯（Thomas Bayes）是一位英国数学家，1701 年生于伦敦，曾是一位神父。后于 1742 年成为英国皇家学会会员。贝叶斯在数学方面主要研究概率论，他创立了贝叶斯统计理论，该理论对现代概率论和数理统计又有很重要的作用，在数学和工程领域都得到了广泛的应用，本节将介绍贝叶斯定理和朴素贝叶斯分类器的基本概念。

5.1.1 贝叶斯定理

那是一个 7 月的傍晚，临近下班，小 C 收拾东西准备去接女朋友。问题来了，小 C 要不要带伞呢？

已知：

天气预报说今日降水概率为 50%——P（A）

晚高峰堵车的概率是 80%——P（B）

如果下雨，晚高峰堵车的概率是 95%——P（$B|A$）

小 C 向窗外望去，看到堵车了，则根据贝叶斯定理：

$$P(A|B)=\frac{P(B|A)\cdot P(A)}{P(B)}$$

求得下雨的概率为 0.5×0.95÷0.8=0.593 75。

小 C 果断地拿起雨伞冲了出去……

5.1.2 朴素贝叶斯的简单应用

过去的 7 天当中，有 3 天下雨，4 天没有下雨。用 0 代表没有下雨，而 1 代表下雨，我们可以用一个数组来表示：

$$y = [0, 1, 1, 0, 1, 0, 0]$$

而在这 7 天当中，还有另外一些信息，包括刮北风、闷热、多云，以及天气预报给出的信息，如表 5-1 所列。

表 5-1 过去 7 天中和气象有关的信息

	刮 北 风	闷 热	多 云	天气预报有雨
第 1 天	否	是	否	是
第 2 天	是	是	是	否
第 3 天	否	是	是	否

续表

	刮 北 风	闷　　热	多　　云	天气预报有雨
第 4 天	否	否	否	是
第 5 天	否	是	是	否
第 6 天	否	是	否	是
第 7 天	是	否	否	是

同样地，我们用 0 代表否，1 代表是，可以得到另外一个数组：

X = [0, 1, 0, 1], [1, 1, 1, 0], [0, 1, 1, 0], [0, 0, 0, 1], [0, 1, 1, 0], [0, 1, 0, 1], [1, 0, 0, 1]

现在我们用 Jupyter Notebook 来看一下数据的关系，在 Jupyter Notebook 中输入代码如下：

```
#导入numpy
import numpy as np
#将X，y赋值为np数组
X = np.array([[0, 1, 0, 1],
              [1, 1, 1, 0],
              [0, 1, 1, 0],
              [0, 0, 0, 1],
              [0, 1, 1, 0],
              [0, 1, 0, 1],
              [1, 0, 0, 1]])
y = np.array( [0, 1, 1, 0, 1, 0, 0])
#对不同分类计算每个特征为1的数量
counts = {}
for label in np.unique(y):
counts[label] = X[y == label].sum(axis=0)
#打印计数结果
print("feature counts:\n{}".format(counts))
```

运行代码，会得到如图 5-1 所示的结果。

```
代码运行结果：
==============================

feature counts:
{0: array([1, 2, 0, 4]), 1: array([1, 3, 3, 0])}

==============================
```

图 5-1　对数组中 0 和 1 的个数进行计数

【结果分析】下面我们来解释一下这个结果的意思，当 y 为 0 时，也就在没有下雨的 4 天当中，有 1 天刮了北风，有 2 天比较闷热，而没有出现多云的情况，但这 4 天天气预报全部播报有雨。同时，在 y 为 1 时，也就是在下雨的 3 天当中，有 1 天刮了北风，

3 天全都比较闷热，且 3 天全部出现了多云的现象，有意思的是，这 3 天的天气预报都没有播报有雨。

那么对于朴素贝叶斯来说，它会根据上述的计算来进行推理。它会认为，如果某一天天气预报没有播报有雨，但出现了多云的情况，它会倾向于把这一天放到“下雨”这一个分类中。

下面我们来验证一下，在 Jupyter Notebook 里输入代码如下：

```
#导入贝努利贝叶斯
from sklearn.naive_bayes import BernoulliNB
#使用贝努利贝叶斯拟合数据
clf = BernoulliNB()
clf.fit(X, y)
#要进行预测的这一天，没有刮北风，也不闷热
#但是多云，天气预报没有说有雨
Next_Day = [[0, 0, 1, 0]]
pre = clf.predict(Next_Day)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
if pre == [1]:
    print("要下雨啦，快收衣服啊！")
else:
print("放心，又是一个艳阳天")
print('\n==============================')
print('\n\n\n')
```

运行代码的结果如图 5-2 所示。

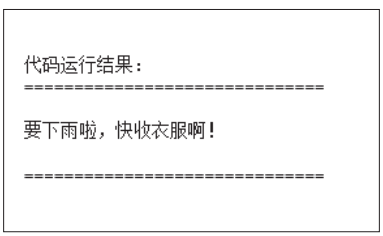

图 5-2　模型预测会下雨

【结果分析】可以看出，朴素贝叶斯分类器把这一天放到了会下雨的分类当中。

那么如果有另外一天，刮了北风，而且很闷热，但云量不多，同时天气预报说有雨，会怎样呢？我们输入代码如下：

```
#假设另外一天的数据如下
Another_day = [[1, 1, 0, 1]]
#使用训练好的模型进行预测
pre2 = clf.predict(Another_day)
print('\n\n\n')
```

```
print('代码运行结果：')
print('==============================\n')
if pre2 == [1]:
    print("要下雨啦，快收衣服啊！")
else:
print("放心，又是一个艳阳天")
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到如图 5-3 所示的结果。

```
代码运行结果：
==============================

放心，又是一个艳阳天

==============================
```

图 5-3　模型预测这一天不会下雨

【结果分析】可以看到，这次分类器把这一天归为不会下雨的分类中了。

现在大家可能很想知道朴素贝叶斯给出的预测准确率怎么样，我们可以用 predict_proba 方法来测试一下，输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#模型预测分类的概率
print(clf.predict_proba(Next_Day))
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到如图 5-4 所示的结果。

```
代码运行结果：
==============================

[[ 0.13848881  0.86151119]]

==============================
```

图 5-4　模型预测数据点所述分类的概率

【结果分析】这个意思是说，我们所预测的第一天，不下雨的概率大约是 13.8%，而下雨的概率是 86.2%，看起来还是很不错的。

再看一下第二天的预测情况。输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印另外一天模型预测的分类概率
print(clf.predict_proba(Another_day))
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
```

运行代码，将得到如图 5-5 所示的结果。

```
代码运行结果：
==============================

[[ 0.92340878  0.07659122]]

==============================
```

图 5-5　模型预测的另外一天的分类概率

【结果分析】也就是说，第二天不下雨的概率是 92.3%，下雨的概率只有 7.7%，这样看起来，朴素贝叶斯做出的预测还不错。

注意　不要太乐观！如果大家在 scikit-learn 官网上查看文档，会发现一段很搞笑的描述——虽然朴素贝叶斯是相当好的分类器，但对于预测具体的数值并不是很擅长，因此 predict_proba 给出的预测概率，大家也不要太当真。

5.2　朴素贝叶斯算法的不同方法

朴素贝叶斯算法包含多种方法，在 scikit-learn 中，朴素贝叶斯有三种方法，分别是贝努利朴素贝叶斯（Bernoulli Naïve Bayes）、高斯贝叶斯（Gaussian Naïve Bayes）和多项式朴素贝叶斯（Multinomial Naïve Bayes），本节将对这几种方法进行介绍。

5.2.1　贝努利朴素贝叶斯

在上面的例子当中，我们使用了朴素贝叶斯算法中的一种方法，称为贝努利朴素贝叶斯（Bernoulli Naïve Bayes），这种方法比较适合于符合贝努利分布的数据集，贝努利分布也被称为“二项分布”或者是“0-1 分布”，比如我们进行抛硬币的游戏，硬币落下

来只有两种可能的结果：正面或者反面，这种情况下，我们就称抛硬币的结果是贝努利分布的。

在刚才我们举的例子当中，数据集中的每个特征都只有 0 和 1 两个数值，在这种情况下，贝努利贝叶斯的表现还不错。但如果我们用更复杂的数据集，结果可能就不一样了，下面我们动手来试一试，输入代码如下：

```
#导入数据集生成工具
from sklearn.datasets import make_blobs
#导入数据集拆分工具
from sklearn.model_selection import train_test_split
#生成样本数量为500，分类数为5的数据集
X, y = make_blobs(n_samples=500, centers=5,random_state=8)
#将数据集拆分成训练集和训练集
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=8)
#使用贝努利贝叶斯拟合数据
nb = BernoulliNB()
nb.fit(X_train,y_train)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型得分
print('模型得分：{:.3f}'.format(nb.score(X_test, y_test)))
print('\n==============================')
print('\n\n\n')
```

这里我们还是使用了非常熟悉的 make_blobs 来生成手工数据集，为了加大难度，我们令样本数量为 500，而分类的数量为 5 个，也就是 centers 参数等于 5，运行代码，会得到如图 5-6 所示的结果。

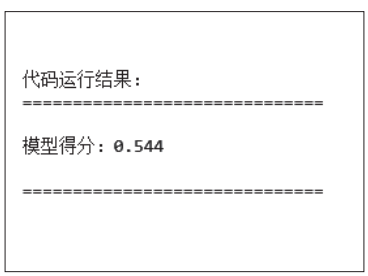

图 5-6　模型在测试集中的得分

【结果分析】可以看到，在我们手工生成的相对复杂的数据集中，贝努利朴素贝叶斯的得分相当糟糕，只有大约一半的数据被放进了正确的分类，这是为什么呢？

下面我们通过图像来了解一下贝努利朴素贝叶斯的工作过程，输入代码如下：

```
#导入画图工具
import matplotlib.pyplot as plt
```

```
#限定横轴与纵轴的最大值
x_min, x_max = X[:,0].min()-0.5, X[:,0].max()+0.5
y_min, y_max = X[:,1].min()-0.5, X[:,1].max()+0.5
#用不同的背景色表示不同的分类
xx,yy = np.meshgrid(np.arange(x_min, x_max,.02),
                    np.arange(y_min, y_max, .02))
z = nb.predict(np.c_[(xx.ravel(),yy.ravel())]).reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap=plt.cm.Pastel1)
#将训练集和测试集用散点图表示
plt.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=plt.cm.cool,edgecolor='k')
plt.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=plt.cm.cool,marker='*',
            edgecolor='k')
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
#定义图题
plt.title('Classifier: BernoulliNB')
#现实图片
plt.show()
```

运行代码，将会得到如图 5-7 所示的结果。

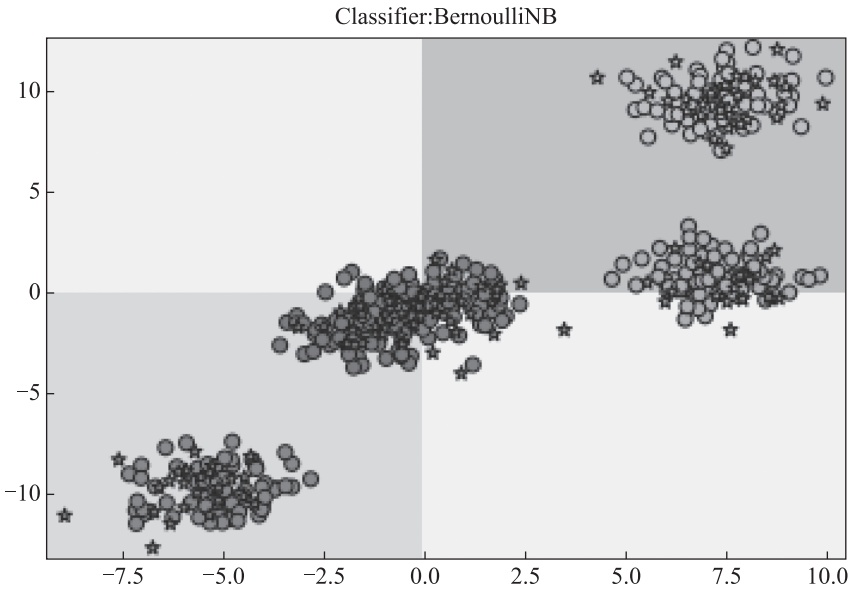

图 5-7　贝努利朴素贝叶斯对 make_blobs 数据集的分类

【结果分析】在图 5-7 中，我们可以看到贝努利朴素贝叶斯的模型十分简单，它分别在横轴等于 0 和纵轴等于 0 的位置画了两条直线，再用这两条直线形成的 4 个象限对数据进行分类。这是因为我们使用了贝努利朴素贝叶斯的默认参数 binarize=0.0，所以模型对于数据的判断是，如果特征 1 大于或等于 0，且特征 2 大于或等于 0，则将数据归为

一类；如果特征 1 小于 0，且特征 2 也小于 0，则归为另一类而其余的数据全部归为第三类，难怪模型的得分这么差了。

所以在这种情况下，我们就不能再使用贝努利朴素贝叶斯，而要用其他的方法，例如下面要讲到的高斯朴素贝叶斯方法。

5.2.2　高斯朴素贝叶斯

高斯朴素贝叶斯，顾名思义，是假设样本的特征符合高斯分布，或者说符合正态分布时所用的算法。接下来我们尝试用高斯朴素贝叶斯对刚刚生成的数据集进行拟合，看看结果如何，输入代码如下：

```
#导入高斯贝叶斯
from sklearn.naive_bayes import GaussianNB
#使用高斯贝叶斯拟合数据
gnb = GaussianNB()
gnb.fit(X_train, y_train)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型得分
print('模型得分：{:.3f}'.format(gnb.score(X_test, y_test)))
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到如图 5-8 所示的结果。

```
代码运行结果：
==============================

模型得分：0.968

==============================
```

图 5-8　高斯贝叶斯的模型得分

【结果分析】看起来，使用高斯朴素贝叶斯方法建立的模型得分要好了很多，准确率达到了 96.8%，这说明我们生成的手工数据集的特征基本上符合正态分布的情况。

下面我们再次用图像来进行演示，以便了解高斯朴素贝叶斯的工作工程，输入代码如下：

```
#用不同色块来表示不同的分类
z = gnb.predict(np.c_[(xx.ravel(),yy.ravel())]).reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap=plt.cm.Pastel1)
#用散点图画出训练集和测试集数据
```

```
plt.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=plt.cm.cool,edgecolor='k')
plt.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=plt.cm.cool,marker='*',
            edgecolor='k')
#设定横轴纵轴的范围
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
#设定图题
plt.title('Classifier: GaussianNB')
#画出图形
plt.show()
```

运行代码，会得到如图 5-9 所示的结果。

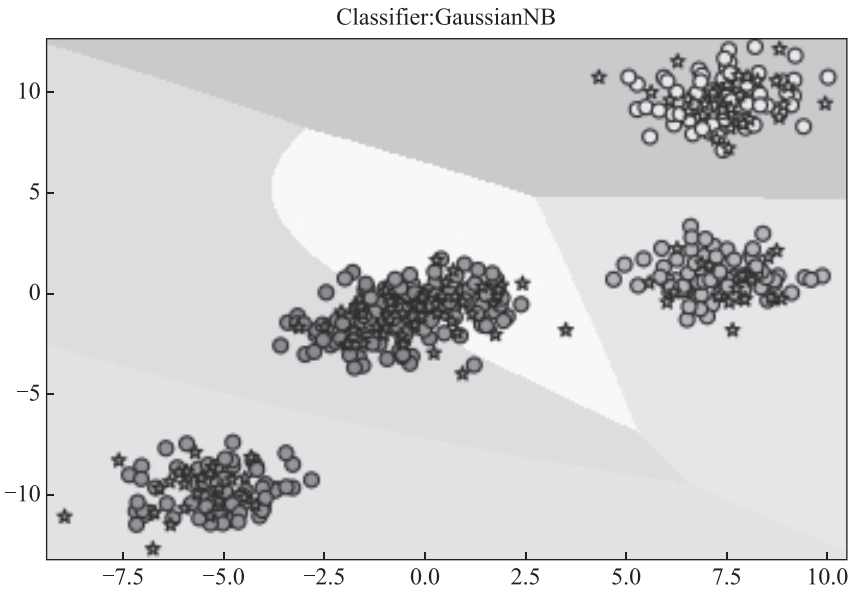

图 5-9　高斯朴素贝叶斯对数据集的分类

【结果分析】从图 5-9 中我们可以看到，高斯朴素贝叶斯的分类边界比贝努利朴素贝叶斯的分类边界要复杂得多，也基本上把数据点都放进了正确的分类当中了。

事实上，高斯朴素贝叶斯也确实是能够胜任大部分的分类任务，这是因为在自然科学和社会科学领域，有大量的现象都是呈现出正态分布的状态。接下来，我们要介绍第三种方法——多项式朴素贝叶斯。

5.2.3　多项式朴素贝叶斯

多项式朴素贝叶斯，从名字也可以推断出它主要是用于拟合多项式分布的数据集。可能多项式分布相对于二项式分布和高斯分布来说，我们会接触得少一些。但如果我们

可以理解二项式分布，那么理解多项式分布也会非常简单。二项式分布可以通过抛硬币的例子来进行理解，那么多项式分布都可以用掷骰子来理解。

我们知道硬币只有两个面，正面和反面，而骰子有 6 个面，因此每掷一次骰子，结果都可能是从 1 ～ 6 这 6 个数字，如果我们掷 n 次骰子，而每个面朝上的次数的分布情况，就是一个多项式分布。

现在我们继续使用生成的手工数据集来对多项式朴素贝叶斯进行实验，输入代码如下：

```
#导入多项式朴素贝叶斯
from sklearn.naive_bayes import MultinomialNB
#用多项式朴素贝叶斯拟合数据
mnb = MultinomialNB()
mnb.fit(X_train, y_train)
mnb.score(X_test, y_test)
```

注意　上面这段代码和我们使用贝努利朴素贝叶斯或是高斯朴素贝叶斯看起来没有什么区别，但是这样使用多项式朴素贝叶斯是错误的。

运行代码，程序会报错，并且给出提示信息如图 5-10 所示。

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-acfb4658330f> in <module>()
      3 #用多项式朴素贝叶斯拟合数据
      4 mnb = MultinomialNB()
----> 5 mnb.fit(X_train, y_train)
      6 mnb.score(X_test, y_test)

c:\program files\python36\lib\site-packages\sklearn\naive_bayes.py in fit(self,
X, y, sample_weight)
    602         self.feature_count_ = np.zeros((n_effective_classes, n_feature
s),
    603                                        dtype=np.float64)
--> 604         self._count(X, Y)
    605         alpha = self._check_alpha()
    606         self._update_feature_log_prob(alpha)

c:\program files\python36\lib\site-packages\sklearn\naive_bayes.py in _count(se
lf, X, Y)
    706         """Count and smooth feature occurrences."""
    707         if np.any((X.data if issparse(X) else X) < 0):
--> 708             raise ValueError("Input X must be non-negative")
    709         self.feature_count_ += safe_sparse_dot(Y.T, X)
    710         self.class_count_ += Y.sum(axis=0)

ValueError: Input X must be non-negative
```

图 5-10　程序报错，提示 X 值必须是非负的

【结果分析】提示信息告诉我们，输入的 X 值必须是非负的，这样的话，我们需要对数据进行一下预处理才行。

所以我们需要把代码改成如下的样子：

```
#导入多项式朴素贝叶斯
from sklearn.naive_bayes import MultinomialNB
#导入数据预处理工具MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
#使用MinMaxScaler对数据进行预处理，使数据全部为非负值
```

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
#使用多项式朴素贝叶斯拟合经过预处理之后的数据
mnb = MultinomialNB()
mnb.fit(X_train_scaled, y_train)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型得分
print('模型得分：{:.3f}'.format(mnb.score(X_test_scaled, y_test)))
print('\n==============================')
print('\n\n\n')
```

重新运行代码，程序不再报错，并且给出模型的分数如图 5-11 所示。

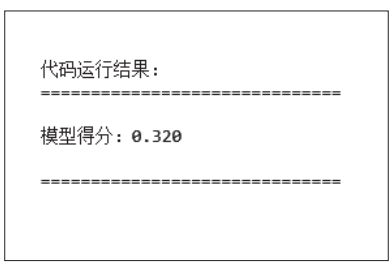

图 5-11　多项式朴素贝叶斯的模型得分

【结果分析】从结果中可以看出，虽然经过了预处理将所有特征值转化为非负的，但是多项式朴素贝叶斯还是不能获得较高的分数，32% 的准确率甚至比贝努利朴素贝叶斯的得分还要更糟糕一点。

如果我们用图形来表示的话，也可以直观地看出多项式朴素贝叶斯并不适合用来拟合这个数据集，输入代码如下：

```
#用不同颜色区分不同的分类
z = mnb.predict(np.c_[(xx.ravel(),yy.ravel())]).reshape(xx.shape)
plt.pcolormesh(xx,yy,z,cmap=plt.cm.Pastel1)
#用散点图表示训练集和测试集
plt.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=plt.cm.cool,edgecolor='k')
plt.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=plt.cm.cool,marker='*',
            edgecolor='k')
#设定横纵轴范围
plt.xlim(xx.min(),xx.max())
plt.ylim(yy.min(),yy.max())
#设定图题
plt.title('Classifier: MultinomialNB')
#显示图片
plt.show()
```

运行代码，将会得到结果如图 5-12 所示。

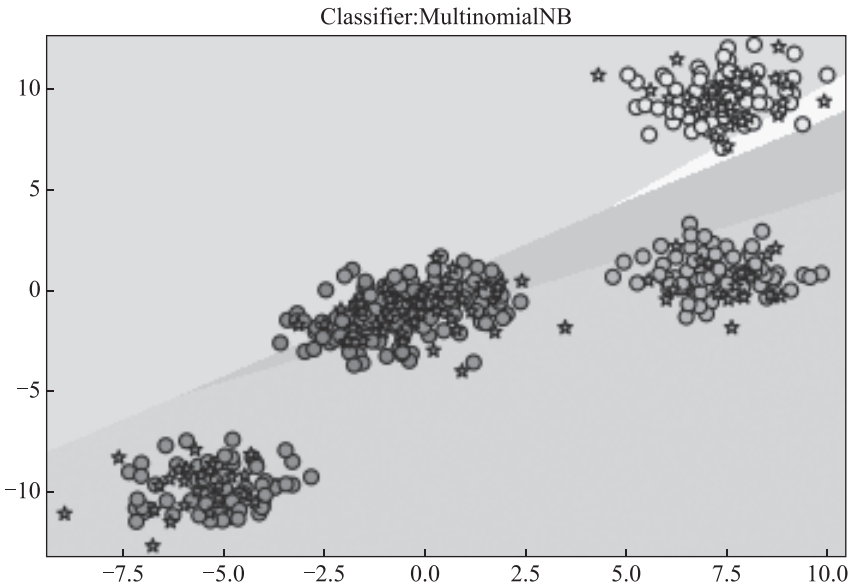

图 5-12　多项式朴素贝叶斯进行的分类

【结果分析】从图 5-12 中可以看出多项式朴素贝叶斯所进行的分类确实比贝努利朴素贝叶斯的还要差一些，大部分数据点都被放到了错误的分类中。

这是因为，多项式朴素贝叶斯只适合用来对非负离散数值特征进行分类，典型的例子就是对转化为向量后的文本数据进行分类。文本数据的处理我们将会在第 13 章中向大家介绍，这里我们暂时略过。

注意　在本例中，我们使用了 MinMaxScaler 对数据进行预处理，MinMaxScaler 的作用是将数据集中的特征值全部转化为 0 ～ 1。更多关于数据预处理的内容，我们将在后面的章节进行讲解。

5.3　朴素贝叶斯实战——判断肿瘤是良性还是恶性

接下来，我们将使用朴素贝叶斯算法来进行一个小的项目实战——判断一个患者的肿瘤是良性还是恶性。这里我们会用到一个来自真实世界的数据集——威斯康星乳腺肿瘤数据集。

5.3.1 对数据集进行分析

威斯康星乳腺肿瘤数据集是一个非常经典的用于医疗病情分析的数据集，它包括 569 个病例的数据样本，每个样本具有 30 个特征值，而样本共分为两类：分别是恶性（Malignant）和良性（Benign）。下面我们就载入这个数据集并了解一下它的样子，输入代码如下：

```
#导入威斯康星乳腺肿瘤数据集
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印数据集键值
print(cancer.keys())
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到该数据集的键值如图 5-13 所示。

```
代码运行结果：
==============================

dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])

==============================
```

图 5-13　乳腺肿瘤数据集中的键值

【结果分析】从这个结果中可以看到，数据集包含的信息有特征数据 data、分类值 target、分类名称 target_names、数据描述 DESCR，以及特征名称 feature_names。

下面我们来看一下分类的名称和特征的名称，输入代码如下：

```
#打印数据集中标注好的肿瘤分类
print('肿瘤的分类：',cancer['target_names'])
#打印数据集中的肿瘤特征名称
print('\n肿瘤的特征：\n',cancer['feature_names'])
```

运行代码，会得到如图 5-14 的结果。

```
肿瘤的分类： ['malignant' 'benign']

肿瘤的特征：
 ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

图 5-14　数据集中肿瘤的分类与特征名称

【结果分析】就像我们之前所说，该数据集中肿瘤的分类包括恶性（Malignant）和良性（Benign），而特征值就多了很多，如半径、表面纹理的灰度值、周长值、表面积值、平滑度等。当然这些都涉及一定的医学知识，我们就不逐一展开了。

下面我们开始使用朴素贝叶斯算法进行建模。

5.3.2　使用高斯朴素贝叶斯进行建模

用我们的直觉来分析的话，这个数据集的特征值并不属于二项式分布，也不属于多项式分布，所以这里我们选择使用高斯朴素贝叶斯（GaussianNB）。不过首先，我们要将数据集拆分为训练集和测试集，输入代码如下：

```
#将数据集的数值和分类目标赋值给x和y
X, y = cancer.data, cancer.target
#使用数据集拆分工具拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=38)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印训练集和测试集的数据形态
print('训练集数据形态：',X_train.shape)
print('测试集数据形态：',X_test.shape)
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到结果如图 5-15 所示。

```
代码运行结果：
==============================

训练集数据形态： (426, 30)
测试集数据形态： (143, 30)

==============================
```

图 5-15　训练集和测试集中数据的形态

【结果分析】从结果中我们可以看到，通过使用 train_test_split 工具进行拆分，现在的训练集中有 426 个样本，而测试集中有 143 个样本，当然，特征数量都是 30 个。

下面我们开始用高斯朴素贝叶斯对训练数据集进行拟合，输入代码如下：

```
#使用高斯朴素贝叶斯拟合数据
gnb = GaussianNB()
gnb.fit(X_train, y_train)
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型评分
```

```
print('训练集得分：{:.3f}'.format(gnb.score(X_train, y_train)))
print('测试集得分：{:.3f}'.format(gnb.score(X_test, y_test)))
print('\n==============================')
print('\n\n\n')
```

运行代码，将得到如图 5-16 所示的结果。

```
代码运行结果：
==============================

训练集得分：0.948
测试集得分：0.944

==============================
```

图 5-16　高斯朴素贝叶斯模型的得分

【结果分析】从结果中可以看到，GaussianNB 在训练集和测试集的得分都非常不错，均在 95% 左右。

下面我们随便用其中一个样本（如第 312 个样本）让模型进行一下预测，看是否可以分到正确的分类中，输入代码如下：

```
print('\n\n\n')
print('代码运行结果：')
print('==============================\n')
#打印模型预测的分类和真实的分类
print('模型预测的分类是：{}'.format(gnb.predict([X[312]])))
print('样本的正确分类是：',y[312])
print('\n==============================')
print('\n\n\n')
```

运行代码，会得到如图 5-17 所示的结果。

```
代码运行结果：
==============================

模型预测的分类是：[1]
样本的正确分类是： 1

==============================
```

图 5-17　模型预测的分类与其真实分类的对比

【结果分析】从结果中我们看到，模型对第 312 个样本所进行的分类和正确的分类完全一致，都是分类 1，也就是说，这个样本的肿瘤是一个良性的肿瘤。

5.3.3　高斯朴素贝叶斯的学习曲线

在机器学习中，有一个概念称为学习曲线（learning curve），指的是随着数据集样

本数量的增加，模型的得分变化情况。下面我们一起来绘制一下高斯朴素贝叶斯在威斯康星乳腺肿瘤数据集中的学习曲线，输入代码如下：

```
#导入学习曲线库
from sklearn.model_selection import learning_curve
#导入随机拆分工具
from sklearn.model_selection import ShuffleSplit
#定义一个函数绘制学习曲线
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
#设定横轴标签
plt.xlabel("Training examples")
#设定纵轴标签
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    plt.grid()

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="lower right")
    return plt

#设定图题
title = "Learning Curves (Naive Bayes)"
#设定拆分数量
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)
#设定模型为高斯朴素贝叶斯
estimator = GaussianNB()
#调用我们定义好的函数
plot_learning_curve(estimator, title, X, y, ylim=(0.9, 1.01), cv=cv, n_jobs=4)
#显示图片
plt.show()
```

运行代码，会得到如图 5-18 所示的结果。

【结果分析】从图 5-18 中可以看到，在训练数据集中，随着样本量的增加，模型的得分是逐渐降低的。这是因为随着样本数量增加，模型要拟合的数据越来越多，难度也越来越大。而模型的交叉验证得分的变化相对没有那么明显，从 10 个样本左右一直到接近 500 个样本为止，分数一直在 0.94 左右浮动。这说明高斯朴素贝叶斯在预测方面，对于样本数量的要求并没有那么苛刻。所以如果你的样本数量比较少的话，应该可以考虑

使用朴素贝叶斯算法来进行建模。

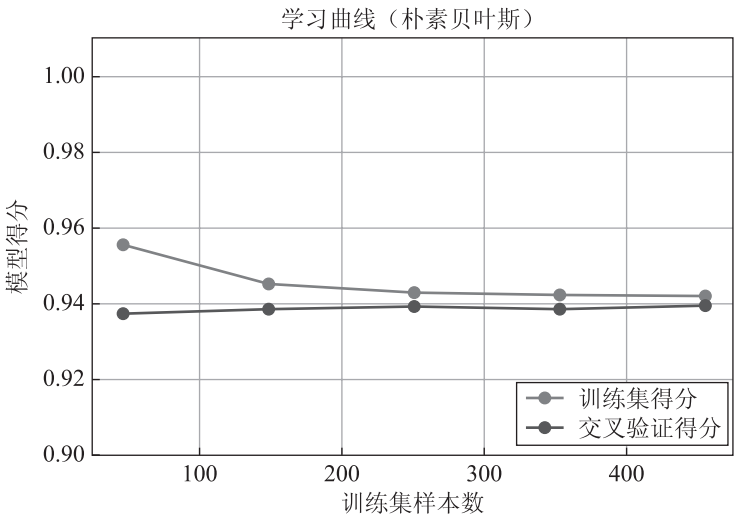

图 5-18　高斯朴素贝叶斯的学习曲线

5.4　小结

在本章中，我们一起学习了朴素贝叶斯算法和它的几种变体——贝努利朴素贝叶斯、高斯朴素贝叶斯和多项式朴素贝叶斯。贝努利朴素贝叶斯适合与二项式分布的数据集，而多项式朴素贝叶斯适合计数类型的数据集，即非负、离散数值的数据集，而高斯朴素贝叶斯适用的面就要广得多，它可以应用于任何连续数值型的数据集当中，当然如果是符合正态分布的数据集的话，高斯朴素贝叶斯模型的得分会更高。

相比起线性模型算法来说，朴素贝叶斯算法的效率要高一些，这是因为朴素贝叶斯算法会把数据集中的各个特征看作完全独立的，而不考虑特征之间的关联关系。但同时模型泛化的能力会稍微弱一点，不过一般情况下并不太影响实际的使用。尤其是在现在这个大数据时代，很多数据集的样本特征可能成千上万，这种情况下，模型的效率要比模型泛化性能多零点几个百分点的得分重要得多。在这种超高维度的数据集中，训练一个线性模型的时间可能会非常长，因此在这种情况下，朴素贝叶斯算法往往是一个更好的选择。

在第 6 章中，我们会一起学习决策树和随机森林算法，它们也是目前非常流行的算法之一，接下来我们马上开启新的旅程。

第 6 章　决策树与随机森林——会玩读心术的算法

按照惯例，我们还是用一个小故事来引入本章的内容：某天，小 C 的表妹小 Q 来找小 C，说她遇到了一点困扰——小 Q 的同事给她介绍了一个对象 Mr. Z，Mr. Z 现年 37 岁，在某省机关做文员工作。但是小 Q 的择偶标准是需要对方月薪在 5 万以上（不要骂小 Q 拜金，我们只是为了引入后面的内容），但是又不好直接问 Mr. Z，所以拿不定主意要不要和 Mr. Z 深入交往，想让小 C 帮忙做个决策。说到决策，小 C 自然想到决策树算法，而说到决策树算法，又自然会想到随机森林。

本章主要涉及的知识点有：

- 决策树的基本原理和构造
- 决策树的优势和不足
- 随机森林的基本原理和构造
- 随机森林的优势和不足
- 实例演示：小Q要不要和相亲对象进一步发展

6.1 决策树

决策树是一种在分类与回归中都有非常广泛应用的算法，它的原理是通过对一系列问题进行 if/else 的推导，最终实现决策。

6.1.1 决策树基本原理

记得有个在公司团建时候经常玩的游戏，称为 “读心术”——在一组人里选出一个出题者，出题者在心中默想一个人或事物，其余的人可以提问题，但是出题者只能回答“是”或者“否”，游戏限定提问者一共只能提出 20 个问题。在 20 个问题内，如果有人猜中了出题者心里想的人或事物，则出题者输掉游戏；如果 20 个问题问完还没有人猜中，则出题者胜利。这个游戏就可以使用决策树的算法来进行表达。

举个例子：假设出题者心里想的是斯嘉丽 • 约翰逊、泰勒斯威夫特、吴彦祖、威尔 • 史密斯 4 个人中的一个，则提问决策树如图 6-1 所示。

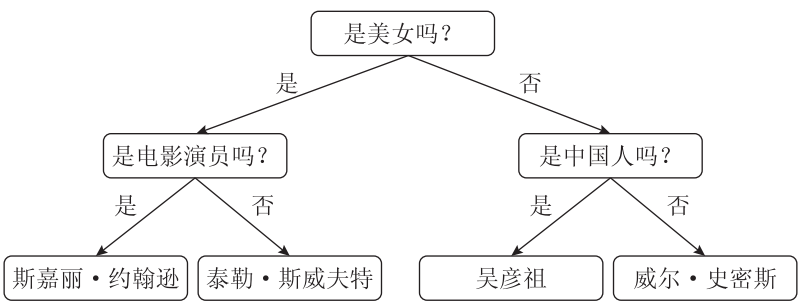

图 6-1 “读心术”游戏中的决策树

图 6-1 中，最终的 4 个节点，也就是 4 个人物的名字，被称为决策树的树叶。例子中的这棵决策树只有 4 片树叶，所以通过手动的方式就可以进行建模。但是如果样本的特征特别多，就不得不使用机器学习的办法来进行建模了。

6.1.2 决策树的构建

下面我们再次使用酒的数据集来演示一下决策树的构建，还记得在第 2 章做的实验吗？下面我们先载入酒的数据集，然后将它做成训练集和数据集，输入代码如下：

```
#导入numpy
import numpy as np
#导入画图工具
import matplotlib.pyplot as plt
```

```
from matplotlib.colors import ListedColormap
#导入tree模型和数据集加载工具
from sklearn import tree, datasets
#导入数据集拆分工具
from sklearn.model_selection import train_test_split
wine = datasets.load_wine()
#只选取数据集的前两个特征
X = wine.data[:,:2]
y = wine.target
#将数据集拆分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

现在完成了数据集的准备，开始用决策树分类器进行分类。

注意 **为了便于用图形进行演示，我们仍然只选取了数据集中样本的前两个特征。**

接下来，输入代码如下：

```
#设定决策树分类器最大深度为1
clf = tree.DecisionTreeClassifier(max_depth=1)
#拟合训练数据集
clf.fit(X_train,y_train)
```

运行代码，会得到如图 6-2 所示的结果。

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

图 6-2　决策树模型的全部参数

【结果分析】Jupyter Notebook 把分类器的参数返回，这些参数中，我们先关注其中之一，就是 max_depth 参数。这个参数指的是决策树的深度，也就是我们在玩“读心术”游戏的时候，所问的问题的数量，问题数量越多，就代表决策树的深度越深。现在我们使用的最大深度是 1，所以 max_depth = 1。

现在看看分类器的表现如何，我们把图形画出来，输入代码如下：

```
#定义图像中分区的颜色和散点的颜色
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

#分别用样本的两个特征值创建图像和横轴和纵轴
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```