python机器学习

通过这本重要的尖端预测分析指南，深入了解机器学习

塞巴斯蒂安·拉什卡

伯明翰-孟买

python机器学习

版权所有©2015 Packt Publishing

版权所有。未经出版商事先书面许可，不得复制、储存本书的任何部分，也不得以任何形式或任何方式传播本书的任何部分，除非关键文章或评论中有简短的引文。

为确保所提供信息的准确性，我们已尽一切努力准备了这本书。然而，本书所包含的信息在出售时没有明示或暗示的保证。作者、packt出版公司及其经销商和分销商对本书直接或间接造成或声称造成的任何损害概不负责。

Packt Publishing致力于通过适当的资本使用，提供本书中提到的所有公司和产品的商标信息。但是，packt发布不能保证此信息的准确性。

初版：2015年9月

生产参考号：1160915

由packt出版有限公司出版。

制服存放处

利维街35号

英国伯明翰B3 2PB。国际标准书号978-1-78355-513-0

信用

马德胡尼基塔·苏尼尔·钦达卡尔

塔比什汗

前言

我们生活在数据洪流之中。根据最近的估计，每天会生成250万（1018）字节的数据。这是如此之多的数据，以至于我们现在存储的超过90%的信息都是在过去的十年中生成的。不幸的是，大多数信息不能被人类使用。要么数据超出了标准分析方法的范围，要么数据太大，我们有限的头脑根本无法理解。

通过机器学习，我们使计算机能够处理、学习和从原本无法穿透的大数据墙中汲取可操作的见解。从支持谷歌搜索引擎的大型超级计算机到我们口袋里的智能手机，我们都依靠机器学习来经常为我们周围的大多数人提供动力，甚至不知道。

作为勇敢的大数据新世界的现代先驱者，我们应该更多地学习机器学习。什么是机器学习，它是如何工作的？我怎样才能用机器学习来了解未知的事物，为我的事业提供动力，或者只是了解互联网对我最喜欢的电影的看法？所有这些以及更多内容将在我的好朋友和同事塞巴斯蒂安·拉什卡（Sebastian Raschka）撰写的以下章节中介绍。

除了驯服我那只脾气暴躁的宠物狗之外，塞巴斯蒂安还不厌其烦地把他的空闲时间投入到开源机器学习社区。在过去的几年中，Sebastian开发了几十个流行的教程，涵盖了机器学习和Python中数据可视化的主题。他还开发并贡献了几个开源的python包，其中一些包现在是核心python机器学习工作流的一部分。

由于塞巴斯蒂安在这一领域拥有丰富的专业知识，我相信，塞巴斯蒂安对Python机器学习世界的洞察对于所有体验级别的用户都将是无价的。我全心全意推荐这本书给任何希望获得更广泛和更实际的机器学习理解的人。

Randal S.Olson博士

宾夕法尼亚大学人工智能和机器学习研究员

关于作者

Sebastian Raschka是密歇根州立大学的博士生，他在计算生物学领域开发了新的计算方法。他被分析公司Vidhya评为Github最具影响力的数据科学家之一。他在Python编程方面有一年的经验，他还举办了几次关于数据科学和机器学习的实际应用的研讨会。谈论和写作数据科学、机器学习和python真的激励塞巴斯蒂安写这本书，以帮助人们开发数据驱动的解决方案，而不必有机器学习的背景。

他还积极地为他所实施的开源项目和方法做出了贡献，这些项目和方法现在已经成功地应用于机器学习竞赛，如kaggle。在业余时间，他做运动预测模型，如果他不在电脑前，他喜欢运动。

我要感谢我的教授阿伦·罗斯和庞宁丹，以及其他许多启发我并激发我对模式分类、机器学习和数据挖掘的极大兴趣的人。

我想借此机会感谢伟大的Python社区和开源软件包的开发人员，他们帮助我为科学研究和数据科学创造了完美的环境。

特别感谢SciKit Learn的核心开发人员。作为这个项目的贡献者，我很高兴与伟大的人一起工作，他们不仅在机器学习方面很有知识，而且是优秀的程序员。

最后，我要感谢大家对这本书表现出的兴趣，我真诚地希望我能传递我的热情，加入巨蟒和机器学习社区。

关于审稿人

Richard Dutton在8岁时就开始编写ZX频谱，他对ZX频谱的痴迷使他在技术和金融领域中经历了一系列令人困惑的技术和角色。

他曾与微软合作过，作为巴克莱银行的董事，他现在的痴迷是将Python、机器学习和区块链结合在一起。

如果他不在电脑前，他可以在健身房或家里看iPhone时喝一杯酒。他称之为平衡。

戴夫·朱利安是一名IT顾问和教师，拥有超过15年的经验。他曾担任过技术员、项目经理、程序员和Web开发人员。他目前的项目包括开发作物分析工具，作为温室害虫综合管理战略的一部分。他对生物学和技术的交叉有着浓厚的兴趣，相信智能机器可以帮助解决世界上最重要的问题。

Vahid Mirjalili在密歇根州立大学获得了机械工程博士学位，在那里他利用分子动力学模拟开发了蛋白质结构精炼的新技术。结合他在统计、数据挖掘和物理领域的知识，他开发了强大的数据驱动方法，帮助他和他的研究小组在2012年和2014年赢得了最近两次全球蛋白质结构预测和精炼竞赛，即casp。

在攻读博士学位的同时，他决定加入密歇根州立大学计算机科学与工程系，专攻机器学习领域。他目前的研究项目包括开发无监督机器学习算法，用于挖掘海量数据集。他也是一个热情的Python程序员，并在他的个人网站上分享他的集群算法实现。

Hamidreza Sattari是一名IT专业人士，曾参与软件工程的多个领域，从编程到架构，以及管理。他拥有赫里奥瓦特大学软件工程硕士学位，

英国，德黑兰阿扎德电气工程（电子）学士学位

伊朗大学。近年来，他的兴趣领域一直是大数据和机器学习。他与人合著了《SpringWebServices2CookBook》，并在上维护了他的博客。

dmytro taranovsky是一名软件工程师，对python、linux和机器学习有兴趣和背景。他原籍乌克兰基辅，1996年移居美国。从很小的时候，他就表现出对科学和知识的热情，赢得了数学和物理竞赛。1999年，他被选为美国物理团队的成员。2005年，他毕业于麻省理工学院，主修数学。后来，他在计算机辅助医学转录（描述）的文本转换系统上担任软件工程师。尽管他最初是在Perl上工作的，但他欣赏Python的强大和清晰，并且能够将系统扩展到非常大的数据大小。之后，他在一家算法贸易公司担任软件工程师和分析师。他还为数学的创立做出了重要贡献，包括创立和发展集合论语言的扩展和它与大基数公理的联系，发展建构真理的概念，并建立有序符号和实现T的系统。蟒蛇的折边。他也喜欢读书，喜欢户外活动，努力使世界变得更好。

网址：www.packtpub.com

支持文件、电子书、折扣优惠等

有关书籍的支持文件和下载，请访问

你知道Packt提供每本已出版书籍的电子书版本，并提供PDF和EPUB文件吗？你可以升级到电子书版本在和作为印刷书客户，你有权对电子书副本的折扣。更多详情请联系我们service@packtpub.com。

您还可以阅读免费的技术文章集，注册一系列免费的新闻通讯，并获得包装书和电子书的独家折扣和优惠。

您是否需要即时解决您的IT问题？packtlib是packt的在线数字图书库。在这里，您可以搜索、访问和阅读packt的整个图书库。

为什么要订阅？

可在packt出版的每本书中完全搜索

复制和粘贴、打印和书签内容•按需提供，可通过Web浏览器访问

packt账户持有人免费访问

如果您在packt有一个帐户，您可以使用它访问packtlib，并查看9本完全免费的书籍。只需使用您的登录凭证即可立即访问。

目录

[二]

目录

[三]

[四]

目录

第10章：预测连续目标变量

[五]

[六]

前言

我可能不需要告诉你机器学习已经成为我们这个时代最令人兴奋的技术之一。谷歌、Facebook、苹果、亚马逊、IBM等大公司出于充分的理由大量投资于机器学习研究和应用。虽然机器学习似乎已经成为我们这个时代的流行语，但它肯定不是一个炒作。这一激动人心的领域开辟了新的可能性，并已成为我们日常生活不可或缺的一部分。在我们的智能手机上与语音助理交谈，为客户推荐合适的产品，阻止信用卡欺诈，从我们的电子邮件收件箱中过滤垃圾邮件，检测和诊断医疗疾病，这个列表持续不断。

如果你想成为一个机器学习实践者，一个更好的问题解决者，甚至可能考虑在机器学习研究的职业，那么这本书是为你！然而，对于一个新手来说，机器学习背后的理论概念是非常压倒一切的。然而，近年来出版的许多实用书籍将通过实现强大的学习算法帮助您开始机器学习。在我看来，使用实际的代码示例有一个重要的目的。他们通过将所学的材料直接付诸行动来阐明这些概念。然而，请记住，拥有强大的力量，责任重大！机器学习背后的概念太美了，太重要了，不能隐藏在黑匣子里。因此，我的个人使命是为您提供一本不同的书；一本讨论有关机器学习概念的必要细节的书，提供关于机器学习算法如何工作、如何使用它们以及最重要的是如何避免注意陷阱。

如果你在google scholator中输入“机器学习”作为搜索词，它会返回一个绝大多数数量的180万份出版物。当然，我们不能讨论过去60年中出现的所有不同算法和应用程序的所有细节。然而，在这本书中，我们将开始一段激动人心的旅程，涵盖所有基本的主题和概念，让您在这一领域有一个领先的开端。如果你发现你对知识的渴望没有得到满足，有许多有用的资源可以用来跟进这一领域的基本突破。

[七]

如果你已经详细学习了机器学习理论，这本书将向你展示如何将你的知识付诸实践。如果你以前使用过机器学习技术，并且想更深入地了解机器学习是如何工作的，那么这本书就是为你准备的！如果你是机器学习领域的新手，不要担心；你有更多的理由感到兴奋。我向你保证，机器学习将改变你对你想要解决的问题的思考方式，并向你展示如何通过释放数据的力量来解决问题。

在深入机器学习领域之前，让我来回答您最重要的问题：“为什么是Python？”答案很简单：它很强大，但却很容易接近。python已经成为数据科学最流行的编程语言，因为它可以让我们忘记编程中繁琐的部分，并为我们提供一个可以快速记下想法并将概念直接付诸实践的环境。

回想我的个人旅程，我可以说，学习机器学习使我成为一个更好的科学家、思想家和问题解决者。在这本书中，我想和你分享这些知识。知识是通过学习获得的，关键是我们的热情，真正掌握技能只能通过实践来实现。未来的道路有时可能会坎坷不平，有些话题可能比其他话题更具挑战性，但我希望你能抓住这个机会，专注于回报。请记住，我们正一起踏上这段旅程，在这本书中，我们将为您的军火库添加许多强大的技术，这些技术将帮助我们以数据驱动的方式解决最棘手的问题。

这本书包括什么

第1章，让计算机能够从数据中学习，向您介绍机器学习的主要子领域，以解决各种问题任务。此外，本文还讨论了创建典型机器学习模型构建管道的基本步骤，这些步骤将指导我们完成以下章节。

第二章，训练机器学习分类算法，追溯机器学习的起源，介绍了二元感知器分类器和自适应线性神经元。本章对模式分类的基本原理进行了温和的介绍，重点介绍了优化算法和机器学习的相互作用。

第3章，介绍了使用SciKit学习的机器学习分类程序，描述了分类的基本机器学习算法，并提供了使用最流行和最全面的开源机器学习库之一SciKit学习的实用示例。

[八]

第4章，构建良好的训练集-数据预处理，讨论如何处理未处理数据集中最常见的问题，如丢失数据。它还讨论了识别数据集中最具信息性特征的几种方法，并教您如何准备不同类型的变量作为机器学习算法的适当输入。

第5章，通过降维压缩数据，描述了将数据集中的特征数量减少到较小集合，同时保留大部分有用和歧视性信息的基本技术。讨论了主成分分析降维的标准方法，并将其与监督变换和非线性变换技术进行了比较。

第六章，学习模型评估和超参数调整的最佳实践，讨论了预测模型性能评估的注意事项。此外，还讨论了衡量模型性能的不同指标以及微调机器学习算法的技术。

第七章，结合不同的集成学习模型，介绍了有效组合多种学习算法的不同概念。它教你如何建立专家团队来克服个别学习者的弱点，从而得到更准确和可靠的预测。

第八章，将机器学习应用于情感分析，探讨了将文本数据转化为有意义的表示的必要步骤，以便机器学习算法根据人们的写作预测他们的意见。

第9章，将机器学习模型嵌入到Web应用程序中，继续上一章的预测模型，并指导您完成使用嵌入式机器学习模型开发Web应用程序的基本步骤。

第10章，用回归分析预测连续目标变量，讨论了在连续尺度上预测目标和响应变量之间线性关系的基本技术。在介绍了不同的线性模型之后，还讨论了多项式回归和基于树的方法。

第11章，使用未标记的数据-聚类分析，将重点转移到机器学习的不同子领域，无监督学习。我们应用三个基本的聚类算法家族的算法来寻找具有一定相似性的对象群。

[九]

第12章，训练用于图像识别的人工神经网络，扩展了基于梯度的优化的概念，我们在第2章，训练用于分类的机器学习算法中首先介绍了这一概念，以建立基于流行的反向传播的强大的多层神经网络。离子算法。

第13章，将神经网络训练与Theano并行，建立在前一章的知识基础上，为更有效地训练神经网络提供实践指导。本章的重点是Theano，一个开放源码的Python库，它允许我们利用现代GPU的多个核心。

这本书你需要什么

本书中提供的代码示例的执行需要在Mac OS X、Linux或Microsoft Windows上安装python 3.4.3或更高版本。我们将在本书中频繁使用python的科学计算基本库，包括scipy、numpy、scikit-learn、matplotlib和pandas。

第一章将为您提供设置Python环境和这些核心库的说明和有用提示。我们将在我们的剧目中添加额外的库，安装说明在相应的章节中提供：自然语言处理的NLTK库（第8章，将机器学习应用于情感分析）、flask web框架（第9章，嵌入机器学习lgorithm into a web application）、Seaborn统计数据可视化库（第10章，用回归分析预测连续目标变量）和Theano，用于图形处理单元的高效神经网络培训（第13章，并行神经网络培训，W.和Theano）。

这本书是给谁的

如果您想了解如何使用python来开始回答数据的关键问题，请学习python机器，无论您是想从头开始，还是想扩展您的数据科学知识，这是一个必不可少的、不可错过的资源。

习俗

在这本书中，你会发现许多不同信息类型的文本样式。下面是这些样式的一些例子，并解释了它们的含义。

[十]

文本中的代码字、数据库表名、文件夹名、文件名、文件扩展名、路径名、虚拟URL、用户输入和Twitter句柄如下所示：“并且可以通过--upgrade标志更新已安装的包。”

代码块设置如下：

>>>导入matplotlib.pyplot为plt

>>>导入numpy为np

>>>Y=df.iloc[0:100，4].值

>>>Y=NP.其中（Y=‘Iris Setosa’，-1，1）

>>>X=df.iloc[0:100，[0，2]].值

>>>PLT.散射（X[：50，0]，X[：50，1]，

……color='red'，marker='x'，label='setosa'）

>>>PLT.散射（X[50:100，0]，X[50:100，1]，

……color='blue'，marker='o'，label='versicolor'）

>>>plt.xlabel（'花瓣长度'）

>>>plt.ylabel（'sepal length'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

任何命令行输入或输出如下：

>点-tpng tree.dot-o tree.png

新术语和重要词汇以粗体显示。您在屏幕上看到的单词，例如，在菜单或对话框中，会出现在这样的文本中：“单击右上角的仪表板按钮后，我们可以访问显示在页面顶部的控制面板。”

警告或重要提示出现在这样的框中。

提示和技巧如下所示。

[西]

读者反馈

欢迎读者的反馈。让我们知道你对这本书的看法，你喜欢或不喜欢什么。读者反馈对我们很重要，因为它可以帮助我们开发出你真正能充分利用的标题。

若要向我们发送一般反馈，只需发送电子邮件feedback@packtpub.com，并在邮件主题中提及本书的标题。

如果您对某个主题有专长，并且您对某本书的写作或贡献感兴趣，请参阅我们的作者指南

客户支持

既然你是一本包装书的骄傲拥有者，我们有很多东西可以帮助你从购买中获得最大的收益。

下载示例代码

对于您购买的所有packt出版书籍，您可以从您的帐户下载示例代码文件。如果您在别处购买了这本书，您可以访问并注册，直接通过电子邮件将文件发送给您。

勘误表

虽然我们已经尽一切努力确保内容的准确性，但确实会出现错误。如果你在我们的一本书中发现错误，也许是文本或代码中的错误，如果你能向我们报告，我们将不胜感激。通过这样做，你可以让其他读者免受挫折，并帮助我们改进本书的后续版本。如果您发现任何勘误表，请访问、选择您的书籍、单击勘误表提交表单链接并输入您的勘误表的详细信息来报告它们。一旦您的勘误表被核实，您的提交将被接受，并且勘误表将上传到我们的网站或添加到该标题的勘误表部分下的任何现有勘误表列表中。

要查看以前提交的勘误表，请转到并在搜索字段中输入书籍的名称。所需信息将显示在勘误表部分下。

[十二]

盗版

互联网上受版权保护的材料的盗版是所有媒体都在持续的问题。在packt，我们非常重视版权和许可证的保护。如果您在互联网上发现任何形式的非法复制品，请立即向我们提供地址或网站名称，以便我们寻求补救。

请通过copyright@packtpub.com与我们联系，链接到可疑的盗版材料。

我们感谢您在保护我们的作者以及我们为您提供宝贵内容的能力方面的帮助。

问题

如果您对本书的任何方面有问题，可以通过questions@packtpub.com与我们联系，我们将尽最大努力解决问题。

[十三]

使计算机能够从数据中学习

在我看来，机器学习、算法的应用和科学是所有计算机科学中最令人兴奋的领域。我们生活在一个数据丰富的时代，利用机器学习领域的自学习算法，我们可以将这些数据转化为知识。由于近年来开发了许多功能强大的开放源码库，因此可能从未有过更好的时间来深入机器学习领域，学习如何利用功能强大的算法来发现数据中的模式并预测未来的事件。

在本章中，我们将学习机器学习的主要概念和不同类型。结合相关术语的基本介绍，我们将为成功使用机器学习技术解决实际问题奠定基础。

在本章中，我们将讨论以下主题：

机器学习的一般概念

三种类型的学习和基本术语

成功设计机器学习系统的基石

安装和设置用于数据分析和机器学习的python

构建智能机器将数据转化为知识

在这个现代技术的时代，我们拥有一种丰富的资源：大量结构化和非结构化数据。在二十世纪后半叶，机器学习发展成为人工智能的一个分支领域，它涉及到自学习算法的发展，以从数据中获取知识，以便做出预测。机器学习不需要人类从分析大量数据中手动派生规则和构建模型，而是提供了一种更有效的选择，用于捕获数据中的知识，以逐步提高预测模型的性能，并做出数据驱动的决策。机器学习不仅在计算机科学研究中越来越重要，而且在我们的日常生活中也扮演着越来越重要的角色。由于机器学习，我们享受强大的电子邮件垃圾邮件过滤器，方便的文本和语音识别软件，可靠的网络搜索引擎，具有挑战性的棋手，并希望不久，安全和高效的自动驾驶汽车。

三种不同类型的机器学习

在本节中，我们将介绍三种机器学习：监督学习、无监督学习和强化学习。我们将了解这三种不同学习类型之间的基本差异，并通过概念示例，我们将为实际问题领域发展一种直觉，这些领域可以应用：

[2]

通过有监督的学习预测未来

监督学习的主要目标是从有标签的训练数据中学习一个模型，使我们能够对看不见的或未来的数据做出预测。这里，术语监督是指已经知道所需输出信号（标签）的一组样本。

考虑到电子邮件垃圾邮件过滤的例子，我们可以使用监控机器学习算法对标记为“垃圾邮件”或“非垃圾邮件”的电子邮件的语料库训练模型，以预测新电子邮件是否属于这两个类别中的任何一个。具有离散类标签的有监督的学习任务，例如在以前的电子邮件垃圾邮件过滤示例中，也被称为分类任务。监督学习的另一个子类别是回归，其中结果信号是一个连续值：

预测类标签的分类

分类是监督学习的一个子类别，其目标是根据过去的观察预测新实例的分类类标签。这些类标签是离散的、无序的值，可以理解为实例的组成员身份。前面提到的电子邮件垃圾邮件检测示例代表了二进制分类任务的典型示例，其中机器学习算法学习一组规则，以便区分两个可能的类：垃圾邮件和非垃圾邮件。

[3]

但是，类标签集不必是二进制的。由监督学习算法学习的预测模型可以将训练数据集中出现的任何类标签分配给一个新的、未标记的实例。多类分类任务的一个典型例子是手写字符识别。在这里，我们可以收集一个训练数据集，它由字母表中每个字母的多个手写示例组成。现在，如果用户通过输入设备提供一个新的手写字符，我们的预测模型将能够以一定的精度预测字母表中的正确字母。但是，我们的机器学习系统将无法正确识别零到九的任何数字，例如，如果它们不是我们的培训数据集的一部分。

下图说明了给定30个训练样本的二元分类任务的概念：15个训练样本标记为负类（圆），15个训练样本标记为正类（加号）。在这个场景中，我们的数据集是二维的，这意味着每个样本都有两个与其相关联的值：x1和x2。现在，我们可以使用监督机器学习算法来学习规则决策边界表示为一条黑色虚线，它可以将这两个类分开，并根据给定的x1和x2值将新数据分为这两类：

预测连续结果的回归

我们在上一节中了解到，分类的任务是为实例分配分类的、无序的标签。第二类监督学习是对连续结果的预测，也称为回归分析。在回归分析中，我们得到了一些预测（解释）变量和一个连续反应变量（结果），我们试图找到这些变量之间的关系，使我们能够预测结果。

[4]

例如，假设我们对预测学生的数学SAT成绩感兴趣。如果考试的学习时间与最终成绩之间存在关系，我们可以将其用作培训数据，学习一个模型，该模型利用学习时间预测未来打算参加考试的学生的考试成绩。

“回归”一词是弗朗西斯·高尔顿在1886年发表的《回归到平庸的遗传地位》一文中提出的。盖尔顿描述了一种生物现象，即一个种群的身高变化不会随着时间的推移而增加。他观察到，父母的身高并没有传给他们的孩子，但孩子的身高正在向人口平均值递减。

下图说明了线性回归的概念。给定一个预测变量x和一个响应变量y，我们将一条直线拟合到该数据中，这条直线最大程度地减小了采样点和拟合线之间的平均平方距离。我们现在可以使用从这些数据中获得的截距和斜率来预测新数据的结果变量：

[5]

用强化学习解决互动问题

另一种机器学习是强化学习。在强化学习中，目标是开发一个基于与环境交互的系统（代理），以提高其性能。由于有关环境当前状态的信息通常还包括所谓的奖励信号，因此我们可以将强化学习视为与监督学习相关的领域。然而，在强化学习中，这种反馈不是正确的基本事实标签或价值，而是一种衡量行为如何通过奖励函数来衡量的指标。通过与环境的交互，代理可以使用强化学习来学习一系列的行为，这些行为通过探索性的尝试和错误的方法或深思熟虑的计划来最大化回报。

强化学习的一个流行例子是象棋引擎。在这里，代理根据董事会（环境）的状态决定一系列动作，并且奖励可以定义为在游戏结束时赢或输：

利用无监督学习发现隐藏结构

在有监督的学习中，当我们训练我们的模型时，我们预先知道正确的答案，在强化学习中，我们定义了代理对特定行为的奖励措施。然而，在无监督学习中，我们处理的是未标记的数据或未知结构的数据。使用无监督学习技术，我们能够探索数据结构，在没有已知结果变量或奖励函数的指导下提取有意义的信息。

[6]

用聚类法寻找子群

集群是一种探索性的数据分析技术，它允许我们将一堆信息组织成有意义的子组（集群），而不必事先知道它们的组成员身份。在分析过程中可能出现的每个聚类定义了一组具有一定相似性但与其他聚类中的对象更为不同的对象，这就是为什么聚类有时也被称为“无监督分类”。聚类是构建例如，信息和在数据之间产生有意义的关系，使营销人员能够根据自己的兴趣发现客户群，从而制定出独特的营销计划。

下图说明了如何根据特征x1和x2的相似性将未标记的数据组织成三个不同的组：

数据压缩的维数约简

无监督学习的另一个子领域是维度约简。我们通常使用高维数据，每次观测都会有大量的测量结果，这些测量结果会对有限的存储空间和机器学习算法的计算性能提出挑战。无监督降维是特征预处理中常用的去除数据噪声的方法，它也会降低某些算法的预测性能，将数据压缩到较小的维子空间，同时保留大部分相关信息。

[七]

有时，维数减少也可用于可视化数据，例如，可以将高维特征集投影到一维、二维或三维特征空间上，以便通过三维或二维散点图或柱状图进行可视化。下图显示了一个示例，其中非线性维数减少用于将三维Swiss Roll压缩到新的二维特征子空间：

基本术语和符号介绍

既然我们已经讨论了机器学习的三大类，即监督学习、无监督学习和强化学习，那么让我们来看看我们将在下一章中使用的基本术语。下表描述了IRIS数据集的摘录，这是机器学习领域的一个典型示例。鸢尾数据集包含了来自三个不同种类的150朵鸢尾花的测量结果：狗尾草、花青草和绿原花。在这里，每个花样本代表我们的数据集中的一行，以厘米为单位的花测量值存储为列，我们也将其称为数据集的特征：

[8]

为了保持符号和实现简单而有效，我们将利用线性代数的一些基础知识。在下面的章节中，我们将使用矩阵和向量表示法来引用我们的数据。我们将遵循共同的约定，将每个样本表示为特征矩阵X中的单独行，其中每个特征都存储为单独的列。

IRIS数据集由150个样本和4个特性组成，可以写成

150×4矩阵X：

[9]

对于本书的其余部分，我们将使用上标（i）引用第i个训练样本，下标j引用训练数据集的第j个维度。

我们使用小写、粗体字母分别表示向量（x∈rn×1）和大写、粗体字母分别表示矩阵。为了引用向量或矩阵中的单个元素，我们分别用斜体x（n）或x（（mn））编写字母。

例如，x1150是指花样本150的第一个尺寸，即花色宽度。因此，该特征矩阵中的每一行代表一个花实例，可以写成四维列向量x，

x（i）=x1（i）x2（i）x3（i）x4（i）。

每个特征尺寸都是一个150维的行向量x，例如：

X J XXJ J（（12））

=

XJ（150）。

同样，我们将目标变量（这里是类标签）存储为

Y（1）

（Y∈setosa，花色，弗吉尼亚）。150维柱向量y=…

Y（150）

构建机器学习系统的路线图

在前面的章节中，我们讨论了机器学习的基本概念以及三种不同类型的学习。在本节中，我们将讨论与学习算法一起使用的机器学习系统的其他重要部分。下图显示了在预测建模中使用机器学习的典型工作流程图，我们将在以下小节中讨论：

[10]

预处理-使数据成形

原始数据很少以最佳学习算法性能所必需的形式和形状出现。因此，数据的预处理是任何机器学习应用程序中最关键的步骤之一。如果我们以前面部分的虹膜花数据集为例，我们可以将原始数据视为一系列花图像，我们希望从中提取有意义的特征。有用的特征可以是颜色、色调、花的强度、高度和花的长度和宽度。许多机器学习算法还要求选择的特征在相同的尺度上以获得最佳性能，这通常是通过转换范围[0，1]中的特征或具有零平均值和单位方差的标准正态分布来实现的，如我们将在后面的章节中看到的。

某些选定的功能可能高度相关，因此在一定程度上是冗余的。在这些情况下，维数约简技术对于将特征压缩到低维子空间非常有用。减小特征空间的维数具有存储空间少、学习算法运行速度快的优点。

[11]

为了确定我们的机器学习算法是否不仅在训练集上表现良好，而且能很好地概括为新的数据，我们还希望将数据集随机划分为单独的训练和测试集。我们使用训练集来训练和优化我们的机器学习模型，而我们将测试集保留到最后评估最终模型。

培训和选择预测模型

正如我们将在后面的章节中看到的，已经开发了许多不同的机器学习算法来解决不同的问题任务。大卫·沃尔伯特著名的“无免费午餐”定理可以总结出一个重要的观点，那就是我们不能“免费”学习（D.H.沃尔伯特，1996；D.H.沃尔伯特和W.G.Macready，1997）。直观地说，我们可以将这个概念与流行的说法联系起来，“我想，如果你唯一拥有的工具是锤子，那么把一切都当作钉子来对待是很诱人的”（Abraham Maslow，1966年）。例如，每一种分类算法都有其固有的偏差，如果我们不对任务做任何假设，任何单一的分类模型都不具有优势。因此，在实践中，为了训练和选择最佳性能的模型，必须对至少几种不同的算法进行比较。但是在我们比较不同的模型之前，我们首先必须决定度量性能的指标。一个常用的度量是分类精度，它定义为正确分类实例的比例。

一个合法的问题是：如果我们不将这个测试集用于模型选择，而是将它用于最终的模型评估，那么我们如何知道哪个模型在最终的测试数据集和真实的数据上表现良好？为了解决这个问题中嵌入的问题，可以使用不同的交叉验证技术，将训练数据集进一步划分为训练和验证子集，以估计模型的泛化性能。最后，我们也不能期望软件库提供的不同学习算法的默认参数对于我们的特定问题任务是最优的。因此，我们将经常使用超参数优化技术，帮助我们在后面的章节中微调模型的性能。直观地说，我们可以将这些超参数视为参数，这些参数不是从数据中学习到的，而是表示模型的旋钮，我们可以利用这些旋钮来提高模型的性能，在后面的章节中，当我们看到实际的例子时，这些参数会变得更加清晰。

[12]

评估模型和预测看不见的数据实例

在我们选择了一个模型，并将其与训练数据集相匹配之后，我们可以使用测试数据集来估计它在这个未知数据上的表现，以估计泛化错误。如果我们对它的性能满意，我们现在可以使用这个模型来预测新的、未来的数据。需要注意的是，上述过程的参数（如特征缩放和维数减少）仅从训练数据集中获得，相同的参数随后被重新应用以转换测试数据集以及任何新的数据样本-T。否则，根据测试数据测量的性能可能过于乐观。

使用python进行机器学习

Python是数据科学中最流行的编程语言之一，因此它拥有大量由其庞大的社区开发的有用的附加库。

虽然解释语言（如python）在计算密集型任务中的性能不如低级编程语言，但是已经开发了扩展库（如numpy和scipy），这些扩展库是在底层fortran和c实现的基础上构建的，以实现快速和矢量化的操作。多维数组上的操作。

对于机器学习编程任务，我们将主要参考SciKit学习库，它是当今最流行和可访问的开放源码机器学习库之一。

安装python包

python可用于所有三个主要操作系统：Microsoft Windows、Mac OS X和Linux，安装程序以及文档可从官方python网站下载：。

本书是为python版本>=3.4.3编写的，建议您使用当前可用的最新版本python 3，尽管大多数代码示例也可能与python>=2.7.10兼容。如果您决定使用python 2.7来执行代码示例，请确保您了解两个python版本之间的主要区别。关于python 3.4和2.7之间的差异的一个很好的总结可以在

【13】

我们将在本书中使用的附加包可以通过pip安装程序安装，该程序自python 3.3以来一直是python标准库的一部分。有关PIP的更多信息，请访问。

在成功安装了python之后，我们可以从命令行终端执行pip来安装其他python包：

pip安装somepackage

可以通过--upgrade标志更新已安装的包：

pip安装somepackage--升级

一个强烈推荐的科学计算用的python发行版是anaconda（通过连续体分析）。Anaconda是一个免费的包括商业用途的企业级python发行版，它在一个用户友好的跨平台发行版中捆绑了用于数据科学、数学和工程的所有基本python包。可从下载“蟒蛇”安装程序，并可从获取“蟒蛇快速入门指南”。

在成功安装了anaconda之后，我们可以使用以下命令安装新的python包：conda install somepackage

可以使用以下命令更新现有包：

Conda更新somepackage

在本书中，我们主要使用Numpy的多维数组来存储和操作数据。偶尔，我们会利用pandas，它是一个建立在numpy之上的库，提供额外的更高级的数据操作工具，使处理表格数据更加方便。为了增强我们的学习体验和可视化定量数据，这通常对于直观地理解它非常有用，我们将使用非常可定制的matplotlib库。

[14]

下面列出了用于编写本书的主要python包的版本号。请确保已安装软件包的版本号等于或大于这些版本号，以确保代码示例正确运行：

数字1.9.1

坐骨神经痛0.14.0

Scikit学习0.15.2

Matplotlib 1.4.0

熊猫0.15.2

总结

在这一章中，我们对机器学习进行了非常高的探索，并熟悉了我们将在下一章中更详细地探讨的大局和主要概念。

我们了解到监督学习由两个重要的子领域组成：分类和回归。虽然分类模型允许我们将对象分类为已知类，但我们可以使用回归分析来预测目标变量的连续结果。无监督学习不仅为发现未标记数据中的结构提供了有用的技术，而且对于特征预处理步骤中的数据压缩也很有用。

我们简要地介绍了将机器学习应用到问题任务的典型路线图，我们将在下面的章节中作为更深层次的讨论和实际例子的基础。最后，我们设置了我们的Python环境，并安装和更新了所需的包，以准备好看到机器学习的实际应用。

[15]

在下一章中，我们将实施最早的分类机器学习算法之一，这将为我们准备第3章，使用SciKit Learn的机器学习分类器之旅，其中我们将介绍使用SciKit Learn开放源代码M的更高级的机器学习算法。阿辛学习图书馆。由于机器学习算法是从数据中学习的，因此我们必须向它们提供有用的信息，在第4章，建立良好的训练集数据预处理中，我们将了解重要的数据预处理技术。在第五章，通过降维压缩数据，我们将学习降维技术，可以帮助我们将数据集压缩到一个低维特征子空间，这有利于计算效率。建立机器学习模型的一个重要方面是评估它们的性能，并评估它们对新的、看不见的数据做出预测的能力。在第6章中，学习模型评估和超参数优化的最佳实践，我们将学习所有关于模型优化和评估的最佳实践。在某些情况下，尽管我们可能花费了数小时或数天的时间进行广泛的调优和测试，但我们仍然可能对预测模型的性能不满意。在第7章中，结合不同的集成学习模型，我们将学习如何结合不同的机器学习模型来构建更强大的预测系统。

在介绍了典型机器学习管道的所有重要概念之后，我们将在第8章中实现一个文本中预测情绪的模型，将机器学习应用于情绪分析，在第9章中，将机器学习模型嵌入到Web应用程序中，我们将把它放到一个Web应用程序中，与世界共享。在第10章，使用回归分析预测连续目标变量，然后我们将使用机器学习算法进行回归分析，从而预测连续输出变量，在第11章，使用未标记的数据-聚类分析，我们将应用聚类分析。允许我们在数据中找到隐藏结构的算法。这本书的最后一章将涵盖人工神经网络，它将使我们能够解决复杂的问题，如图像和语音识别，这是目前机器学习研究的热点之一。

[16]

训练机器学习

分类算法

在本章中，我们将使用第一个算法描述的分类机器学习算法，感知器和自适应线性神经元。我们将从在python中逐步实现感知器开始，并训练它在虹膜数据集中对不同的花进行分类。这将帮助我们理解机器学习分类算法的概念，以及如何在Python中有效地实现它们。讨论使用自适应线性神经元的优化基础，然后将为通过Scikit学习机器学习库使用更强大的分类器奠定基础，第3章，使用Scikit学习机器学习分类器的教程。

我们将在本章中讨论的主题如下：

机器学习算法的直觉构建

使用panda、numpy和matplotlib读取、处理和可视化数据

在python中实现线性分类算法

【17】

人工神经元——机器学习早期历史的简要一瞥

在我们更详细地讨论感知器和相关算法之前，让我们先简单地了解一下机器学习的早期阶段。为了理解生物大脑是如何工作来设计人工智能的，沃伦·麦考洛克和沃尔特·皮特于1943年发表了第一个简化脑细胞的概念，即所谓的麦考洛克·皮特（mcp）神经元（W.S.麦考洛克和W.皮特）。神经活动中内在的思想的逻辑演算。《数学生物物理学公报》，5（4）：115–133，1943）。神经元是大脑中相互连接的神经细胞，参与化学和电信号的处理和传输，如下图所示：

McCullock和Pitts将这种神经细胞描述为具有二进制输出的简单逻辑门；多个信号到达树突，然后集成到细胞体中，如果累积信号超过某个阈值，则会生成一个输出信号，该信号将由轴突传递。

[18]

几年后，Frank Rosenblatt发表了基于MCP神经元模型的感知器学习规则的第一个概念（F.Rosenblatt，感知器，一种感知和识别的自动机）。康奈尔航空实验室，1957年）。罗森布拉特利用感知器规则，提出了一种自动学习最优权重系数的算法，然后将其与输入特征相乘，以决定是否触发一个神经元。在有监督学习和分类的背景下，这种算法可以用来预测样本是否属于某个类。

更正式地说，我们可以将这个问题作为一个二进制分类任务来处理，为了简单起见，我们将两个类称为1（正类）和-1（负类）。然后，我们可以定义一个激活函数φ（z），它采用特定输入值x和相应的权重向量w的线性组合，其中z是所谓的净输入（z=w1x1++wmxm）：

w1 x 1 w=，x=wm xm

现在，如果特定样本x（i）的激活，即φ（z）的输出大于定义的阈值θ，我们预测1类和1类，否则，在感知器算法中，激活函数φ（）是一个简单的单位阶跃函数，有时也称为重侧台阶功能：

φ（z）=−11其他wisif z≥θe

【19】

为了简单起见，我们可以将阈值θ移到方程的左侧，并将权重零定义为w0=−θ和x0=1，这样我们可以将z以更紧凑的形式写入z=w0x0+w1x1++wmxm=wt x和φ（z）=−11其他wisif z≥θe。

在下面的部分中，我们将经常使用线性代数中的基本符号。例如，我们将使用向量点积来缩写x和w中值的乘积之和，而上标t表示转置，这是一种将列向量转换为行向量的操作，反之亦然：

Z=w0x0+w1x1++wmxm=∑mj=0 x jw j=wt x

4\_

例如：【1 2 3】×1×4+2×5+3×6=32。

6\_

此外，转置操作还可以应用于矩阵，以在其对角线上反映矩阵，例如：

1 2\_t

\_

53 6 4\_=12 34 56\_\_

 

在这本书中，我们将只使用线性代数中非常基本的概念。但是，如果您需要快速复习，请看一下Zico Kolter的优秀线性代数复习和参考，这是免费提供的。

在

下图说明了网络输入z=wt x如何被感知器（左子图）的激活函数压缩成二进制输出（-1或1），以及如何使用它来区分两个线性可分离类（右子图）：

[20]

MCP神经元和Rosenblatt的阈值感知器模型背后的全部思想是使用还原论方法来模拟大脑中单个神经元的工作方式：它要么激发要么不激发。因此，Rosenblatt的初始感知器规则相当简单，可以通过以下步骤进行总结：附言：

将权重初始化为0或小的随机数。

对于每个培训样本x（i），执行以下步骤：

计算输出值y\_。

更新权重。

这里，输出值是我们前面定义的单位阶跃函数预测的类标签，权重向量w中每个权重wj的同时更新可以更正式地写为：

WJ：=WJ+∆WJ。

用于更新权重wj的∆wj的值由感知器学习规则计算：

∆wj=η（y（i）−y\_（i））x（ji）

[21]

式中，η为学习率（常数在0.0和1.0之间），y（i）为第i个训练样本的真实班级标签，y\_（i）为预测班级标签。需要注意的是，权重向量中的所有权重都会同时更新，这意味着在所有权重∆wj更新之前，我们不会重新计算y\_（i）。具体来说，对于二维数据集，我们将编写如下更新：

∆w0=η（y（i）−输出（i））

∆w1=η（y（i）−输出（i））x1（i）

∆W2=η（y（i）−输出（i））x2（i）

在用python实现感知器规则之前，让我们做一个简单的思考实验来说明这个学习规则是多么的简单。在感知器正确预测类标签的两个场景中，权重保持不变：

∆wj=η（−1（i）−1（i））x（ji）=0

∆wj=η（1（i）−1（i））x（ji）=0

然而，在预测错误的情况下，权重分别被推向正目标类或负目标类的方向：

∆wj=η（1（i）−1（i））x（ji）=η（2）x（ji）

∆wj=η（−1（i）−1（i））x（ji）=η（−2）x（ji）

为了更好地理解乘法因子x（ji），让我们再举一个简单的例子，其中：

y\_（ji）=+1，y（i）=-1，η=1

[22]

假设x（ji）=0.5，我们将这个样本误分类为-1。在这种情况下，我们将相应的权重增加1，以便下次我们遇到该样本时，激活x（ji）=w（ji）将更为正值，因此更可能高于单位阶跃函数的阈值，将样本分类为+1：∆w（ji）=（1（i）−1（i）。）0.5（i）=（2）0.5（i）=1

权重更新与x（ji）的值成比例。例如，如果我们有另一个样本x（ji）=2被错误地分类为-1，我们会将决策边界扩大一个更大的范围，以便下次正确地分类该样本：∆wj=（1（i）−1（i））2（i）=（2）2（i）=4

需要注意的是，只有当两个类是线性可分的并且学习率足够小时，才能保证感知器的收敛性。如果两个类不能用线性决策边界分开，我们可以设置训练数据集（epoch）上的最大通过次数和/或允许错误分类次数的阈值，否则感知器将永远不会停止更新权重：

【23】

现在，在我们进入下一节的实现之前，让我们用一个简单的图来概括我们刚刚学到的内容，这个图说明了感知器的一般概念：

上图说明了感知器如何接收样本x的输入，并将其与权重w结合，以计算净输入。净投入

然后传递给激活函数（这里是单元步骤函数），该函数生成二进制输出-1或+1——样本的预测类标签。在学习阶段，该输出用于计算预测的误差并更新权重。

在python中实现感知器学习算法

在前一节中，我们学习了Rosenblatt的感知器规则是如何工作的；现在让我们继续用python实现它，并将它应用到我们在第1章中介绍的IRIS数据集，使计算机能够从数据中学习。我们将采用面向对象的方法将感知器接口定义为python类，它允许我们初始化新的感知器对象，这些对象可以通过fit方法从数据中学习，并通过单独的预测方法进行预测。作为一种惯例，我们向属性添加下划线，这些属性不是在对象初始化时创建的，而是通过调用对象的其他方法（例如self.w\_u）来创建的。

【24】

如果您还不熟悉python的科学库或需要更新，请参阅以下资源：

麻木：

熊猫：

Matplotlib:

此外，为了更好地遵循代码示例，我建议您从packt网站下载ipython笔记本。有关ipython笔记本的一般介绍，请访问

导入numpy作为np类感知器（对象）：“”感知器分类器。

参数

-------eta:浮动

学习率（介于0.0和1.0之间）n\_Iter:int通过训练数据集。

属性

-------W\_uu1d数组

安装后的重量。

错误：列出每个时期的错误分类数量。

“定义初始化”（self，eta=0.01，n\_Iter=10）：

self.eta=eta self.n\_iter=n\_iter

def fit（self，x，y）：“调整训练数据。

参数

---------

x：类似数组，shape=[n\_samples，n\_features]

训练向量，其中n\_samples是样本数，并且

[25]

n\_Features是功能的数量。

y:数组状，shape=[n\_samples]目标值。

返回------self:object

“”“

self.w\_=np.zeros（1+x.shape[1]）self.errors=[]

对于“范围内”（self.n\_iter）：

误差＝0为XI，目标为zip（x，y）：更新＝SELF.ETA\*（目标-自我。预测（XI））Surviv.Wy[1：] += Update \*XI Soad。WY[ 0 ] +=更新错误+INT（更新）！=0.0）self.errors\_.append（errors）返回self

def net\_输入（self，x）：

“calculate net input”“返回np.dot（x，self.w\_[1:）+self.w\_[0]

def预测（self，x）：

“单元步骤后返回类标签”返回np.where（self.net\_input（x）>=0.0，1，-1）

使用这个感知器实现，我们现在可以用一个给定的学习速率eta和n iter初始化新的感知器对象，这是一个时间段（通过训练集）。通过fit方法，我们将self.w中的权重初始化为零向量m+1，其中m表示数据集中的维数（特征），其中我们为零权重（即阈值）添加1。

一维数组的numpy索引与使用方括号（[]）表示法的python列表类似。对于二维数组，第一个索引器引用行数，第二个索引器引用列数。例如，我们将使用x[2，3]来选择二维数组x的第三行和第四列。

[26]

权重初始化后，fit方法循环训练集中的所有单个样本，并根据上一节讨论的感知器学习规则更新权重。类标签是通过预测方法预测的，在fit方法中也被称为预测权重更新的类标签，但是predict也可以在我们拟合模型后用来预测新数据的类标签。此外，我们还收集了self.errors列表中每个时期错误分类的次数，以便稍后分析我们的感知器在培训期间的表现。网络输入法中使用的np.dot函数只计算矢量点积wt x。

不使用numpy通过a.dot（b）或np.dot（a，b）计算两个数组a和b之间的矢量点积，

我们还可以通过sum在纯python中执行计算（[j\*j表示i，j表示zip（a，b））。然而，其优势在于

在经典的python循环结构上使用numpy是因为它的算术运算是矢量化的。矢量化意味着元素算术运算会自动应用于数组中的所有元素。通过将我们的算术运算定义为数组上的一系列指令，而不是一次为每个元素执行一组操作，我们可以更好地利用我们的现代CPU体系结构，并支持单指令、多数据（SIMD）。此外，numpy使用高度优化的线性代数库，如basic

用C或FORTRAN编写的线性代数子程序（BLAS）和线性代数包（LAPACK）。最后，numpy还允许我们使用线性代数的基础知识，如向量和矩阵点积，以更紧凑和直观的方式编写代码。

在虹膜数据集上训练感知器模型

为了测试感知器的实现，我们将从IRIS数据集中加载两个flower类setosa和versicolor。虽然感知器规则不局限于二维，但为了可视化的目的，我们只考虑了两个特征：萼片长度和花瓣长度。此外，出于实际的原因，我们只选择了两个花类：狗尾草和花色。然而，感知器算法可以扩展到多类分类，例如，通过一对一技术。

【27】

one-vs.-all（ova），有时也称为one-vs.-rest（ovr），是一种将二进制分类器扩展到多类问题的技术。使用ova，我们可以为每个类训练一个分类器，其中特定类被视为正类，所有其他类的样本被视为负类。如果我们要对一个新的数据样本进行分类，我们将使用我们的φ（z）分类器，其中n是类标签的数量，并将具有最高置信度的类标签分配给特定的样本。对于感知器，我们将使用ova选择与最大绝对净输入值相关联的类标签。

首先，我们将使用pandas库将IRIS数据集直接从UCI机器学习库加载到一个数据帧对象中，并通过tail方法打印最后五行，以检查数据是否正确加载：

>>>将熊猫作为PD导入

>>>df=pd.read\_csv（'https://archive.ics.uci.edu/ml/'

…'机器学习数据库/iris/iris.data'，header=none）

>>>df.tail（）。

接下来，我们分别提取对应于50朵鸢尾花和50朵鸢尾花的前100个类标签，并将这些类标签转换为两个整数类标签1（鸢尾花）和-1（狗尾花），我们将它们分配给一个向量y，其中熊猫数据帧yi的values方法ELDS相应的numpy表示。同样，我们提取100个训练样本的第一个特征列（sepal length）和第三个特征列（patal length），并将它们分配给特征矩阵x，我们可以通过二维散点图来可视化：

>>>导入matplotlib.pyplot为plt

>>>导入numpy为np

>>>Y=df.iloc[0:100，4].值

【28】

>>>Y=NP.其中（Y=‘Iris Setosa’，-1，1）

>>>X=df.iloc[0:100，[0，2]].值

>>>PLT.散射（X[：50，0]，X[：50，1]，

……color='red'，marker='o'，label='setosa'）

>>>PLT.散射（X[50:100，0]，X[50:100，1]，

……color='blue'，marker='x'，label='versicolor'）

>>>plt.xlabel（'花瓣长度'）

>>>plt.ylabel（'sepal length'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

在执行上述代码示例之后，我们现在应该看到以下散点图：

现在是时候在我们刚刚提取的虹膜数据子集上训练我们的感知器算法了。此外，我们还将绘制每个历代的错误分类错误，以检查算法是否收敛，并找到一个分隔两个虹膜花类的决策边界：

>>>ppn=感知器（eta=0.1，n\_Iter=10）

>>>ppn.fit（x，y）>>>plt.plot（range（1，len（ppn.errors\_）+1），ppn.errors\_u，

[29]

……marker='o'）

>>>plt.xlabel（'epochs'）

>>>plt.ylabel（'错误分类数'）

>>>plt.show（）。

在执行上述代码之后，我们应该看到错误分类错误与周期数的关系图，如下所示：

正如我们在前面的情节中看到的，我们的感知器在第六个时代之后已经收敛，现在应该能够对训练样本进行完美的分类。让我们实现一个小的方便函数来可视化二维数据集的决策边界：从matplotlib.colors导入listedcolormap

def plot\_decision\_regions（x，y，分类器，分辨率=0.02）：

#设置标记生成器和颜色映射标记=（'s'、'x'、'o'、'^'、'v'）

颜色=（'red'、'blue'、'lightgreen'、'gray'、'cyan'）

[30]

cmap=listedColormap（颜色[：len（np.unique（y））]）

#绘制决策面x1\_min，x1\_max=x[：，0].min（）-1，x[：，0].max（）+1 x2\_min，x2\_max=x[：，1].min（）-1，x[：，1].max（）+1 xx1，xx2=np.meshgrid（np.arange（x1\_min，x1\_max，resolution），np.arange（x2\_min，x2\_max，resolution））z=0.predictt（np.array（[xx1.ravel（），xx2.ravel（））.t）z=z.remote（xx1.shape）plt.contourf（xx1，xx2，z，alpha=0.4，cmap=cmap）plt.xlim（xx1.min（），xx1.max（））plt.ylim（xx2.min（），xx2.max（））

#枚举（np.unique（y））中IDX、CL的绘图类样本：plt.scatter（x=x[y==cl，0]，y=x[y==cl，1]，alpha=0.8，c=cmap（idx），marker=markers[idx]，label=cl）

首先，我们定义一些颜色和标记，并通过listedcolormap从颜色列表中创建颜色映射。然后，我们确定这两个特征的最小值和最大值，并使用这些特征向量通过numpy meshgrid函数创建一对网格数组xx1和xx2。由于我们在两个特征维上训练了感知分类器，因此我们需要将网格数组展平，并创建一个与虹膜训练子集具有相同列数的矩阵，以便使用预测方法预测相应网格点的类标签Z。在将预测类标签Z重塑为与XX1和XX2尺寸相同的网格后，我们现在可以通过Matplotlib的Contourf函数绘制轮廓图，该函数将网格数组中每个预测类的不同决策区域映射为不同的颜色：

>>>绘制决策区域（x，y，classifier=ppn）

>>>plt.xlabel（“sepal length[cm]”）

>>>plt.ylabel（'花瓣长度[cm]'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

[31]

在执行前面的代码示例之后，我们现在应该看到决策区域的图，如下图所示：

正如我们在前面的图中看到的那样，感知器学习了一个决策边界，能够对虹膜训练子集中的所有花样本进行完美分类。

虽然感知器对这两类鸢尾花进行了很好的分类，但收敛性是感知器最大的问题之一。Frank Rosenblatt在数学上证明了如果两个类可以用一个线性超平面分开，则感知器学习规则收敛。然而，如果类不能被这样一个线性决策边界完美地分隔开，那么权重将永远不会停止更新，除非我们设置一个最大的epoch数。

[32]

自适应线性神经元与学习的收敛性

在本节中，我们将介绍另一种类型的单层神经网络：自适应线性神经元（adaline）。Adaline在Frank Rosenblatt的感知算法发表几年后，由Bernard Widrow和他的博士生Tedd Hoff出版，可以认为是对后者的改进（B.Widrow等人自适应“adaline”神经元使用化学“memistors”。数字

技术报告1553-2。斯坦福电子。实验室。加利福尼亚州斯坦福，1960年10月）。Adaline算法特别有趣，因为它说明了定义和最小化成本函数的关键概念，这将为理解更先进的分类机器学习算法（如逻辑回归和支持向量机）奠定基础。以及我们将在未来章节中讨论的回归模型。

Adaline规则（也称为widrow-hoff规则）和Rosenblatt的感知器之间的关键区别在于，权重是基于线性激活函数而不是像感知器中那样的单位阶跃函数来更新的。在Adaline中，线性激活函数φ（z）只是网络输入的同一函数，因此φ（wt x）=wt x。

虽然线性激活函数用于学习权重，但是量化器（类似于我们以前看到的单位阶跃函数）随后可用于预测类标签，如下图所示：

[33]

如果我们将前面的图与前面看到的感知器算法的示例进行比较，区别在于我们知道使用线性激活函数的连续值输出来计算模型错误和更新权重，而不是使用二进制类标签。

利用梯度下降最小化成本函数

监督机器学习算法的一个关键组成部分是定义一个在学习过程中要优化的目标函数。这个目标函数通常是我们想要最小化的成本函数。在Adaline的情况下，我们可以定义成本函数j来学习权重的和

计算结果和真类标签之间的平方误差（SSE）

J=I（Y（I）−φ（Z（I）））2.

添加这个术语只是为了我们的方便；它将使推导渐变更容易，正如我们将在下面的段落中看到的。这种连续线性激活函数的主要优点是与单位阶跃函数相比，成本函数变得可微。这个成本函数的另一个优点是它是凸的；因此，我们可以使用一个简单而强大的优化算法，称为梯度下降，来找到最小化成本函数的权重，以便对IRIS数据集中的样本进行分类。

如下图所示，我们可以将梯度下降背后的原理描述为爬下山，直到达到本地或全球最低成本。在每次迭代中，我们都会从梯度中迈出一步，其中阶跃大小由学习速率的值以及梯度的斜率决定：

[34]

使用梯度下降，我们现在可以通过远离成本函数j（w）的梯度j（w）来更新权重：

W：=W+∆W

这里，重量变化∆w定义为负梯度乘以学习速率η：

∆w=−η∆j（w）

.

为了计算成本函数的梯度，我们需要计算成本函数对每个权重的偏导数wj wj=−∑i（y（i）−φ（z（i）））x（ji）

因此，我们可以将权重wj的更新写为：∆wj=−ηwjj=µ∑i（y（i）−φ（z（i）））x（ji）：因为我们同时更新所有权重，所以我们的adaline学习规则变为w：=w+∆w。

[35]

对于那些熟悉微积分的人，SSE成本函数关于j加权的偏导数可以得到如下：

w j j=w j 12∑i（y（i）−φ（z（i）））2

=∑（y（i）−φ（z（i）））w j y（i）−∑i（w（ji）x（ji））

我

=-∑（Y（I）−φ（Z（I）））x

我

虽然Adaline学习规则看起来与感知器规则相同，但是z（i）=wt x（i）的φ（z（i））是实数，而不是整数类标签。此外，权重更新是基于训练集中的所有样本计算的（而不是在每个样本之后递增更新权重），这就是为什么这种方法也被称为“批”梯度下降。

在Python中实现自适应线性神经元

由于感知器规则和adaline非常相似，我们将采用前面定义的感知器实现，并更改拟合方法，以便通过梯度下降最小化成本函数来更新权重：

类adalinegd（object）：“”自适应线性神经元分类器。

参数

-----------

[36]

ETA：浮动

学习率（介于0.0和1.0之间）n\_iter:int

传递训练数据集。

属性

-------W\_uu1d数组

安装后的重量。

错误：列出每个时期的错误分类数量。

“定义初始化”（self，eta=0.01，n\_Iter=50）：

self.eta=eta self.n\_iter=n\_iter

def fit（self，x，y）：“调整训练数据。

参数

---------

x：类似数组，shape=[n\_samples，n\_features]

训练向量，

其中n\_samples是样本数，n\_features是特征数。y:数组状，shape=[n\_samples]目标值。

返回------self:object

“”“

self.w\_=np.zeros（1+x.shape[1]）self.cost=[]

对于i i n range（self.n\_iter）：output=self.net\_input（x）errors=（y-output）self.w\_[1:+=self.eta\*x.t.dot（errors）self.w\_[0]+=self.eta\*errors.sum（）

【37】

cost=（错误\*\*2）.sum（）/2.0 self.cost\_u.append（cost）返回self

def net\_输入（self，x）：

“calculate net input”“返回np.dot（x，self.w\_[1:）+self.w\_[0]

DEF激活（自我，X）：

“计算线性激活”返回self.net输入（x）

def预测（self，x）：

“单元步骤后返回类标签”返回np.where（self.activation（x）>=0.0，1，-1）

与感知器一样，在评估每个单独的训练样本后，我们不更新权重，而是通过self.eta\*errors.sum（）为零权重计算整个训练数据集的梯度，通过self.eta\*x.t.dot（错误）为权重1到m，其中x.t.dot（错误）i是特征矩阵和误差向量之间的矩阵向量乘法。与之前的感知器实现类似，我们将成本值收集到一个列表self.cost中，以检查算法在训练后是否收敛。

执行矩阵向量乘法类似于计算向量点积，其中矩阵中的每一行都被视为单行向量。这种矢量化方法代表了一种更紧凑的表示法，使用numpy可以得到更有效的计算结果。例如：

7\_

1 2 3\_

4 5 6\_×9 8\_=14×77++2 5×88++36×99=122 50。

\_

\_

【38】

在实际应用中，往往需要进行一些实验，以找到一个良好的学习率η的最优收敛。因此，让我们从两个不同的学习率η=0.1和η=0.0001开始，并绘制成本函数与epoch数的关系，以了解Adaline实现如何从培训数据中学习。

学习速率η以及周期数是所谓的感知器超参数和自适应学习算法。在第四章，建立良好的训练集数据预处理，我们将研究不同的技术，自动找到不同超参数的值，以获得分类模型的最佳性能。

现在，让我们根据两种不同的学习速率的时间段数来绘制成本：

>>>图，ax=plt.子图（nrows=1，ncols=2，figsize=（8，4））

>>>ADA1=ADalineGD（n坿iter=10，eta=0.01）.fit（x，y）

>>>ax[0].绘图（范围（1，长度（ada1.cost\_uux）+1），

……np.log10（ada1.成本），marker='o'）

>>>ax[0].设置xlabel（'epochs'）

>>>ax[0].设置“ylabel（'log（sum squared error）'”

>>>ax[0].设置\_title（'adaline-学习率0.01'）

>>>ADA2=ADalineGD（n iter=10，eta=0.0001）.fit（x，y）

>>>ax[1].绘图（范围（1，长度（ada2.cost\_uux）+1），

……ada2.成本，marker='o'）

>>>ax[1].设置xlabel（'epochs'）

>>>ax[1].设置“ylabel”（sum-squared-error'）

>>>ax[1].设置\_title（'adaline-学习率0.0001'）

>>>plt.show（）。

[39]

正如我们在接下来的成本函数图中看到的，我们遇到了两种不同类型的问题。左图显示了如果我们选择一个太大的学习率而不是最小化成本函数，那么会发生什么情况，因为我们超过了全局最小值，在每个时期的错误都会变得更大：

虽然我们可以看到，当我们看正确的图时，成本会降低，但是所选的学习率η=0.0001非常小，算法需要大量的时间来收敛。下图说明了如何更改特定权重参数的值以最小化成本函数j（左子图）。右边的子图说明了如果我们选择的学习率太大，我们会超出全局最小值：

[40]

我们将在本书中遇到的许多机器学习算法都需要某种功能缩放以获得最佳性能，我们将在第3章（使用SciKit Learn的机器学习分类器教程）中更详细地讨论这一点。梯度下降算法是许多受益于特征缩放的算法之一。在这里，我们将使用一种称为标准化的特征缩放方法，它使我们的数据具有标准正态分布的特性。每个特征的平均值以0为中心，特征列的标准偏差为1。例如，为了标准化j th特性，我们只需要从每个训练样本中减去样本平均值μj，然后除以其标准偏差σj:x j−μj。

x′j=σj

这里x j是一个向量，由所有训练样本n的j th特征值组成。

使用numpy方法mean和std很容易实现标准化：

>>>x\_std=np.副本（x）

>>>X\_-std[：，0]=（X[：，0]-X[：，0].mean（））/X[：，0].std（）

>>>x\_std[：，1]=（x[：，1]-x[：，1].mean（））/x[：，1].std（）

标准化后，我们将再次对adaline进行培训，并看到它现在使用学习率η=0.01收敛：

>>>Ada=Adalinegd（n\_Iter=15，eta=0.01）

>>>Ada.Fit（X\_Std，Y）

>>>绘制决策区域（x\_std，y，classifier=ada）

>>>plt.title（'adaline-渐变下降'）

>>>plt.xlabel（'sepal length[标准化]）

>>>plt.ylabel（'花瓣长度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

>>>plt.plot（range（1，len（ada.cost\_ux）+1），ada.cost\_uuu，marker='o'）

>>>plt.xlabel（'epochs'）

>>>plt.ylabel（'平方和误差'）

>>>plt.show（）。

【41】

在执行上述代码之后，我们应该看到决策区域的图以及成本下降的图，如下图所示：

正如我们在前面的图中看到的，Adaline现在在使用学习率η=0.01对标准化特性进行培训后收敛。但是，请注意，即使所有样本都正确分类，SSE仍为非零。

大规模机器学习与随机梯度下降

在前一节中，我们学习了如何通过向从整个训练集计算的梯度的相反方向迈出一步来最小化成本函数；这就是为什么这种方法有时也被称为批梯度下降。现在假设我们有一个拥有数百万个数据点的非常大的数据集，这在许多机器学习应用程序中并不少见。在这种情况下，运行批处理梯度下降在计算上可能非常昂贵，因为我们每次朝着全局最小值迈出一步时都需要重新评估整个训练数据集。

批梯度下降算法的一个流行的替代方法是随机梯度下降，有时也称为迭代或在线梯度下降。而不是根据所有样本x（i）的累积误差之和更新权重：

∆w =η∑（y（i）−φ（z（i）））x（i）、

我

我们递增更新每个训练样本的权重：η（y（i）−φ（z（i）））x（i）

【42】

虽然随机梯度下降可以被看作是梯度下降的近似值，但由于权重更新的频繁性，其收敛速度通常更快。由于每个梯度都是基于一个训练实例计算的，因此误差面比梯度下降时噪声更大，这也具有随机梯度下降更容易避开浅局部极小的优点。为了通过随机梯度下降获得准确的结果，重要的是以随机的顺序将数据呈现出来，这就是为什么我们要对每个历代的训练集进行洗牌以防止循环的原因。

在随机梯度下降实现中，固定学习率η经常被随时间减少的自适应学习率c 1所取代。

例如，[numbero f iterations]+c2，其中c1和c2是常量。

请注意，随机梯度下降不会达到全局最小值，而是一个非常接近它的区域。通过使用一个自适应学习速率，我们可以进一步退火到一个更好的全局最小值。

随机梯度下降的另一个优点是我们可以利用它进行在线学习。在在线学习中，随着新的培训数据的到来，我们的模型将进行实时培训。如果我们正在积累大量数据（例如，典型Web应用程序中的客户数据），那么这一点尤其有用。使用在线学习，系统可以立即适应变化，如果问题中存在存储空间，更新模型后可以丢弃培训数据。

批梯度下降和随机梯度下降之间的折衷是所谓的小批量学习。小批量学习可以理解为将批量梯度下降应用于较小的训练数据子集，例如一次50个样本。相对于批梯度下降的优势在于，由于权重更新更频繁，通过小批更快地实现收敛。此外，小批量学习允许我们用向量运算代替随机梯度下降（SGD）中训练样本的for循环，从而进一步提高学习算法的计算效率。

【43】

由于我们已经使用梯度下降实现了Adaline学习规则，我们只需要做一些调整来修改学习算法，通过随机梯度下降更新权重。在fit方法中，我们现在将在每个训练样本之后更新权重。此外，我们还将实现一个附加的部分拟合方法，它不会重新初始化权重，用于在线学习。为了检验算法在训练后是否收敛，我们将计算出每个时期训练样本的平均成本。此外，在优化成本函数时，我们还将添加一个选项，在每个历元之前对训练数据进行无序排列，以避免循环；通过随机状态参数，我们允许指定一个随机种子来保持一致性：从numpy.random导入种子

类adalinesgd（object）：“”自适应线性神经元分类器。

参数

-------eta:浮动

学习率（介于0.0和1.0之间）n\_Iter:int通过训练数据集。

属性

-------W\_uu1d数组

安装后的重量。

错误列表

每个时代的错误分类数目。shuffle:bool（默认值：true）如果为true，则在每个历元中对训练数据进行无序处理，以防止循环。random\_state:int（默认值：none）设置随机状态，用于解组和初始化权重。

“”“

def uu init\_uuuu（self，eta=0.01，n\_Iter=10，shuffle=true，random\_state=none）：

self.eta=eta self.n\_iter=n\_iter self.w\_initialized=false self.shuffle=shuffle

【44】

如果是随机状态：

seed（random\_state）def fit（self，x，y）：“”匹配训练数据。

参数

---------

x：类似数组，shape=[n\_samples，n\_features]

训练向量，其中n\_samples是样本数，n\_features是特征数。

y:数组状，shape=[n\_samples]目标值。

返回------self:object

“”“

self.\_初始化范围内i的“权重”（x.shape[1]）self.cost\_uu=[]（self.n\_iter）：如果self.shuffle：

x，y＝Syfle（x，y）成本＝[]为zip，目标在zip（x，y）：成本。追加（自我。

def-partial-fit（self，x，y）：“在不重新初始化权重的情况下调整训练数据”如果不是self.w“初始化：

如果y.ravel（），self.\u初始化权重（x.shape[1]）。shape[0]>1：

对于XI，zip（x，y）中的目标：

I. UpUpDead权重（XI，Target）其他：

self。\_update\_weights（x，y）返回self def\_shuffle（self，x，y）：

【45】

“shuffle training data”“”r=np.random.permutation（len（y））返回x[r]，y[r]def初始化权重（self，m）：

“初始化权重为零”“自我”WA= NP.0（1 +M）Self.Wx初始化=真DEF-UpDATE权重（自我，XI，目标）：

“应用AdLink学习规则来更新权重”“输出= No.IdIn输入（XI）错误＝（目标输出）自我。W[ 1：] = Self.ETA\*XI.DOT（错误）自。WY[ 0 ] += Self.ETA\*错误成本＝0.5 \*错误\*\* 2返回成本DEF NETLION输入（自我，X）：

“”“计算净输入”“”

返回np.dot（x，self.w\_[1:）+self.w\_[0]

DEF激活（自我，X）：

“计算线性激活”返回self.net输入（x）

def预测（self，x）：

“单元步骤后返回类标签”返回np.where（self.activation（x）>=0.0，1，-1）

我们现在在adalinesgd分类器中使用的\_shuffle方法的工作原理如下：通过numpy.random中的置换函数，我们生成一个0到100范围内唯一数字的随机序列。然后，这些数字可以用作索引来无序排列我们的特征矩阵和类标签向量。

然后，我们可以使用fit方法来训练adalinesgd分类器，并使用我们的绘图决策区域来绘制我们的培训结果：

>>>Ada=Adalinesgd（n\_Iter=15，eta=0.01，Random\_State=1）

>>>Ada.Fit（X\_Std，Y）

>>>绘制决策区域（x\_std，y，classifier=ada）

>>>plt.title（'adaline-随机梯度下降'）

>>>plt.xlabel（'sepal length[标准化]）

>>>plt.ylabel（'花瓣长度[标准化]）

【46】

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

>>>plt.plot（range（1，len（ada.cost\_ux）+1），ada.cost\_uuu，marker='o'）

>>>plt.xlabel（'epochs'）

>>>plt.ylabel（平均成本）

>>>plt.show（）。

我们通过执行前面的代码示例得到的两个图如下图所示：

如我们所见，平均成本下降很快，15个时期后的最终决策边界看起来类似于Adaline的批梯度下降。例如，如果我们想更新我们的模型，在一个带有流数据的在线学习场景中，我们可以简单地对单个样本调用部分拟合方法，例如，ada.partial-fit（x\_-std[0，：]，y[0]）。

总结

在这一章中，我们对线性分类器的基本概念有了很好的理解。在实现了感知器之后，我们看到了如何通过梯度下降的矢量化实现有效地训练自适应线性神经元，以及通过随机梯度下降的在线学习。既然我们已经了解了如何在python中实现简单的分类器，那么我们准备好进入下一章，在这里我们将使用python scikit学习机器学习库来访问Academ中常用的更高级和更强大的现成机器学习分类器。工业界也是如此。

【47】

机器学习之旅

使用SciKit学习的分类器

在本章中，我们将介绍一些在学术界和行业中常用的流行和强大的机器学习算法。在了解几种分类监督学习算法之间的差异的同时，我们还将直观地了解它们各自的优缺点。此外，我们将在SciKit学习库中迈出第一步，该库提供了一个用户友好的界面，可以高效、高效地使用这些算法。

我们将在本章中了解的主题如下：

常用分类算法概念介绍

使用SciKit学习机器学习库

选择机器学习算法时要问的问题

选择分类算法

为一个特定的问题任务选择一个合适的分类算法需要实践：每个算法都有自己的特性，并且基于某些假设。重述“没有免费午餐”定理：没有一个分类器在所有可能的场景中都能发挥最好的作用。在实践中，总是建议您比较至少几种不同的学习算法的性能，为特定的问题选择最佳的模型；这些可能在特性或样本的数量、数据集中的噪声量以及类是否为lin方面有所不同。早期可分离与否。

【49】

最后，分类器的性能，计算能力和预测能力，在很大程度上取决于可供学习的基础数据。培训机器学习算法涉及的五个主要步骤可概括如下：

功能选择。

选择性能指标。

选择分类器和优化算法。

评估模型的性能。

调整算法。

由于本书的方法是逐步建立机器学习知识，因此我们将主要关注本章中不同算法的主要概念，并重新探讨诸如特征选择和预处理、性能度量和超参数调整等主题，以获得更多的详细信息。这本书后面的讨论失败了。

SciKit学习的第一步

在第2章，分类的机器学习算法培训中，您学习了两种相关的分类学习算法：感知器规则和adaline，我们自己在python中实现。现在我们来看看Scikit Learn API，它结合了一个用户友好的界面和一些分类算法的高度优化的实现。然而，SciKit学习库不仅提供了各种各样的学习算法，而且提供了许多方便的功能来预处理数据和微调和评估我们的模型。我们将更详细地讨论这一点，以及第4章“构建良好的训练集——数据预处理”和第5章“通过降维压缩数据”中的基本概念。

通过Scikit学习训练感知器

为了开始使用SciKit学习库，我们将训练类似于第2章“训练机器学习分类算法”中所实现的感知器模型。为了简单起见，我们将在以下各节中使用已经熟悉的IRIS数据集。很方便，IRIS数据集已经通过Scikit Learn提供，因为它是一个简单但流行的数据集，经常用于测试和试验算法。此外，我们将仅使用虹膜花数据集中的两个功能进行可视化。

[50]

我们将把150个花样本的花瓣长度和花瓣宽度分配给特征矩阵x，并将花物种的相应类别标签分配给向量y：

>>>从sklearn导入数据集

>>>导入numpy为np

>>>iris=datasets.load\_iris（）。

>>>X=iris.data[：，[2，3]]

>>>Y=iris.target

如果我们执行np.unique（y），返回存储在iris中的不同类标签。目标，我们将看到虹膜花类名，鸢尾属，花色鸢尾和弗吉尼亚鸢尾，已经存储为整数（0，1，2），这是为许多机器学习库的最佳性能推荐的。

为了评估一个经过训练的模型对未知数据的性能，我们将进一步将数据集拆分为单独的训练和测试数据集。在第5章的后面，通过降维压缩数据，我们将更详细地讨论模型评估的最佳实践：

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>X\_train，X\_test，Y\_train，Y\_test=列车\_test\_split（

……x，y，测试尺寸=0.3，随机状态=0）

使用Scikit Learn交叉验证模块的train\_test\_split函数，我们将x和y数组随机分成30%的测试数据（45个样本）和70%的培训数据（105个样本）。

许多机器学习和优化算法也需要特征缩放以获得最佳性能，正如我们从第2章“训练机器学习算法进行分类”中的梯度下降示例中所记得的那样。在这里，我们将使用Scikit Learn预处理模块中的Standardscaler类标准化功能：

>>>来自sklearn.preprocessing import standardscaler

>>>SC=标准缩放器（）

>>>SC.fit（X\_系列）

>>>x\_train\_std=sc.transform（x\_train）

>>>x\_test\_std=sc.transform（x\_test）

【51】

使用前面的代码，我们从预处理模块加载了Standardscaler类，并初始化了一个新的Standardscaler对象，我们将该对象分配给变量sc。Standardscaler使用fit方法估计了每个特征dim的参数μ（样本平均值）和σ（标准偏差）。从培训数据中得出的结论。然后，通过调用转换方法，我们使用这些估计参数μ和σ标准化训练数据。请注意，我们使用相同的缩放参数来标准化测试集，以便训练和测试数据集中的值可以相互比较。

标准化训练数据后，我们现在可以训练感知模型。SciKit中的大多数算法默认情况下都支持通过一对一（One-vs.-Rest，OVR）方法进行多类分类，这允许我们一次将三个花类都输入感知器。代码如下：

>>>从sklearn.linear\_model导入感知器

>>>ppn=感知器（n\_Iter=40，eta0=0.1，Random\_State=0）

>>>ppn.fit（x\_train\_std，y\_train）

SciKit学习界面提醒我们第2章“训练机器分类算法”中的感知器实现：从线性模型模块加载感知器类后，我们初始化了一个新的感知器对象，并通过fit方法对模型进行训练。在这里，模型参数eta0相当于我们在自己的感知器实现中使用的学习率eta，参数n iter定义了时间段数（通过训练集）。正如我们在第2章中所记得的，训练机器学习算法进行分类，找到合适的学习速度需要一些实验。如果学习率太大，算法将超过全局成本最小值。如果学习速度太小，算法需要更多的时间段才能收敛，这会使学习速度变慢，特别是对于大数据集。此外，我们还使用了随机状态参数来再现每个时期后训练数据集的初始配置。

在Scikit Learn中训练了一个模型之后，我们可以通过预测方法进行预测，就像在第2章“训练机器学习分类算法”中我们自己的感知器实现中一样。代码如下：

>>>y\_pred=ppn.predict（x\_test\_std）

>>>print（'错误分类的样本数：%d%”（y\_test！=y\_pred）.sum（））

错误分类样品：4个

【52】

在执行前面的代码时，我们看到感知器在

45个花样本。因此，测试数据集上的错误分类误差为0.089或8.9%（4/45≈0.089）。

许多机器学习实践者报告模型的分类精度，而不是错误分类错误，其简单计算如下：

1-错误分类错误=0.911或91.1%。

SciKit Learn还实现了各种不同的性能指标，这些指标可通过Metrics模块获得。例如，我们可以计算测试集上感知器的分类精度，如下所示：

>>>来自sklearn.metrics import accuracy\_score

>>>print（'准确度：%.2f'%准确度得分（y\_test，y\_pred））

0.91美元

这里，y\_测试是真正的类标签，y\_pred是我们之前预测的类标签。

注意，我们根据本章中的测试集评估模型的性能。在第5章中，通过降维压缩数据，您将了解一些有用的技术，包括图形分析（如学习曲线），以检测和防止过度拟合。过度拟合意味着该模型很好地捕获了训练数据中的模式，但未能很好地概括为未发现的数据。

最后，我们可以使用第2章“训练机器学习分类算法”中的“绘图决策区域”功能，绘制新训练的感知器模型的决策区域，并可视化它如何区分不同的花样本。但是，让我们添加一个小修改，通过小圆圈突出显示测试数据集中的样本：

从matplotlib.colors导入listedcolormap导入matplotlib.pyplot as plt

def plot\_decision\_regions（x，y，classifier，test\_idx=none，resolution=0.02）：35;设置标记生成器和颜色图

【53】

标记=（'s'、'x'、'o'、'^'、'v'）

颜色=（'red'、'blue'、'lightgreen'、'gray'、'cyan'）cmap=listedColormap（颜色[：len（np.unique（y））]）

#绘制决策图面

x1\_min，x1\_max=x[：，0].min（）-1，x[：，0].max（）+1 x2\_min，x2\_max=x[：，1].min（）-1，x[：，1].max（）+1 xx1，xx2=np.meshgrid（np.arrange（x1\_min，x1\_max，resolution），np.arrange（x2\_min，x2\_max，resolution））z=classifier.predict（np.array（[xx1.ravel（），xx2.ravel（））.t）

Z=Z.整形（XX1.形状）

plt.contourf（xx1，xx2，z，alpha=0.4，cmap=cmap）plt.xlim（xx1.min（），xx1.max（））plt.ylim（xx2.min（），xx2.max（））

#绘制所有样本

x\_test，y\_test=x[test\_idx，：]，y[test\_idx]用于idx，cl用于枚举（np.unique（y））：plt.scatter（x=x[y==cl，0]，y=x[y==cl，1]，alpha=0.8，c=cmap（idx），marker=markers[idx]，label=cl）

#如果测试IDX，突出显示测试样本：

x\_test，y\_test=x[测试\_idx，：]，y[测试\_idx]plt.散射（x\_test[：，0]，x\_test[：，1]，c='，alpha=1.0，lineidth=1，marker='o'，s=55，label='test set'）

通过对plot\_decision\_regions函数（在前面的代码中突出显示）所做的微小修改，我们现在可以指定要在结果图上标记的样本的索引。代码如下：

>>>X\_combined\_std=np.vstack（（X\_train\_std，X\_test\_std））。

>>>Y\_combined=NP.hsTACK（（Y\_train，Y\_test））。

……Y=Y\_组合，

……分类器=ppn，

……test\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

【54】

正如我们在结果图中看到的，三个花类不能通过线性决策边界完全分开：

我们记得在第2章“分类训练机器学习算法”中的讨论中，感知器算法从不收敛于不完全线性可分离的数据集，这就是为什么在实践中通常不建议使用感知器算法的原因。在下面的部分中，我们将看到更强大的线性分类器，它收敛到成本最低，即使类不是完全线性可分离的。

感知器以及其他SciKit学习函数和类都有额外的参数，为了清晰起见，我们省略了这些参数。您可以使用python中的帮助函数（例如，帮助（perceptron））或通过浏览优秀的scikit了解在线文档了解更多关于这些参数的信息，网址为

[55]

通过逻辑回归建模类概率

虽然感知器规则为机器学习分类算法提供了一个很好且容易理解的介绍，但是它最大的缺点是如果类不是完全线性可分的，它永远不会收敛。前一节中的分类任务就是这样一个场景的例子。直观地说，我们可以认为原因是权重一直在不断更新，因为每个时代都存在至少一个错误分类的样本。当然，你可以改变学习速度，增加时间段的数量，但是要注意，感知器永远不会收敛于这个数据集。为了更好地利用我们的时间，我们现在将看看另一个简单但更强大的线性和二进制分类问题的算法：逻辑回归。请注意，尽管逻辑回归的名称不同，但它是一个分类模型，而不是回归模型。

逻辑回归直觉和条件概率

逻辑回归是一种分类模型，很容易实现，但在线性可分类上表现很好。它是工业上应用最广泛的分类算法之一。与感知器和Adaline相似，本章中的逻辑回归模型也是一个二元分类的线性模型，可以通过OVR技术扩展到多类分类。

为了解释逻辑回归作为概率模型背后的思想，我们首先介绍优势比，它是有利于特定事件的优势。几率P

比率可以写为（1-p），其中p代表正事件的概率。术语“阳性事件”并不一定意味着好的，而是指我们想要预测的事件，例如，患者患某种疾病的概率；我们可以将阳性事件视为类标签y=1。然后我们可以进一步定义logit函数，它只是优势比（对数优势）的对数：

logit（p）=log（1−pp）

[第56条]

logit函数接受0到1范围内的输入值，并将其转换为整个实数范围内的值，我们可以使用这些值来表示特征值和对数赔率之间的线性关系：

n逻辑（p（y=1\_x））=w0x0+w1x1+wmxm=∑wmxm=wt x

i=0

这里，p（y=1\_x）是特定样本属于1类的条件概率，给定其特征x。

现在我们真正感兴趣的是预测某个样本属于某一类的概率，这是逻辑函数的逆形式。它也被称为logistic函数，由于其S形的特点，有时简称为sigmoid函数。

φ（z）=1+1e−z

这里，z是净输入，即权重和样本特征的线性组合，可以计算为z=wtx=w0+w1x1++wmxm。

现在，让我们简单地为范围-7到7中的一些值绘制sigmoid函数，看看它是什么样子的：

>>>导入matplotlib.pyplot为plt

>>>导入numpy as np>>def sigmoid（z）：

……返回1.0/（1.0+np.exp（-z））

>>>Z=np.arange（-7，7，0.1）

>>>phi\_z=乙状结肠（Z）

>>>plt.绘图（Z，phi\_Z）

>>>plt.axvline（0.0，color='k'）

>>>plt.axhspan（0.0，1.0，facecolor='1.0'，alpha=1.0，ls='dotted'）

>>>plt.axhline（y=0.5，ls='dotted'，color='k'）

>>>请点击（[0.0，0.5，1.0]）

>>>plt.ylim（-0.1，1.1）

>>>plt.xlabel（'z'）

>>>plt.ylabel（'$\phi（z）$'）

>>>请显示（））

[第57条]

由于执行了前面的代码示例，我们现在应该看到S形（乙状）曲线：

我们可以看到，如果z趋于无穷大（z→∞），则φ（z）接近1，因为对于z的大值，e−z变得非常小。同样，由于分母越来越大，因此φ（z）趋于0。因此，我们得出结论，这个sigmoid函数将实数值作为输入，并将它们转换为范围[0，1]内的值，截距为φ（z）=0.5。

为了建立逻辑回归模型的一些直观性，我们可以将其与我们在第2章“分类的训练机器学习算法”中以前的Adaline实现联系起来。在Adaline中，我们使用单位函数φ（z）=z作为激活函数。在逻辑回归中，这个激活函数简单地成为我们前面定义的乙状结肠函数，如下图所示：

【58】

然后将sigmoid函数的输出解释为属于1类φ（z）=p（y=1 x；w）的特定样本的概率，给定其特征x由权重w参数化。例如，如果我们计算特定花样本的φ（z）=0.8，则意味着此样本的概率是鸢尾花的百分之八十。同样，这种花是一种鸢尾花的概率可以计算为p（y=0\_x；w）=1−p（y=0\_x；w）=0.2或20%。预测的概率可以简单地通过量化器（单位阶跃函数）转换成二进制结果：

Y\_=1，如果φ（z）≥0.5 0，否则

如果我们看前面的乙状结肠图，这相当于：

如果z≥0.0，y\_=1

0否则

事实上，有许多应用程序，我们不仅对预测类标签感兴趣，而且对估计类成员概率特别有用。例如，在天气预报中，逻辑回归不仅用来预测某一天是否会下雨，还用来报告下雨的可能性。同样，logistic回归可以用来预测患者在特定症状下患特定疾病的可能性，这也是logistic回归在医学领域广泛流行的原因。

学习物流成本函数的权重

您了解了如何使用逻辑回归模型来预测概率和类标签。现在，让我们简单地讨论一下模型的参数，例如权重w。在上一章中，我们定义了平方和误差成本函数：

j（w）=∑1（φ（z（i））−y（i））2 i 2

[59]

为了学习Adaline分类模型的权重w，我们将其最小化。为了解释我们如何得出逻辑回归的成本函数，我们首先定义在构建逻辑回归模型时要最大化的可能性L，假设数据集中的单个样本彼此独立。公式如下：

L（W）=P（Y X；W）=N P（Y（I）X（I）；W）=（φ（Z（I）））Y（I）（1−φ（Z（I）））1−Y（I）

I=1

实际上，该方程的（自然）对数更容易最大化，称为对数似然函数：

L（W）=Log L（W）=∑N Log（φ（Z（I））＋（1−Y（I））Log（1−φ（Z（I）））

I=1

首先，应用对数函数可以降低数值下溢的可能性，如果可能性很小，则可能发生下溢。其次，我们可以将因子的积转化为因子的总和，这使得通过加法技巧更容易获得这个函数的导数，正如您可能记得的微积分。

现在我们可以使用一个优化算法，比如梯度上升来最大化这个对数似然函数。或者，让我们将日志可能性重写为成本函数j，使用梯度下降将其最小化，如第2章“培训机器学习分类算法：

j（w）=∑n−log（φ（z（i）））−（1−y（i））log（1−φ（z（i）））

I=1

为了更好地理解这个成本函数，让我们看看我们为一个示例实例计算的成本：

j（φ（z），y；w）=−y对数（φ（z））−（1−y）对数（1−φ（z））。

[60]

从前面的方程可以看出，如果y=0，第一项变为零；如果y=1，第二项变为零：

j（φ（z），y；w）=−log（φ（z）），如果y=1

−log（1−φ（z）），如果y=0

下图说明了不同值φ（z）下单个样本分类的成本：

如果我们正确地预测一个样本属于1类，我们可以看到成本接近0（纯蓝线）。同样，我们可以在y轴上看到，如果我们正确预测y=0（虚线），成本也接近0。然而，如果预测是错误的，那么成本将走向无穷大。道德上讲，我们用越来越大的成本来惩罚错误的预测。

【61】

用SciKit学习训练逻辑回归模型

如果我们自己来实现逻辑回归，我们可以简单地用新的成本函数j（w）=-∑y（i）log（φ（z（i）））+（1−y（i））log（1−φ（z（i））替换第2章“培训机器学习算法”中的Adaline实现中的成本函数j。

我

这将计算每个时期对所有训练样本进行分类的成本，最后我们将得到一个有效的逻辑回归模型。但是，由于Scikit Learn实现了一个高度优化的逻辑回归版本，也支持现成的多类设置，因此我们将跳过该实现，并使用sklearn.linear\_model.logistic regression类以及熟悉的fit方法在展台上对模型进行培训。心花训练数据集：

>>>来自sklearn.linear\_model import logisticregression

>>>lr=logisticregression（c=1000.0，random\_state=0）

>>>左后配合（X\_-train\_-std，Y\_-train）

>>>绘制决策区域（x\_组合标准，

……Y\_组合，分类器=lr，

……test\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

在训练数据上拟合模型后，我们绘制了决策区域、训练样本和测试样本，如下所示：

【62】

看看前面我们用来训练逻辑回归模型的代码，您可能会想，“这个神秘的参数c是什么？”

我们将在一秒钟内讨论这个问题，但是让我们在下一小节中首先简要地讨论过拟合和正则化的概念。

此外，我们还可以通过预测概率法来预测样本的类成员概率。例如，我们可以预测第一个刚毛鸢尾样本的概率：

>>>lr.Predict\_Proba（x\_Test\_Std[0，：]）

这将返回以下数组：

数组（[[0.000，0.063，0.937]]）

【63】

前面的数组告诉我们，模型预测样本属于维吉尼亚鸢尾类的概率为93.7%，而样本属于花色鸢尾类的概率为6.3%。

我们可以证明，通过梯度下降的逻辑回归中的权重更新确实等于我们在第2章，分类训练机器学习算法中使用的Adaline方程。让我们从计算对数似然函数相对于第j个权的偏导数开始：

L Y

wjφ（z）1−φ（z）wj

在继续之前，我们先计算一下sigmoid函数的偏导数：

wφ（z）=z 1+1e−z=（1+1e−z）2 e−z=1+1e−z 1−1+1e−z

J

=φ（z）（1−φ（z））。

（z）=φ（z）（1−φ（z））在我们的第一个方程中得到

现在我们可以恢复wjφ，如下：

yφ1−（1−y）1（z）w jφ（z）

（z）1−φ

=yφ（1z）−（1−y）1−φ1（z）φ（z）（1−φ（z））w j z

=（Y（1−φ（Z））−（1−Y）φ（Z））xj

=（y−φ（z））xj

[64]

记住，目标是找到最大化日志可能性的权重，以便我们对每个权重执行以下更新：

wj：=wj+η∑n（y（i）−φ（z（i）））x（i）

I=1

由于我们同时更新了所有权重，因此我们可以编写如下一般更新规则：

W：=W+∆W

我们定义∆w如下：

∆w=ηl（w）

由于最大化对数似然等于最小化我们前面定义的成本函数j，我们可以编写如下的梯度下降更新规则：

∆wj=−η\_ wjj=η∑i=n 1（y（i）−φ（z（i））x（i））。

W：=W+∆W，∆W=−ηJ（W）

这相当于第2章“训练机器学习分类算法”中ADALINE中的梯度下降规则。

通过正则化处理过拟合

过度拟合是机器学习中的一个常见问题，模型在训练数据上表现良好，但不能很好地概括为未发现的数据（测试数据）。如果一个模型存在过度拟合的问题，我们也会说这个模型有很高的方差，这可能是由于有太多的参数导致一个模型在给定基础数据的情况下过于复杂。同样，我们的模型也可能会出现未满配（高偏差），这意味着我们的模型不够复杂，无法很好地捕获训练数据中的模式，因此也会在未看到的数据上表现不佳。

[65]

虽然目前我们只遇到了用于分类的线性模型，但通过使用更复杂的非线性决策边界，可以更好地说明过度拟合和欠拟合问题，如下图所示：

找到一个好的偏差-方差权衡的方法是通过正则化调整模型的复杂性。正则化是处理共线性（特征间相关性高）、滤除数据噪声、最终防止拟合过度的一种非常有用的方法。正则化背后的概念是引入额外的信息（偏差）来惩罚极端参数权重。最常见的正则化形式是所谓的l2正则化（有时也称为l2收缩或重量衰减），可写如下：

λ2 w 2=λ2∑jm=1 w2 j

【66】

这里，λ是所谓的正则化参数。

规范化是特征缩放（如标准化）很重要的另一个原因。为了使规范化正常工作，我们需要确保我们所有的特性都具有可比的规模。

为了应用正则化，我们只需要将正则化项添加到我们为逻辑回归定义的成本函数中，以缩小权重：

n（）（）（）λw 2

J

通过正则化参数λ，我们可以在保持较小权重的同时控制训练数据的拟合程度。通过增加λ的值，我们增加了正则化强度。

Scikit Learn中为LogisticRegression类实现的参数C来自支持向量机中的约定，这将是下一节的主题。c与正则化参数λ直接相关，后者为其逆：

C=

因此，我们可以将逻辑回归的正则化成本函数改写为：

J C N（）（）1 W 2

【67】

因此，减小反正则化参数c的值意味着我们正在增加正则化强度，我们可以通过绘制两个权重系数的l2正则化路径来可视化：

>>>权重，参数=[]，[]>>对于np.arange中的c（-5，5）：

……lr=逻辑回归（c=10\*\*c，随机状态=0）

……Lr.fit（x\_train\_std，y\_train）

……weights.append（lr.coef\_[1]）

……附加参数（10\*\*c）

>>>权重=np.数组（权重）

>>>plt.plot（参数，权重[：，0]，

……label='花瓣长度'）

>>>plt.plot（参数，权重[：，1]，linestyle=--'，

……label='花瓣宽度'）

>>>plt.ylabel（'重量系数'）

>>>plt.xlabel（'c'）

>>>Plt.Legend（loc='左上角'）

>>>plt.xscale（'log'）

>>>plt.show（）。

通过执行前面的代码，我们为反正则化参数c拟合了10个具有不同值的逻辑回归模型。为了便于说明，我们只收集了类2与所有分类器的权重系数。记住，我们正在使用OVR技术进行多类分类。

正如我们在结果图中看到的，如果我们减少参数c，即如果我们增加正则化强度，权重系数就会收缩：

【68】

由于对单个分类算法的深入报道超出了本书的范围，我热情地推荐Scott Menard博士的逻辑回归：从入门到高级概念和应用、SAGE出版物，到想了解更多逻辑回归的读者。

支持向量机的最大边缘分类

另一种功能强大、应用广泛的学习算法是支持向量机（SVM），它可以看作是感知器的扩展。使用感知器算法，我们最小化错误分类错误。然而，在支持向量机中，我们的优化目标是使利润最大化。边界定义为分离超平面（决策边界）与最接近该超平面的训练样本之间的距离，即所谓的支持向量。如下图所示：

【69】

最大利润直觉

具有较大利润的决策边界的基本原理是，它们往往具有较低的泛化误差，而具有较小利润的模型更容易过度拟合。为了获得边际最大化的直觉，让我们仔细看看那些与决策边界平行的正、负超平面，可以表示为：

w0+wt xpos=1（1）w0+wt xneg=−1（2）

如果我们将这两个线性方程（1）和（2）相减，我们得到：

？重量（xpos−xneg）=2

我们可以通过向量w的长度将其规范化，定义如下：

W=∑MJ=1 W2 J

因此，我们得出如下方程：

wt（xpos−xneg）=2

上述方程的左侧可以解释为正超平面和负超平面之间的距离，即我们想要最大化的所谓边界。

【70】

现在，支持向量机的目标函数变成了这个裕度2的最大化。

通过在正确分类样品的约束下最大化w，可写为：w0+wt x（i）≥1，如果y（i）=1 w0+wt x（i）<−1，如果y（i）=1

这两个方程基本上说，所有的负样本都应该落在负超平面的一边，而所有的正样本都应该落在正超平面的后面。这也可以更简洁地写如下：

y（i）（w0+wt x（i））≥1 i

然而，在实际中，将倒数项12w 2最小化更容易，这可以通过二次规划来解决。然而，关于二次规划的详细讨论超出了本书的范围，但是如果您感兴趣，您可以在Vladimir Vapnik的《统计学习理论的本质》、Springer Science&Business Media或Chris J.C.Burges的Exc中进一步了解支持向量机（SVM）。Ellent在模式识别支持向量机教程中的解释（数据挖掘和知识发现，2（2）：121-1671998）。

用松弛变量处理非线性可分情形

尽管我们不想深入到边缘分类背后更复杂的数学概念中，让我们简单地提到松弛变量ξ。1995年，弗拉基米尔·瓦普尼克（Vladimir Vapnik）提出了该分类法，并导致了所谓的软利润分类法。引入松弛变量ξ的动机是，需要放宽非线性可分离数据的线性约束，以便在适当的成本惩罚下，在存在错误分类的情况下，使优化收敛。

【71】

将正值松弛变量简单地添加到线性约束：wt x（i）≥1，如果y（i）=1−ξ（i）wt x（i）<−1，如果y（i）=1+ξ（i）

因此，要最小化的新目标（受制于前面的限制）变成：

12瓦2+c∑iξ（i）

使用变量c，我们就可以控制误分类的惩罚。C的大值对应于大的误差惩罚，而如果我们选择C的小值，我们对错误分类的错误就不那么严格了。然后我们可以使用参数C来控制保证金的宽度，从而调整偏差方差权衡，如下图所示自然资源：

这个概念与正则化有关，我们之前在正则化回归中讨论过，在正则化回归中，增加c的值会增加偏差并降低模型的方差。

【72】

既然我们了解了线性支持向量机背后的基本概念，那么让我们训练一个支持向量机模型来对虹膜数据集中的不同花进行分类：

>>>从sklearn.svm导入svc

>>>SVM=SVC（kernel='linear'，C=1.0，random\_state=0）

>>>SVM.fit（X\_train\_std，Y\_train）

>>>绘制决策区域（x\_组合标准，

……Y\_组合，分类器=SVM，

……test\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

在执行上述代码示例后，显示的SVM决策区域如下图所示：

【73】

逻辑回归与支持向量机

在实际的分类任务中，线性逻辑回归和线性支持向量机常常产生非常相似的结果。逻辑回归试图最大化训练数据的条件相似性，这使得它比SVM更容易出现异常值。SVM主要关注离决策边界最近的点（支持向量）。另一方面，逻辑回归的优点是它是一个更简单的模型，可以更容易地实现。此外，逻辑回归模型可以很容易地更新，这在处理流数据时很有吸引力。

SciKit学习中的替代实现

感知器和Logistic回归类，我们使用在以前的章节通过SCIKIT学习利用LyBLULL库，这是一个高度优化的C/C++库在台湾大学（）。类似地，我们用来训练SVM的SVC类使用LIbSVM，它是专门用于SVMs（）的C/C++类库。

与本机python实现相比，使用liblinear和libsvm的优势在于，它们允许对大量线性分类器进行非常快速的培训。然而，有时我们的数据集太大，无法装入计算机内存。因此，SciKit Learn还通过sgdclassifier类提供了其他实现，该类还支持通过部分匹配方法进行在线学习。sgdclassifier类背后的概念类似于我们在第2章中实现的随机梯度算法，即Adaline的培训机学习分类算法。我们可以用以下默认参数初始化感知器的随机梯度下降版本、逻辑回归和支持向量机：

>>>来自sklearn.linear\_model import sgdclassifier

>>>ppn=sgdclassifier（loss='perceptron'）

>>>lr=sgdclassifier（loss='log'）

>>>SVM=SGDClassifier（损耗='铰链'）

【74】

用核支持向量机求解非线性问题

支持向量机在机器学习实践者中很受欢迎的另一个原因是，它们可以很容易地进行内核化以解决非线性分类问题。在讨论内核SVM背后的主要概念之前，让我们先定义并创建一个示例数据集，看看这样一个非线性分类问题可能看起来如何。

使用以下代码，我们将创建一个简单的数据集，该数据集的形式为xor gate，使用numpy的logical\_xor函数，其中100个样本将被分配给类标签1，100个样本将分别被分配给类标签1：

>>>np.random.seed（0）

>>>x\_xor=np.random.randn（200，2）

>>>y\_xor=np.逻辑\_xor（x\_xor[：，0]>0，x\_xor[：，1]>0）

>>>Y诳or=np.其中（Y诳or，1，-1）

>>>散点图（x\_xor[y\_xor==1，0]，x\_xor[y\_xor==1，1]，

……c='b'，marker='x'，label='1'）

>>>散点图（x\_xor[y\_xor==-1，0]，x\_xor[y\_xor==-1，1]，

……c='r'，marker='s'，label='1'）

>>>plt.ylim（-3.0）

>>>Plt.Legend（）。

>>>plt.show（）。

执行完代码后，我们将得到一个带有随机噪声的XOR数据集，如下图所示：

[75]

显然，我们不能很好地通过前面讨论的线性逻辑回归或线性支持向量机模型，使用线性超平面作为决策边界，从正类和负类中分离样本。

处理这种线性不可分割数据的核心方法的基本思想是创建原始特征的非线性组合，通过映射函数φ（）将它们投影到更高的空间中，在该空间中，原始特征可以线性分离。如下图所示，我们可以将二维数据集转换为一个新的三维特征空间，在该空间中，类通过以下投影可分离：

φ（x1，x2）=（z1，z2，z3）=（x1，x2，x12+x22）

这允许我们通过一个线性超平面将图中显示的两个类分离，如果我们将其投影回原始特征空间，该超平面将成为非线性决策边界：

【76】

利用核技巧在高维空间中寻找分离超平面

为了利用支持向量机解决非线性问题，我们通过映射函数φ（）将训练数据转化为高维特征空间，并训练线性支持向量机模型对新特征空间中的数据进行分类。然后，我们可以使用相同的映射函数φ（）来转换新的、未看到的数据，并使用线性SVM模型对其进行分类。

然而，这种映射方法的一个问题是，新特性的构造在计算上非常昂贵，特别是在处理高维数据时。这就是所谓的内核技巧发挥作用的地方。虽然我们没有详细讨论如何解决二次规划任务来训练支持向量机，但实际上我们所需要的是用φ（x（i））tφ（x（j））替换点积x（i）t x（j）。为了节省计算两点间点积的昂贵步骤，我们定义了一个所谓的核函数：k（x（i），x（j））=φ（x（i））tφ（x（j））。

最广泛使用的内核之一是径向基函数内核（RBF内核）或高斯内核：

K（x（i），x（j））=exp−x（i）2-σx2（j）2

 

这通常被简化为：

K（x（i），x（j））=exp（−γx（i）-x（j）2）

这里，γ=是要优化的自由参数。

粗略地说，术语内核可以解释为一对样本之间的相似函数。减号将距离测量值转化为相似度得分，由于指数项的原因，所得相似度得分将介于1（完全相似的样本）和0（非常不同的样本）之间。

【77】

既然我们定义了内核技巧背后的大局，那么让我们看看是否可以训练一个内核SVM，它能够绘制一个将XOR数据分离得很好的非线性决策边界。这里，我们只需使用scikit中的svc类，了解之前导入的内容，并将参数kernel='linear'替换为kernel='rbf'：

>>>SVM=SVC（kernel='rbf'，random\_state=0，gamma=0.10，c=10.0）

>>>Svm.Fit（X\_xor，Y\_xor）

>>>绘制决策区域（x\_xor，y\_xor，classifier=svm）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

正如我们在结果图中看到的，内核SVM比较好地分离了XOR数据：

γ参数，我们设置为γ=0.1，可以理解为高斯球的截止参数。如果我们增加γ值，我们会增加训练样本的影响或影响范围，从而导致更软的决策边界。为了更好地理解γ，让我们将RBF内核SVM应用于我们的虹膜花数据集：

>>>svm=svc（kernel='rbf'，random\_state=0，gamma=0.2，c=1.0）

>>>SVM.fit（X\_train\_std，Y\_train）

>>>绘制决策区域（x\_组合标准，

【78】

……Y\_组合，分类器=SVM，

……test\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

由于我们选择了相对较小的γ值，因此RBF核支持向量机模型的决策边界相对较软，如下图所示：

现在让我们增加γ的值，观察对决策边界的影响：

>>>svm=svc（kernel='rbf'，random\_state=0，gamma=100.0，c=1.0）

>>>SVM.fit（X\_train\_std，Y\_train）

>>>绘制决策区域（x\_组合标准，

……Y\_组合，分类器=SVM，

……test\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

【79】

在结果图中，我们现在可以看到，使用相对较大的γ值，类0和1周围的决策边界要紧得多：

虽然模型非常适合训练数据集，但这种分类器对未知数据的泛化误差很高，说明γ的优化在控制过度拟合中也起着重要作用。

决策树学习

如果我们关心可解释性，决策树分类器是有吸引力的模型。就像决策树的名字所暗示的那样，我们可以把这个模型看作是通过在一系列问题的基础上做出决策来分解我们的数据。

[80]

让我们考虑下面的示例，其中我们使用决策树来决定特定日期的活动：

基于我们训练集的特点，决策树模型学习了一系列问题来推断样本的类标签。虽然前面的图说明了基于分类变量的决策树的概念，但是如果我们的特性是这样的，同样的概念也适用。如果我们的特性是实数（如IRIS数据集中的实数），这也可以工作。例如，我们可以简单地沿着sepal宽度特征轴定义一个截止值，并问一个二进制问题“sepal宽度≥2.8cm？”

使用决策算法，我们从树根开始，将导致最大信息增益（ig）的特征上的数据进行拆分，这将在下一节中进行更详细的解释。在一个迭代过程中，我们可以在每个子节点上重复这个拆分过程，直到叶子是纯的。这意味着每个节点上的样本都属于同一类。在实践中，这会导致非常深的树有许多节点，这很容易导致过度拟合。因此，我们通常希望通过设置树的最大深度限制来修剪树。

【81】

最大限度地提高信息收益——获得最大的回报

为了在信息量最大的地方分割节点，我们需要通过树学习算法定义一个要优化的目标函数。在这里，我们的目标函数是最大化每个拆分处的信息获取，我们定义如下：

（）（）∑m n j i（dj）ig dp，f=i dp−n p

J=1

这里，f是进行拆分的特性，dp和dj是父节点和jth子节点的数据集，i是我们的杂质度量，n p是父节点的样本总数，n j是jth子节点的样本数。如我们所见，信息增益只是父节点的杂质和子节点杂质之和之间的差异，子节点的杂质越低，信息增益越大。然而，为了简化和减少组合搜索空间，大多数库（包括scikit-learn）实现了二进制决策树。这意味着每个父节点被拆分为两个子节点，分别是dleft和dright：

（）左（左）右（右）右（右）

ig dp，a=i dp−n p

目前，二元决策树中常用的三种杂质度量或分裂标准是基尼指数（IG）、熵（IH）和分类误差（IE）。让我们从所有非空类（p（i\_t）≠0）的熵定义开始：

Ih（t）=-∑c p（i\_t）对数p（i\_t）

我

[82]

这里，p（i\_t）是特定节点t中属于C类的样本的比例。因此，如果一个节点上的所有样本都属于同一类，则熵为0；如果我们具有均匀的类分布，则熵最大。例如，在二进制类设置中，如果p（i=1\_t）=1或p（i=0\_t）=0，则熵为0。如果类均匀分布，p（i=1\_t）=0.5，p（i=0\_t）=0.5，则熵为1。因此，我们可以说，熵准则试图最大化树中的相互信息。

直观地说，基尼指数可以被理解为最小化错误分类概率的标准：

C C C

ig（t）=∑p（i\_t）（−p（i\_t））=1−∑p（i\_t）2

I=1 I=1

与熵类似，如果类是完全混合的，那么基尼指数是最大的，例如，在二进制类设置（c=2）中：

C

1−0.52=0.5

然而，在实践中，基尼指数和熵通常会产生非常相似的结果，而且通常不值得花太多时间用不同的杂质标准来评估树木，而不是尝试不同的修剪截止。

另一个杂质测量是分类误差：

Ie=1−最大p（i t）

【83】

这是修剪的一个有用标准，但不推荐用于生长决策树，因为它对节点类概率的变化不太敏感。我们可以通过查看下图所示的两个可能的拆分场景来说明这一点：

我们从父节点dp处的一个数据集dp开始，该数据集dp由来自类1的40个样本和来自类2的40个样本组成，我们分别拆分为两个数据集dleft和dright。在方案A和方案B中，使用分类误差作为分割标准的信息增益将相同（ige=0.25）：

ie（dp）=1−0.5=0.5

A:ie（dLeft）=1−=0.25

A:IE（干燥）=1−=0.25

A:ige=0.5−84 0.25−8 4 0.25=0.25

b:ie（dLeft）=1−64=1 3

b:ie（干燥）=1−1=0

B:ige=0.5−86×13−0=0.25

【84】

然而，基尼指数倾向于情景B（igg=0.16）中的分割，而不是情景A（igg=0.125），这确实更纯粹：

ig（dp）=1−（0.52+0.52）=0.5

A:ig（dleft）=1−\_\_\_34 2+14 2 8 3=0.375

A:ig（干燥）=1−2+34 2=8 3=0.375

A:ig=0.5−84 0.375−8 4 0.375=0.125

b:ig（dleft）=1−62 2+64 2=9 4=0.4

B:ig（干燥）=1−（12+02）=0

B:igg=0.5−0.4−0=0.16

同样地，熵准则将倾向于方案B（igh=0.19）而不是方案A（igh=0.31）：

IH（dp）=-（0.5 log2（0.5）+0.5 log2（0.5））=1

A:ih（dleft）=-34 log2 34+14 log2 14=0.81

【85】

A:ih（干燥）=-14+34 log2 34=0.81

A:igh=1−8 4 0.81−8 4 0.81=0.19

B:ih（dleft）=-62 log2+62+64 log2+64=0.92

b:ih（光照）=0

B:igh=1−0.92−0=0.31

为了更直观地比较前面讨论过的三种不同的杂质标准，让我们绘制1类概率范围[0，1]的杂质指数。注意，我们还将添加一个按比例缩放的熵（熵/2），以观察到基尼指数是熵和分类误差之间的中间度量。代码如下：

>>>导入matplotlib.pyplot为plt

>>>导入numpy as np>>def gini（p）：

……返回（P）\*（1-（P））+（1-P）\*（1-（1-P））>>>定义熵（P）：

……返回-p\*np.log2（p）-（1-p）\*np.log2（（1-p））>>>定义错误（p）：

……返回1-np.max（[p，1-p]）

>>>X=np.arange（0.0，1.0，0.01）

>>>ent=[熵（p）如果p！=0，否则p in x不适用]

>>>sc\_ent=[e\*0.5 if e else none for e in ent]

>>>错误=[X中I的错误（I）]

>>>图=PLT.Figure（）

>>>AX=plt.subflot（111）

>>>对于i、lab、ls、c、in-zip（[ent、sc-ent、gini（x），err]，

……[“熵”，“熵（按比例）”，

…'基尼杂质'，

【86】

…'错误分类错误“]，

……['-'、'-'、'--'、'-.']，

……[黑'，'浅灰色'，…'红色“、”绿色“、”青色“]）：

……line=ax.图（x，i，label=lab，

……linestyle=ls，lw=2，color=c）

>>>AX.图例（loc='upper center'，bbox\_to\_anchor=（0.5，1.15），

……ncol=3，fancybox=true，shadow=false）

>>>ax.axhline（y=0.5，lineidth=1，color='k'，linestyle=--'）

>>>ax.axhline（y=1.0，lineidth=1，color='k'，linestyle=--'）

>>>plt.ylim（[0，1.1]）

>>>plt.xlabel（'p（i=1）'）

>>>plt.ylabel（'杂质指数'）

>>>plt.show（）。

上述代码示例生成的绘图如下：

【87】

构建决策树

决策树可以通过将特征空间划分为矩形来构建复杂的决策边界。但是，由于决策树越深，决策边界越复杂，很容易导致过度拟合。使用Scikit Learn，我们现在将使用熵作为杂质的标准来训练一个最大深度为3的决策树。尽管出于可视化的目的可能需要特征缩放，但请注意，对于决策树算法来说，特征缩放不是一项要求。代码如下：

>>>从sklearn.tree导入decisiontreeclassifier

>>>tree=decisionTreeClassifier（criteria='entropy'，

……最大深度=3，随机状态=0）

>>>Tree.Fit（X轴、Y轴）

>>>X\_combined=np.vstack（（X\_train，X\_test））。

>>>Y\_combined=NP.hsTACK（（Y\_train，Y\_test））。

>>>绘制决策区域（x\_组合，y\_组合，

……分类器=树，测试范围（105150）

>>>plt.xlabel（'花瓣长度[cm]'）

>>>plt.ylabel（'花瓣宽度[cm]'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

在执行上述代码示例后，我们得到决策树的典型轴平行决策边界：

【88】

Scikit Learn中的一个很好的特性是，它允许我们在训练后将决策树导出为.dot文件，我们可以使用graphviz程序可视化该文件。此程序在Linux、Windows和Mac OS X上免费提供并受其支持。

首先，我们使用树子模块中的export\_graphviz函数通过scikit learn创建.dot文件，如下所示：

>>>来自sklearn.tree import export\_graphviz

>>>导出graphviz（tree，

……out\_file='tree.dot'，

……特征\_name=['花瓣长度'，'花瓣宽度']）

在我们的计算机上安装了graphviz之后，我们可以从保存tree.dot文件的位置的命令行中执行以下命令，将tree.dot文件转换为png文件：

>点-tpng tree.dot-o tree.png

查看我们通过graphviz创建的决策树图，我们现在可以很好地跟踪决策树从我们的培训数据集中确定的拆分。我们从根部的105个样本开始，将其分成两个子节点，每个子节点有34个和71个样本，每个样本使用切掉值小于等于0.75 cm的花瓣。在第一次分割之后，我们可以看到左子节点已经是纯的，并且只包含来自IRIS SETOSA类（熵=0）的样本。然后，右边的进一步分割用于从花色鸢尾和弗吉尼亚鸢尾类中分离样品。

[89]

通过随机森林把弱者和强者结合起来

随机森林由于其良好的分类性能、可扩展性和易用性，在过去十年中在机器学习的应用中获得了巨大的普及。从直观上讲，一个随机森林可以看作是一组决策树。集成学习的思想是将弱学习者结合起来，建立一个更健壮的模型，一个强学习者，具有更好的泛化误差，不易过度拟合。随机森林算法可以概括为四个简单步骤：

随机抽取一个大小为n的引导样本（从替换的训练集中随机选择n个样本）。

从引导程序示例中生长决策树。在每个节点：

随机选择D个特征，不替换。

使用根据目标函数提供最佳拆分的功能拆分节点，例如，通过最大化信息增益。

重复步骤1至2 k次。

按每棵树聚合预测，以多数票分配类标签。多数投票将在第7章中更详细地讨论，结合不同的集成学习模式。

当我们训练单个决策树时，步骤2中有一个微小的修改：我们不评估所有特性来确定每个节点的最佳分割，只考虑其中的一个随机子集。

尽管随机森林不能提供与决策树相同水平的可解释性，但随机森林的一大优势在于我们不必太担心如何选择好的超参数值。我们通常不需要修剪随机森林，因为集成模型对于来自单个决策树的噪声非常健壮。我们在实践中真正需要关心的唯一参数是我们为随机森林选择的树数k（步骤3）。通常，树的数量越大，随机森林分类器的性能越好，但代价是增加了计算成本。

[90]

尽管在实践中并不常见，但是我们将在第5章中讨论的可以使用技术优化的随机森林分类器的其他超参数，通过降维压缩数据是引导样本的大小n（步骤1）和S分别为每个分割随机选择（步骤2.1）。通过引导样本的样本大小n，控制随机森林的偏差方差权衡。通过选择较大的N值，我们减少了随机性，从而使森林更容易过度适应。另一方面，我们可以通过为n选择较小的值来降低过度拟合的程度，而牺牲了模型的性能。在大多数实现中，包括SciKit Learn中的RandomForestClassifier实现，引导样本的样本大小被选择为等于原始训练集中的样本数，这通常提供了一个良好的偏差方差权衡。对于每个拆分处的功能数d，我们希望选择一个小于训练集中功能总数的值。SciKit学习和其他实现中使用的一个合理的默认值是d=m，其中m是培训集中的功能数量。

方便的是，我们不必自己从单个决策树构建随机森林分类器；SciKit中已经有了一个实现，我们可以使用：

>>>从sklearn.ensegle导入RandomForestClassifier

>>>forest=randomforestClassifier（criteria='entropy'，

……n\_估计数=10，

……随机状态=1，

……n\_作业=2）

>>>Forest.Fit（X\_火车，Y\_火车）

>>>绘制决策区域（x\_组合，y\_组合，

……分类器=森林，测试范围（105150）

>>>plt.xlabel（'花瓣长度'）

>>>plt.ylabel（'花瓣宽度'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

【91】

执行完上述代码后，我们可以看到随机林中的树集合形成的决策区域，如下图所示：

利用前面的代码，我们通过n\_估计参数从10个决策树中训练了一个随机森林，并使用熵准则作为一个杂质度量来分割节点。虽然我们从一个非常小的训练数据集发展一个非常小的随机森林，但我们使用n\_jobs参数进行演示，这允许我们使用计算机的多个核心（这里是两个）并行化模型训练。

k-最近邻——一种懒惰的学习算法

我们在本章中要讨论的最后一个监督学习算法是k-最近邻分类器（knn），它特别有趣，因为它与我们目前讨论的学习算法有根本不同。

KNN是懒惰学习者的典型例子。之所以称之为懒惰，并不是因为它明显简单，而是因为它不从训练数据中学习区分函数，而是记忆训练数据集。

【92】

参数模型与非参数模型

机器学习算法可以分为参数模型和非参数模型。使用参数模型，我们从训练数据集估计参数，以学习一个函数，该函数可以对新的数据点进行分类，而不再需要原始的训练数据集。参数模型的典型例子有感知器、逻辑回归和线性支持向量机。相比之下，非参数模型不能用一组固定的参数来描述，参数的数量随着训练数据的增加而增加。到目前为止，我们看到的两个非参数模型的例子是决策树分类器/随机森林和内核支持向量机。

KNN属于非参数模型的子类别，称为基于实例的学习。基于实例学习的模型具有记忆训练数据集的特点，而懒惰学习是基于实例学习的一种特殊情况，在学习过程中与零成本相关。

KNN算法本身相当简单，可以通过以下步骤进行总结：

选择K数和距离度量。

找到我们要分类的样本的k最近邻。

通过多数票分配班级标签。

下图说明了新数据点（？）根据五个最近的邻居中的多数票分配三角形类标签。

【93】

根据选择的距离度量，knn算法在训练数据集中找到与我们要分类的点最近（最相似）的k样本。然后，新数据点的类标签由其k个最近邻中的多数票决定。

这种基于内存的方法的主要优点是，当我们收集新的训练数据时，分类器会立即进行自适应。但是，缺点是，在最坏的情况下，分类新样本的计算复杂性随着训练数据集中样本的数量而线性增长，除非数据集的维度（特征）非常少，并且算法是使用有效的数据结构来实现的。结构，如杜松树。J.H.Friedman、J.L.Bentley和R.A.Finkel。在对数期望时间内寻找最佳匹配的算法。ACM数学软件汇刊（TOMS），3（3）：209–2261977年。此外，我们不能丢弃培训样本，因为不涉及培训步骤。因此，如果我们使用大型数据集，存储空间可能会成为一个挑战。

通过执行以下代码，我们现在将使用欧几里得距离度量在Scikit Learn中实现一个KNN模型：

>>>从sklearn.neighbors导入Kneighbors分类器

>>>knn=Kneighbors分类器（n\_neighbors=5，p=2，

……metric='minkowski'）

>>>knn.fit（x\_train\_std，y\_train）

>>>绘制决策区域（x\_Combined\_Std，y\_Combined，

……分类器=knn，测试\_idx=范围（105150）

>>>plt.xlabel（'花瓣长度[标准化]）

>>>plt.ylabel（'花瓣宽度[标准化]）

>>>plt.show（）。

通过在该数据集的KNN模型中指定五个邻居，我们得到一个相对平滑的决策边界，如下图所示：

【94】

正确选择K对于在过量和不足之间找到一个良好的平衡至关重要。我们还必须确保选择适合数据集中特征的距离度量。通常，一个简单的欧几里得距离测量用于实际值的样本，例如，我们虹膜数据集中的花，其特征以厘米为单位测量。但是，如果我们使用欧几里得距离测量，那么标准化数据也很重要，这样每个特征对距离的贡献都是相等的。我们在前面的代码中使用的“minkowski”距离只是欧几里得距离和曼哈顿距离的概括，可以写成如下：

d（x（i），x（i））=p∑xk（i）xk（j）p

K

【95】

如果我们分别将参数p=2或manhatten距离设为p=1，它就变成欧几里得距离。Scikit Learn中提供了许多其他距离度量，可以将其提供给度量参数。它们列在

维度的诅咒

值得一提的是，由于维度的诅咒，knn很容易过度拟合。维数的诅咒描述了一种现象，即随着固定尺寸训练数据集维数的增加，特征空间变得越来越稀疏。直观地说，我们可以想到，即使是最近的邻居，在高维空间中也离得太远，无法给出一个很好的估计。

我们在关于逻辑回归的章节中讨论了正则化的概念，这是避免过度拟合的一种方法。然而，在决策树和KNN等不适用正则化的模型中，我们可以使用特征选择和维数约简技术来避免维数的诅咒。这将在下一章中更详细地讨论。

总结

在本章中，您学习了许多用于处理线性和非线性问题的不同机器算法。我们已经看到，如果我们关心可解释性，决策树尤其具有吸引力。逻辑回归不仅是一个有用的随机梯度下降在线学习模型，而且允许我们预测特定事件的概率。虽然支持向量机是强大的线性模型，可以通过核技巧扩展到非线性问题，但它们有许多参数必须调整才能做出良好的预测。相比之下，随机森林等集成方法不需要太多的参数调整，也不象决策树那样容易过拟合，这使得它成为许多实际问题领域的一个有吸引力的模型。K-最近邻分类器提供了一种通过懒惰学习进行分类的替代方法，它允许我们在没有任何模型训练的情况下进行预测，但具有更昂贵的计算预测步骤。

[96]

然而，比选择适当的学习算法更重要的是我们的培训数据集中的可用数据。没有信息性和歧视性的特征，任何算法都无法做出好的预测。

在下一章中，我们将讨论有关数据预处理、特征选择和降维的重要主题，我们需要建立强大的机器学习模型。在第6章的后面，学习模型评估和超参数调整的最佳实践，我们将看到如何评估和比较模型的性能，并学习有用的技巧来微调不同的算法。

【97】

培养良好的培训

集合–数据预处理

数据的质量和包含的有用信息量是决定机器学习算法学习能力的关键因素。因此，在将数据集输入到学习算法之前，确保对其进行检查和预处理是绝对关键的。在本章中，我们将讨论有助于我们建立良好机器学习模型的基本数据预处理技术。

我们将在本章中讨论的主题如下：

从数据集中删除和输入缺少的值

机器学习算法中分类数据的成形

选择模型构造的相关特征

处理丢失的数据

在实际应用程序中，由于各种原因，我们的示例缺少一个或多个值并不罕见。例如，在数据收集过程中可能会出现错误，某些测量值不适用，特定字段在调查中可能只是空白。我们通常将缺失值视为数据表中的空白空间或占位符字符串，如楠（不是数字）。

[99]

不幸的是，大多数计算工具无法处理这些丢失的值，或者如果我们简单地忽略它们，就会产生不可预测的结果。因此，在进行进一步分析之前，我们必须处理掉这些缺失的值。但在讨论处理缺失值的几种技术之前，让我们先从csv（逗号分隔值）文件创建一个简单的示例数据帧，以便更好地理解问题：

>>>将熊猫作为PD导入

>>>从IO导入Stringio

>>>csv\_data=''a，b，c，d

……1.0、2.0、3.0、4.0

……5.0、6.0、8.0

……0.0,11.0,12.0，“”

>>>如果您使用的是python 2.7，则需要>>>将字符串转换为Unicode：

>>>csv\_data=unicode（csv\_data）

>>>df=pd.read\_csv（stringio（csv\_data））。

>>>测向

甲、乙、丙、丁

1 2 3 4

5 6南8

0 11 12南

使用前面的代码，我们通过read\_csv函数将csv格式的数据读取到熊猫数据帧中，注意到丢失的两个单元格被nan替换。前面代码示例中的Stringio函数仅用于说明目的。它允许我们将分配给csv\_数据的字符串读取到熊猫数据框中，就像它是我们硬盘上的常规csv文件一样。

对于较大的数据帧，手动查找缺少的值可能很麻烦；在这种情况下，我们可以使用isNull方法返回一个具有布尔值的数据帧，该布尔值指示单元格是否包含数值（false）或数据是否丢失（true）。然后，使用sum方法，我们可以返回每列缺少值的数量，如下所示：

>>>df.isNull（）.sum（）。

0

0

[100]

这样，我们可以计算每列缺少的值的数量；在下面的小节中，我们将了解如何处理这些缺少的数据的不同策略。

尽管Scikit Learn是为使用numpy数组而开发的，但有时使用pandas的数据帧对数据进行预处理更为方便。我们总是可以通过values属性访问数据帧的基础numpy数组，然后将其输入SciKit学习估计量：

>>>df.values数组（[[1.，2.，3.，4.]，[5.，6.，NaN，8.]，

[10.，11.，12.，南部]]）

消除缺少值的样本或特征

处理丢失数据的最简单方法之一是从数据集中完全删除相应的特性（列）或示例（行）；可以通过dropna方法轻松删除缺少值的行：

>>>df.dropna（）。

甲、乙、丙、丁

0 1 2 3 4

同样，通过将axis参数设置为1，我们可以删除任何行中至少有一个NaN的列：

>>>df.dropna（轴=1）

A B

1 2个

5 6个

0 11个

Dropna方法支持一些其他参数，这些参数可以派上用场：

#只删除所有列均为NaN的行

>>>df.dropna（how='all'）

#删除至少有4个非NaN值的行

>>>df.dropna（阈值=4）

#只删除NaN出现在特定列中的行（此处为：“C”）。

>>>df.dropna（subset=['c']）

【101】

虽然删除丢失的数据似乎是一种方便的方法，但它也有一定的缺点，例如，我们可能会最终删除太多的样本，这将使可靠的分析成为不可能。或者，如果删除太多的特性列，我们将面临丢失有价值的信息的风险，而我们的分类器需要区分类。在下一节中，我们将讨论处理缺失值最常用的替代方法之一：插值技术。

输入缺失值

通常，删除样本或删除整个特性列是不可行的，因为我们可能会丢失太多有价值的数据。在这种情况下，我们可以使用不同的插值技术来估计数据集中其他训练样本的缺失值。最常见的插值技术之一是均值插补，我们只需将缺失值替换为整个特征列的均值。实现这一点的一个方便方法是使用Scikit Learn中的输入器类，如以下代码所示：

>>>来自sklearn.preprocessing import imputer

>>>IMR=输入（缺少\_值='nan'，strategy='mean'，axis=0）

>>>imr=imr.fit（df）

>>>输入的\_data=imr.transform（df.values）

>>>输入的数据

数组（[[1.，2.，3.，4.]，[5.，6.，3.，8.]，

【10、11、12、4】）

这里，我们用相应的平均值替换每个NaN值，这是为每个特征列单独计算的。如果我们将设置轴=0改为轴=1，我们将计算行的平均值。策略参数的其他选项是中间值或最频繁的，后者用最频繁的值替换丢失的值。这对于输入分类特征值很有用。

了解SciKit学习评估器API

在上一节中，我们使用Scikit Learn中的输入类来输入数据集中缺少的值。输入器类属于Scikit Learn中用于数据转换的所谓Transformer类。这些估计量的两个基本方法是拟合和变换。拟合方法用于从训练数据中学习参数，变换方法使用这些参数对数据进行变换。任何要转换的数据数组都需要具有与用于适应模型的数据数组相同数量的功能。下图说明了如何使用安装在培训数据上的变压器来转换培训数据集以及新的测试数据集：

【102】

我们在第3章中使用的分类器，使用scikit-learn的机器学习分类器教程，属于scikit-learn中的所谓估值器，其API在概念上与transformer类非常相似。估计量有一个预测方法，但也可以有一个转换方法，我们稍后将看到。您可能还记得，当我们训练这些估计量进行分类时，我们还使用fit方法来学习模型的参数。然而，在有监督的学习任务中，我们还提供了用于拟合模型的类标签，该类标签随后可用于通过预测方法对新数据样本进行预测，如下图所示：

【103】

处理分类数据

到目前为止，我们只研究数值。然而，现实数据集包含一个或多个分类特征列并不少见。当我们谈论分类数据时，我们必须进一步区分名义特征和序数特征。序数特征可以理解为可以排序或排序的分类值。例如，T恤的尺寸将是一个有序的特征，因为我们可以定义一个订单xl>l>m。相反，象征性特征并不意味着任何订单，并且，为了继续上一个示例，我们可以将T恤的颜色视为象征性特征，因为通常说来没有意义，f例如，红色比蓝色大。

在我们探索处理此类分类数据的不同技术之前，让我们创建一个新的数据框架来说明问题：

>>>将熊猫作为PD导入

>>>df=pd.dataframe（[

……[“绿色”，“M”，10.1，“1类”]

……[红色'，'L'，13.5，'Class2']

……[蓝色'，'XL'，15.3，'Class1']）

>>>df.columns=['color'、'size'、'price'、'classlabel']

>>>测向

颜色尺寸价格等级标签0绿色M 10.1等级1

红色L 13.5 2级

蓝色XL 15.3 1级

正如我们在前面的输出中看到的，新创建的数据帧包含一个名义特征（颜色）、一个序数特征（大小）和一个数字特征（价格）列。类标签（假设我们为受监督的学习任务创建了一个数据集）存储在最后一列中。我们在本书中讨论的分类学习算法在类标签中不使用顺序信息。

映射顺序特征

为了确保学习算法正确地解释顺序特征，我们需要将分类字符串值转换为整数。不幸的是，没有一个方便的函数可以自动得出我们尺寸特征的标签的正确顺序。因此，我们必须手动定义映射。在下面的简单示例中，假设我们知道特性之间的区别，例如，xl=l+1=m+2。

>>>大小映射={

…'XL”：3，

…'L'：2，

【104】

…'M'：1

>>>df['size']=df['size']映射（大小映射）

>>>测向

颜色尺寸价格等级标签0绿色1 10.1等级1

红色2 13.5 2级

蓝色3 15.3 1级

如果要将整数值转换回原始字符串

在后期，我们可以简单地定义一个反向映射字典in v size mapping=v:k代表k，v代表size mapping.items（），然后可以通过pandas的map方法在转换后的特征列上使用，类似于我们以前使用的size mapping字典。

编码类标签

许多机器学习库要求类标签编码为整数值。尽管SciKit中分类的大多数估计数在内部学习将类标签转换为整数，但将类标签提供为整数数组以避免技术故障被认为是一种良好的实践。为了对类标签进行编码，我们可以使用一种类似于前面讨论的序列特征映射的方法。我们需要记住，类标签不是序数，我们为特定的字符串标签分配的整数并不重要。因此，我们可以简单地枚举从0开始的类标签：

>>>导入numpy为np

>>>class\_mapping=label:idx用于idx，label in

……枚举（np.unique（df['classlabel']）

>>>类映射

'class1'：0，'class2'：1

接下来，我们可以使用映射字典将类标签转换为整数：

>>>df['classlabel']=df['classlabel'].映射（class\_mapping）

>>>测向

颜色大小价格等级标签0绿色1 10.1 0

红色2 13.5 1

蓝色3 15.3 0

【105】

我们可以将映射字典中的键值对反转如下，以将转换后的类标签映射回原始字符串表示形式：

>>>in v\_class\_mapping=v:k代表k，v代表class\_mapping.items（）

>>>df['classlabel']=df['classlabel']映射（inv\_class\_mapping）

>>>测向

颜色尺寸价格等级标签0绿色1 10.1等级1

红色2 13.5 2级

蓝色3 15.3 1级

或者，在SciKit Learn中直接实现了一个方便的Labelencoder类，以实现相同的目标：

>>>来自sklearn.preprocessing import labelencoder

>>>class\_le=labelencoder（）。

>>>Y=class\_le.fit\_transform（df['classlabel']值）

>>>Y数组（[0，1，0]）

请注意，fit\_转换方法只是单独调用fit和transform的快捷方式，我们可以使用inverse\_转换方法将整数类标签转换回其原始字符串表示形式：

>>>class\_le.Inverse\_Transform（Y）数组（['class1'、'class2'、'class1']，dtype=object）

对名义特征执行一次热编码

在前一节中，我们使用一种简单的字典映射方法将序数大小特性转换为整数。由于Scikit-Learn的估计量处理类标签没有任何顺序，因此我们使用方便的Labelencoder类将字符串标签编码为整数。似乎我们可以使用类似的方法来转换数据集的名义颜色列，如下所示：

>>>X=df['color'、'size'、'price']。值

>>>color\_le=labelencoder（）。

>>>X[：，0]=颜色匹配变换（X[：，0]）

>>>X阵列（[[1，1，10.1]，[2，2，13.5]，

[0，3，15.3]]，dtype=对象）

【106】

执行上述代码后，numpy数组x的第一列现在保存新的颜色值，这些值的编码如下：

蓝色0

绿色1

红色2

如果我们在这一点上停止并将数组馈送给分类器，我们将在处理分类数据时犯最常见的错误之一。你能找出问题所在吗？虽然颜色值没有任何特定的顺序，但是学习算法现在假设绿色大于蓝色，红色大于绿色。虽然这个假设是错误的，但是算法仍然可以产生有用的结果。然而，这些结果并不是最佳的。

解决这个问题的一个常见方法是使用一种称为“热编码”的技术。这种方法背后的思想是为“名义特征”列中的每个唯一值创建一个新的虚拟特征。在这里，我们将颜色特性转换为三个新特性：蓝色、绿色和红色。然后可以使用二进制值指示样本的特定颜色；例如，蓝色样本可以编码为blue=1、green=0、red=0。要执行此转换，我们可以使用SciKit-Learn.Preprocessing模块中实现的OneHotEncoder：

>>>来自sklearn.preprocessing导入OneHotEncoder

>>>ohe=OneHotEncoder（分类功能=[0]）

>>>ohe.fit\_transform（x）.toarray（）数组（[[0.，1.，0。，1.，10.1]，[0.，0。，1.，2.，13.5]，

[1.，0。，0。，3.，15.3]）

初始化OneHotEncoder时，我们定义了要通过categorical\_features参数转换的变量的列位置（注意，颜色是特征矩阵x中的第一列）。默认情况下，OneHotEncoder在使用转换方法时返回稀疏矩阵，我们将稀疏矩阵表示转换为常规（密集）numpy数组，以便通过ToArray方法进行可视化。稀疏矩阵只是存储大型数据集的一种更有效的方法，而且许多SciKit学习函数都支持稀疏矩阵，如果稀疏矩阵包含大量的零，它尤其有用。为了省略to array步骤，我们可以将编码器初始化为OneHotEncoder（…，sparse=false），以返回常规的numpy数组。

【107】

通过一个热编码创建这些虚拟特性的更方便的方法是使用在熊猫中实现的get-dummies方法。应用于数据帧时，get-dummies方法将只转换字符串列并保持所有其他列不变：

>>>pd.获取\_dummies（df['price'、'color'、'size']）价格大小颜色\_蓝色\_绿色\_红色0 10.1 1 0 1 0

13.5 2 0 0 1

15.3 3 1 0 0

在训练和测试集中对数据集进行分区

我们在第1章中简要介绍了将数据集划分为单独的数据集进行培训和测试的概念，使计算机能够从数据中学习，第3章介绍了使用SciKit Learn的机器学习分类器。请记住，测试集可以被理解为我们模型的最终测试，然后我们才将其放在现实世界中。在本节中，我们将准备一个新的数据集，即葡萄酒数据集。在对数据集进行预处理之后，我们将探索不同的特征选择技术，以减少数据集的维数。

葡萄酒数据集是另一个开源数据集，可从UCI机器学习库（）获得；它由178个葡萄酒样品组成，其中13个特征描述了它们的不同化学性质。

使用熊猫图书馆，我们将直接从UCI机器学习库中读取开源葡萄酒数据集：

>>>df\_wine=pd.read\_csv（'https://archive.ics.uci.edu/ml/machine-

学习数据库/wine/wine.data'，header=none）>>>df\_wine.columns=['class label'，'alcohol'，

…'苹果酸“，”灰分“，

…'灰的碱性，镁的碱性，

…'总酚类、黄酮类，

…'非黄酮类酚类，

…'原花青素类，

…'颜色强度'，'色调'，

…'稀释葡萄酒的OD280/OD315’，

…'脯氨酸

>>>打印（“类标签”，np.unique（df\_wine[“类标签”））

类标签[1 2 3]

>>>df\_wine.head（）。

【108】

下表列出了葡萄酒数据集中描述178个葡萄酒样品化学性质的13个不同特征：

这些样品属于三个不同类别（1、2和3）中的一个，它们指的是在意大利不同地区种植的三种不同类型的葡萄。

将此数据集随机分为单独的测试和培训数据集的一种方便方法是使用Scikit Learn的交叉验证子模块中的train\_test\_split函数：

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>X，Y=df\_wine.iloc[：，1:].值，df\_wine.iloc[：，0].值

>>>X轴、X轴、Y轴、Y轴测试=\

……列车\测试\拆分（x，y，测试\大小=0.3，随机\状态=0）

首先，我们将特征列1-13的numpy数组表示形式分配给变量x，并将类标签从第一列分配给变量

然后，我们使用train\_test\_split函数将x和y随机分割成单独的训练和测试数据集。通过设置测试尺寸=0.3，我们将30%的葡萄酒样品分配给X-U测试和Y-U测试，其余70%的样品分别分配给X-U和Y-U测试。

如果我们将一个数据集划分为训练和测试数据集，我们必须记住，我们保留了学习算法可能受益的有价值的信息。因此，我们不想给测试集分配太多的信息。然而，测试集越小，泛化误差的估计就越不准确。将数据集划分为训练集和测试集就是为了平衡这种权衡。实际上，最常用的分割是60:40、70:30或80:20，这取决于初始数据集的大小。然而，对于大型数据集，90:10或99:1划分为训练和测试子集也是常见和适当的。不要在模型训练和评估后丢弃分配的测试数据，最好在整个数据集上重新训练分类器以获得最佳性能。

【109】

使功能具有相同的规模

在我们的预处理管道中，特征缩放是一个很容易被遗忘的关键步骤。决策树和随机森林是很少的机器学习算法之一，我们不需要担心特征缩放。然而，当我们实现梯度下降优化算法时，大多数机器学习和优化算法在相同的尺度上表现得更好，正如我们在第2章“训练机器学习分类算法”中看到的那样。

特征缩放的重要性可以通过一个简单的例子来说明。假设我们有两个特征，其中一个特征以1到10的比例测量，第二个特征以1到100000的比例测量。当我们在第2章“训练机器学习分类算法”中考虑Adaline中的平方误差函数时，可以直观地说，该算法主要是根据第二个特征中的较大误差来优化权重。另一个例子是带有欧几里得距离度量的k-最近邻（knn）算法；计算出的样本之间的距离将由第二个特征轴控制。

现在，有两种将不同的特性带到同一个尺度的常见方法：规范化和标准化。这些术语通常在不同的领域中使用得相当松散，其含义必须从上下文中派生出来。通常，标准化是指将特性重新调整到[0，1]的范围，这是最小-最大缩放的一种特殊情况。为了规范化我们的数据，我们可以简单地对每个特征列应用最小-最大比例，其中样本x（i）的新值xnor（i）m可以计算如下：

xnor（i）m=xxma（i）x−−xx分钟

这里，x（i）是一个特定的示例，xmin是特征列中最小的值，xmax是最大的值。

最小-最大缩放过程在SciKit Learn中实现，可以如下使用：

>>>来自sklearn.preprocessing import minmaxscaler

>>>MMS=MinMaxscaler（）。

>>>x\_train\_norm=mms.fit\_transform（x\_train）

>>> x\_test\_norm=mms.transform（x\_test）

[110]

虽然通过最小-最大缩放进行规范化是一种常用的技术，当我们需要有界区间内的值时，这种技术很有用，但是对于许多机器学习算法来说，标准化更为实用。原因是许多线性模型，如逻辑回归和支持向量机，我们记得从第3章，一个机器学习分类器使用scikit学习，初始化权重为0或小的随机值接近0。使用标准化方法，我们将特征列的中心值设为平均值0，标准偏差为1，这样特征列就可以采用正态分布的形式，从而更容易了解权重。此外，标准化维护了关于离群值的有用信息，并使算法对离群值的敏感度降低，而最小-最大缩放则将数据缩放到有限的值范围。

标准化程序可以用以下公式表示：

xst（id）=x（iσ）−x微x

这里，μx是特定特征列的样本平均值，σx是相应的标准偏差。

下表说明了两种常用的特征缩放技术（标准化和标准化）在由数字0到5组成的简单样本数据集上的区别：

与minmaxscaler类似，scikit-learn还实现了一个标准化类：

>>>来自sklearn.preprocessing import standardscaler

>>>STDSC=StandardScaler（）。

>>>X\_train\_std=stdsc.fit\_transform（X\_train）

>>>x\_test\_std=stdsc.transform（x\_test）

【111】

再次强调，我们只在训练数据上安装一次标准定标器，并使用这些参数来转换测试集或任何新的数据点也是很重要的。

选择有意义的功能

如果我们注意到一个模型在训练数据集上的性能比在测试数据集上好得多，那么这个观察结果就是过度拟合的一个有力指标。过度拟合意味着模型与训练数据集中的特定观测值过于吻合，但不能很好地概括为实际数据。我们认为模型具有很高的方差。过度拟合的一个原因是我们的模型对于给定的训练数据过于复杂，为了减少泛化误差，常用的解决方案如下：

收集更多培训数据

通过正则化引入对复杂性的惩罚

选择参数较少的简单模型

减少数据的维数

收集更多的培训数据通常不适用。在下一章中，我们将学习一种有用的技术来检查更多的培训数据是否有用。在下面的部分和小节中，我们将讨论通过特征选择通过正则化和降维来减少过度拟合的常见方法。

具有l1正则化的稀疏解

我们从第3章（使用SciKit Learn的机器学习分类器的教程）回忆起，L2正则化是一种通过惩罚大的单个权重来降低模型复杂性的方法，其中我们定义了权重向量w的L2范数，如下所示：

米

L2：宽W2J

J=1

另一种降低模型复杂性的方法是相关的L1正则化：

米

l1:w 1=∑w j

J=1

【112】

这里，我们简单地用权重的绝对值之和替换了权重的平方。与二级正则化不同，一级正则化产生稀疏特征向量；大多数特征权重为零。如果我们有一个具有许多不相关特性的高维数据集，那么稀疏性在实践中是有用的，特别是在我们具有比样本更多不相关维度的情况下。从这个意义上说，L1正则化可以理解为一种特征选择技术。

为了更好地理解l1正则化如何鼓励稀疏性，让我们后退一步，看看正则化的几何解释。让我们绘制两个权重系数w1和w2的凸成本函数的轮廓。在这里，我们将考虑我们在第2章训练机器学习分类算法中用于Adaline的平方误差（SSE）成本函数的总和，因为它比逻辑回归的成本函数对称且易于绘制；但是，同样的概念也适用于后者。记住，我们的目标是找到权重系数的组合，以最小化培训数据的成本函数，如下图所示（椭圆中间的点）：

现在，我们可以把正规化看作是在成本函数中添加一个惩罚项来鼓励较小的权重；或者换句话说，我们惩罚较大的权重。

【113】

因此，通过正则化参数λ增加正则化强度，我们将权重缩小到零，并减少模型对训练数据的依赖性。让我们在下图中为二级罚款期限说明这个概念。

二次L2正则化项用阴影球表示。这里，我们的权重系数不能超过我们的正规化预算，权重系数的组合不能超出阴影区域。另一方面，我们仍然希望将成本函数最小化。在罚分约束下，我们的最大努力是选择L2球与非货币化成本函数的等值线相交的点。正则化参数λ的值越大，惩罚成本函数增长越快，导致L2球越窄。例如，如果我们将正则化参数向无穷大方向增加，则权重系数将有效地变为零，由l2球的中心表示。为了总结这个例子的主要信息：我们的目标是最小化未实现的成本函数加上惩罚项的总和，这可以理解为增加偏差，在缺乏足够的训练数据来适应模型的情况下，更倾向于使用更简单的模型来减少方差。

【114】

现在让我们来讨论l1正则化和稀疏性。L1正则化背后的主要概念与我们在这里讨论的类似。但是，由于l1惩罚是绝对权重系数的总和（记住l2项是二次的），我们可以将其表示为菱形预算，如下图所示：

在上图中，我们可以看到成本函数的轮廓在w1=0时与l1菱形相接触。由于一个l1正则化系统的轮廓是尖锐的，所以最可能的情况是，成本函数的椭圆和l1菱形的边界之间的交点位于轴上，这会导致稀疏性。为什么L1正则化可以导致稀疏解的数学细节超出了本书的范围。如果您感兴趣，可以在统计学习要素的第3.4节，Trevor Hastie、Robert Tibshirani和Jerome Friedman，Springer中找到关于二级与一级正则化的优秀章节。

对于SciKit中支持L1正则化的正则化模型，我们可以简单地将惩罚参数设置为“L1”，以生成稀疏解：

>>>来自sklearn.linear\_model import logisticregression

>>>逻辑回归（惩罚='l1'）

【115】

应用于标准化葡萄酒数据，L1正则化逻辑回归将产生以下稀疏解：

>>>lr=逻辑回归（惩罚='l1'，c=0.1）

>>>左后配合（X\_-train\_-std，Y\_-train）

>>>打印（‘培训精度：’，lr.score（x\_train\_std，y\_train））。

培训精度：0.983870967742

>>>print（'测试精度：'，lr.score（x\_test\_std，y\_test））

测试精度：0.981481481481

培训和测试精度（均为98%）均未表明我们的模型有任何过度拟合。当我们通过lr.intercept属性访问截获项时，我们可以看到数组返回三个值：

>>>lr.intercept\_u array（[-0.38379237，-0.1580855，-0.70047966]）

由于我们将logisticregression对象拟合到多类数据集中，因此在默认情况下，它使用一对休息（ovr）方法，其中第一个截取属于适合1类对2类和3类的模型；第二个值是适合2类对1类和3类的模型的截取；以及第三个值是模型的截距，分别适用于3级和1级和2级：

>>>左心室系数\_

数组（[[0.280，0.000，0.000，-0.0282，0.000，0.000，0.710，0.000，0.000，0.000，

0.000、0.000、1.236、[-0.644、-0.0688、-0.0572、0.000、0.000、

0.000，0.000，0.000，0.000，-0.927，

0.060，0.000，-0.371]，[0.000，0.061，0.000，0.000，0.000，

0.000，-0.637，0.000，0.000，0.499，-0.358，-0.570，0.000

]]）

我们通过lr.coef属性访问的权重数组包含三行权重系数，每个类有一个权重向量。每行包含13个权重，其中每个权重乘以13维葡萄酒数据集中的各自特征，以计算净输入：

Z=w1x1++wmxm=∑mj=0 xjwj=wt x

【116】

我们注意到权重向量是稀疏的，这意味着它们只有几个非零的条目。作为特征选择方法的L1正则化的结果，我们刚刚训练了一个模型，该模型对数据集中可能不相关的特征具有鲁棒性。

最后，让我们绘制正则化路径，即不同正则化强度下不同特征的权重系数：

>>>导入matplotlib.pyplot为plt

>>>图=PLT.Figure（）

>>>AX=PLT.子批次（111）

>>>颜色=['blue'、'green'、'red'、'cyan'，

…'洋红'、'黄色'、'黑色'，

…'粉色“、”浅绿色“、”浅蓝色“，

…'灰色“、”靛蓝“、”橙色“]

>>>权重，参数=[]，[]>>对于np.arange中的c（-4，6）：

……lr=逻辑回归（惩罚='l1'，

……C=10\*\*C，

……随机状态=0）

……Lr.fit（x\_train\_std，y\_train）

……weights.append（lr.coef\_[1]）

……附加参数（10\*\*c）

>>>weights=np.array（weights）>>>对于列，颜色为zip（range（weights.shape[1]），颜色）：

……plt.plot（参数，权重[：，列]，

……label=df\_wine.columns[列+1]，

……颜色=颜色）

>>>plt.axhline（0，color='black'，linestyle=--'，lineidth=3）

>>>plt.xlim（[10\*\*（-5），10\*\*5]）

>>>plt.ylabel（'重量系数'）

>>>plt.xlabel（'c'）

>>>plt.xscale（'log'）

>>>Plt.Legend（loc='左上角'）

>>>AX.图例（loc='upper center'，

……B箱锚（（1.38，1.03）

……ncol=1，fancybox=真）

>>>plt.show（）。

【117】

由此得出的图为我们提供了关于L1正则化行为的进一步见解。如我们所见，如果我们用一个强正则化参数（c<0.1）惩罚模型，所有特征权重都将为零；c是正则化参数λ的倒数。

序列特征选择算法

另一种降低模型复杂度和避免过度拟合的方法是通过特征选择减少维度，这对于非规范化模型尤其有用。降维技术主要有两类：特征选择和特征提取。使用特征选择，我们选择原始特征的一个子集。在特征提取中，我们从特征集中提取信息来构造新的特征子空间。在本节中，我们将介绍一个经典的特征选择算法系列。在下一章，第5章，通过降维压缩数据，我们将学习不同的特征提取技术，将数据集压缩到低维特征子空间。

序列特征选择算法是一系列贪婪搜索算法，用于将初始的D维特征空间减少到k维特征子空间，其中k<d。特征选择算法背后的动机是自动选择一个子集，其中e与问题最相关的是通过去除不相关的特征或噪声来提高计算效率或减少模型的泛化误差，这对于不支持正则化的算法是有用的。经典的序列特征选择算法是序列后向选择（SBS），它的目标是在分类器性能衰减最小的情况下降低初始特征子空间的维数，以提高计算效率。在某些情况下，如果模型过度拟合，SBS甚至可以提高模型的预测能力。

[118]

贪婪算法在组合搜索问题的每个阶段都会做出局部最优选择，通常会产生一个次优解，而非穷尽搜索算法，它评估所有可能的组合，并保证找到最优解。然而，在实践中，穷尽搜索通常在计算上是不可行的，而贪婪算法允许一个不那么复杂、计算上更有效的解决方案。

SBS算法背后的思想非常简单：SBS顺序地从完整的功能子集中删除功能，直到新的功能子空间包含所需数量的功能。为了确定在每个阶段要删除哪个特征，我们需要定义要最小化的标准函数j。由标准函数计算的标准可以简单地说是特定特征去除前后分类器性能的差异。然后，在每个阶段要删除的特性可以简单地定义为最大化该标准的特性；或者，更直观地说，在每个阶段，我们都会删除在删除后导致性能损失最小的特性。根据前面对sbs的定义，我们可以用4个简单的步骤概述算法：

用k=d初始化算法，其中d是整个特征空间xd的维数。

确定最大化标准x−=argmaxj（xk−x））的特征x−其中x∈xk。

从功能集中删除功能X−：XK–1=XK-1=XK−X−，K=K−1。

如果k等于所需功能的数量，则终止，否则转到步骤2。

在大规模特征选择技术的比较研究中，您可以找到对几种顺序特征算法的详细评估，F.Ferri、P.Pudil、M.Hatef和J.Kittler。

大规模特征选择技术的比较研究。

《实践四》中的模式识别，第403-413页，1994年。

不幸的是，SBS算法还没有在SciKit学习中实现。但是，由于它非常简单，让我们从头开始用Python实现它：

从sklearn.base导入克隆从itertools导入组合导入numpy为np

从sklearn.cross-validation-import-train-split-from sklearn.metrics-import-accuracy-score

[119]

class sbs（）：def\_uu init\_uuu（self，estimator，k\_features，

得分=准确度得分，测试\大小=0.25，随机\状态=1）：self.scoring=得分self.estimator=克隆（estimator）self.k \功能=k \功能self.test \大小=测试\大小self.random \状态=随机\状态

def fit（自我，x，y）：

x轴试验，x轴试验，y轴试验，y轴试验=\

列车\测试\拆分（x，y，测试\大小=自检\大小，随机\状态=自检.随机\状态）

dim=x\_train.shape[1]self.indexs u=tuple（range（dim））self.subsets=[self.indexs\_ux]score=self.\_calc\_score（x\_train，y\_train，

x检验，y检验，self.指数，self.分数

而dim>self.k\_特性：得分=[]子集=[]

对于组合中的p（self.indexes\_u，r=dim-1）：score=self.\_calc\_score（x\_train，y\_train，x\_test，y\_test，p）

分数。附加（分数）子集。附加（P）

best=np.argmax（分数）self.indexs=subsets[best]self.subsets.append（self.indexes uuu）dim-=1

self.scores\_u.append（scores[best]）self.k\_score\_u=self.scores\_[-1]返回self

def transform（self，x）：返回x[：，self.indexes\_uux]

[120]

定义计算分数（自我，x训练，y训练，

x检验，y检验，指数）：

self.estimator.fit（x\_train[：，indexes]，y\_train）y\_pred=self.estimator.predict（x\_test[：，indexes]）score=self.scoring（y\_test，y\_pred）返回分数

在前面的实现中，我们定义了k\_features参数来指定要返回的所需功能数量。默认情况下，我们使用SciKit Learn中的精度分数来评估模型的性能，并使用评估器对特征子集进行分类。在拟合方法的while循环中，对itertools.combination函数创建的特征子集进行评估和缩减，直到特征子集具有所需的维数。在每次迭代中，最佳子集的准确度得分被收集到一个列表中。基于内部创建的测试数据集x\_测试得分。我们稍后将使用这些分数来评估结果。最后一个特征子集的列索引被分配给self.indexes\_uu，我们可以通过transform方法使用self.indexes返回带有所选特征列的新数据数组。注意，我们没有在fit方法中显式地计算标准，而是简单地删除了不包含在最佳性能特征子集中的特征。

现在，让我们看看使用scikit中的knn分类器的SBS实现的实际情况，了解：

>>>从sklearn.neighbors导入Kneighbors分类器

>>>导入matplotlib.pyplot为plt

>>>Knn=Kneighbors分类器（n\_neighbors=2）

>>>sbs=sbs（knn，k\_features=1）

>>>sbs.fit（x\_train\_std，y\_train）

尽管我们的SBS实现已经将数据集拆分为fit函数内的测试和培训数据集，但我们仍然将培训数据集x\_train提供给算法。然后，sbs-fit方法将为测试（验证）和培训创建新的培训子集，这就是为什么这个测试集也被称为验证数据集的原因。这种方法是必要的，以防止我们的原始测试集成为培训数据的一部分。

记住，我们的SBS算法在每个阶段收集最佳特征子集的分数，所以让我们继续讨论实现中更令人兴奋的部分，并绘制在验证数据集上计算的KNN分类器的分类精度。代码如下：

>>>K\_Feat=[len（k）for k in sbs.subsets\_uux（k）]

>>>plt.plot（K\_Feat，sbs.scores\_uu，marker='o'）

>>>plt.ylim（[0.7，1.1]）

【121】

>>>plt.ylabel（'准确度'）

>>>plt.xlabel（'功能数量'）

>>>plt.grid（）。

>>>plt.show（）。

正如我们在下面的图表中看到的，当我们减少特征的数量时，验证数据集上的knn分类器的精度得到了提高，这可能是由于我们在第3章，机器学习之旅中讨论的knn算法的上下文中减少了维度的诅咒。使用SciKit学习RNing分类器。此外，我们可以在下面的图中看到，分类器对于k=5、6、7、8、9、10达到了100%的精度：

为了满足我们自己的好奇心，让我们看看这五个特性是什么，它们在验证数据集上产生了如此好的性能：

>>>K5=列表（sbs.subsets\_[8]）

>>>打印（df\_wine.columns[1:][k5]）

指数（[‘酒精’、‘苹果酸’、‘灰分碱度’、‘色调’，

'proline']，dtype='object'）

使用前面的代码，我们从sbs.subsets u属性的第9个位置获取了5个特征子集的列索引，并从熊猫酒数据帧的列索引返回了相应的特征名称。

【122】

接下来，让我们在原始测试集上评估KNN分类器的性能：

>>>knn.fit（x\_train\_std，y\_train）

>>>打印（‘培训精度：’，knn.分数（x\_train\_std，y\_train））

培训精度：0.983870967742

>>>print（'测试精度：'，knn.分数（x\_test\_std，y\_test））

测试精度：0.9444444444

在前面的代码中，我们使用了完整的特征集，在训练数据集上获得了大约98.4%的准确度。然而，测试数据集的准确度略低（约94.4%），这表明有轻微的过度拟合。现在，让我们使用所选的5特征子集，看看knn的性能如何：

>>>knn.fit（x\_train\_std[：，k5]，y\_train）

>>>打印（'训练精度：'，

……knn.分数（x\_train\_std[：，k5]，y\_train）

训练精度：0.959677419355

>>>print（'测试精度：'，

……knn.分数（x\_test\_std[：，k5]，y\_test）

测试精度：0.962962962963

使用葡萄酒数据集中不到一半的原始特征，测试集的预测精度提高了近2%。此外，我们还减少了过度拟合，这可以从测试（约96.3%）和培训（约96.0%）准确性之间的小差距中看出。

SciKit学习中的特征选择算法

SciKitLearn提供了更多的功能选择算法。其中包括基于特征权重的递归后向消除、基于树的按重要性选择特征的方法以及单变量统计测试。对不同的特征选择方法的全面讨论超出了本书的范围，但是可以在中找到一个很好的总结和示例。

[123]

用随机森林评估特征重要性

在前面的部分中，您学习了如何通过逻辑回归使用l1正则化来消除不相关的特性，并使用sbs算法进行特性选择。从数据集中选择相关特性的另一个有用方法是使用随机林，这是我们在第3章中介绍的一种集成技术，即使用SciKit Learn的机器学习分类器教程。使用一个随机森林，我们可以测量特征重要性，因为从森林中所有决策树计算出的平均杂质减少量，而不需要做任何假设，我们的数据是否是线性可分离的。很方便，SciKitLearn中的随机森林实现已经为我们收集了特征导入，这样我们可以在拟合RandomForestClassifier之后通过特征导入属性访问它们。通过执行以下代码，我们现在将在葡萄酒数据集上训练10000棵树的森林，并根据其各自的重要性度量对13个特征进行排名。记得

（从我们在第3章中的讨论，使用SciKit Learn的机器学习分类器教程）我们不需要使用标准化或标准化的基于树的模型。代码如下：

>>>从sklearn.ensegle导入RandomForestClassifier

>>>Feat\_Labels=df\_wine.columns[1:]

>>>forest=randomforestclassifier（n\_estimators=10000，

……随机状态=0，

……n\_作业=-1）

>>>Forest.Fit（X\_火车，Y\_火车）

>>>importances=forest.feature\_importances\_

>>>indexs=np.argsort（importances）[：：-1]>>>对于范围内的F（x\_train.shape[1]）：

……打印（”%2d）%-\*s%f“%（f+1，30，

……专长标签[F]，

……进口[指数[f]]）

酒精0.182508

苹果酸0.158574

灰0.150954

灰分碱度0.131983

镁0.106564

总酚0.078249

黄酮类0.060717

非酚类0.032039

原花青素0.025385

颜色强度0.022369

色调0.022070

【124】

稀释葡萄酒的OD280/OD315 0.014655

脯氨酸0.013933

>>>plt.title（'功能导入'）

>>>plt.bar（范围（x\_train.shape[1]），

……进口[指数]，

……color='lightblue'，

……align='居中'）

>>>plt.xticks（范围（x\_train.shape[1]），

……特征标签，旋转=90）

>>>plt.xlim（[-1，x\_train.shape[1]]）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

在执行上述代码之后，我们创建了一个图，根据葡萄酒数据集中不同的特征的相对重要性对其进行排序；请注意，特征重要性是规范化的，因此它们的总和为1.0。

[125]

根据10000个决策树的平均杂质减少量，我们可以得出结论，葡萄酒中的酒精含量是数据集中最具辨别性的特征。有趣的是，前面图中排名靠前的三个功能也是我们在前一节中实现的SBS算法选择的前五个功能。然而，就可解释性而言，随机森林技术有一个重要的启示值得一提。例如，如果两个或多个功能高度相关，则一个功能的排名可能非常高，而其他功能的信息可能无法完全捕获。另一方面，如果我们仅仅对模型的预测性能感兴趣，而不是对特征重要性的解释，我们就不需要担心这个问题。为了总结关于特征导入和随机林的这一节，值得一提的是，SciKit Learn还实现了一种转换方法，该方法在模型拟合后根据用户指定的阈值选择特征，如果我们想将RandomForestClassifier用作特征，这很有用。scikit学习管道中的选择器和中间步骤，允许我们将不同的预处理步骤与估计器连接起来，如第6章“学习模型评估和超参数调整的最佳实践”所示。例如，我们可以将阈值设置为0.15，以使用以下代码将数据集减少到3个最重要的功能，即酒精、苹果酸和灰分：

>>>X\_selected=forest.transform（X\_train，threshold=0.15）

>>>X\_selected.shape

（124，3）

总结

本章首先介绍了一些有用的技术，以确保正确处理丢失的数据。在我们将数据输入机器学习算法之前，我们还必须确保正确编码分类变量，并且我们已经了解了如何将序数和名义特征值映射到整数表示。

此外，我们还简要讨论了L1正则化，它可以通过降低模型的复杂性来帮助我们避免过度拟合。作为一种去除不相关特征的替代方法，我们使用顺序特征选择算法从数据集中选择有意义的特征。

在下一章中，您将了解另一种有效的降维方法：特征提取。它允许我们将特征压缩到低维子空间，而不是像在特征选择中那样完全删除特征。

【126】

通过压缩数据

降维

在第4章，构建良好的训练集-数据预处理中，您学习了使用不同的特征选择技术降低数据集维数的不同方法。另一种用于降维的特征选择方法是特征提取。在本章中，您将了解三种基本技术，它们将帮助我们通过将数据集转换为比原始数据集低维度的新特征子空间来总结数据集的信息内容。数据压缩是机器学习中的一个重要课题，它有助于我们存储和分析现代技术时代产生和收集的越来越多的数据。在本章中，我们将讨论以下主题：

无监督数据压缩的主成分分析（PCA）

线性判别分析（LDA）作为一种监督降维技术，用于最大化类可分离性

基于核主成分分析的非线性降维

【127】

基于主成分分析的无监督降维

与特征选择类似，我们可以使用特征提取来减少数据集中的特征数量。然而，当我们使用特征选择算法（如顺序向后选择）时，虽然我们保持了原始特征，但我们使用特征提取将数据转换或投影到新的特征空间。在维数约简的背景下，特征提取可以理解为一种数据压缩方法，其目的是维护大部分相关信息。特征提取通常用于提高计算效率，但也有助于减少维数的诅咒，特别是当我们使用非规范化模型时。

主成分分析（PCA）是一种无监督线性变换技术，广泛应用于各个领域，在降维中最为突出。PCA的其他流行应用包括股票市场交易中的探索性数据分析和信号去噪，以及生物信息学领域的分析基因组数据和基因表达水平。PCA帮助我们根据特征之间的相关性识别数据中的模式。简言之，PCA的目标是找到高维数据中最大方差的方向，并将其投影到一个新的子空间中，该子空间的维数等于或小于原始子空间的维数。新子空间的正交轴（主分量）可以解释为最大方差方向，前提是新特征轴相互正交，如下图所示。这里，x1和x2是原始特征轴，pc1和pc2是主要组件：

[128]

如果我们使用PCA进行维数约简，我们构造了一个D×K维变换矩阵W，它允许我们将一个样本向量X映射到一个新的K维特征子空间，该子空间的维数小于原始的D维特征空间：

x=[x1，x2，…，xd]，x∈rd

↓xw，w∈rd×k

Z=[Z1，Z2，…，ZK]，Z∈RK

由于将原始的D维数据转换成这个新的K维子空间（通常是k<<d），第一个主分量将

最大可能方差和所有随后的主分量将具有最大可能方差，因为它们与其他主分量不相关（正交）。请注意，PCA方向对数据缩放高度敏感，如果在不同尺度上测量特征，我们需要在PCA之前对特征进行标准化，并且我们希望对所有特征赋予同等重要性。

在更详细地研究用于降维的PCA算法之前，让我们用几个简单的步骤来总结该方法：

标准化三维数据集。

构造协方差矩阵。

将协方差矩阵分解为特征向量和特征值。

选择与k最大特征值相对应的k特征向量，其中k是新特征子空间的维数（k≤d）。

从“顶部”k特征向量构造投影矩阵w。

利用投影矩阵W变换三维输入数据集X，得到新的K维特征子空间。

【129】

总方差和解释方差

在这一小节中，我们将处理主成分分析的前四个步骤：标准化数据，构造协方差矩阵，获得协方差矩阵的特征值和特征向量，并通过降序对特征值进行排序。

首先，我们将从加载我们在第4章中一直使用的葡萄酒数据集开始，构建良好的培训集-数据预处理：

>>>将熊猫作为PD导入

>>>df\_wine=pd.read\_csv（'https://archive.ics.uci.edu/ml/machineLearning databases/wine/wine.data'，header=none）

接下来，我们将分别使用70%和30%的数据将葡萄酒数据处理成单独的培训和测试集，并将其标准化为单位差异。

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>来自sklearn.preprocessing import standardscaler

>>>X，Y=df\_wine.iloc[：，1:].值，df\_wine.iloc[：，0].值

>>>X轴、X轴、Y轴、Y轴测试=\

……列车测试分割（x，y，

……测试\_大小=0.3，随机\_状态=0）

>>>SC=标准缩放器（）

>>>x\_train\_std=sc.fit\_transform（x\_train）

>>>x\_test\_std=sc.fit\_transform（x\_test）

在通过执行前面的代码完成了强制的预处理步骤之后，让我们进入第二步：构造协方差矩阵。对称d×d维协方差矩阵，其中d是数据集中的维数，存储不同特征之间的成对协方差。例如，两个特征x j和xk在总体水平上的协方差可以通过以下公式计算：

σjk=−

这里，μj和μk分别是特征和k的样品手段。注意，如果我们对数据集进行标准化，那么样本平均值为零。两个特征之间的正协方差表示特征一起增加或减少，而负协方差表示特征在相反方向上变化。例如，三个特征的协方差矩阵可以写成（注意∑代表希腊字母sigma，不要与和符号混淆）：

[130]

σ2\_

∑=σ21σ2σ23

\_\_

协方差矩阵的特征向量表示主分量（最大方差方向），而相应的特征值将定义其大小。在葡萄酒数据集的情况下，我们将从13×13维协方差矩阵中获得13个特征向量和特征值。

现在，让我们得到协方差矩阵的特征对。正如我们从线性代数或微积分的入门课程中所记得的，特征值v满足以下条件：

∑V=λV

这里，λ是一个标量：特征值。由于手工计算特征向量和特征值有点繁琐和复杂，我们将使用numpy的linalg.eig函数来获得葡萄酒协方差矩阵的特征对：

>>>导入numpy为np

>>>cov\_mat=np.cov（x\_train\_std.t）

>>>eigen-vals，eigen-vecs=np.linalg.eig（cov-mat）

>>>打印（'\neigenvalues\n%s'%eigen-vals）

特征值

[4.8923083 2.46635032 1.42809973 1.01233462 0.84906459

0.60181514

0.5251546 0.08414846 0.33051429 0.29595018 0.16831254 0.21432212

0.2399553条]

利用numpy.cov函数，我们计算了标准化训练数据集的协方差矩阵。利用Linalg.eig函数，我们进行了特征成分分析，得到了一个由13个特征值和相应的特征向量组成的向量（特征值），这些特征向量存储在13×13维矩阵（特征值向量）的列中。

由于我们希望通过将数据集压缩到一个新的特征子空间来减少数据集的维数，所以我们只选择包含大部分信息（方差）的特征向量（主成分）的子集。由于特征值定义了特征向量的大小，因此我们必须通过减小的大小对特征值进行排序；我们对基于其相应特征值的顶部k个特征向量感兴趣。但是在我们收集那些K最有用的特征向量之前，让我们先画出特征值的方差解释比。

【131】

特征值λj的方差解释比只是特征值λj的分数和特征值的总和：λj

使用numpy-cumsum函数，我们可以计算解释的方差的累积和，我们将通过matplotlib的阶跃函数绘制：

>>>TOT=总和（本征值）

>>>var\_exp=[（i/tot）对于i in

……排序（本征值，反向=真）]

>>>cum\_var\_exp=np.cumsum（var\_exp）

>>>导入matplotlib.pyplot为plt

>>>plt.bar（范围（1,14），var\_exp，alpha=0.5，align='center'，

……label='个别解释差异'）

>>>plt.step（range（1,14），cum\_var\_exp，where='mid'，

……label='累计解释方差'）

>>>plt.ylabel（'解释的方差比'）

>>>plt.xlabel（'主要组件'）

>>>Plt.Legend（loc='best'）

>>>plt.show（）。

结果图表明，第一个主要成分单独占方差的40%。此外，我们还可以看到前两个主要组成部分的组合解释了数据中几乎60%的差异：

【132】

虽然解释的方差图提醒我们，我们在第4章“建立良好的训练集-数据预处理”中计算的特征重要性，但是我们应该提醒自己，PCA是一种无监督的方法，这意味着关于类标签的信息被忽略了。当一个随机森林使用类成员信息来计算节点杂质时，方差度量值沿特征轴的传播。

特征转换

在我们成功地将协方差矩阵分解为特征对之后，现在让我们继续进行最后三个步骤，将葡萄酒数据集转换为新的主成分轴。在本节中，我们将按特征值的降序对特征值对进行排序，从选定的特征向量构造投影矩阵，并使用投影矩阵将数据转换到低维子空间。

我们首先通过特征值的降序对特征值对进行排序：

>>>Eigen\_Pairs=[（np.abs（Eigen\_Vals[i]），Eigen\_Vecs[：，i]）

……对于i inrange（len（eigen-vals））]

>>>eigen\_pairs.sort（reverse=true）

接下来，我们收集与两个最大值对应的两个特征向量，以捕获该数据集中大约60%的方差。注意，我们只选择了两个特征向量来进行说明，因为我们将在本小节后面通过二维散点图来绘制数据。在实践中，主要成分的数量必须根据计算效率和分类器性能之间的权衡来确定：

>>>W=np.hstack（（eigen\_pairs[0][1][：，np.newaxis]，

……本征对[1][1][：，np.newaxis]））

>>>打印（'matrix w:\n'，w）matrix w:

[0.14669811 0.50417079]

[-0.24224554 0.24216889]

[-0.02993442 0.28698484]

[-0.25519002-0.06468718]

[0.12079772 0.22995385]

[0.38934455 0.09363991]

[0.42326486 0.01088622]

[-0.30634956 0.01870216]

【0.30572219 0.03040352】

[-0.09869191 0.54527081]

【133】

[0.30032535-0.27924322]

[0.36821154-0.174365]

[0.29259713 0.36315461]]

通过执行上述代码，我们从顶部的两个特征向量创建了一个13×2维投影矩阵W。使用投影矩阵，我们现在可以将样本x（表示为1×13维行向量）转换为PCA子空间，获得x′，这是一个由两个新特征组成的二维样本向量：

x′=xw

>>>X\_train\_std[0].点（W）数组（[2.59891628，0.00484089]）

同样，我们可以通过计算矩阵点积将整个124×13维训练数据集转换为两个主要组件：

x′=xw

>>>X\_train\_pca=X\_train\_std.dot（w）

最后，让我们在二维散点图中可视化转换后的葡萄酒训练集，现在存储为124×2维矩阵：

>>>颜色=['R'、'B'、'G']

>>>markers=['s'、'x'、'o']>>>对于L、C、M in-zip（np.unique（y\_train）、colors、markers）：

……plt.散射（x\_train\_pca[y\_train==l，0]，

……x\_train\_pca[y\_train==l，1]，

……c=c，label=l，marker=m）

>>>plt.xlabel（'pc 1'）

>>>plt.ylabel（'pc 2'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

正如我们在结果图中看到的（如下图所示），数据沿x轴（第一主成分）的分布比第二主成分（y轴）的分布更广，这与我们在上一小节中创建的解释方差比图一致。然而，我们可以直观地看到，线性分类器很可能能够很好地分离类：

【134】

尽管为了在前面的散点图中进行说明，我们对类标签信息进行了编码，但我们必须记住，PCA是一种不使用类标签信息的无监督技术。

Scikit学习中的主成分分析

虽然前面小节中的详细方法帮助我们跟踪PCA的内部工作，但是现在我们将讨论如何使用Scikit Learn中实现的PCA类。PCA是Scikit Learn的另一个变压器类，我们首先使用培训数据拟合模型，然后使用相同的模型参数转换培训数据和测试数据。现在，让我们在葡萄酒培训数据集中使用ScikitLearn中的PCA，通过逻辑回归对转换后的样品进行分类，并通过我们在第2章“培训机器学习分类算法”中定义的绘图决策区域功能来可视化决策区域：

从matplotlib.colors导入listedcolormap def plot ou decision ou regions（x，y，classifier，resolution=0.02）：

#设置标记生成器和颜色映射标记=（'s'、'x'、'o'、'^'、'v'）

颜色=（'red'、'blue'、'lightgreen'、'gray'、'cyan'）cmap=listedColormap（颜色[：len（np.unique（y））]）绘制决策图面

[135]

x1\_min，x1\_max=x[：，0].min（）-1，x[：，0].max（）+1 x2\_min，x2\_max=x[：，1].min（）-1，x[：，1].max（）+1 xx1，xx2=np.meshgrid（np.arrange（x1\_min，x1\_max，resolution），np.arrange（x2\_min，x2\_max，resolution））z=classifier.predict（np.array（[xx1.ravel（），xx2.ravel（））.t）

Z=Z.整形（XX1.形状）

plt.contourf（xx1，xx2，z，alpha=0.4，cmap=cmap）plt.xlim（xx1.min（），xx1.max（））plt.ylim（xx2.min（），xx2.max（））

#枚举（np.unique（y））中IDX、CL的绘图类样本：plt.scatter（x=x[y==cl，0]，y=x[y==cl，1]，alpha=0.8，c=cmap（idx），marker=markers[idx]，label=cl）

>>>来自sklearn.linear\_model import logisticregression

>>>来自sklearn.composition import pca

>>>PCA=PCA（n\_组件=2）

>>>LR=LogisticRegression（）。

>>>x\_train\_pca=pca.fit\_transform（x\_train\_std）

>>>x\_test\_pca=pca.transform（x\_test\_std）

>>>LR.fit（X\_train\_PCA，Y\_train）

>>>绘制决策区域（x\_train\_pca，y\_train，classifier=lr）

>>>plt.xlabel（'pc1'）

>>>plt.ylabel（'pc2'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

通过执行前面的代码，我们现在应该看到训练模型的决策区域减少到两个主要组件轴。

[136]

如果我们将通过SciKit Learn的PCA投影与我们自己的PCA实现进行比较，我们会注意到上面的图是通过我们的逐步方法得到的前一个PCA的镜像。请注意，这不是由于这两种实现中的任何一种都有错误造成的，但造成这种差异的原因是，根据Eigensolver，EigenVector可以有负符号或正符号。这并不重要，但是我们可以简单地通过将数据乘以-1来恢复镜像，如果我们愿意的话；请注意，特征向量通常被缩放到单位长度1。为了完整起见，让我们在转换后的测试数据集上绘制逻辑回归的决策区域，看看它是否能够很好地分离类：

>>>绘制决策区域（x\_test\_pca，y\_test，classifier=lr）

>>>plt.xlabel（'pc1'）

>>>plt.ylabel（'pc2'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

在我们通过执行前面的代码为测试集绘制决策区域之后，我们可以看到逻辑回归在这个小的二维特征子空间上表现得很好，并且只对测试数据集中的一个样本进行了错误分类。

如果我们对不同主成分的解释方差比感兴趣，我们可以简单地初始化PCA类，将n\_components参数设置为none，这样所有主成分都会保留，然后通过解释方差比访问解释的方差比。o属性：

>>>PCA=PCA（n\_组件=无）

>>>x\_train\_pca=pca.fit\_transform（x\_train\_std）

>>>PCA.解释的方差比\_

【137】

数组（[0.37329648，0.18818926，0.10896791，0.07724389，

0.06478595，

0.04592014，0.03986936，0.0252914，0.02258181，0.01830924，

0.01635336、0.01284271、0.00642076]）

注意，我们在初始化PCA类时设置了n\_components=none，这样它将按排序顺序返回所有主要组件，而不是执行维数缩减。

基于线性判别分析的监督数据压缩

线性判别分析（LDA）作为一种特征提取技术，可以提高非规范模型的计算效率，降低因维数诅咒而导致的过度拟合程度。

LDA背后的一般概念与PCA非常相似，而PCA试图在数据集中找到最大方差的正交分量轴；LDA的目标是找到优化类可分性的特征子空间。LDA和PCA都是线性变换技术，可以用来减少数据集中的维数；前者是无监督的算法，而后者是有监督的。因此，我们可以直观地认为，与PCA相比，LDA是一种优越的分类任务特征提取技术。然而，A.M.Martinez报告说，在某些情况下，通过PCA的预处理往往会在图像识别任务中产生更好的分类结果，例如，如果每个类仅包含少量样本（A.M.Martinez和A.C.Kak）。PCA对LDA。图案

《分析与机器智能》，IEEE汇刊，23（2）：228–2332001）。

虽然lda有时也被称为fisher's lda，但Ronald A.Fisher最初在1936年为两类分类问题制定了fisher的线性判别式（R.A.Fisher）。在分类问题中使用多重测量。优生学年鉴，7（2）：179–1881936）。1948年，C.Radhakrishna Rao在等类协方差和正态分布类的假设下，将Fisher线性判别式推广到多类问题，我们现在称之为lda（C.R.Rao）。多用途

生物分类问题的测量。皇家统计学会杂志。系列B（方法学），10（2）：159-2031948）。

【138】

下图总结了两类问题的lda概念。1类样品显示为十字，2类样品分别显示为圆：

线性判别法，如x轴（ld 1）所示，将两个正态分布类很好地分开。虽然Y轴（ld 2）上显示的示例性线性判别法捕获了数据集中的大量方差，但由于它不捕获任何类的歧视性信息，因此它将作为一个良好的线性判别法失败。

LDA中的一个假设是数据是正态分布的。此外，我们假设类具有相同的协方差矩阵，并且这些特征在统计上彼此独立。然而，即使这些假设中的一个或多个被轻微违反，用于降维的LDA仍然可以相当好地工作（R.O.Duda、P.E.Hart和D.G.Stork）。模式分类。第二个。版本。纽约，2001年）。

在我们在下面的小节中研究LDA的内部工作之前，让我们总结一下LDA方法的关键步骤：

标准化三维数据集（D是功能的数量）。

对于每个类，计算D维平均向量。

构造类间散射矩阵sb和类内散射矩阵sw。

【139】

计算矩阵sw-1sb的特征向量和相应的特征值。

选取K最大特征值对应的K特征向量，构造D×K维变换矩阵W，特征向量为该矩阵的列。

使用变换矩阵w将样本投影到新的特征子空间。

我们在使用LDA时所做的假设是，这些特性是正态分布的，彼此独立。此外，LDA算法假定各个类的协方差矩阵是相同的。然而，即使我们在某种程度上违反了这些假设，LDA在降维和分类任务（R.O.Duda、P.E.Hart和D.G.Stork）中仍然可以相当好地工作。模式分类。第二个。版本。纽约，2001年）。

计算散射矩阵

由于我们已经在本章开头的PCA部分对葡萄酒数据集的特性进行了标准化，因此我们可以跳过第一步，继续计算平均向量，我们将使用平均向量来构造类内和类间散射矩阵，分别。每个平均矢量m i存储相对于I类样品的平均特征值μm：

m i=n1i x∑∈cd i x m

这会产生三个平均向量：

\_，酒精

\_

Mi=\_\_\_i∈1，2，3

\_，脯氨酸\_

\_

[140]

>>>np.设置打印选项（精度=4）

>>>对于范围（1,4）中的标签，平均值vecs=[]>>：

……均值附加（

……x\_train\_std[y\_train==label]，axis=0]）

……print（“mv%s:%s\n%”（label，mean\_vecs[label-1]））

MV 1:[0.9259-0.3091 0.2592-0.7989 0.3039 0.9608 1.0515-0.6306

0.5354个

0.2209 0.4855 0.798 1.2017年]

中压2：[-0.8727-0.3854-0.4437 0.2481-0.2409-0.1059 0.0187-0.0164

0.1095-0.8796 0.4392 0.2776-0.7016]

MV 3:[0.1637 0.8929 0.3249 0.5658-0.01-0.9499-1.228 0.7436

-0.7652个

0.979-1.1698-1.3007-0.3912]

使用平均向量，我们现在可以计算类内散射矩阵sw：

C

sw=∑s i

I=1

这是通过对每个I类的单个散射矩阵si求和来计算的：

C

Si=∑（x−m）（x−m）t

>>>D=13功能数量

>>>s\_w=np.zeros（（d，d））>>>对于标签，mv in zip（范围（1,4），平均值\_vecs）：

……\_类散射=np.零（（d，d））…对于x中的行[y==label]：

……row，mv=row.重塑（d，1），mv.重塑（d，1）

……\_类散射+=（行mv）.dot（（行mv）.t）

……s\_w+=类\_散射

>>>打印（'类内散点矩阵：%sx%s'

……%（S形[0]，S形[1]））

类内散点矩阵：13x13

【141】

我们在计算散点矩阵时所做的假设是训练集中的类标签是均匀分布的。但是，如果我们打印类标签的数量，我们会发现违反了这个假设：

>>>print（'类标签分发：%s'

……%Np.BinCount（Y\_列车）[1:]

类标签分布：【40 49 35】

因此，我们想在将单个散点矩阵求和为散点矩阵sw之前对其进行比例缩放。当我们将散射矩阵除以类样本数ni时，我们可以看到计算散射矩阵实际上与计算协方差矩阵∑i相同。协方差矩阵是散射矩阵的标准化版本：

∑i=n1i sw=n1i x∑∈cd i（x−mi）（x−mi）t

>>>D=13功能数量

>>>s\_w=np.zeros（（d，d））>>>对于标签，mv in zip（范围（1，4），平均值\_vecs）：

……\_类散射=np.cov（x\_train\_std[y\_train==label].t）

……s\_w+=类\_散射

>>>print（'在类散点矩阵中缩放：%sx%s'

……%（S形[0]，S形[1]））

类内散点矩阵：13x13

在我们计算了类内散点矩阵（或协方差矩阵）的比例之后，我们可以继续下一步，计算类间散点矩阵b：

C

sb=∑ni（mi−m）（mi−m）t

I=1

这里，m是计算的总平均值，包括所有类的样本。

>>>总平均值=NP.平均值（X列标准，轴=0）

>>>D=13功能数量

>>>s\_b=np.zeros（（d，d））>>>对于i，表示枚举中的\_vec（mean\_vecs）：

……n=x[y==i+1，：]形状[0]

……mean\_vec=平均\_vec.重塑（d，1）

……mean\_overall=mean\_overall.重塑（d，1）

s\_b+=n\*（mean\_vec-mean\_overall）.dot（

【142】

……（mean\_vec-mean\_overall）.t）print（'类间散射矩阵：%sx%s'

……%（S形[0]，S形[1]））

类间散射矩阵：13x13

为新特征子空间选择线性判别法

LDA的其余步骤与PCA的步骤相似。但是，我们不在协方差矩阵上进行特征成分分析，而是解决矩阵sw-1sb的广义特征值问题：

>>>本征车辆，本征车辆=\

…np.linalg.eig（np.linalg.inv（s\_w）.dot（s\_b））。

计算特征值对后，我们现在可以按降序对特征值进行排序：

>>>Eigen\_Pairs=[（np.abs（Eigen\_Vals[i]），Eigen\_Vecs[：，i]）

……对于范围内的i（len（本征值））]

>>>eigen\_pairs=排序（eigen\_pairs，

……key=lambda k:k[0]，reverse=true）

>>>打印（“特征值降序排列）：\n”）>>>对于特征值对：

……打印（eigen\_val[0]）

特征值递减：

643.015384346

225.086981854

1.37146633984E-13 5.68434188608E-14 4.16877714935E-14 4.16877714935E-14

3.76733516161E-14号

3.7544790902E-14号

3.7544790902E-14号

2.30295239559E-14号

2.30295239559E-14 1.9101018959E-14 3.86601693797E-16

【143】

那些更熟悉线性代数的人可能知道，d×d维协方差矩阵的秩至多可以是d-1，我们确实可以看到，我们只有两个非零的特征值（特征值3-13不完全是零，但这是由于浮点运算我麻木了）。请注意，在很少的完全共线情况下（所有对齐的采样点都位于直线上），协方差矩阵的秩为1，这将导致只有一个特征向量具有非零特征值。

为了测量线性判别器（特征向量）捕获了多少类判别信息，让我们通过减少特征值来绘制线性判别器，类似于我们在PCA部分创建的解释方差图。为了简单起见，我们将类的内容称为区分信息。

>>>TOT=总和（eigen-valu.real）

>>>discr=[（i/tot）对于已排序的i（eigen-vals.real，reverse=true）]

>>>cum\_discr=np.cumsum（discr）

>>>plt.bar（范围（1，14），discr，alpha=0.5，align='center'，

……label='individual“discriminability”'）

>>>plt.step（range（1，14），cum\_discr，where='mid'，

……label='cumulative“discriminability”'）

>>>plt.ylabel（“可辨别性”比率）

>>>plt.xlabel（'线性判别式'）

>>>plt.ylim（[-0.1，1.1]）

>>>Plt.Legend（loc='best'）

>>>plt.show（）。

正如我们在结果图中看到的，前两个线性判别法捕获了葡萄酒培训数据集中大约100%的有用信息：

【144】

现在让我们将两个最具辨别力的特征向量列叠加起来，以创建转换矩阵w：

>>>W=np.hstack（（eigen\_pairs[0][1][：，np.newaxis].real，

……本征对[1][1][：，np.newaxis].real）

>>>打印（'matrix w:\n'，w）matrix w:

[-0.0707-0.3778]

[0.0359-0.2223]

[0.0263-0.3813]

[0.1875年0.2955年]

[-0.0033 0.0143年]

[0.2328 0.0151]

[0.7719 0.2149]

[-0.0803 0.0726]

[0.0896 0.1767]

[0.1815-0.2909]

[0.0631 0.2376]

[0.3794 0.0867]

[-0.3355-0.586]]

将样本投影到新的特征空间

使用我们在上一小节中创建的转换矩阵w，我们现在可以通过将这些矩阵相乘来转换训练数据集：

x′=xw

>>>X\_train\_lda=X\_train\_std.dot（W）

>>>颜色=['R'、'B'、'G']

>>>markers=['s'、'x'、'o']>>>对于L、C、M in-zip（np.unique（y\_train）、colors、markers）：

……plt.散射（x\_train\_lda[y\_train==l，0]，

……x\_train\_lda[y\_train==l，1]，

……c=c，label=l，marker=m）

>>>plt.xlabel（'ld 1'）

>>>plt.ylabel（'ld 2'）

>>>Plt.Legend（loc='upper right'）

>>>plt.show（）。

【145】

正如我们在结果图中看到的，这三种葡萄酒类别现在在新的特征子空间中是线性可分的：

通过Scikit学习的lda

分步实施是理解LDA内部工作原理和了解LDA与PCA区别的一个很好的练习。现在，让我们来看一下SciKit中实现的lda类，了解：

>>>从sklearn.lda导入lda

>>>LDA=LDA（n\_组件=2）

>>>x\_train\_lda=lda.fit\_transform（x\_train\_std，y\_train）

接下来，让我们看看逻辑回归分类器如何在LDA转换后处理低维训练数据集：

>>>LR=LogisticRegression（）。

>>>lr=lr.fit（x\_train\_lda，y\_train）

>>>绘制决策区域（x\_train\_lda，y\_train，classifier=lr）

>>>plt.xlabel（'ld 1'）

>>>plt.ylabel（'ld 2'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

【146】

查看结果图，我们发现logistic回归模型对2类中的一个样本进行了错误分类：

通过降低正则化强度，我们可以改变决策边界，从而使逻辑回归模型正确地对训练数据集中的所有样本进行分类。但是，让我们看看测试集上的结果：

>>>x\_test\_lda=lda.transform（x\_test\_std）

>>>绘制决策区域（x\_test\_lda，y\_test，classifier=lr）

>>>plt.xlabel（'ld 1'）

>>>plt.ylabel（'ld 2'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

正如我们在结果图中看到的那样，逻辑回归分类器仅使用二维特征子空间而不是最初的13种葡萄酒特征，就能够获得对测试数据集中的样品进行分类的完美准确度评分：

【147】

非线性映射的核主成分分析

许多机器学习算法对输入数据的线性可分性进行假设。你了解到知觉器甚至需要完全线性可分离的训练数据来收敛。到目前为止，我们所讨论的其他算法都假定，缺乏完美的线性可分离性是由于噪声：Adaline、逻辑回归和（标准）支持向量机（SVM）等等。然而，如果我们处理的是非线性问题，在实际应用中可能会经常遇到，那么用于降维的线性变换技术，如PCA和LDA，可能不是最佳选择。在本节中，我们将了解PCA的内核化版本，即内核PCA，它与我们在第3章中记得的内核SVM概念有关，该章介绍了

机器学习分类器使用SciKit学习。使用核PCA，我们将学习如何将不可线性分离的数据转换成适合线性分类器的新的低维子空间。

内核函数和内核技巧

正如我们在第3章（使用scikit-learn的机器学习分类器之旅）中讨论的核心支持向量机一样，我们可以通过将非线性问题投影到一个更高维度的新特征空间中来解决，在这个空间中类可以线性地分离。将样本x∈rd转换为更高的

k维子空间，我们定义了一个非线性映射函数φ：

φ：d→k（k>>d）

【148】

我们可以把φ看作是一个函数，它创建原始特征的非线性组合，将原始的D维数据集映射到一个更大的K维特征空间。例如，如果我们有二维（d=2）的特征向量x∈rd（x是由d个特征组成的列向量），则到三维空间的潜在映射可以如下：

x=[x1，x2]t

↓φ

Z=x12，2x1x2，x2 2\_t

换句话说，通过核PCA，我们执行一个非线性映射，将数据转换为一个更高的维空间，并使用这个更高维空间中的标准PCA将数据投影回一个较低的维空间，在这个空间中，样本可以通过一个线性分类器（在在输入空间中，可以用密度分离样本的条件。然而，这种方法的一个缺点是计算成本非常高，这就是我们使用内核技巧的地方。利用核技巧，我们可以计算原始特征空间中两个高维特征向量之间的相似性。

在我们继续使用内核技巧来解决这个计算上昂贵的问题之前，让我们回顾一下在本章开头实现的标准PCA方法。我们计算了两个特征k和j之间的协方差，如下所示：

σjk=−

例如，由于特征的标准化使它们的中心值为零，因此我们可以将该方程简化为：

σjk=1n∑i=n1 x（ji）xk（i）

【149】

注意，前面的方程是指两个特征之间的协方差；现在，让我们编写一个通用方程来计算协方差矩阵∑：

∑=1∑n x（i）x（i）t n i=1

Bernhard Scholkopf推广了这种方法（B.Scholkopf、A.Smola和K.-R.Muller。核主成分分析。第583–588页，1997），以便我们可以通过φ将原始特征空间中样本之间的点积替换为非线性特征组合：

T

为了从协方差矩阵中获得主成分的特征向量，我们必须解出以下方程：

T

？V=λV

⇒v=1∑nφ（x（i））φ（x（i））t v=1∑n a（i）φ（x（i））。

nλi=1 n i=1

这里，λ和v是协方差矩阵的特征值和特征向量，通过提取核（相似）矩阵k的特征向量可以得到a。

我们将在下面的段落中看到。

内核矩阵的推导如下：

首先，我们把协方差矩阵写成矩阵表示法，其中φ（x）是一个n×k维矩阵：

[150]

现在，我们可以把特征向量方程写成：v a t a

因为∑v=λv，我们得到：

1φ（x）tφ（x）φ（x）t a=λφ（x）t a n

将其两边乘以φ（x），得出以下结果：

1φ（x）φ（x）tφ（x）φ（x）t a=λφ（x）φ（x）t a n

⇒1φ（x）φ（x）t a=λa n

⇒1ka=λa n

这里，k是相似性（内核）矩阵：

K=φ（x）φ（x）t

正如我们在第3章的SVM部分使用scikit-learn的机器学习分类器之旅所回顾的那样，我们使用内核技巧避免使用内核函数k显式计算样本x在φ下的成对点积，这样我们就不需要计算特征向量exp。含蓄地：

K（x（i），x（j））=φ（x（i））tφ（x（j））

换句话说，在内核PCA之后，我们得到的是已经投影到各个组件上的样本，而不是像标准PCA方法那样构造转换矩阵。基本上，内核函数（或简单的内核）可以理解为计算两个向量之间的点积的函数——相似度的度量。

【151】

最常用的谷粒如下：

多项式核：

k（x（i），x（j））=（x（i）t x（j）+θ）p

这里，θ是阈值，p是必须由用户指定的功率。

双曲正切（乙状）核：k（x（i），x（j））=tanh（ηx（i）t x（j）+θ）

我们将在下一小节的以下示例中使用的径向基函数（RBF）或高斯核：

K（x（i），x（j））=exp−x（i）2−σx2（j）2

 

它也写如下：

K（x（i），x（j））=exp（−γx（i）−x（j）2）

为了总结我们目前讨论的内容，我们可以定义以下三个步骤来实现RBF内核PCA：

1.我们计算核心（相似性）矩阵k，其中我们需要计算以下内容：

K（x（i），x（j））=exp（−γx（i）−x（j）2）

[152]

我们对每对样本都这样做：

κ（x（1），x（1））κ（x（1），x（2））κ（x（1），x（n））

k=κ（x（2），x（1））（x（2），x（2））κ（x（2），x（n））

  

\_

κ（x（n），x（1））κ（x（d），x（2））κ（x（n），x（n））

例如，如果我们的数据集包含100个训练样本，那么成对相似性的对称核矩阵将是100×100维。

我们使用以下公式将核矩阵k居中：

k′=k−1n k−k1n+1n k1n

这里，1n是一个n×n维矩阵（与内核的尺寸相同

1个

矩阵）其中所有值都等于。

N号

我们根据中心核矩阵的特征值，收集了中心核矩阵的顶部k特征向量，这些特征值按降幅排序。与标准PCA相比，特征向量不是主分量轴，而是投影到这些轴上的样本。

在这一点上，您可能想知道为什么我们需要在第二步将内核矩阵居中。我们以前假设我们使用的是标准化数据，当我们制定协方差矩阵，并通过φ将点积替换为非线性特征组合时，所有特征均为零。因此，在第二步中，核矩阵的中心变得必要，sinc我们不明确地计算新的特征空间，也不能保证新的特征空间也以零为中心。

在下一节中，我们将通过在Python中实现内核PCA，将这三个步骤付诸实施。

[153]

在Python中实现内核主成分分析

在前面的小节中，我们讨论了内核PCA背后的核心概念。现在，我们将按照总结内核PCA方法的三个步骤，在python中实现一个RBF内核PCA。使用scipy和numpy helper函数，我们将看到实现内核PCA实际上非常简单：

从scipy.space.distance import pdist，从scipy import exp到scipy.linalg import eigh import numpy as np的平方改革

def rbf\_kernel\_pca（x，gamma，n\_components）：“rbf kernel pca实现。

参数

-----------

x：numpy ndarray，shape=[n\_samples，n\_features]

gamma：RBF内核的浮点调整参数

n\_components:int要返回的主要组件的数目

退换商品

-----------

x pc:numpy ndarray，shape=[n\_samples，k\_features]

投影数据集

“”“

#计算成对平方欧几里得距离

#在MXN维度数据集中。sq\_dists=pdist（x，'sqeucliden'）

#将成对距离转换为一个正方形矩阵。mat\_sq\_dists=平方改革（sq\_dists）

k=exp（-gamma\*mat\_sq\_dists）

【154】

#将内核矩阵居中。

n=k.形状[0]

一个n=np.ones（（n，n））/n k=k-一个n.dot（k）-k.dot（一个n）+一个n.dot（k）.dot（一个n）

#从中心核矩阵中获取特征对

#numpy.eigh按排序顺序返回特征值，eigvecs=eigh（k）

#收集顶部k特征向量（投影样本）

x\_pc=np.列\_堆栈（（eigvecs[：，-i]

对于范围内的i（1，n\_分量+1）））返回x\_pc

使用RBF核PCA进行维数约简的一个缺点是我们必须指定参数γa priori。为γ找到合适的值需要实验，最好使用参数调整算法，例如网格搜索，我们将在第6章学习模型评估和超参数调整的最佳实践中详细讨论。

示例1–分离半月形状

现在，让我们将RBF\_内核PCA应用于一些非线性示例数据集。我们将首先创建一个包含100个代表两个半月形状的采样点的二维数据集：

>>>从sklearn.datasets导入make\u moons

>>>X，Y=制造月球（n\_samples=100，random\_state=123）

>>>PLT.散射（X[Y==0，0]，X[Y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>PLT.散射（X[Y==1，0]，X[Y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>plt.show（）。

【155】

为了便于说明，三角形符号的半月代表一个等级，圆形符号所描绘的半月代表另一等级的样品：

显然，这两个半月形状不是线性可分离的，我们的目标是通过核PCA展开半月，以便数据集可以作为线性分类器的合适输入。但首先，让我们看看如果我们通过标准PCA将数据集投影到主要组件上，数据集是什么样子的：

>>>来自sklearn.composition import pca

>>>SciKit\_PCA=PCA（n\_组件=2）

>>>x\_spca=scikit\_pca.fit\_transform（x）

>>>图，ax=plt.子图（nrows=1，ncols=2，figsize=（7,3））

>>>ax[0].散点图（x\_spca[y==0，0]，x\_spca[y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>ax[0].散点图（x\_spca[y==1，0]，x\_spca[y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>ax[1].散点图（x\_spca[y==0，0]，np.零（（50，1））+0.02，

……color='red'，marker='^'，alpha=0.5）

>>>ax[1].散点图（x\_spca[y==1，0]，np.零（（50，1））-0.02，…color='blue'，marker='o'，alpha=0.5）

【156】

>>>AX[0].设置xlabel（'pc1'）

>>>ax[0].设置“ylabel（'pc2'）

>>>ax[1].设置ylim（[-1，1]）

>>>AX[1].设置时钟（[]）

>>>AX[1].设置xlabel（'pc1'）

>>>plt.show（）。

很明显，我们可以在结果图中看到，线性分类器无法很好地处理通过标准PCA转换的数据集：

注意，当我们只绘制第一个主成分（右子地块）时，我们将三角形样本稍微向上移动，而圆形样本稍微向下移动，以便更好地显示类重叠。

请记住，PCA是一种无监督的方法，不使用类标签信息来最大化vare，而不是lda。在这里，三角形和圆形符号只是为了显示分离程度而添加的。

现在，让我们来试试我们的内核PCA函数rbf\_kernel\_pca，我们在前面的小节中实现了它：

>>>从matplotlib.ticker导入formatstrformatter

>>>x\_kpca=rbf\_kernel\_pca（x，gamma=15，n\_components=2）

>>>图，ax=plt.子图（nrows=1，ncols=2，figsize=（7,3））

>>>ax[0].散点图（x\_kpca[y==0，0]，x\_kpca[y==0，1]，

……color='red'，marker='^'，alpha=0.5）>>>ax[0].散点图（x\_kpca[y==1，0]，x\_kpca[y==1，1]，

【157】

……color='blue'，marker='o'，alpha=0.5）

>>>ax[1].散点图（x\_kpca[y==0，0]，np.零（（50，1））+0.02，

……color='red'，marker='^'，alpha=0.5）

>>>ax[1].散射（x\_kpca[y==1，0]，np.零（（50，1））-0.02，

……color='blue'，marker='o'，alpha=0.5）

>>>AX[0].设置xlabel（'pc1'）

>>>ax[0].设置“ylabel（'pc2'）

>>>ax[1].设置ylim（[-1，1]）

>>>AX[1].设置时钟（[]）

>>>AX[1].设置xlabel（'pc1'）

>>>AX[0].xaxis.set\_major\_formatter（formatstrformatter（'%0.1f'））

>>>AX[1].xaxis.set\_major\_formatter（formatstrformatter（'%0.1f'））

>>>plt.show（）。

现在我们可以看到这两个类（圆和三角形）是线性的，并且它们之间的距离很好，因此它成为线性分类器的合适训练数据集：

不幸的是，对于不同的数据集，没有适用于调整参数γ的通用值。要找到适合给定问题的γ值，需要进行实验。在第6章，学习模型评估和超参数调优的最佳实践中，我们将讨论能够帮助我们自动化优化此类调优参数的任务的技术。这里，我将使用我发现产生良好结果的γ值。

【158】

示例2-分离同心圆

在前面的小节中，我们向您展示了如何通过核PCA分离半月形状。既然我们在理解核PCA的概念上做了很多努力，那么让我们来看一个非线性问题的另一个有趣的例子：同心圆。

代码如下：

>>>从sklearn.datasets导入make\_circles

>>>X，Y=生成\_圆（n\_samples=1000，

……随机状态=123，噪声=0.1，系数=0.2）

>>>PLT.散射（X[Y==0，0]，X[Y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>PLT.散射（X[Y==1，0]，X[Y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>plt.show（）。

同样，我们假设一个两类问题，三角形代表一个类，圆形代表另一个类：

【159】

让我们从标准PCA方法开始，将其与RBF内核PCA的结果进行比较：

>>>SciKit\_PCA=PCA（n\_组件=2）

>>>x\_spca=scikit\_pca.fit\_transform（x）

>>>图，ax=plt.子图（nrows=1，ncols=2，figsize=（7,3））

>>>ax[0].散点图（x\_spca[y==0，0]，x\_spca[y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>ax[0].散点图（x\_spca[y==1，0]，x\_spca[y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>ax[1].散点图（x\_spca[y==0，0]，np.零（（500，1））+0.02，

……color='red'，marker='^'，alpha=0.5）

>>>ax[1].散点图（x\_spca[y==1，0]，np.零（（500，1））-0.02，

……color='blue'，marker='o'，alpha=0.5）

>>>AX[0].设置xlabel（'pc1'）

>>>ax[0].设置“ylabel（'pc2'）

>>>ax[1].设置ylim（[-1，1]）

>>>AX[1].设置时钟（[]）

>>>AX[1].设置xlabel（'pc1'）

>>>plt.show（）。

同样，我们可以看到，标准PCA无法产生适合训练线性分类器的结果：

[160]

给定γ的适当值，让我们看看使用RBF内核PCA实现是否更幸运：

>>>x\_kpca=rbf\_kernel\_pca（x，gamma=15，n\_components=2）

>>>图，ax=plt.子图（nrows=1，ncols=2，figsize=（7,3））

>>>ax[0].散点图（x\_kpca[y==0，0]，x\_kpca[y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>ax[0].散点图（x\_kpca[y==1，0]，x\_kpca[y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>ax[1].散点图（x\_kpca[y==0，0]，np.零（（500，1））+0.02，

……color='red'，marker='^'，alpha=0.5）

>>>ax[1].散射（x\_kpca[y==1，0]，np.零（（500，1））-0.02，

……color='blue'，marker='o'，alpha=0.5）

>>>AX[0].设置xlabel（'pc1'）

>>>ax[0].设置“ylabel（'pc2'）

>>>ax[1].设置ylim（[-1，1]）

>>>AX[1].设置时钟（[]）

>>>AX[1].设置xlabel（'pc1'）

>>>plt.show（）。

同样，RBF内核PCA将数据投影到一个新的子空间中，在该子空间中，两个类可以线性分离：

【161】

投影新数据点

在前面两个核心PCA的示例应用中，半月形状和同心圆，我们将一个数据集投影到一个新特性上。然而，在实际的应用程序中，我们可能有不止一个要转换的数据集，例如，培训和测试数据，通常还有在模型构建和评估之后收集的新样本。在本节中，您将学习如何投影不属于培训数据集的数据点。

我们记得，在本章开头的标准PCA方法中，我们通过计算转换矩阵和输入样本之间的点积来投影数据；投影矩阵的列是我们从协方差矩阵获得的顶部k特征向量（v）。现在，问题是我们如何将这个概念转移到核PCA？如果我们回想一下核PCA背后的思想，我们记得我们得到了中心核矩阵（而不是协方差矩阵）的特征向量（A），这意味着这些样本已经投射到主分量轴V上。因此，如果我们想将一个新的样本x'投射到这个主分量轴上，我们需要计算出以下公式：φ（x’）t v

幸运的是，我们可以使用内核技巧，这样就不必显式计算投影φ（x’）t v。然而，值得注意的是，与标准PCA相比，内核PCA是一种基于内存的方法，这意味着我们必须每次重复使用原始的训练集来投射新的样本。我们必须计算训练数据集中每个第i个样本与新样本x′之间的成对RBF内核（相似性）：

φ（x’）t v=∑a（i）φ（x’）tφ（x（i））

我

=∑a（i）k（x’，x（i））t

我

这里，核矩阵k的特征向量a和特征值λ满足方程中的以下条件：

ka=λa

[162]

在计算了新样本与训练集样本之间的相似性之后，我们必须通过特征值对特征向量A进行归一化。因此，让我们修改前面实现的rbf\_kernel\_pca函数，以便它也返回内核矩阵的特征值：

从scipy.space.distance import pdist，从scipy import exp到scipy.linalg import eigh import numpy as np的平方改革

def rbf\_kernel\_pca（x，gamma，n\_components）：“rbf kernel pca实现。

参数

-----------

x：numpy ndarray，shape=[n\_samples，n\_features]

gamma：RBF内核的浮点调整参数

n\_components:int要返回的主要组件的数目

退换商品

-----------

x pc:numpy ndarray，shape=[n\_samples，k\_features]

投影数据集

lambdas:列出特征值

“”“

#计算成对平方欧几里得距离

#在MXN维度数据集中。sq\_dists=pdist（x，'sqeucliden'）

#将成对距离转换为一个正方形矩阵。mat\_sq\_dists=平方改革（sq\_dists）

#计算对称核矩阵。

k=exp（-gamma\*mat\_sq\_dists）

【163】

#将内核矩阵居中。

n=k.形状[0]

一个n=np.ones（（n，n））/n k=k-一个n.dot（k）-k.dot（一个n）+一个n.dot（k）.dot（一个n）

#从中心核矩阵中获取特征对

#numpy.eigh按排序顺序返回特征值，eigvecs=eigh（k）

#收集顶部k个特征向量（投影样本）alphas=np.列\_堆栈（i在范围（1，n\_分量+1））中的i的特征向量[：，-i]）

#收集相应的特征值

lambdas=[特征值[-i]对于范围内的i（1，n\_分量+1）]返回alphas，lambdas

现在，让我们创建一个新的半月数据集，并使用更新的RBF内核PCA实现将其投影到一维子空间：

>>>X，Y=制造月球（n\_samples=100，random\_state=123）

>>>alphas，lambdas=rbf\_kernel\_pca（x，gamma=15，n\_components=1）为了确保我们实现新样本的投影代码，假设半月数据集的第26个点是新的数据点x′，我们的任务是将其投影到这个新的子空间：

>>>X\_new=X[25]

>>>X\_新

数组（[1.8713187，0.00928245]）

>>>X\_proj=alphas[25]原始投影

>>>X项目阵列（[0.07877284]）

>>>def项目\_x（x\_new，x，gamma，alphas，lambdas）：

……对距离=np.数组（[np.和（

……（X\_新行）\*\*2）对于X]中的行）

……k=np.exp（-gamma\*对距离）

……返回k.dot（alphas/lambdas）

【164】

通过执行以下代码，我们能够重现原始投影。使用project\_x函数，我们还可以投射任何新的数据样本。代码如下：

>>>x\_reproj=项目x（x\_new，x，

……gamma=15，alphas=alphas，lambdas=lambdas）

>>>X撊reproj array（[0.07877284]）

最后，让我们可视化第一个主要组件上的投影：

>>>PLT.散射（alphas[y==0，0]，np.零（（50）），

……color='red'，marker='^'，alpha=0.5）

>>>PLT.散射（alphas[y==1，0]，np.零（（50）），

……color='blue'，marker='o'，alpha=0.5）

>>>请分散（x\_proj，0，color='black'，

……label='点X的原始投影[25]，

……marker='^'，s=100）

>>>PLT.散射（x\_reproj，0，color='green'，

……label='remapped point x[25]'，

……marker='x'，s=500）

>>>PLT.图例（散点=1）

>>>plt.show（）。

正如我们在下面的散点图中看到的，我们正确地将样本x'映射到第一个主分量上：

【165】

Scikit学习中的核主成分分析

为了方便起见，scikit-learn在sklearn.deposition子模块中实现了一个内核PCA类。用法类似于标准PCA类，我们可以通过kernel参数指定kernel：

>>>来自sklearn.composition导入kernelpca

>>>X，Y=制造月球（n\_samples=100，random\_state=123）

>>>SciKit\_kpca=Kernelpca（n\_components=2，

……kernel='rbf'，gamma=15）

>>>X诳skernpca=SciKit诳kpca.Fit诳Transform（X）

为了查看我们得到的结果是否与我们自己的内核PCA实现一致，让我们将转换后的半月形状数据绘制到前两个主要组件上：

>>>PLT.散点图（X诳skernpca[Y==0，0]，X诳skernpca[Y==0，1]，

……color='red'，marker='^'，alpha=0.5）

>>>PLT.散点图（X诳skernpca[Y==1，0]，X诳skernpca[Y==1，1]，

……color='blue'，marker='o'，alpha=0.5）

>>>plt.xlabel（'pc1'）

>>>plt.ylabel（'pc2'）

>>>plt.show（）。

【166】

Scikit Learn还实现了本书范围之外的非线性维数减少的高级技术。你可以找到一个很好的当前概况

SciKit中的实现将在中补充说明性示例。

总结

在本章中，您学习了三种用于特征提取的基本维数约简技术：标准PCA、LDA和内核PCA。使用PCA，我们将数据投影到一个低维子空间，以最大化沿正交特征轴的方差，同时忽略类标签。与PCA相比，LDA是一种监督降维技术，即在训练数据集中考虑类信息，以最大限度地提高线性特征空间中的类可分性。最后，您了解了PCA的内核化版本，它允许您将非线性数据集映射到一个低维特征空间，在那里类可以线性分离。

有了这些基本的预处理技术，您现在已经做好充分的准备，可以在下一章中了解有效地结合不同的预处理技术和评估不同模型性能的最佳实践。

[167]

学习模型评估和超参数调整的最佳实践

在前面的章节中，您学习了分类的基本机器学习算法，以及在将数据输入到这些算法之前如何使数据成形。现在，是时候学习如何通过微调算法和评估模型性能来构建良好的机器学习模型的最佳实践了！在本章中，我们将学习如何：

获得模型性能的无偏估计

机器学习算法常见问题的诊断

微调机器学习模型

使用不同的性能指标评估预测模型

使用管道简化工作流

当我们在前几章中应用不同的预处理技术时，例如第4章中的特征缩放标准化、构建良好的训练集（数据预处理）或第5章中数据压缩的主成分分析（通过降维压缩数据）。，您了解到，我们必须重用在拟合训练数据时获得的参数，以缩放和压缩任何新数据，例如，单独测试数据集中的样本。在本节中，您将学习一个非常方便的工具，SciKit学习中的Pipeline类。它允许我们拟合一个模型，包括任意数量的转换步骤，并应用它来预测新数据。

【169】

载入威斯康星乳腺癌数据集

在本章中，我们将使用威斯康星乳腺癌数据集，其中包含569个恶性和良性肿瘤细胞样本。数据集中的前两列分别存储样本的唯一ID号和相应的诊断（M=恶性，B=良性）。第3-32列包含30个真实值特征，这些特征是根据细胞核的数字化图像计算得出的，可用于建立一个模型来预测肿瘤是良性还是恶性。威斯康星州乳腺癌数据集已存放在UCI上。

有关此数据集的更多详细信息，请访问

在本节中，我们将读取数据集，并将其分为培训和测试数据集，分为三个简单步骤：

我们将从直接从UCI网站上读取数据集开始，使用熊猫：

>>>将熊猫作为PD导入

>>>df=pd.read\_csv（'https://archive.ics.uci.edu/ml/machineLearning databases/乳腺癌wisconsin/wdbc.data'，header=none）

接下来，我们将30个特性分配给一个numpy数组x。使用labelencoder，我们将类标签从其原始字符串表示形式（m和b）转换为整数：

>>>来自sklearn.preprocessing import labelencoder

>>>X=df.loc[：，2:..值

>>>Y=df.loc[：，1].值

>>>LE=Labelencoder（）。

>>>Y=le.fit\_变换（Y）

在数组y中对类标签（诊断）进行编码后，恶性肿瘤现在表示为1类，良性肿瘤分别表示为0类，我们可以通过调用两个虚拟类标签上的labelencoder转换方法来说明：

>>>le.transform（['m'，'b']）数组（[1，0]）

【170】

三。在我们在下面的小节中构建第一个模型管道之前，让我们将数据集分为单独的训练数据集（80%的数据）和单独的测试数据集（20%的数据）：

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>X轴、X轴、Y轴、Y轴测试=\

……列车测试分割（x，y，测试尺寸=0.20，随机状态=1）

管道中变压器和估计器的组合

在上一章中，您了解到许多学习算法都需要相同规模的输入特性来获得最佳性能。因此，我们需要标准化乳腺癌威斯康星数据集中的列，然后才能将它们输入线性分类器，如逻辑回归。此外，假设我们希望通过主成分分析（PCA）将最初30维的数据压缩到一个较低的二维子空间，这是我们在第5章中介绍的一种用于降维的特征提取技术，即通过维数r压缩数据。教育。我们不需要分别对培训和测试数据集进行拟合和转换步骤，而是可以将标准缩放器、PCA和逻辑回归对象链接到一个管道中：

>>>来自sklearn.preprocessing import standardscaler

>>>来自sklearn.composition import pca

>>>来自sklearn.linear\_model import logisticregression

>>>来自sklearn.pipeline导入pipeline

>>>pipe\_lr=管道（[（'scl'，standardscaler（）），

……（‘PCA’，PCA（n\_分量=2）），

……（‘CLF’，逻辑回归（随机状态=1）））

>>>管道安装（X轴、Y轴）

>>>print（'测试精度：%.3f'%管道\u.score（x\_测试，y\_测试））

测试精度：0.947

pipeline对象接受一个元组列表作为输入，其中每个元组中的第一个值是一个任意的标识符字符串，我们可以使用它来访问管道中的各个元素，正如我们在本章后面将看到的那样，每个元组中的第二个元素是一个scikit学习转换器或估计量。

【171】

管道中的中间步骤构成了Scikit学习变压器，最后一步是估计器。在前面的代码示例中，我们构建了一个由两个中间步骤组成的管道，一个标准定标器和一个PCA转换器，以及一个逻辑回归分类器作为最终估计。当我们在管道上执行拟合方法时，标准定标器对训练数据进行拟合和转换，然后将转换后的训练数据传递到管道中的下一个对象PCA。与前一步类似，PCA也对按比例缩放的输入数据执行拟合和转换，并将其传递给管道的最终元素估计量。我们应该注意到，在这个管道中，中间步骤的数量没有限制。管道如何工作的概念概括在下图中：

【172】

使用k-折叠交叉验证评估模型性能

建立机器学习模型的关键步骤之一是根据模型以前没有见过的数据来估计其性能。让我们假设我们的模型适合一个训练数据集，并使用相同的数据来估计它在实践中的表现。我们记得在第3章的“通过正则化处理过度拟合”一节中，使用scikit-learn的机器学习分类器之旅中，如果模型过于简单，模型可能会出现欠匹配（高偏差），或者如果模型为对于基础培训数据来说太复杂。为了找到一个可接受的偏差方差权衡，我们需要仔细评估我们的模型。在本节中，您将了解到有用的交叉验证技术holdout cross validation和k-fold cross validation，这有助于我们获得模型的泛化错误的可靠估计，也就是说，模型对未知数据的性能。

维持方法

对于机器学习模型的泛化性能评估，一种经典而流行的方法是保持交叉验证。使用维持方法，我们将初始数据集分成单独的训练和测试数据集，前者用于模型训练，后者用于评估其性能。然而，在典型的机器学习应用程序中，我们也对调整和比较不同的参数设置感兴趣，以进一步提高对未知数据进行预测的性能。这个过程称为模型选择，其中术语模型选择指的是一个给定的分类问题，我们要为该问题选择优化参数的最佳值（也称为超参数）。但是，如果我们在模型选择过程中反复使用相同的测试数据集，那么它将成为我们培训数据的一部分，因此模型将更容易过度拟合。尽管如此，许多人仍然使用测试集进行模型选择，这不是一个好的机器学习实践。

【173】

使用维持方法进行模型选择的更好方法是将数据分为三部分：训练集、验证集和测试集。训练集用于适应不同的模型，然后将验证集的性能用于模型选择。在培训和模型选择步骤中，拥有一个模型以前从未见过的测试集的好处是，我们可以获得一个对其概括为新数据能力的较少偏向的估计。下图说明了维持交叉验证的概念，其中我们使用验证集在培训后使用不同的参数值重复评估模型的性能。一旦我们对参数值的调整感到满意，我们就在测试数据集上估计模型的泛化误差：

维持方法的一个缺点是，性能估计对我们如何将训练集划分为训练和验证子集很敏感；对于不同的数据样本，估计值会有所不同。在下一小节中，我们将研究一种更强大的性能估计技术，k倍交叉验证，在这里我们对训练数据的k个子集重复保持方法k次。

【174】

交叉检验

在k-折叠交叉验证中，我们将训练数据集随机拆分为k折叠而不替换，其中k-1折叠用于模型训练，一个折叠用于测试。这个过程重复k次，因此我们得到k模型和性能估计。

如果您不熟悉“采样”和“不采样”这两个术语，我们来进行一个简单的思维实验。假设我们在玩一个抽奖游戏，从骨灰盒中随机抽取数字。我们从一个能容纳5个唯一数字0、1、2、3和4的瓮开始，并且每一圈只画一个数字。在第一轮中，从骨灰盒中提取特定数字的机会是1/5。现在，在不更换的采样中，我们不会在每次旋转后将编号放回瓮中。因此，在下一轮中从剩余数字集合中提取特定数字的概率取决于前一轮。例如，如果我们还有一组数字0、1、2和4，那么在下一个回合中绘制数字0的机会将变成1/4。

但是，在替换随机抽样中，我们总是将绘制的数字返回到urn，这样在每个回合绘制特定数字的概率不会改变；我们可以多次绘制相同的数字。换句话说，在替换抽样中，样本（数字）是独立的，协方差为零。例如，五轮随机数绘制的结果如下：

不更换的随机抽样：2、1、3、4、0

随机抽样替换：1、3、3、4、1

然后，我们根据不同的、独立的折叠计算模型的平均性能，以获得一个性能估计，与维持方法相比，该估计对训练数据的子分区不太敏感。通常，我们使用k-折叠交叉验证进行模型优化，也就是说，找到产生令人满意的泛化性能的最佳超参数值。一旦我们找到了令人满意的超参数值，我们就可以在完整的训练集上重新训练模型，并使用独立的测试集获得最终的性能估计。

【175】

由于k-fold交叉验证是一种无需替换的重采样技术，因此该方法的优点是每个采样点都将恰好一次成为训练和测试数据集的一部分，从而产生比holdout方法更低的模型性能方差估计。下图总结了k=10的k次交叉验证背后的概念。训练数据集分为10个折叠，在10次迭代中，9个折叠用于训练，1个折叠用作模型评估的测试集。此外，每个折叠的估计性能ei（例如分类精度或误差）用于计算模型的估计平均性能e：

k-折叠交叉验证中k的标准值是10，这对于大多数应用来说通常是一个合理的选择。但是，如果我们使用的是相对较小的训练集，那么增加折叠的数量是很有用的。如果我们增加k的值，那么在每次迭代中将使用更多的训练数据，这就降低了通过平均单个模型估计值来估计泛化性能的偏差。但是，k的大值也会增加交叉验证算法的运行时间，并且由于训练折叠之间的相似性，因此产生更大的方差。另一方面，如果我们处理的是大数据集，我们可以为k选择一个较小的值，例如k=5，并且仍然可以获得模型平均性能的精确估计。

同时降低了模型在不同褶皱上的重装和评价的计算成本。

【176】

K-折叠交叉验证的一个特殊情况是“左一出（loo）”交叉验证方法。在loo中，我们将折叠的数量设置为训练样本的数量（k=n），以便在每次迭代期间只使用一个训练样本进行测试。这是处理非常小的数据集的推荐方法。

与标准的k倍交叉验证方法相比，稍微有一点改进的是分层k倍交叉验证，这可以产生更好的偏差和方差估计，特别是在阶级比例不相等的情况下，如R.Kohavi等人的一项研究所示。（R.Kohavi等人一个交叉验证和引导精度估计和模型选择的研究。在Ijcai，第14卷，第1137-1145页，1995年）。在分层交叉验证中，每个折叠中都保留了类比例，以确保每个折叠都代表培训数据集中的类比例，我们将通过使用SciKit学习中的分层折叠迭代器来说明：

>>>导入numpy为np

>>>从sklearn.cross\_验证导入分层折叠

>>>KFOD=分层折叠（Y=Y\_火车，

……n\_折叠=10，

……随机状态=1）

>>>在Enumerate（Kfold）中k（训练、测试）的得分=[]>>>>：

……管道安装（X列[列车]、Y列[列车]）

……得分=管道得分（X\_列车[试验]，Y\_列车[试验]）

……分数。附加（分数）

……print（'折叠：%s，类分布：%s，acc:%.3f'%（k+1，

……Np.BinCount（Y\_列车[列车]），得分）

折叠：1，等级距离：【256 153】，acc:0.891

折叠：2，等级距离：【256 153】，acc:0.978

折叠：3，等级距离：【256 153】，acc:0.978

折叠：4，等级距离：【256 153】，acc:0.913

折叠：5，等级距离：【256 153】，acc:0.935

折叠：6，等级距离：【257 153】，acc:0.978

折叠：7，等级距离：【257 153】，acc:0.933

折叠：8，等级距离：【257 153】，acc:0.956

折叠：9，等级距离：【257 153】，acc:0.978

折叠：10，等级距离：【257 153】，acc:0.956

>>>打印（‘cv精度：%.3f+/-%.3f%’（

……NP.平均（分数），NP.标准（分数）））

cv精度：0.950+/-0.029

【177】

首先，我们用训练集中的类标签y\_train初始化sklearn.cross\_验证模块中的分层折叠迭代器，并通过n\_folds参数指定折叠数。当我们使用Kfold迭代器循环遍历k折叠时，我们使用train中返回的索引来适应我们在本章开头设置的逻辑回归管道。使用Pile\_lr管道，我们确保在每次迭代中正确缩放样本（例如，标准化）。然后，我们使用测试指标计算模型的准确度得分，我们收集在得分列表中计算平均准确度和估计的标准差。

尽管前面的代码示例有助于说明k-fold交叉验证的工作原理，但SciKit Learn还实现了k-fold交叉验证记分器，这使我们能够更有效地使用分层k-fold交叉验证来评估模型：

>>>来自sklearn.cross\_验证导入cross\_val\_分数

>>>分数=交叉值分数（估计值=管道分数，

……X=X\_列车，

……Y=Y\_列车，

……cv=10，

……n\_作业=1）

>>>print（'cv准确度分数：%s'%分数）

简历准确度得分：【0.89130435 0.97826087 0.97826087】

0.91304348 0.93478261 0.97777778

0.93333333 0.955555556 0.97777778

0.95555556条]

>>>打印（‘cv准确度：%.3f+/-%.3f’%（np.平均（分数），np.标准（分数）））cv准确度：0.950+/-0.029

Cross-Val-Score方法的一个非常有用的特性是，我们可以在机器上的多个CPU上分布不同折叠的评估。如果我们将n\_jobs参数设置为1，那么将只使用一个CPU来评估性能，就像前面的分层示例中那样。但是，通过设置n\_jobs=2，我们可以将10轮交叉验证分布到两个CPU（如果我们的机器上可用），并且通过设置n\_jobs=-1，我们可以使用机器上所有可用的CPU并行进行计算。

【178】

请注意，在交叉验证中如何估计泛化性能的方差的详细讨论超出了本书的范围，但是您可以在M.Markatou等人（M.Markatou、H.Tian、S.Biswa和G.M.Hricpsak）的这篇优秀文章中找到详细讨论。广义误差交叉验证估计量方差分析。机器学习研究杂志，6:1127–1168，2005）。

您还可以阅读其他交叉验证技术，例如.632引导交叉验证方法（b.efron和r.tibshirani）。交叉验证的改进：632+引导方法。美国统计协会杂志，92（438）：548-5601997）。

用学习和验证曲线调试算法

在本节中，我们将介绍两个非常简单但功能强大的诊断工具，它们可以帮助我们改进学习算法的性能：学习曲线和验证曲线。在下一小节中，我们将讨论如何使用学习曲线来诊断学习算法是否存在过拟合（高方差）或欠匹配（高偏差）问题。此外，我们将看一看验证曲线，它可以帮助我们解决学习算法的常见问题。

【179】

用学习曲线诊断偏差和方差问题

如果一个模型对于一个给定的训练数据集来说太复杂，那么这个模型中的自由度或参数太多，那么这个模型倾向于过度拟合训练数据，并且不能很好地概括为未看到的数据。通常，它可以帮助收集更多的训练样本，以减少过度拟合的程度。然而，在实践中，收集更多的数据往往非常昂贵或根本不可行。通过绘制模型训练和验证精度作为训练集大小的函数，我们可以很容易地检测模型是否存在高方差或高偏差，以及收集更多的数据是否有助于解决这个问题。但是，在讨论如何在SCKIT Learn中绘制学习曲线之前，让我们通过以下示例来讨论这两个常见的模型问题：

[180]

左上角的图表显示了一个高偏差的模型。该模型训练精度和交叉验证精度均较低，说明其训练数据不足。解决这个问题的常见方法是增加模型的参数数量，例如，通过收集或构造附加的特征，或者通过减少正则化程度，例如，在SVM或逻辑回归分类器中。右上角的图表显示了一个具有高方差的模型，这是由训练和交叉验证精度之间的巨大差距表示的。为了解决这种过度拟合的问题，我们可以收集更多的训练数据或降低模型的复杂性，例如，通过增加正则化参数；对于非规范化模型，它还可以通过特征选择帮助减少特征数量（第4章，构建良好的tra）。Ining集合-数据预处理）或特征提取（第5章，通过降维压缩数据）。我们应该注意到，收集更多的训练数据会减少过度拟合的可能性。然而，当训练数据非常嘈杂或模型已经非常接近最优时，这可能并不总是有帮助的。

在下一小节中，我们将看到如何使用验证曲线来解决这些模型问题，但首先让我们看看如何使用Scikit Learn中的Learning curve函数来评估模型：

>>>导入matplotlib.pyplot为plt

>>>从sklearn.learning\_curve导入learning\_curve

>>>pipe\_lr=管道（[

……（'scl'，standardscaler（）），

……（‘CLF’，逻辑回归（

……惩罚='l2'，随机状态=0））]

>>>训练尺寸、训练分数、测试分数=\

……学习曲线（估计量=管道

……X=X\_列车，

……Y=Y\_列车，

……列车尺寸=np.linspace（0.1、1.0、10）

……cv=10，

……n\_作业=1）

>>>列车平均值=np.平均值（列车分数，轴=1）

>>>列车标准=NP.标准（列车得分，轴=1）

>>>测试平均值=np.平均值（测试分数，轴=1）

>>>测试标准=NP.标准（测试分数，轴=1）

>>>plt.plot（列车尺寸，列车意思是，

……color='blue'，marker='o'，

……markersize=5，

……label='训练精度'）

>>>请在（列车尺寸，

……列车平均值+列车标准，

……火车-火车标准，

【181】

……alpha=0.15，颜色为蓝色）

>>>plt.plot（列车尺寸，测试意味着，

……color='green'，linestyle='--'，

……marker='s'，markersize=5，

……label='验证准确性'）

>>>请在（列车尺寸，

……测试平均值+测试标准，

……测试平均值-测试标准，

……α=0.15，颜色=绿色）

>>>plt.grid（）。

>>>plt.xlabel（'培训样本数'）

>>>plt.ylabel（'准确度'）

>>>Plt.Legend（loc='右下'）

>>>plt.ylim（[0.8，1.0]）

>>>plt.show（）。

在成功执行上述代码后，我们将获得以下学习曲线图：

[182]

通过学习曲线函数中的训练尺寸参数，我们可以控制用于生成学习曲线的训练样本的绝对或相对数量。这里，我们将列车尺寸设置为np.linspace（0.1，1.0，10），以使用10个均匀间隔的相对间隔作为列车尺寸。默认情况下，学习曲线函数使用分层k倍交叉验证来计算交叉验证精度，我们通过cv参数设置k=10。然后，我们简单地从返回的交叉验证训练和测试分数中计算出不同规模训练集的平均精度，我们使用matplotlib的绘图函数绘制了这些训练集。此外，我们使用Fill\_Between函数将平均精度的标准偏差添加到绘图中，以指示估计的方差。

正如我们在前面的学习曲线图中看到的，我们的模型在测试数据集上表现得相当好。然而，它可能稍微过拟合训练数据，表明训练和交叉验证精度曲线之间的差距相对较小，但可见。

用验证曲线解决过拟合和欠拟合问题

验证曲线是一个有用的工具，可以通过解决诸如过拟合或欠拟合等问题来提高模型的性能。验证曲线与学习曲线有关，但我们不将训练和测试精度作为样本大小的函数来绘制，而是改变模型参数的值，例如逻辑回归中的逆正则化参数c。让我们来看看我们如何通过SCKIT创建验证曲线，了解：

>>>来自sklearn.learning\_curve import validation\_curve

>>>参数范围=[0.001，0.01，0.1，1.0，10.0，100.0]

>>>训练分数，测试分数=验证曲线（

……Estimator=管道

……X=X\_列车，

……Y=Y\_列车，

……param\_name='clf\_uuu c'，

……参数范围=参数范围，

……cv=10）

>>>列车平均值=np.平均值（列车分数，轴=1）

>>>列车标准=NP.标准（列车得分，轴=1）

>>>测试平均值=np.平均值（测试分数，轴=1）

>>>测试标准=NP.标准（测试分数，轴=1）

【183】

>>>plt.plot（参数范围，列车意思是，

……color='blue'，marker='o'，

……markersize=5，

……label='训练精度'）

>>>请在（参数范围，列车均值+列车标准，

……列车平均值-列车标准，α=0.15，

……color='blue'）

>>>plt.plot（参数范围，测试含义，

……color='green'，linestyle='--'，

……marker='s'，markersize=5，

……label='验证准确性'）

>>>请在（参数范围，

……测试平均值+测试标准，

……测试平均值-测试标准，

……α=0.15，颜色=绿色）

>>>plt.grid（）。

>>>plt.xscale（'log'）

>>>Plt.Legend（loc='右下'）

>>>plt.xlabel（'参数C'）

>>>plt.ylabel（'准确度'）

>>>plt.ylim（[0.8，1.0]）

>>>请显示（））

利用前面的代码，我们得到了参数c的验证曲线图：

【184】

与学习曲线函数类似，如果我们使用分类算法，则验证曲线函数默认使用分层k倍交叉验证来估计模型的性能。在验证曲线函数中，我们指定了要评估的参数。在这种情况下，它是逻辑回归分类器的逆正则化参数c，我们将其写为“clf\_uuu c”，以访问SciKit学习管道中通过参数range参数设置的指定值范围的逻辑回归对象。与前一节中的学习曲线示例类似，我们绘制了平均培训和交叉验证精度以及相应的标准偏差。

虽然不同C值的精度差异很小，但我们可以看到，当我们增加正则化强度（C的小值）时，模型稍微低估了数据。然而，对于较大的C值，它意味着降低了正则化的强度，因此该模型倾向于稍微过拟合数据。在这种情况下，最佳点似乎在c=0.1左右。

通过网格搜索微调机器学习模型

在机器学习中，我们有两种类型的参数：从训练数据中学习的参数，例如逻辑回归中的权重，以及单独优化的学习算法的参数。后者是模型的调整参数，也称为超参数，例如逻辑回归中的正则化参数或决策树的深度参数。

在前一节中，我们使用验证曲线通过调整模型的一个超参数来提高模型的性能。在本节中，我们将介绍一种称为网格搜索的强大超参数优化技术，它可以通过找到超参数值的最佳组合来进一步帮助改进模型的性能。

【185】

通过网格搜索调整超参数

网格搜索的方法非常简单，它是一种强力穷尽的搜索范式，我们为不同的超参数指定一个值列表，计算机对这些值的每一个组合的模型性能进行评估，以获得最佳集：

>>>来自sklearn.grid\_search import gridsearchcv

>>>从sklearn.svm导入svc

>>>pipe\_svc=管道（[（'scl'，standardscaler（）），

……（'clf'，svc（random\_state=1）））

>>>参数范围=[0.0001，0.001，0.01，0.1，1.0，10.0，100.0，1000.0]

>>>参数网格=['clf\_uuu c'：参数范围，

…'clf\_uu内核“：[”线性“]，

……'clf\_uuu c'：参数范围，

…'clf\_uuu gamma：参数范围，

…'clf\_uuu内核“：['rbf']]

>>>GS=GridSearchCV（Estimator=Pipe\_Svc，

……参数网格=参数网格，

……得分='准确度'，

……cv=10，

……n\_作业=-1）

>>>GS=GS.fit（X\_train，Y\_train）

>>>打印（GS.最佳分数）

0.978021978022>>打印（gs.best\_u参数）

'clf\_uu c'：0.1，'clf\_u kernel'：'linear'

使用前面的代码，我们初始化了sklearn.grid\_搜索模块中的gridsearchcv对象，以训练和优化支持向量机（SVM）管道。我们将gridsearchcv的param\_grid参数设置为字典列表，以指定要优化的参数。对于线性支持向量机，我们只评估逆正则化参数c；对于RBF核支持向量机，我们同时调整了c和gamma参数。注意gamma参数是特定于内核SVM的。在使用训练数据进行网格搜索后，我们通过best-score\_uu属性获得了最佳性能模型的得分，并查看了其参数，这些参数可以通过best-params\_u属性访问。在这种特殊情况下，“clf\_uuu c”=0.1”的线性支持向量机模型产生了最佳的k倍交叉验证精度：97.8%。

[186年]

最后，我们将使用独立的测试数据集来估计最佳选择模型的性能，该模型通过gridsearchcv对象的最佳评估器属性提供：

>>>clf=gs.best\_估计量\_

>>>clf.fit（x\_train，y\_train）

>>>print（'测试精度：%.3f'%clf.score（x\_test，y\_test））。

测试精度：0.965

虽然网格搜索是寻找最佳参数集的一种有效方法，但是对所有可能的参数组合的评估也非常昂贵。使用Scikit Learn对不同参数组合进行抽样的另一种方法是随机搜索。使用Scikit Learn中的RandomzedSearchcv类，我们可以从具有指定预算的采样分布中提取随机参数组合。有关其用法的更多详细信息和示例，请参阅。

嵌套交叉验证算法选择

将k-折叠交叉验证与网格搜索结合使用是一种很有用的方法，可以通过改变机器学习模型的超参数值来微调机器学习模型的性能，如前一小节所示。如果我们想在不同的机器学习算法中进行选择，另一个推荐的方法是嵌套交叉验证，在一个关于误差估计偏差的研究中，Varma和Simon得出结论，当嵌套时，估计的真实误差几乎与测试集无偏。使用交叉验证（S.Varma和R.Simon。使用交叉验证进行模型选择时的误差估计偏差。BMC生物信息学，7（1）：91，2006）。

[187年]

在嵌套交叉验证中，我们有一个外部k-折叠交叉验证循环，将数据拆分为训练和测试折叠，并使用一个内部循环来选择在训练折叠上使用k-折叠交叉验证的模型。选择模型后，使用测试折叠来评估模型性能。下图说明了具有五个外部折叠和两个内部折叠的嵌套交叉验证的概念，这对于计算性能很重要的大型数据集很有用；这种特定类型的嵌套交叉验证也称为5x2交叉验证：

在SciKit学习中，我们可以执行嵌套交叉验证，如下所示：

>>>GS=GridSearchCV（Estimator=Pipe\_Svc，

……参数网格=参数网格，

……得分='准确度'，

……cv=10，

……n\_作业=-1）

>>>得分=交叉得分（GS，X，Y，scoring='accuracy'，cv=5）

>>>打印（‘cv精度：%.3f+/-%.3f%’（

……NP.平均（分数），NP.标准（分数）））

cv精度：0.978+/-0.012

【188】

返回的平均交叉验证精度为我们提供了一个很好的估计，如果我们调整模型的超参数，然后在未看到的数据上使用它，将会发生什么。例如，我们可以使用嵌套交叉验证方法将SVM模型与简单的决策树分类器进行比较；为了简单起见，我们只调整其深度参数：

>>>从sklearn.tree导入decisiontreeclassifier

>>>GS=GridSearchCV（

……Estimator=DecisionTreeClassifier（随机状态=0），

……参数网格=[

……'Max\_Depth'：[1，2，3，4，5，6，7，none]]，

……得分='准确度'，

……cv=5）

>>>分数=交叉值分数（Gs，

……X列，

……你的火车，

……得分='准确度'，

……cv=5）

>>>打印（‘cv精度：%.3f+/-%.3f%’（

……NP.平均（分数），NP.标准（分数）））

cv精度：0.908+/-0.045

正如我们在这里看到的，SVM模型的嵌套交叉验证性能（97.8%）明显优于决策树的性能（90.8%）。因此，我们希望它可能是对来自与此特定数据集相同总体的新数据进行分类的更好选择。

查看不同的绩效评估指标

在前面的章节中，我们使用模型精度来评估我们的模型，这是一个很有用的度量标准，通常可以量化模型的性能。但是，还有其他几个性能指标可用于衡量模型的相关性，例如精度、召回和F1分数。

【189】

读取混淆矩阵

在我们了解不同评分标准的细节之前，让我们先打印一个所谓的混淆矩阵，一个列出学习算法性能的矩阵。混淆矩阵只是一个方形矩阵，它报告分类器的真正、真负、假正和假负预测的计数，如下图所示：

虽然可以通过比较真实和预测的类标签轻松地手动计算这些度量，但是SciKit Learn提供了一个方便的混淆矩阵函数，我们可以使用如下所示：

>>>从sklearn.metrics导入混淆矩阵

>>>管道SVC.fit（x\_train，y\_train）

>>>y\_pred=管道\u svc.predict（x\_测试）

>>>confmat=混淆矩阵（y\_true=y\_test，y\_pred=y\_pred）

>>>打印（confmat）

【71 1】

[2 40]]

执行上述代码后返回的数组为我们提供了有关测试数据集上分类器所产生的不同类型错误的信息，我们可以使用matplotlib的matshow函数将这些错误映射到上一图中的混淆矩阵图中：

>>>图，ax=plt.子图（图尺寸=（2.5，2.5））

>>>ax.matshow（confmat，cmap=plt.cm.blues，alpha=0.3）>>>对于范围内的i（confmat.shape[0]）：

……对于范围内的J（confmat.shape[1]）：

……ax.文本（x=j，y=i，

……S=confmat[i，j]，

……va='center'，ha='center'）

【190】

>>>plt.xlabel（'预测标签'）

>>>plt.ylabel（'真标签'）

>>>plt.show（）。

现在，这里所示的混淆矩阵图应该使结果更容易解释：

假设在本例中1类（恶性）为阳性，我们的模型正确地分类了0类（假阴性）的71个样本和1类（真阳性）的40个样本。然而，我们的模型也错误地将0级的2个样本分类为1级（假阴性），并且预测1个样本是良性的，尽管它是恶性肿瘤（假阳性）。在下一节中，我们将学习如何使用这些信息来计算各种不同的错误度量。

优化分类模型的精度和召回

预测误差（err）和准确度（acc）都提供了有关多少样本被错误分类的一般信息。误差可以理解为所有错误预测的和除以总预测数，准确度计算为正确预测的和除以总预测数，分别为：

错误=fp+fn

Fp+Fn+Tp+Tn

[191年]

然后可以直接从误差中计算预测精度：

acc=tp+tn=1−err

Fp+Fn+Tp+Tn

真阳性率（TPR）和假阳性率（FPR）是对不平衡类问题特别有用的性能指标：

fpr=fp=fp

N fp+tn

tpr=tp=tp

p-fn+tp

例如，在肿瘤诊断中，我们更关注恶性肿瘤的检测，以帮助患者进行适当的治疗。然而，减少被错误地归类为恶性（假阳性）的良性肿瘤的数量也很重要，不要不必要地关注患者。与FPR相比，真正的阳性率提供了有关阳性（或相关）样本在总阳性（P）池中正确识别的部分的有用信息。

精确性（pre）和召回（rec）是与真实阳性率和真实阴性率相关的绩效指标，事实上，召回是真实阳性率的同义词：

pre=tp tp+fp

rec=tpr=tp=tp

p-fn+tp

在实践中，通常采用精确性和回忆相结合的方法，即所谓的F1分数：

f1=2 pre×rec

预+回收

【192】

这些评分指标都在SciKit Learn中实现，可以从skLearn.metrics模块导入，如下所示：

>>>来自sklearn.metrics import precision\_score

>>>来自sklearn.metrics import recall\_score，f1\_score

>>>打印（'精度：%.3f'%精度分数（

……y\_真=y\_检验，y\_pred=y\_pred）

精度：0.976

>>>print（'召回：%.3f'%召回分数（

……y\_真=y\_检验，y\_pred=y\_pred）

召回：0.952

>>>打印（'f1:%.3f'%f1分数（

……y\_真=y\_检验，y\_pred=y\_pred）

一层：0.964

此外，我们可以通过评分参数在网格搜索中使用不同的评分指标，而不是精度。评分参数接受的不同值的完整列表可在

记住，SciKit Learn中的正类是标记为Class1的类。如果我们想要指定一个不同的正面标签，我们可以通过make\_scorer函数构造我们自己的记分员，然后我们可以直接将其作为参数提供给gridsearchcv中的记分参数：

>>>从sklearn.metrics导入make\_scorer，f1\_score

>>>得分者=得分者（F1得分，位置标签=0）

>>>GS=GridSearchCV（Estimator=Pipe\_Svc，

……参数网格=参数网格，

……得分=得分者，

……cv=10）

绘制接收机工作特性

接收算子特征（ROC）图是根据模型对假阳性率和真阳性率的性能来选择分类模型的有用工具，假阳性率和真阳性率是通过改变分类器的决策阈值来计算的。ROC图的对角线可以解释为随机猜测，低于对角线的分类模型被认为比随机猜测更糟糕。一个完美的分类器将落在图的左上角，真阳性率为1，假阳性率为0。基于ROC曲线，我们可以计算所谓的曲线下面积（AUC），来表征分类模型的性能。

[193]

与ROC曲线类似，我们可以计算分类器不同概率阈值的精确召回曲线。绘图功能

这些精确的回忆曲线也在Scikit Learn中实现，并且

记录于

通过执行以下代码示例，我们将绘制一个分类器的ROC曲线，该分类器仅使用来自威斯康星乳腺癌数据集的两个特征来预测肿瘤是良性还是恶性。虽然我们将使用与前面定义的相同的逻辑回归管道，但我们正在使分类任务对分类器更具挑战性，从而使生成的ROC曲线在视觉上变得更有趣。出于类似的原因，我们还将分层折叠验证器中的折叠数减少到三个。代码如下：

>>>从sklearn.metrics导入roc\_曲线，auc

>>>从scipy导入interp

>>>X诳train2=X诳train[：，[4，14]]

>>>cv=分层折叠（y\_train，

……n\_折叠=3，

……随机状态=1）

>>>图=PLT.图（图尺寸=（7，5））

>>>平均温度=0.0

>>>平均值\_fpr=np.linspace（0，1，100）

>>>所有=[]

>>>对于i，（训练，测试）在枚举（cv）中：

……Probas=管接头（X轴2[传动系]，

>>>Y\_train[训练]。预测\_proba（X\_train2[测试]）…fpr，tpr，thresholds=roc\_曲线（y\_列[测试]，

[194年]

……Probas[：，1]，

……位置标签=1）

……平均\_-tpr+=interp（平均\_-fpr、fpr、tpr）

……平均温度=0.0

……roc\_auc=auc（fpr，tpr）

……plt.绘图（fpr，

……TPR，

……Lw=1，

……label='roc fold%d（区域=%0.2f）'

……%（I+1，Roc\_-Auc）

>>>plt.绘图（[0，1]，

……[0，1]，

……lineStyle=--'，

……颜色=（0.6、0.6、0.6）

……label='随机猜测'）

>>>平均温度/=len（cv）

>>>平均温度[-1]=1.0

>>>mean\_auc=auc（mean\_fpr，mean\_tpr）

>>>plt.plot（mean\_fpr，mean\_tpr，'k--'，

……label='mean roc（area=%0.2f）'%平均值，lw=2）

>>>plt.绘图（[0，0，1]，

……[0，1，1]，

……Lw=2，

……lineStyle=''，

……color='black'，

……label='完美性能'）

>>>plt.xlim（[-0.05，1.05]）

>>>plt.ylim（[-0.05，1.05]）

>>>plt.xlabel（'假阳性率'）

>>>plt.ylabel（'真阳性率'）

>>>plt.title（'接收器操作员特征'）

>>>Plt.Legend（loc=“Lower Right”）。

>>>plt.show（）。

[195]年

在前面的代码示例中，我们使用了Scikit Learn中已经熟悉的分层折叠类，并使用sklearn.metrics模块中的roc-curve函数分别为每个迭代计算了pipe-lr管道中logisticregression分类器的roc性能。此外，我们通过从scipy导入的interp函数从三个褶皱中插入平均roc曲线，并通过auc函数计算曲线下的面积。得出的ROC曲线表明，不同褶皱之间存在一定程度的差异，平均ROC AUC（0.75）介于完美分数（1.0）和随机猜测（0.5）之间：

如果我们只是对ROC AUC分数感兴趣，我们还可以直接从sklearn.metrics子模块导入ROC\_AUC\_分数函数。以下代码在将分类器与两个特性训练集相匹配后，计算独立测试数据集上分类器的ROC AUC得分：

>>>管道\u svc=管道\u svc.fit（x\_train2，y\_train）>>y\_pred2=管道\u svc.predict（x\_test[：，[4，14]]）

[196年]

>>>从sklearn.metrics导入roc\_auc\_分数

>>>来自sklearn.metrics import accuracy\_score

>>>打印（‘ROC AUC:%.3f’%ROC AUC评分（

……y\_真=y\_检验，y\_得分=y\_pred2）

ROC AUC:0.671

>>>打印（'准确度：%.3f'%准确度\得分（

……y\_真=y\_检验，y\_pred=y\_pred2）

精度：0.728

将分类器的性能报告为ROC AUC可以进一步了解分类器对不平衡样本的性能。然而，虽然准确度得分可以解释为ROC曲线上的一个单一临界点，但A.P.布拉德利表明，ROC AUC和准确度指标基本一致（A.P.布拉德利）。在机器学习算法评估中使用ROC曲线下的面积。模式识别，30（7）：1145–1159，1997）。

多类分类的评分指标

我们在本节中讨论的评分指标是特定于二进制分类系统的。然而，Scikit Learn还实现了宏观和微观平均方法，通过一对所有（ova）分类将这些评分指标扩展到多类问题。微平均值是根据系统的单个真阳性、真阴性、假阳性和假阴性计算的。例如，K级系统的精密度得分的微观平均值可计算如下：

预聚物=TP1+…+TPTP1 K+…+FPTP 1+K…+FP K

宏观平均值简单计算为不同系统的平均分数：pre macro=pre1+…k+pre k

如果我们希望对每个实例或预测进行平均权重，则微观平均非常有用，而宏观平均对所有类进行平均权重，以评估分类器相对于最频繁的类标签的总体性能。

【197】

如果我们使用二进制性能度量来评估SciKit Learn中的多类分类模型，那么默认情况下使用宏平均值的规范化或加权变量。加权宏平均值是通过计算平均值时按真实实例数加权每个类标签的得分来计算的。如果我们要处理类不平衡（即每个标签的实例数不同），加权宏平均值是有用的。

虽然加权宏平均值是SciKit学习中多类问题的默认值，但我们可以通过从sklean.metrics模块导入的不同评分函数中的平均参数指定平均方法，例如，精度评分或生成评分函数：

>>>pre\_scorer=得分者（score\_func=精确分数，

……位置标签=1，

……越大越好=真的，

……平均值='micro'）

总结

在本章的开头，我们讨论了如何在方便的模型管道中链接不同的转换技术和分类器，从而帮助我们更有效地培训和评估机器学习模型。然后，我们使用这些管道执行k-折叠交叉验证，这是模型选择和评估的基本技术之一。利用k次交叉验证，绘制了学习和验证曲线，对学习算法中常见的过拟合和欠拟合问题进行了诊断。使用网格搜索，我们进一步优化了我们的模型。我们通过查看一个混淆矩阵和各种不同的性能指标来结束本章，这些指标对于进一步优化特定问题任务的模型性能非常有用。现在，我们应该很好地掌握必要的技术，成功地建立监督机器学习模型进行分类。

在下一章中，我们将介绍集成方法，这些方法允许我们将多个模型和分类算法结合起来，以进一步提高机器学习系统的预测性能。

[198]

组合不同的集成学习模型

在前一章中，我们重点介绍了优化和评估不同分类模型的最佳实践。在本章中，我们将以这些技术为基础，探索构建一组分类器的不同方法，这些分类器通常比任何单个成员都具有更好的预测性能。您将学习如何：

基于多数投票做出预测

通过随机抽取重复训练集的组合来减少过度拟合。

从从错误中吸取教训的弱者身上建立强有力的榜样

合奏学习

集成方法的目标是将不同的分类器组合成一个元分类器，它比单个分类器具有更好的泛化性能。例如，假设我们收集了10位专家的预测，集合方法将允许我们战略性地将10位专家的这些预测结合起来，得出比每个专家的预测更准确和可靠的预测。正如我们将在本章后面看到的，有几种不同的方法来创建分类器的集合。在本节中，我们将介绍一种基本的理解，即合奏如何工作，以及为什么它们通常被认为能够产生良好的泛化性能。

[一九九]

在本章中，我们将重点介绍使用多数投票原则的最流行的合奏方法。多数投票仅仅意味着我们选择了由大多数分类器预测的类标签，即收到超过50%的投票。严格来说，“多数票”一词仅指二进制类设置。然而，很容易将多数投票原则归纳为多类设置，即所谓的多投票。在这里，我们选择获得最多投票的类标签（模式）。下图说明了10个分类器集合的多数和多数投票的概念，其中每个唯一符号（三角形、正方形和圆形）代表唯一的类标签：

使用训练集，我们首先训练M个不同的分类器（c1，，cm）。根据这一技术，集成可以从不同的分类算法中构建，例如决策树、支持向量机、逻辑回归分类器等。或者，我们也可以使用相同的基本分类算法来拟合训练集的不同子集。这种方法的一个突出例子是随机森林算法，它结合了不同的决策树分类器。下图说明了使用多数投票的一般合奏方法的概念：

[200]

为了通过简单多数或多个投票来预测类标签，我们结合每个分类器CJ的预测类标签并选择获得最多投票的类标签y\_：

y\_=模式c1（x），c2（x），，cm（x）

例如，在Class1=-1和Class2=+1的二进制分类任务中，我们可以编写如下多数票预测：

C（x）=符号∑MJ CJ（x）=−11其他wisif∑I CJ E（x）≥0

为了说明为什么集成方法比单独的分类器更有效，让我们应用组合数学的简单概念。在下面的例子中，我们假设一个二进制分类任务的所有n个基分类器都有一个相等的错误率ε。此外，我们假设分类器是独立的，错误率不相关。在这些假设下，我们可以简单地将基本分类器集合的误差概率表示为二项式分布的概率质量函数：

p（y≥k）=∑n nkεk（1−ε）n−k=ε系综

K

N号

这里，k是二项式系数n，选择k。换句话说，我们计算了合集预测错误的概率。现在让我们来看一个更具体的例子，11个基本分类器（n=11），错误率为0.25（ε=0.25）：

P（Y≥K）=∑N 11 0.25K（1−ε）11−K=0.034 K=6 K

【201】

如我们所见，在满足所有假设的情况下，合集（0.034）的错误率远低于每个分类器（0.25）的错误率。注意，在这个简化的示例中，由偶数个分类器n分割的50-50被视为一个错误，而这只是正确的一半时间。为了在不同的基本错误率范围内将这种理想的集合分类器与基本分类器进行比较，让我们在python中实现概率质量函数：

>>>来自scipy.misc导入梳

>>>导入数学>>def ensegle\_error（n\_classifier，error）：

……k\_start=math.ceil（n\_分类器/2.0）

……probs=[梳（n\_分类器，k）\*

……错误\*\*K\*

……（1-错误）\*\*（n\_分类器-k）

……对于范围内的k（k\_开始，n\_分类器+1）]

……收益总额（probs）

>>>整体误差（n\_分类器=11，误差=0.25）

0.034327507019042969

在我们实现了系综误差函数之后，我们可以计算0.0到1.0范围内不同基本误差的系综误差率，从而在一个线性图中可视化系综与基本误差之间的关系：

>>>导入numpy为np

>>>错误范围=np.arange（0.0，1.01，0.01）

>>>ens\_错误=[集合\_错误（n\_分类器=11，错误=错误）

……错误范围内的错误]

>>>导入matplotlib.pyplot为plt

>>>plt.plot（错误范围，ens错误，

……label='集合错误'，

……线条宽度=2）

>>>plt.plot（错误范围，错误范围，

……linestyle=--，label='base错误'，

……线条宽度=2）

>>>plt.xlabel（'基本错误'）

>>>plt.ylabel（'基本/集合错误'）

>>>Plt.Legend（loc='左上角'）

>>>plt.grid（）。

>>>plt.show（）。

正如我们在结果图中看到的那样，只要基分类器比随机猜测（ε<0.5）执行得更好，那么集合的错误概率总是比单个基分类器的错误概率好。注意，Y轴表示基本误差（虚线）和系综误差（连续线）：

【202】

实现简单的多数票分类器

在上一节中简短介绍了集成学习之后，让我们从一个热身练习开始，并实现一个简单的集成分类器，用于Python中的多数投票。虽然下面的算法也通过多投票归纳为多类设置，但我们将使用术语“多数投票”来简化，正如文献中经常做的那样。

我们将要实现的算法将允许我们结合与单个权重相关的不同分类算法以获得置信度。我们的目标是建立一个更强的元分类器，以平衡单个分类器在特定数据集上的弱点。在更精确的数学术语中，我们可以写下加权多数票如下：

m y\_=argmaxi∑j=1 wjχa（cj（x）=i）

【203】

这里，wj是一个与基分类器相关的权重，cj，y\_是集合的预测类标签，χa（希腊语chi）是特征函数cj（x）=i∈a，a是唯一类标签集。对于等量，我们可以简化这个方程，并将其写成：y\_=模式c1（x），c2（x），，cm（x）

为了更好地理解权重的概念，我们现在将看一个更具体的例子。假设我们有一个三个基本分类器的集合cj（j∈0,1），并且想要预测给定样本实例x的类标签。三个基本分类器中的两个预测类标签0，一个c3预测样本属于类1。如果我们平均地对每个基本分类器的预测进行加权，多数票将预测样本属于0类：

c1（x）→0，c2（x）→0，c3（x）→1 y=模式0,0,1=0

现在让我们分别将0.6的重量分配给c3，将c1和c2的重量分配给系数0.2。

m y\_=argmaxi∑j=1 wjχa（cj（x）=i）

=argmaxi[0.2×i0+0.2×i0+0.6×i1]=1

更直观地说，由于3×0.2=0.6，我们可以说，c3的预测比c1或c2的预测分别重三倍。我们可以这样写：

y\_=模式0,0,1,1,1=1

【204】

为了将加权多数投票的概念转换为python代码，我们可以使用numpy方便的argmax和bincount函数：

>>>导入numpy为np

>>>np.argmax（np.bincount（[0，0，1]，

……权重=[0.2、0.2、0.6]）

1个

如第3章所讨论的，使用scikit-learn的机器学习分类器之旅，scikit-learn中的某些分类器也可以通过predict-proba方法返回预测类标签的概率。如果我们集合中的分类器经过了很好的校准，那么使用预测类概率而不是类标签进行多数投票是很有用的。根据概率预测类标签的多数票修改版本可写如下：

m y\_=argmaxi∑j=1 wj pij

这里，pij是类标签i的jth分类器的预测概率。为了继续前面的例子，假设我们有一个类标签i∈0,1和三个分类器cj（j∈1,2,3）的二元分类问题。假设分类器cj返回特定样本x的以下类成员概率：

c1（x）→[0.9,0.1]、c2（x）→[0.8,0.2]、c3（x）→[0.4,0.6]

然后，我们可以计算出单个类的概率：p（i0 x）=0.2×0.9+0.2×0.8+0.6×0.4=0.58 p（i1 x）=0.2×0.1+0.2×0.2+0.6×0.06=0.42 y\_=argmaxi p（i0 x），p（i1 x）=0.

【205】

为了实现基于类概率的加权多数投票，我们可以再次使用numpy.average和np.argmax:

>>>ex=np.array（[[0.9，0.1]，

……[0.8，0.2]，

……[0.4，0.6]）

>>>P=NP.平均值（例如，轴=0，权重=[0.2，0.2，0.6]）

>>>P

数组（[0.58，0.42]）

>>>np.argmax（P）

0

把所有东西放在一起，现在让我们在python中实现一个主要的voteclassifier：

从sklean.base导入baseEstimator从sklean.base导入分类器mixin从sklean.preprocessing导入labelencoder从sklean.externals导入6从sklean.base导入克隆从sklean.pipeline导入\_name\_Estimators导入numpy作为np导入运算符

大类Voteclassifier（baseEstimator，ClassifierMixin）：

“多数票集合分类器

参数---------

分类器：类似数组的，shape=[n\_分类器]集合的不同分类器

投票：str，'classlabel'、'probability'

默认值：“ClassLabel”

如果“classlabel”，则预测基于类标签的argmax。否则如果

“概率”，概率总和的argmax用于预测类标签（推荐用于已校准的分类器）。

权重：数组状，形状=[n\_分类器]

可选，默认值：无

如果“int”或“float”值的列表是

[206]

前提是，分类器按重要性加权；如果“weights=none”，则使用统一的权重。

“”“

def uu init\_uuu（self，分类器，vote='classlabel'，weights=none）：

self.classifiers=分类器self.name\_classifiers=key:键的值，中的值

\_ name\_estimators（分类器）self.vote=投票self.weights=权重

def fit（self，x，y）：“匹配分类器。

参数

---------

x：类数组，稀疏矩阵，shape=[n\_samples，n\_features]训练样本矩阵。

y：数组状，形状=目标类标签的[n\_samples]向量。

返回------self:object

“”“

#使用labelencoder确保类标签开始

#其中0对np.argmax很重要

#为self.classifiers中的clf调用self.predict self.labelnc=labelencoder（）self.labelnc.fit（y）self.classes=self.labelnc.classes u.self.classifiers=[]：

fitted\_clf=clone（clf）.fit（x，self.labelnc\_.transform（y））self.classifiers\_.append（fitted\_clf）返回self

【207】

为了更好地理解各个部分，我在代码中添加了很多注释。然而，在我们实现其余方法之前，让我们快速地休息一下，首先讨论一些看起来可能令人困惑的代码。我们使用父类baseEstimator和ClassifierMixin免费获得一些基本功能，包括get\_-params和set\_-params方法来设置和返回分类器的参数，以及score方法来分别计算预测精度。还要注意，我们导入了六个函数，使majorityvoteclassifier与python 2.7兼容。

下一步，如果我们用vote='class label'初始化一个新的majorityvoteclassifier对象，我们将添加predict方法，通过基于类标签的多数投票来预测类标签。或者，我们可以使用vote='probability'初始化集合分类器，根据类成员概率预测类标签。此外，我们还将添加一个预测概率法来返回平均概率，这对于计算曲线下的接收算子特征面积（ROC AUC）很有用。

def predict（self，x）：“”预测x的类标签。

参数

---------

x：类似数组，稀疏矩阵，

shape=训练样本的矩阵。

退换商品

---------

Maj\_投票：类似数组，shape=[n\_samples]预测类标签。

“if self.vote='probability'：maj\_vote=np.argmax（self.predict\_proba（x），

轴=1）其他：“ClassLabel”投票

#从clf.predict calls predictions=np.asarray（[clf.predict（x）for clf in self.classifiers”）.t收集结果。

Maj\_Vote=np.沿\_轴应用\_（lambda x:

np.argmax（np.bincount（x，

【208】

weights=self.weights），axis=1，arr=predictions）maj\_-vote=self.lablnc\_u.inverse\_-transform（maj\_-vote）返回maj\_-vote

def predict\_Proba（self，x）：“预测x的类概率。

参数

---------

x：类数组，稀疏矩阵，形状=[n\_样本，n\_特征]

训练向量，其中n\_samples是样本数，n\_features是特征数。

退换商品

---------avg\_proba:数组状，shape=[n\_samples，n\_classes]每个样本的每个类的加权平均概率。

“”“

probas=np.asarray（[clf.predict ou proba（x）for clf in self.classifiers\_u]）avg ou proba=np.average（probas，

轴=0，权重=self.weights）返回平均概率

def get\_params（self，deep=true）：“获取用于GridSearch的分类器参数名”如果不是deep:

返回super（majorityvoteclassifier，self）。获取\_参数（deep=false）否则：

out=self.named\_classifiers.copy（）for name，step in \six.iteritems（self.named\_classifiers）：

对于键，六个.iteritems中的值（step.get\_params（deep=true））：out[”%s\_uu%s“%（name，key）]=值返回out

[209]

另外，请注意，我们定义了自己的get-params方法的修改版本，以使用“name”估计函数来访问集合中单个分类器的参数。起初，这看起来有点复杂，但在后面的部分中，当我们使用网格搜索进行超参数优化时，这将是非常有意义的。

虽然我们的MajorityVoteclassifier实现对于演示非常有用，但是我在Scikit Learn中也实现了更复杂版本的多数投票分类器。它

将在下一个版本（v0.17）中作为sklearn.ensegle.votingClassifier提供。

将不同的分类算法与多数投票相结合

现在是时候将我们在前一节中实现的主要voteclassifier投入使用了。但首先，让我们准备一个可以在其上测试的数据集。由于我们已经熟悉从csv文件加载数据集的技术，我们将采取一种快捷方式，从scikit learn的数据集模块加载iris数据集。此外，我们将只选择两个特征，即萼片宽度和花瓣长度，使分类任务更具挑战性。虽然我们的大多数voteclassifier归纳为多类问题，但我们只将花色鸢尾和弗吉尼亚鸢尾这两类花的样本分类，以计算ROC AUC。代码如下：

>>>从sklearn导入数据集

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>来自sklearn.preprocessing import standardscaler

>>>来自sklearn.preprocessing import labelencoder

>>>iris=datasets.load\_iris（）。

>>>x，y=iris.data[50:，[1，2]]，iris.target[50:]

>>>LE=Labelencoder（）。

>>>Y=le.fit\_变换（Y）

【210】

请注意，Scikit Learn使用Predict\_Proba方法（如果适用）计算ROC AUC分数。在第三章，机器学习分类器使用Scikit学习，我们看到了如何在逻辑回归模型中计算类概率。在决策树中，概率是根据训练时为每个节点创建的频率向量计算的。矢量收集每个类标签的频率值，这些值是根据该节点上的类标签分布计算得出的。然后，对频率进行归一化，使其总和为1。同样，k-最近邻的类标签被聚合，以返回k-最近邻算法中的标准化类标签频率。尽管决策树和k-最近邻分类器返回的归一化概率看起来可能与逻辑回归模型中获得的概率相似，但我们必须注意，这些概率实际上不是从概率质量函数推导出来的。

接下来，我们将虹膜样本分成50%的培训和50%的测试数据：

>>>X轴、X轴、Y轴、Y轴测试=\

……列车测试分割（x，y，

……试验尺寸=0.5，

……随机状态=1）

使用训练数据集，我们现在将训练三个不同的分类器——逻辑回归分类器、决策树分类器和K-最近邻分类器，并在合并它们之前，通过训练数据集的10倍交叉验证来查看它们的个人表现。集成分类器：

>>>来自sklearn.cross\_验证导入cross\_val\_分数

>>>来自sklearn.linear\_model import logisticregression

>>>从sklearn.tree导入decisiontreeclassifier

>>>从sklearn.neighbors导入Kneighbors分类器

>>>来自sklearn.pipeline导入pipeline

>>>导入numpy为np

>>>clf1=逻辑回归（惩罚='l2'，

……C=0.001，

……随机状态=0）

>>>CLF2=决策树分类器（最大深度=1，

……准则='熵'，

……随机状态=0）

>>>CLF3=Kneighbors分类器（n\_neighbors=1，

……P=2，

……metric='minkowski'）

>>>pipe1=管道（['sc'，standardscaler（）]，

……['CLF'，CLF1]）

【211】

>>>pipe3=管道（['sc'，standardscaler（）]，

……['CLF'，CLF3]）

>>>clf\_labels=['logistic regression'、'decision tree'、'knn']

>>>打印（“10倍交叉验证：\n”）>>>对于CLF，在zip中添加标签（[pipe1，clf2，pipe3]，clf\_标签）：

……分数=交叉值分数（估计值=CLF，

>>>X=X\_列车，

>>>Y=Y\_列车，

>>>cv=10，

>>>得分='roc\_u auc'）

>>>打印（“ROC AUC:%0.2f（+/-%0.2f）[%s]”

……%（scores.mean（），scores.std（），label）

我们收到的输出，如下面的代码片段所示，表明各个分类器的预测性能几乎相等：

10倍交叉验证：

ROC AUC:0.92（+/-0.20）[逻辑回归]

ROC AUC:0.92（+/-0.15）[决策树]

ROC AUC:0.93（+/-0.10）[knn]

您可能想知道为什么我们将逻辑回归和k-最近邻分类器作为管道的一部分进行了培训。其背后的原因是，正如第3章所讨论的，使用SciKit Learn的机器学习分类器巡演，逻辑回归和k-最近邻算法（使用欧几里得距离度量）与决策树相比不是尺度不变的。虽然虹膜特征都是以相同的比例（cm）测量的，但是使用标准化特征是一个好习惯。

现在，让我们继续讨论更令人兴奋的部分，并结合在我们的大数voteclassifier中用于多数规则投票的单个分类器：

>>>mv\_clf=主要voteclassifier（

……分类器=[管道1、CLF2、管道3]）

>>>CLF U Labels+=['多数投票']

>>>所有clf=[pipe1，clf2，pipe3，mv clf]>>>对于clf，在zip中添加标签（所有clf，clf \_标签）：

……分数=交叉值分数（估计值=CLF，

……X=X\_列车，

……Y=Y\_列车，

……cv=10，

……scoring='roc\_u auc'）

……print（“精度：%0.2f（+/-%0.2f）[%s]”..%（scores.mean（），scores.std（），label）

【212】

ROC AUC:0.92（+/-0.20）[逻辑回归]

ROC AUC:0.92（+/-0.15）[决策树]

ROC AUC:0.93（+/-0.10）[knn]

ROC AUC:0.97（+/-0.10）[多数投票]

如我们所见，在10倍交叉验证评估中，主投票分类器的性能比单个分类器显著提高。

评价和调整集成分类器

在本节中，我们将从测试集中计算ROC曲线，以检查主要voteclassifier是否很好地概括为未看到的数据。我们应该记住，测试集不能用于模型选择；它的唯一目的是报告分类器系统的泛化性能的无偏估计。代码如下：

>>>来自sklearn.metrics import roc\_curve>>来自sklearn.metrics import auc

>>>颜色=['black'、'orange'、'blue'、'green']

>>>linestyles=['：'、'--'、'-.'、'-']>>>对于clf、label、clr、ls\…在zip中（所有标签、颜色、线条样式）：

……#假设正类的标签是1…y\_pred=clf.fit（x\_列车，

……y\_train.预测\_proba（x\_test）[：，1]…fpr，tpr，thresholds=roc\_曲线（y\_true=y\_测试，

……y\_分数=y\_pred）

……roc\_auc=auc（x=fpr，y=tpr）……plt.绘图（fpr，tpr，

……颜色=clr，

……linestyle=ls，

……label='s（auc=%0.2f）'%（label，roc\_auc）

>>>Plt.Legend（loc='Lower Right'）>>>Plt.Plot（[0，1]，[0，1]，

……lineStyle=--'，

……color='gray'，…线条宽度=2）>>>plt.xlim（[-0.1，1.1]）

>>>plt.ylim（[-0.1，1.1]）>>>plt.grid（）。

>>>plt.xlabel（'假阳性率'）

>>>plt.ylabel（“真阳性率”）>>>plt.show（）。

【213】

正如我们在得到的ROC中所看到的，在测试集上集成分类器也表现良好（ROC AUC=0.95），而k-最近邻分类器似乎过度拟合了训练数据（训练ROC AUC=0.93，测试ROC AUC=0.86）：

因为我们只为分类示例选择了两个特性，所以看到集成分类器的决策区域实际上是什么样子会很有趣。虽然在模型拟合之前不需要标准化训练特性，因为我们的逻辑回归和k-最近邻管道将自动处理这些特性，但我们将标准化训练集，使决策树的决策区域相同。用于视觉目的的比例。代码如下：

>>>SC=标准缩放器（）

>>>x\_train\_std=sc.fit\_transform（x\_train）

>>>从ITertools导入产品

>>>x\_min=x\_train\_std[：，0].min（）-1

>>>x\_max=x\_train\_std[：，0].max（）+1

>>>Y\_min=X\_train\_std[：，1].min（）-1

>>>Y\_max=X\_train\_std[：，1].max（）+1

【214】

>>>x x，yy=np.meshgrid（np.arange（x\_min，x\_max，0.1），

……np.arange（y\_最小，y\_最大，0.1）

>>>F，axarr=plt.子批次（nrows=2，ncols=2，

……sharex='col'，

……shary='行'，

……图尺寸=（7，5）

>>>对于IDX、CLF、TT In-Zip（产品（[0，1]，[0，1]），…所有标签：

……clf.fit（x\_train\_std，y\_train）

……z=clf.predict（np.c\_[xx.ravel（），yy.ravel（）]）

……Z=Z.重塑（XX.形状）

……axarr[idx[0]，idx[1]].轮廓（xx，yy，z，alpha=0.3）

……axarr[idx[0]，idx[1]].散射（x\_train\_std[y\_train==0，0]，

……x\_train\_std[y\_train==0，1]，

……C='蓝色'，

……标记='^'，

……S=50）

……axarr[idx[0]，idx[1]].散射（x\_train\_std[y\_train==1，0]，

……x\_train\_std[y\_train==1，1]，

……C='红色'，

……marker='o'，

……S=50）

……axarr[idx[0]，idx[1]].设置标题（TT）

>>>plt.文本（-3.5，-4.5，

……s='sepal width[标准化]，

……ha='center'，va='center'，fontsize=12）

>>>plt.文本（-10.5，4.5，

……s='花瓣长度[标准化]，

……ha='center'，va='center'，

……字体大小=12，旋转角度=90）

>>>plt.show（）。

【215】

有趣的是，正如预期的那样，集合分类器的决策区域似乎是来自单个分类器的决策区域的混合。乍一看，多数投票决策边界看起来很像K-最近邻分类器的决策边界。然而，我们可以看到，对于sepal宽度≥1，它与y轴是正交的，就像决策树树桩：

在学习如何为集合分类调整各个分类器参数之前，让我们调用get\_params方法来了解如何访问gridsearch对象中的各个参数：

>>>mv\_clf.get\_params（）。

'decisiontreeclassifier'：decisiontreeclassifier（class\_weight=none，criteria='entropy'，max\_depth=1，

max\_features=none，max\_leaf\_nodes=none，min\_samples\_leaf=1，

min\_samples\_split=2，min\_weight\_fraction\_leaf=0.0，random\_state=0，splitter='best'），'decisiontreeclassifier\_uu class\_weight'：无，

'decisiontreeclassifier\_uuu criteria'：'entropy'，

[…]

“DecisionTreeClassifier\_uu Random\_State”：0，

'decisiontreeclassifier\_uuu splitter'：'best'，

[216]

“pipeline-1”：pipeline（steps=[（'sc'，standardscaler（copy=true，with\_mean=true，with\_std=true）），（'clf'，logisticregression（c=0.001，class\_weight=none，dual=false，fit\_intercept=true，

截距缩放=1，最大值ITER=100，多变量class='ovr'，惩罚='l2'，随机变量state=0，解算器='liblinear'，tol=0.0001，

verbose=0））），

“pipeline-1\_uuu clf”：逻辑回归（c=0.001，class\_weight=none，dual=false，fit\_intercept=true，

截距缩放=1，最大值Iter=100，多变量class='ovr'，惩罚='l2'，随机变量state=0，解算器='liblinear'，tol=0.0001，详细值=0），

'pipeline-1\_uuu clf\_uuu c'：0.001，

'pipeline-1\_uu clf\_u class\_weight'：无，

'pipeline-1\_uuu clf\_uuu dual'：错误，

[…]

'pipeline-1\_uuu sc\_uu with\_std'：对，

“pipeline-2”：pipeline（steps=[（'sc'，standardscaler（copy=true，with\_mean=true，with\_std=true）），（'clf'，kneighborssClassifier（algorithm='au to'，leaf\_size=30，metric='minkowski'，metric\_params=none，n\_neighbors=1，p=2，weights='uniform'）），

“pipeline-2\_uuclf”：KneighborsClassifier（algorithm='auto'，leaf\_ux size=30，metric='minkowski'，

metric\_params=none，n\_neighbors=1，p=2，weights='uniform'），

'pipeline-2\_uuu clf\_uu algorithm'：'auto'，

[…]

'pipeline-2\_uuu sc\_uu with\_std'：真

根据get\_params方法返回的值，我们现在知道如何访问单个分类器的属性。现在让我们通过网格搜索来调整逻辑回归分类器的逆正则化参数c和决策树深度，以便于演示。代码如下：

>>>来自sklearn.grid\_search import gridsearchcv

>>>params='决策树分类程序\_uu max\_depth'：[1，2]，

…'pipeline-1\_uuu clf\_uuu c'：[0.001，0.1，100.0]

>>>网格=GridSearchCv（Estimator=mv\_clf，

……参数网格=参数，

……cv=10，

……scoring='roc\_u auc'）

>>>网格拟合（X轴、Y轴）

[217]

在网格搜索完成后，我们可以打印不同的超参数值组合以及通过10倍交叉验证计算的平均ROC AUC分数。代码如下：

>>>对于参数，mean\_score，scores in grid.grid\_scores\_uux:

……打印（“%0.3f+/-%0.2f%r”

……%（平均分，scores.std（）/2，参数）

0.967+/-0.05管道-1 clf\_uuu c'：0.001，'决策树分类器uu max\_u depth'：1

0.967+/-0.05管道-1 clf\_uuu c'：0.1，'决策树分类程序u最大u深度'：1

1.000+/-0.00管道-1 UU CLF UUU C'：100.0，'决策树分类器UUU最大深度'：1

0.967+/-0.05管道-1 clf\_uuu c'：0.001，'决策树分类器uu最大深度'：2

0.967+/-0.05管道-1 clf\_uuu c'：0.1，'决策树分类程序u最大u深度'：2

1.000+/-0.00管道-1 57436; clf\_uuu c'：100.0，'决策树分类器uu最大深度'：2

>>>print（'最佳参数：%s'%grid.best u参数uu）最佳参数：'pipeline-1 u clf uu c'：100.0，

'decisiontreeclassifier\_uu max\_depth'：1\_

>>>打印（‘准确度：%.2f’%grid.best\_u score uu）

精度：1.00

如我们所见，当我们选择一个较低的正则化强度（c=100.0）时，我们得到了最佳的交叉验证结果，而树的深度似乎根本不会影响性能，这表明决策树桩足以分离数据。为了提醒我们自己，多次使用测试数据集进行模型评估是一种糟糕的实践，我们不打算在本节中评估调优超参数的泛化性能。我们将迅速采取另一种合奏学习方法：装袋。

我们在本节中实现的多数投票方法有时也被称为堆叠。然而，堆叠算法通常与逻辑回归模型结合使用集成中单个分类器的预测作为输入来预测最终的类标签，这一点在D.H.Wolpert中由David H.Wolpert进行了更详细的描述。叠加泛化。神经网络，5（2）：241–2591992年。

[218]

装袋-从引导样本构建分类器集合

打包是一种集成学习技术，它与

我们在上一节中实现的主要voteclassifier，

【219】

然而，我们并没有使用相同的训练集来适应集成中的单个分类器，而是从初始训练集中提取引导样本（随机样本和替换样本），这就是为什么打包也被称为引导聚合。为了提供一个关于引导如何工作的更具体的示例，让我们考虑下图中所示的示例。这里，我们有七个不同的训练实例（表示为指数1-7），随机抽样，在每一轮装袋中进行替换。然后，每个引导程序样本都用于匹配分类器CJ，后者通常是未运行的决策树：

装袋也与我们在第3章介绍的随机森林分类器有关，这是一个使用scikit-learn的机器学习分类器的教程。事实上，随机森林是一种特殊的袋装情况，我们也使用随机特征子集来拟合单个决策树。1994年Leo Breiman在一份技术报告中首次提出了装袋，他还表明装袋可以提高不稳定模型的精度，降低过拟合程度。我强烈推荐你读一下他在布莱曼的研究。装袋预测。机器学习，24（2）：123-140，1996年，这是免费在线提供，以了解更多关于包装。

【220】

要了解装袋的实际情况，让我们使用我们在第4章中介绍的Wine数据集创建一个更复杂的分类问题，即构建良好的培训集-数据预处理。在这里，我们只考虑2级和3级葡萄酒，我们选择了两个特征：酒精和色调。

>>>将熊猫作为PD导入

>>>df\_wine=pd.read\_csv（'https://archive.ics.uci.edu/ml/machineLearning databases/wine/wine.data'，header=none）>>>df\_wine.columns=['类标签'，'酒精'，

…'苹果酸“，”灰分“，

…'灰的碱性

…'镁'，'总酚'，

…'黄酮类化合物“、”非黄酮类酚类“，

…'原花青素类，

…'颜色强度'，'色调'，

…'稀释葡萄酒的OD280/OD315’，

…'脯氨酸

>>>df\_wine=df\_wine[df\_wine['class label']！=1]

>>>Y=df\_wine['class label']值

>>>X=df\_wine[[“酒精”，“色调”]。值

接下来，我们将类标签编码为二进制格式，并将数据集分别分成60%的培训和40%的测试集：

>>>来自sklearn.preprocessing import labelencoder

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>LE=Labelencoder（）。

>>>Y=le.fit\_变换（Y）

>>>X轴、X轴、Y轴、Y轴测试=\

……列车测试分割（x，y，

……测试尺寸=0.40，

……随机状态=1）

在Scikit学习中已经实现了一种包分类器算法，我们可以从集成子模块中导入该算法。在这里，我们将使用一个未运行的决策树作为基本分类器，并在训练数据集的不同引导样本上创建一个包含500个决策树的集合：

>>>来自sklearn.ensegle导入包分类器

>>>tree=decisionTreeClassifier（criteria='entropy'，

……最大深度=无）

>>>bag=装袋分类器（base\_estimator=tree，

【221】

……n\_估计数=500，

……最大样本数=1.0，

……最大特征=1.0，

……引导=真，

……bootstrap\_features=false，

……n\_作业=1，

……随机状态=1）

接下来，我们将计算训练和测试数据集预测的准确度分数，以将装袋分类器的性能与单个未运行决策树的性能进行比较：

>>>来自sklearn.metrics import accuracy\_score

>>>tree=tree.fit（x\_train，y\_train）

>>>Y\_train\_pred=树。预测（X\_train）

>>>y\_test\_pred=树.预测（x\_test）

>>>tree\_train=准确度得分（y\_train，y\_train\_pred）

>>>tree\_test=准确度\_分数（y\_test，y\_test\_pred）

>>>打印（“决策树训练/测试精度%.3f/.3f”

……%（树木试验）

决策树训练/测试精度1.000/0.854

根据我们通过执行前面的代码片段打印的精度值，未运行的决策树正确地预测培训样本的所有类标签；但是，测试精度显著降低表明模型的高方差（过度拟合）：

>>>bag=bag.fit（x\_train，y\_train）

>>>Y\_train\_pred=bag.predict（X\_train）

>>>Y\_test\_pred=bag.predict（X\_test）

>>>bag\_train=准确度得分（y\_train，y\_train\_pred）

>>>bag\_test=准确度得分（y\_test，y\_test\_pred）

>>>打印（“包装火车/测试精度%.3f/.3f”

……%（袋式列车，袋式试验）

装袋列车/测试精度1.000/0.896

尽管决策树分类器和装袋分类器在训练集上的训练精度相似（均为1.0），但可以看出，装袋分类器在测试集上的泛化性能稍好。接下来，让我们比较决策树和装袋分类器之间的决策区域：

>>>x\_min=x\_train[：，0].min（）-1

>>>X\_max=X\_train[：，0].max（）+1

>>>Y\_min=X\_train[：，1].min（）-1

>>>Y\_max=X\_train[：，1].max（）+1

>>>x x，yy=np.meshgrid（np.arange（x\_min，x\_max，0.1），…np.arange（y\_最小，y\_最大，0.1）

【222】

>>>F，axarr=plt.子批次（nrows=1，ncols=2，

……sharex='col'，

……shary='行'，

……图尺寸=（8，3）

>>>对于IDX、CLF、TT In-Zip（[0，1]，

……[树，包]，…[‘决策树’，‘装袋’）：

……clf.fit（x\_火车，y\_火车）

……z=clf.predict（np.c\_[xx.ravel（），yy.ravel（）]）

……Z=Z.重塑（XX.形状）

……axarr[idx].轮廓（xx，yy，z，alpha=0.3）

……axarr[idx].散射（x\_train[y\_train==0，0]，

……x列[y列==0，1]，

……c='blue'，marker='^'）

……axarr[idx].散射（x\_train[y\_train==1，0]，

……x列[y列==1，1]，

……c='red'，marker='o'）

……axarr[idx].设置标题（tt）

>>>axarr[0].设置“依拉贝尔”（酒精），fontsize=12）

>>>plt.文本（10.2，-1.2，

……S=色调'，

……ha='center'，va='center'，fontsize=12）

>>>plt.show（）。

正如我们在结果图中看到的，三节点深度决策树的分段线性决策边界在打包集合中看起来更平滑：

在本节中，我们只看了一个非常简单的装袋示例。在实践中，更复杂的分类任务和数据集的高维性很容易导致单决策树的过度拟合，而这正是装箱算法真正发挥其优势的地方。最后，我们要注意的是，装袋算法是减少模型方差的有效方法。然而，装袋在减少模型偏差方面是无效的，这就是为什么我们要选择一组低偏差的分类器，例如未运行的决策树。

通过适应性增强利用弱势学习者

在关于集成方法的这一节中，我们将特别讨论增强，重点讨论它最常见的实现adaboost（自适应增强的缩写）。

Adaboost背后的最初想法是由Robert Schapire于年提出的。

1990年（R.E.Schapire.学习能力弱的力量。机器学习，

5（2）：197–227，1990年）。在罗伯特·夏皮尔和约夫·弗伦德提出

Adaboost算法在第十三届国际会议论文集（ICML 1996）中，Adaboost成为随后几年使用最广泛的集成方法之一（Y.Freund、R.E.Schapire等。一种新的助推算法的实验。在ICML中，第96卷，第148-156页，1996年）。2003年，Freund和Schapire因其开创性的工作获得了Goedel奖，这是计算机科学领域最杰出的出版物的一个著名奖项。

在提升中，合奏由非常简单的基本分类器组成，也通常被称为弱学习者，与随机猜测相比，这些分类器的性能优势很小。一个学习能力较弱的典型例子是决策树树桩。提升背后的关键概念是集中在难以分类的训练样本上，也就是说，让弱者随后从错误分类的训练样本中学习，以提高合奏的表现。与初始的助推式装袋相比，该算法使用从训练数据集中提取的训练样本的随机子集，而无需替换。原始增压程序总结为以下四个关键步骤：

从训练集d中随机抽取训练样本d1的一个子集，不进行替换，以训练弱者c1。

从训练集中抽取第二个随机训练子集d2，不替换，并添加先前错误分类的样本的50%，以训练一个弱者c2。

[224]

找到训练集中d的训练样本d3，c1和c2不同意训练第三个弱者c3。

通过多数投票将弱势学生C1、C2和C3结合起来。

正如Leo Breiman（L.Breiman）所讨论的。偏差、方差和弧分类器。1996年），与装袋模型相比，增加可导致偏差和方差减少。然而，在实践中，诸如adaboost之类的增强算法也因其高方差而闻名，即倾向于过度拟合训练数据（G.Raetsch、T.Onoda和K.R.Mueller）。ADaboost的改进，以避免过拟合。进行中。关于神经信息处理的国际会议。Citeseer，1998年）。

与这里描述的原始提升过程不同，Adaboost使用完整的训练集来训练弱学习者，在每个迭代中对训练样本进行重新加权，以构建一个从以前弱学习者在集成中的错误中学习的强分类器。在深入了解adaboost算法的具体细节之前，让我们先看一下下图，以便更好地了解adaboost背后的基本概念：

[225]

为了一步一步地浏览adaboost演示，我们从子图1开始，它表示一个二进制分类的训练集，其中所有训练样本都被分配了相等的权重。基于此训练集，我们训练了一个决策树桩（以虚线表示），它试图通过最小化成本函数（或决策树集合特殊情况下的杂质分数）来尽可能地对两个类（三角形和圆形）的样本进行分类。对于下一轮（子图2），我们为之前错误分类的两个样本（圆）指定了更大的权重。此外，我们还降低了正确分类样品的重量。下一个决策树桩现在将更加关注那些权重最大的训练样本，也就是那些被认为很难分类的训练样本。子图2所示的弱学习者错误地将来自Circle类的三个不同样本分类，然后将其分配给更大的权重，如子图3所示。假设我们的Adaboost合奏只包含三轮提升，那么我们将通过加权多数投票将在不同的重新加权训练子集上训练的三个弱学习者结合起来，如子图4所示。

既然已经对adaboost的基本概念有了更好的理解，那么让我们更详细地了解一下使用伪代码的算法。为了清楚起见，我们将分别用交叉符号（×表示元素相乘，用点符号（）表示两个向量之间的点积。步骤如下：

将权重向量w设置为均匀权重，其中∑i wi=1

对于J in m助推轮，执行以下操作：

训练一个加权弱的学习者：cj=训练（x，y，w）。

预测类标签：y\_=预测（cj，x）。

计算加权误差率：ε=w（y\_=y）。

计算系数：.

更新权重：w：=w×exp（−αj×y\_×y）。

将权重归一化为1:w：=w/∑i wi。

计算最终预测：y\_=×Predict（cj，x））>0）。

请注意，步骤5中的表达式（y\_==y）指的是1和0的向量，如果预测正确，则指定1，否则指定0。

[226]

虽然Adaboost算法看起来非常简单，但是让我们通过一个更具体的例子，使用一个包含10个训练样本的训练集，如下表所示：

表的第一列描述了训练样本1到10的样本指数。在第二列中，我们看到了假设这是一维数据集的单个样本的特征值。第三列显示了每个训练样本Xi的真类标记Yi，其中yi { 1，-1 }。初始权重如第四列所示；我们将权重初始化为统一的，并将它们规范化为和。因此，对于10个样本训练集，我们将0.1分配给权重向量w中的每个权重wi。假设我们的分割标准是x≤3.0，预测的类标签y\_如第五列所示。然后，表的最后一列根据我们在伪代码中定义的更新规则显示更新后的权重。

由于权重更新的计算一开始可能有点复杂，现在我们将逐步进行计算。我们首先计算加权误差率ε，如步骤5所述：

ε=0.1×0+0.1×0+0.1×0+0.1×0+0.1×0+0.1×0+0.1×0+0.1×0+0.1×0

+0.1×0==0.3

接下来，我们计算系数αj（如步骤6所示），该系数随后在步骤7中用于更新权重以及多数票预测中的权重（步骤10）：

0.5αj=

【227】

在我们计算了系数αj之后，我们现在可以使用以下公式更新权重向量：

W：=W×exp（−αj×y\_×y）

这里，y\_×y分别是预测类标签和真类标签向量之间的元素相乘。因此，如果预测y\_i是正确的，y\_i×yi将有一个正符号，因此我们减小了第i个权重，因为αj也是一个正数：

0.1×exp（−0.424×1×1）≈0.066

同样，如果Y\_i错误地预测了标签，我们将对第i个权重进行加权，如下所示：

0.1×exp（−0.424×1×1（−1））≈0.153

或者像这样：

0.1×exp（-0.424×1）×1）≈0.153

在更新权重向量中的每个权重之后，我们对权重进行规格化，使其总和为1（步骤8）：

W：=W

∑I W I

这里，∑i wi=7×0.065+3×0.153=0.914。

因此，对于下一轮增压，与正确分类的样品相对应的每个重量将从初始值0.1减少到0.066/0.914≈0.072。同样，每个错误分类样品的重量将从0.1增加到0.153/0.914≈0.167。

【228】

简言之，这就是阿达博。跳到更实际的部分，现在让我们通过SciKit学习来训练Adaboost集成分类器。我们将使用上一节中使用的相同的葡萄酒子集来训练装袋元分类器。通过基\_estimator属性，我们将在500个决策树树桩上训练adaboostClassifier：

>>>来自sklearn.ensegle导入adaboostclassifier

>>>tree=decisionTreeClassifier（criteria='entropy'，

……最大深度=1）

>>>Ada=AdaBoostClassifier（Base\_Estimator=Tree，

……n\_估计数=500，

……学习率=0.1，

……随机状态=0）

>>>tree=tree.fit（x\_train，y\_train）

>>>Y\_train\_pred=树。预测（X\_train）

>>>y\_test\_pred=树.预测（x\_test）

>>>tree\_train=准确度得分（y\_train，y\_train\_pred）

>>>tree\_test=准确度\_分数（y\_test，y\_test\_pred）

>>>打印（“决策树训练/测试精度%.3f/.3f”

……%（树木试验）

决策树训练/测试精度0.845/0.854

正如我们所看到的，决策树树桩似乎与上一节中看到的未运行的决策树相比，过度匹配了培训数据：

>>>Ada=Ada.Fit（X\_火车，Y\_火车）

>>>Y\_train\_pred=Ada.Predict（X\_train）

>>>y\_test\_pred=ada.predict（x\_test）

>>>Ada\_train=准确度得分（Y\_train，Y\_train\_pred）

>>>Ada\_test=准确度得分（Y\_test，Y\_test\_pred）

>>>打印（“Adaboost列车/测试精度%.3f/.3f”

……%（Ada ou列车，Ada ou试验）

ADABOOST系列/测试精度1.000/0.875

如我们所见，adaboost模型正确地预测了训练集的所有类标签，并且与决策树树桩相比，测试集的性能略有提高。然而，我们也看到我们通过尝试减少模型偏差引入了额外的方差。

[229]

尽管我们使用了另一个简单的示例进行演示，但与决策树桩相比，Adaboost分类器的性能稍有改善，并且与我们在上一节中培训的袋装分类器获得了非常相似的精度分数。但是，我们应该注意的是，根据测试集的重复使用来选择模型被认为是不好的做法。泛化性能的估计可能过于乐观，我们在第6章“学习模型评估和超参数调整的最佳实践”中对此进行了更详细的讨论。

最后，让我们检查一下决策区域是什么样子的：

>>>x\_min=x\_train[：，0].min（）-1

>>>X\_max=X\_train[：，0].max（）+1

>>>Y\_min=X\_train[：，1].min（）-1

>>>Y\_max=X\_train[：，1].max（）+1

>>>x x，yy=np.meshgrid（np.arange（x\_min，x\_max，0.1），

……np.arange（y\_最小，y\_最大，0.1）

>>>F，axarr=plt.子批次（1，2，

……sharex='col'，

……shary='行'，

……图尺寸=（8，3）

>>>对于IDX、CLF、TT In-Zip（[0，1]，

……[树，艾达]，…[“决策树”，“Adabost”]）：

……clf.fit（x\_系列，y\_系列）

……z=clf.predict（np.c\_[xx.ravel（），yy.ravel（）]）

……Z=Z.重塑（XX.形状）

……axarr[idx].轮廓（xx，yy，z，alpha=0.3）

……axarr[idx].散射（x\_train[y\_train==0，0]，

……x列[y列==0，1]，

……C='蓝色'，

……标记='^'）

……axarr[idx].散射（x\_train[y\_train==1，0]，

……x列[y列==1，1]，

……C='红色'，

……marker='o'）

……axarr[idx].设置标题（tt）

……axarr[0].设置“依拉贝尔”（“酒精”，fontsize=12）

>>>plt.文本（10.2，-1.2，

……S=色调'，

……ha='中心'，

……va='center'，

……字体大小=12）

>>>plt.show（）。

【230】

通过观察决策区域，我们可以看到Adaboost模型的决策边界比决策树桩的决策边界要复杂得多。此外，我们注意到Adaboost模型将特征空间与我们在前一节中培训的装袋分类器非常相似地分隔开。

作为对集成技术的总结，值得注意的是，集成学习比单个分类器增加了计算的复杂性。在实践中，我们需要仔细考虑是否要为预测性能的相对适度改进支付增加的计算成本的代价。

这种权衡的一个经常被引用的例子是著名的价值100万美元的Netflix奖，该奖是通过合奏技术获得的。有关该算法的详细信息发表在A.Toescher、M.Jahrer和R.M.Bell上。Netflix大奖的大混乱解决方案。Netflix奖文档，2009年（可从以下网址获取）。虽然获奖团队获得了100万美元的奖金，但Netflix从未实施过他们的模型，因为它的复杂性，这使得它不可能在现实世界中应用。引用他们确切的话（：

“[…]我们测量的额外精度提高似乎不能证明将它们引入生产环境所需的工程努力是合理的。”

【231】

总结

在这一章中，我们研究了一些最流行和广泛使用的合奏学习技术。集成方法将不同的分类模型结合起来，以消除它们各自的弱点，这通常会导致稳定且性能良好的模型，这些模型对工业应用和机器学习竞争非常有吸引力。

在本章的开头，我们在python中实现了一个主要的voteclassifier，它允许我们结合不同的分类算法。然后我们研究了bagging，这是一种通过从训练集中提取随机引导样本，并通过多数票组合单独训练的分类器来减少模型方差的有用技术。然后我们讨论了Adaboost，它是一种基于弱学习者的算法，而后者从错误中学习。

在前几章中，我们讨论了不同的学习算法、调优和评估技术。在下一章中，我们将看到机器学习、情感分析的一个特殊应用，这无疑成为互联网和社交媒体时代一个有趣的话题。

[232]

机器学习在情绪分析中的应用

在这个互联网和社会媒体时代，人们的意见、评论和建议已经成为政治科学和商业的宝贵资源。多亏了现代技术，我们现在能够最有效地收集和分析这些数据。在本章中，我们将深入探讨自然语言处理（NLP）的一个子领域，即情感分析，并学习如何使用机器学习算法根据文档的极性对文档进行分类：作者的态度。我们将在以下章节中讨论的主题包括：

清理和准备文本数据

从文本文档构建特征向量

训练机器学习模型，将正面和负面电影评论分类

使用核心外学习处理大文本数据集

获取IMDB电影评论数据集

情感分析，有时也称为意见挖掘，是NLP更广泛领域的一个热门分支，它分析了文档的极性。情感分析中的一个常见任务是根据作者对某一特定主题表达的意见或情感对文档进行分类。

[233]

在本章中，我们将使用来自

Maas等人收集的互联网电影数据库（IMDB）。（A.L.Maas、R.E.Daly、P.T.Pham、D.Huang、A.Y.Ng和C.Potts。学习情绪分析的词汇载体。在计算语言学协会第49届年度会议记录：人类语言技术，第142-150页，美国俄勒冈州波特兰，2011年6月。计算语言学协会）。电影评论数据集由50000个极地电影评论组成，这些评论被标记为正片或负片；这里，正片意味着一部电影在IMDB上被评为六颗星以上，负片意味着一部电影在IMDB上被评为五颗星以下。在下面的部分中，我们将学习如何从这些电影评论的子集中提取有意义的信息，以构建一个机器学习模型，该模型可以预测某个评论家是否喜欢电影。

电影评论数据集（84.1 MB）的压缩存档可作为gzip压缩tarball存档下载：

如果您使用的是Linux或Mac OS X，您可以打开一个新的终端窗口，使用cd进入下载目录，然后执行tar-zxf aclimdb\_v1.tar.gz来解压缩数据集。

如果您使用的是Windows，则可以下载免费的存档程序，如7-zip（从下载存档文件中提取文件

在成功地提取了数据集之后，我们现在将把解压下载档案中的单个文本文档组装成一个csv文件。

在下面的代码部分，我们将把电影评论读到一个熊猫数据帧对象中，在标准的台式计算机上，这个对象可能需要10分钟。

为了可视化进度和预计完成前的时间，我们将使用Pyprind（Python进度指示器，我几年前为此开发的包）。可以通过执行命令：pip install pyprind来安装pyprind。

>>>导入Pyprind

>>>将熊猫作为PD导入

>>>导入操作系统

>>>PBAR=pyprind.progbar（50000）

>>>labels='pos'：1，'neg'：0

>>>df=pd.dataframe（）>>>对于s in（“test”，“train”）：

……对于L in（“pos”，“neg”）：

……路径='./aclimdb/%s/%s'%（s，l）…对于os.listdir中的文件（路径）：

……以open（os.path.join（path，file），“r”）作为内嵌：

[234]

……txt=infile.read（）。

……df=df.append（[[txt，labels[l]]]，ignore\_index=true）

……pbar.update（）。

>>>df.columns=['评论'，'情绪']

0%100%

总运行时间：725.001秒

在执行前面的代码时，我们首先用50000次迭代初始化了一个新的进度条对象pbar，这是我们将要读取的文档数。使用嵌套的for循环，我们在列车上进行迭代，并在主aclimdb目录中测试子目录，并从pos和neg子目录中读取单独的文本文件，这些子目录最终与一个整数类标签（1=正，0=负）一起附加到数据帧df。

由于已组装的数据集中的类标签已排序，因此我们现在将无序排列

使用np.random子模块的排列函数的数据帧这将有助于在后面的章节中将数据集拆分为训练集和测试集，我们将直接从本地驱动器流式传输数据。为了方便起见，我们还将组装和无序播放的电影评论数据集存储为一个csv文件：

>>>导入numpy为np

>>>np.random.seed（0）

>>>df=df.reindex（np.random.permutation（df.index））。

>>>df.to\_csv（'./movie\_data.csv'，index=false）

由于我们将在本章的后面部分使用此数据集，因此，让我们快速确认我们通过读取csv并打印前三个示例的摘录，成功地以正确的格式保存了数据：

>>>df=pd.read\_csv（'./movie\_data.csv'）

>>>测向头（3）

如果您在ipython笔记本中运行代码示例，那么现在应该看到数据集的前三个示例，如下表所示：

回顾情绪

1974年，少年玛莎·莫斯利（玛吉1组）

好啊。。。所以…我真的很喜欢克里斯·克里斯托弗森，一个……0，2，剧透……如果你认为是……0，不要读这个。

[235]

介绍文字袋模型

我们记得从第4章，建立良好的训练集-数据预处理，我们必须将分类数据，如文本或单词，转换成数字形式，然后才能将其传递给机器学习算法。在这一部分中，我们将介绍一组单词模型，它允许我们将文本表示为数字特征向量。单词袋模型背后的思想非常简单，可以总结如下：

我们创建一个具有唯一标记的词汇表，例如，来自整个文档集的单词。

我们从每个文档构造一个特征向量，其中包含每个单词在特定文档中出现的频率计数。

由于每个文档中的唯一单词仅代表单词词汇袋中所有单词的一小部分，因此特征向量将主要由零组成，这就是我们称之为稀疏的原因。如果这听起来太抽象，请不要担心；在下面的小节中，我们将逐步创建一个简单的单词袋模型。

将单词转换为特征向量

为了根据文档中的字数构建一个单词包模型，我们可以使用SciKit Learn中实现的CountVectorizer类。正如我们将在下面的代码部分看到的，Countvectorizer类获取一个文本数据数组，可以是文档，也可以只是句子，并为我们构造一个单词袋模型：

>>>导入numpy为np

>>>从sklearn.feature\_extraction.text导入countvectorizer

>>>count=countvectorizer（）。

>>>文档=np.array（[

…'阳光灿烂，

…'天气很好，

…'阳光明媚，天气宜人。'）

>>>bag=count.fit\_transform（docs）

通过调用countvectorizer上的fit\_变换方法，我们构建了单词包模型的词汇表，并将以下三个句子转换为稀疏特征向量：

阳光明媚

天气很好

阳光明媚，天气宜人

[236]

现在，让我们打印词汇表的内容，以便更好地理解基本概念：

>>>打印（count.词汇表）

“The”：5，“Shining”：2，“Weather”：6，“Sun”：3，“Is”：1，“Sweet”：4，

'和'：0

正如我们在执行前面的命令时看到的，词汇表存储在一个python字典中，该字典映射映射到整数索引的唯一词汇。接下来，让我们打印刚刚创建的特征向量：

>>>打印（bag.toarray（））

[0 1 1 1 0 1 0]

1 0 0 1 1 1]

2 1 1 1 2 1]]

这里显示的特征向量中的每个索引位置对应于作为字典项存储在Countvectorizer词汇表中的整数值。例如，索引位置0处的第一个特征类似于单词的计数，并且只出现在最后一个文档中，并且单词在索引位置1处（文档向量中的第二个特征）出现在所有三个句子中。特征向量中的这些值也称为原始项频率：tf（t，d）-文档d中出现项t的次数。

我们刚刚创建的单词袋模型中的项目序列也称为1-gram或unigram模型。词汇表中的每个项目或标记表示一个单词。更一般地说，NLP单词、字母或符号中项目的连续序列也被称为N-gram。n-gram模型中数字n的选择取决于特定的应用；例如，Kanaris等人的研究。结果显示，大小为3和4的N克邮件在电子邮件（ioannis kanaris、konstantinos kanaris、ioannis houvardas和efstathios stamatotos）的反垃圾邮件过滤方面具有良好的性能。单词与字符的反垃圾邮件过滤n克。国际人工智能工具杂志，16（06）：1047-1067，2007）。为了总结n-gram表示的概念，我们的第一个文档“太阳照耀”的1-gram和2-gram表示将构建如下：

1克：“The”、“Sun”、“Is”、“Shining”

2克：“太阳”、“太阳”、“照耀”

Scikit Learn中的Countvectorizer类允许我们通过其n gram\_range参数使用不同的n-gram模型。当默认情况下使用1-gram表示时，我们可以通过使用ngram\_range=（2,2）初始化一个新的countvectorizer实例来切换到2-gram表示。

[237]

用术语频率反比文档频率评估单词相关性

在分析文本数据时，我们经常会遇到在两个类的多个文档中出现的单词。那些经常出现的词通常不包含有用的或歧视性的信息。在本小节中，我们将学习一种称为“频率反转文档频率（tf idf）”的有用技术，它可以用来降低特征向量中经常出现的单词的权重。tf idf可以定义为术语频率和文档反向频率的乘积：

tf idf（t，d）=tf（t，d）×idf（t，d）

这里，tf（t，d）是我们在前一节中介绍的术语频率，反文档频率idf（t，d）可以计算为：

idf（t，d）=log1+df nd（d，t），

式中，nd是文档总数，df（d，t）是包含术语t的文档数d。请注意，在分母中添加常量1是可选的，用于为所有训练样本中出现的术语分配非零值；日志用于确保在较低的文档频率下，不会给予过多的权重。

Scikit Learn还实现了另一个变压器tfidftransformer，它将countvectorizer的原始项频率作为输入，并将其转换为tf idfs：

>>>从sklearn.feature\_extraction.text导入tfidftransformer

>>>tfidf=tfidftransformer（）。

>>>np.设置打印选项（精度=2）

>>>打印（tfidf.fit\_transform（count.fit\_transform（docs））.toArray（））

[0.0.43 0.56 0.56 0。0.43 0.]

[0.0.43 0.0.0.56 0.43 0.56]

[0.4 0.48 0.31 0.31 0.31 0.48 0.31]]

【238】

正如我们在前一小节中看到的，单词在第三个文档中的词频是最大的，是出现最频繁的单词。然而，在将相同的特征向量转换为tf idf之后，我们发现这个词现在与文档3中相对较小的tf idf（0.31）相关联，因为它也包含在文档1和2中，因此不太可能包含任何有用的、歧视性的信息。

但是，如果我们手动计算特征向量中单个项的tf-idfs，我们会注意到tfidftransformer计算tf-idfs与我们前面定义的标准教科书公式略有不同。在SciKit Learn中实现的IDF和TF IDF的方程式如下：

idf（t，d）=log1+1d+f n（d，td）

在SciKit学习中实现的tf-idf方程如下：tf-idf（t，d）=tf（t，d）×（idf（t，d）+1）

虽然在计算tf idfs之前对原始项频率进行归一化更为典型，但是tfidftransformer直接对tf idfs进行归一化。

默认情况下（norm='l2'），SciKit Learn的tfidftTransformer应用l2规范化，它返回长度为1的向量，方法是将未规范化的特征向量v除以l2规范：

vnorm=v v 2=v12+v22v++v n 2=（∑i n=v 1 vi）1/2

为了确保我们了解tfidftransformer是如何工作的，让我们通过一个例子来计算第三个文档中单词的tf idf。

在文档3中，单词is的词频为2（tf=2），由于该词频出现在所有三个文档中（df=3），因此该词频为3。因此，我们可以计算IDF如下：

idf（“is”，d3）=log=0

[239]

现在，为了计算tf idf，我们只需要将1添加到反向文档频率，并将其乘以术语频率：

tf idf（“is”，d3）=2×（0+1）=2

如果我们对第3个文档中的所有术语重复这些计算，我们将获得以下tf idf向量：[1.69、2.00、1.29、1.29、1.29、2.00和1.29]。但是，我们注意到这个特征向量中的值与我们从以前使用的tfidftransformer中获得的值不同。在这个tf idf计算中我们缺少的最后一个步骤是l2标准化，它可以应用如下：

tf idf（“is”，d3）标准1.69 2[，1.692.00，22.00+1.29，1.292+，1.291.292，+1.291.29，2.002+2.00，1.292+]1.292

=[0.40、0.48、0.31、0.31、0.31、0.48、0.31]

如我们所见，结果现在与Scikit Learn的tfidftTransformer返回的结果相匹配。既然我们现在了解了tf idfs是如何计算的，那么让我们继续下一节，并将这些概念应用到电影评论数据集中。

正在清除文本数据

在前面的小节中，我们学习了单词包模型、术语频率和tf idf。然而，在我们构建单词包模型之前的第一个重要步骤是清除文本数据中所有不需要的字符。为了说明这一点的重要性，让我们在重新编排的电影评论数据集中显示第一个文档的最后50个字符：

>>>df.loc[0，'评论'][-50:]

'是七。<br/><br/><br/>标题（巴西）：不可用'

正如我们在这里看到的，文本包含HTML标记、标点和其他非字母字符。虽然HTML标记不包含太多有用的语义，但标点符号可以表示某些NLP上下文中有用的附加信息。不过，为了简单起见，我们现在将删除所有标点符号，但只保留符号符号，如“：）”，因为这些符号对情绪分析肯定有用。为了完成这项任务，我们将使用python的正则表达式（regex）库re，如下所示：

>>>导入RE>>DEF预处理器（文本）：

【240】

……text=re.sub（'<[^>]\*>'，''，text）

……表情符号=re.findall（'（？）：：=）（？：-）？（？：\）\（d\_p）’，文本）

……text=re.sub（'[\w]+'，''，text.lower（））.+\

'加入（表情符号）。替换'-'，''

……返回文本

通过前面代码部分中的第一个regex<[^>]\*>，我们尝试删除电影评论中包含的整个HTML标记。尽管许多程序员通常建议不要使用regex来解析HTML，但是这个regex应该足以清除这个特定的数据集。删除HTML标记后，我们使用稍微复杂一点的regex查找图释，图释暂时存储为图释。接下来，我们通过regex[\w]+从文本中删除所有非单词字符，将文本转换为小写字符，并最终将临时存储的表情符号添加到处理过的文档字符串的末尾。此外，为了保持一致性，我们从表情符号中删除了鼻子字符（-）。

虽然正则表达式提供了一种有效和方便的方法来搜索字符串中的字符，但它们也有一条陡峭的学习曲线。不幸的是，对正则表达式的深入讨论超出了本书的范围。不过，你可以在谷歌开发者门户网站上找到一个很好的教程

或

查看python的re模块的官方文档，网址为

尽管将表情符号添加到清理过的文档字符串的末尾看起来可能不是最优雅的方法，但是如果我们的词汇表只包含一个单词标记，那么在我们的单词包模型中，单词的顺序并不重要。但是，在我们进一步讨论将文档拆分为单个术语、单词或令牌之前，让我们确认预处理器工作正常：

>>>预处理器（df.loc[0，'评论']-50:）

'七标题巴西不可用'

>>>预处理器（“<a>this:）is:（a test:-）！”）

'这是一个测试：）：（：）'

最后，由于我们将在下一节中反复使用清理过的文本数据，现在让我们将预处理器功能应用于数据框中的所有电影评论：

>>>df['review']=df['review']应用（预处理器）

【241】

将文档处理成令牌

在成功地准备好电影评论数据集之后，我们现在需要考虑如何将文本语料库拆分为各个元素。标记文档的一种方法是将文档拆分为单独的单词，方法是将清理过的文档拆分为空白字符：

>>>def tokenizer（文本）：

……返回文本。拆分（）

>>>tokenizer（'跑步者喜欢跑步，因此他们跑步'）

[跑步者'，'喜欢'，'跑步'，'和'，'因此'，'他们'，'跑步']

在标记化技术的上下文中，另一种有用的技术是词干，它是将一个词转换为其根形式的过程，允许我们将相关的词映射到相同的词干。原始的词干算法由Martin F.Porter于1979年开发，因此被称为波特词干算法（Martin F.Porter）。一种去除后缀的算法。课程：电子图书馆和信息系统，14（3）：130–137，1980）。python的自然语言工具包（nltk，）实现了porter stemming算法，我们将在下面的代码部分中使用该算法。为了安装NLTK，您可以简单地执行pip install NLTK。

>>>从nltk.stem.porter导入porterstemmer

>>>Porter=Porterstemmer（）>>>def tokenizer\_porter（文本）：

……返回[porter.stem（word）for word in text.split（）]

>>>tokenizer-porter（“跑步者喜欢跑步，因此他们跑步”）。

[跑步者'，'喜欢'，'跑步'，'和'，'周四'，'他们'，'跑步']

虽然NLTK不是本章的重点，但我强烈建议您访问NLTK网站和正式的NLTK手册，如果您对NLP中的更高级应用程序感兴趣，可以在免费获取。

使用NLTK包中的porterstemmer，我们修改了我们的tokenizer函数，将单词减少到其根形式，前面的简单示例说明了这一点，其中运行的单词源于其根形式的运行。

【242】

波特词干算法可能是最古老和最简单的词干算法。其他流行的词干算法包括更新的雪球词干（porter2或“english”stemmer）或兰开斯特词干（paice husk stemmer），后者比porter stemmer更快，但也更具攻击性。这些替代的词干算法也可以通过NLTK包获得。

虽然词干可以创建非实词，如thu（由此而来），如前一个示例所示，但一种称为引理化的技术旨在获得单个词的规范（语法正确）形式，即所谓的引理。然而，与词干化相比，词干化在计算上更为困难和昂贵，并且在实践中观察到词干化和词干化对文本分类的性能影响很小（Michal Toman、Roman Tesar和Karel Jezek）。词规范化对文本分类的影响。INSCIT会议录，第354-358页，2006年）。

在我们进入下一节之前，在这里将使用单词袋模型培训机器学习模型，让我们简单地讨论另一个有用的主题，即停止单词删除。停止词只是那些在各种文本中极为常见的词，可能没有（或只有很少）有用的信息，可以用来区分不同类别的文档。停止词的例子有is、and、has等。如果我们使用原始或规范化的词频而不是tf idfs，那么删除停止词可能很有用，因为tf idfs已经降低了频繁出现的词的权重。

为了从电影评论中删除停止词，我们将使用NLTK库中提供的127个英文停止词集，可通过调用nltk.download函数获得：

>>>导入NLTK

>>>nltk.download（'stopwords'）

下载完停止词集后，我们可以按如下方式加载和应用英文停止词集：

>>>从nltk.corpus导入stopwords

>>>stop=stop words.words（'english'）

>>>[W代表W在tokenizer\_-porter中（‘跑步者喜欢跑步，经常跑步’）[-10:]if W不在stop中][‘跑步者’、‘喜欢’、‘跑步’、‘跑步’、‘lot’]

【243】

训练文档分类的逻辑回归模型

在本节中，我们将培训一个逻辑回归模型，将电影评论分为正面评论和负面评论。首先，我们将清理文本文档的数据框架划分为25000份用于培训的文档和25000份用于测试的文档：

>>>X\_train=df.loc[：25000，'查看']值

>>>Y\_train=df.loc[：25000，'情绪']值

>>>X\_test=df.loc[25000:，'review']值

>>>Y\_test=df.loc[25000:，'情绪']值

接下来，我们将使用一个GridSearchCv对象，通过5倍分层交叉验证，找到我们的逻辑回归模型的最佳参数集：

>>>来自sklearn.grid\_search import gridsearchcv>>来自sklearn.pipeline import pipeline

>>>来自sklearn.linear\_model import logisticregression

>>>从sklearn.feature\_extraction.text导入tfidfvectorizer

>>>tfidf=tfidfvectorizer（strip\_accents=none，……小写=假，

……预处理器=无）

>>>参数\_grid=['vect\_uu ngram\_range'：[（1,1）]

…'vect\_uuu stop\_words'：[停止，无]，…'vect\_uuu tokenizer'：[tokenizer，

……标记器“波特”，……clf\_uuu惩罚：['l1'、'l2']，

…'clf\_uuu c'：[1.0，10.0，100.0]，…'vect\_uuu ngram\_range'：[（1,1）]，

…'vect\_uuu stop\_words'：[停止，无]，…'vect\_uuu tokenizer'：[tokenizer，

……标记器波特]，

…'vect\_uuu使用\_idf'：[false]…'vect\_uuu norm'：[无]，

…'clf\_uuu惩罚：['l1'、'l2']，

…'clf\_uuu c'：[1.0，10.0，100.0]…]

>>>lr\_tfidf=管道（[（'vect'，tfidf），…（'clf'，

……逻辑回归（随机状态=0））]

>>>GS\_lr\_tfidf=gridsearchcv（lr\_tfidf，param\_grid，

……得分='准确度'，

……cv=5，verbose=1，

……n\_作业=-1）

>>>GS-Lr-Tfidf.fit（X-列车，Y-列车）

[公元244年]

当我们使用前面的代码初始化gridsearchcv对象及其参数网格时，我们将自己限制在有限数量的参数组合中，因为特征向量的数量以及大量的词汇表可以使网格搜索的计算成本相当高；使用标准台式电脑，我们的网格搜索可能需要40分钟才能完成。

在前面的代码示例中，我们将前面小节中的countvectorizer和tfidftransformer替换为tfidfvectorizer，后者结合了后一个Transformer对象。我们的参数网格由两个参数字典组成。在第一个字典中，我们使用带有默认设置的tfidfVectorizer（使用\_idf=true、Smooth\_idf=true和norm='l2'）来计算tf idf；在第二个字典中，我们将这些参数设置为使用\_idf=false、Smooth\_idf=false和norm=none，以便根据原始术语训练模型。频率。此外，对于逻辑回归分类器本身，我们通过惩罚参数对模型进行了L2和L1正则化训练，并通过定义反正则化参数c的一系列值比较了不同的正则化强度。

网格搜索完成后，我们可以打印出最佳参数集：

>>>print（'最佳参数集：%s'%gs lr\_tfidf.best\_params\_u）最佳参数集：'clf\_uu c'：10.0，'vect\_u stop\_words'：无，

'clf\_uuu penalty'：'l2'，'vect\_uu tokenizer'：<函数tokenizer at

0x7F6C704948C8>，'vect\_uuu ngram\_range'：（1，1）

正如我们在这里看到的，我们使用不带波特词干的规则标记器、无停止词库和tf idfs以及使用规则化强度c=10.0的L2正则化的逻辑回归分类器获得了最佳的网格搜索结果。

使用此网格搜索的最佳模型，让我们打印训练集上的5倍交叉验证精度分数和测试数据集上的分类精度：

>>>打印（‘cv精度：%.3f’

……%最好的分数

cv精度：0.897

>>>clf=gs\_lr\_tfidf.best\_estimator\_

>>>打印（“测试精度：%.3f”

……%clf.分数（x\_测试，y\_测试）

测试精度：0.899

结果表明，我们的机器学习模型可以预测电影评论是正面的还是负面的，准确率为90%。

[245]

一个仍然非常流行的文本分类分类器是NA\_ve Bayes分类器，它在电子邮件垃圾邮件过滤应用中得到了广泛的应用。与其他算法相比，NA\_ve Bayes分类器易于实现、计算效率高，并且在相对较小的数据集上表现特别出色。虽然我们在这本书中没有讨论na ve bayes分类器，但是感兴趣的读者可以找到我的文章，关于nave文本分类，我在arxiv（s.raschka）上免费提供。

朴素的贝叶斯与文本分类I——引言与理论。计算研究库（CORR），ABS/1410.5329，2014年。）.

使用更大的数据-在线算法和核心外学习

如果您在前一节中执行了代码示例，您可能会注意到，在网格搜索期间为50000个电影评论数据集构建特征向量的计算成本可能非常高。在许多实际应用中，使用甚至可能超过计算机内存的更大数据集并不罕见。由于并非每个人都能使用超级计算机设施，我们现在将应用一种称为“核心外学习”的技术，使我们能够处理如此庞大的数据集。

在第二章，分类训练机学习算法中，我们引入了随机梯度下降的概念，这是一种优化算法，一次使用一个样本更新模型的权重。在本节中，我们将利用SciKit中sgdclassifier的部分拟合函数，学习直接从本地驱动器流式传输文档，并使用小批量文档训练逻辑回归模型。

首先，我们定义一个tokenizer函数，该函数从我们在本章开头构建的电影data.csv文件中清除未处理的文本数据，并在删除停止字的同时将其分离为字标记。

>>>导入numpy为np

>>>导入RE

>>>从nltk.corpus导入stopwords

>>>stop=stop words.words（'english'）>>>def tokenizer（文本）：

……text=re.sub（'<[^>]\*>'，''，text）

……表情符号=re.findall（'（？）：：=）（？：-）？（？：\）\（d p）'，

……text.lower（））…text=re.sub（'[\w]+'，''，text.lower（））.\

[公元246年]

……+“”.Join（表情符号）。替换（“-”，“”）

……tokenized=[w for w in text.split（）if w not in stop]

……返回标记化

接下来，我们定义一个生成器函数stream\_docs，它一次读取并返回一个文档：

>>>def stream\_docs（路径）：

……将open（path，'r'）作为csv:

……下一个（csv）跳过标题…对于csv中的行：

……text，label=line[：-3]，int（line[-2]）

……产量文本，标签

为了验证stream\_docs功能是否正常工作，让我们从movie\_data.csv文件中读取第一个文档，该文件将返回一个由审阅文本和相应的类标签组成的元组：

>>>下一步（stream\_docs（path='./movie\_data.csv'））

（'1974年，少年玛莎·莫斯利…'，1）

现在，我们将定义一个函数get\_minibatch，它将从stream\_docs函数获取文档流，并返回由size参数指定的特定数量的文档：

>>>def get\_minibatch（文档流，大小）：

……文档，Y=[]，[]…尝试：

……对于范围内的（尺寸）：

……text，label=next（文档流）

……附加文档（文本）

……y.append（标签）除了StopIteration：

……无返回，无

……返回文档，Y

不幸的是，我们不能使用countvectorizer进行核心外学习，因为它需要在内存中保存完整的词汇表。此外，tfidfvectorizer需要将训练数据集的所有特征向量保存在内存中，以计算反向文档频率。然而，在SciKit学习中实现的另一个有用的文本处理向量器是HashingVectorizer。HashingVectorizer独立于数据，并通过AustinAppleby的32位杂音哈希3算法利用了哈希技巧。

>>>来自sklearn.feature\_extraction.text import hashingvectorizer>>来自sklearn.linear\_model import sgdclassifier

[公元247年]

>>>vect=hashingvectorizer（解码错误='ignore'，

……n\_功能=2\*\*21，

……预处理器=无，

……标记器=标记器）

>>>clf=sgdclassifier（loss='log'，random\_state=1，n\_iter=1）

>>>doc\_stream=stream\_docs（path='./movie\_data.csv'）

使用前面的代码，我们用记号赋予器函数初始化了hashingvectorizer，并将特性数量设置为221。此外，我们通过将sgdclassifier的loss参数设置为log来重新初始化一个逻辑回归分类器。注意，通过在hashingvectorizer中选择大量的特性，我们减少了引起哈希冲突的机会，但是我们也增加了逻辑回归模型。

现在是真正有趣的部分。在设置了所有的补充功能之后，我们现在可以使用以下代码开始核心外学习：

>>>导入Pyprind

>>>PBAR=pyprind.progbar（45）

>>>classes=np.array（[0，1]）>>>用于范围内（45）：

……x\_train，y\_train=get\_minibatch（doc\_stream，size=1000）……如果不是X列：

……打破

……x\_train=矢量变换（x\_train）

……clf.部分匹配（x轴列，y轴列，类=类）

……pbar.update（）。

0%100%

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\124;]预计到达时间[秒]：0.000

总运行时间：50.063秒

我们再次利用Pyprind包来估计学习算法的进度。我们用45次迭代初始化了进度条对象，在下面的for循环中，我们迭代了45个小批量的文档，其中每个小批量包含1000个文档。

完成增量学习过程后，我们将使用最后5000个文档来评估模型的性能：

>>>X\_test，Y\_test=get\_minibatch（doc\_stream，size=5000）

>>>X\_test=vect.transform（X\_test）

>>>print（'准确度：%.3f'%clf.score（x\_test，y\_test））

精度：0.868

【248】

如我们所见，模型的精度为87%，略低于我们在上一节中使用网格搜索进行超参数调整所获得的精度。然而，核心外学习是非常有效的记忆和不到一分钟完成。最后，我们可以使用最后5000个文档来更新我们的模型：

>>>clf=clf.部分拟合（x\_测试，y\_测试）

如果您打算直接继续第9章，将机器学习模型嵌入到Web应用程序中，我建议您保持当前的Python会话处于打开状态。在下一章中，将使用我们刚刚培训过的模型，学习如何将其保存到磁盘以供以后使用，并将其嵌入到Web应用程序中。

虽然袋词模型仍然是最常用的文本分类模型，但它不考虑句子结构和语法。词语袋模型的一个流行扩展是潜在dirichlet分配，它是一个考虑词语潜在语义的主题模型（D.M.Blei、A.Y.Ng和M.I.约旦）。潜在的迪里克莱分配。《机器学习研究杂志》，3:993–10222003）。

Word2vec是一个更现代的词汇袋模型替代品，它是谷歌在2013年发布的算法（T.Mikolov、K.Chen、G.Corrado和J.Dean）。矢量空间中字表示的有效估计。ARXIV预印ARXIV:1301.37812013）。word2vec算法是一种基于神经网络的无监督学习算法，尝试自动学习单词之间的关系。word2vec背后的思想是将具有相似含义的单词放入相似的簇中；通过巧妙的向量间距，该模型可以使用简单的向量数学（例如，king–man+woman=queen）来复制某些单词。

原始的C-implementation与相关论文和备选实现之间有有用的链接，可以在

【249】

总结

在本章中，我们学习了如何使用机器学习算法根据文本文档的极性对其进行分类，这是自然语言处理领域情感分析的基本任务。我们不仅学习了如何使用词语袋模型将文档编码为特征向量，而且还学习了如何使用词语频率反比文档频率通过相关性加权词语频率。

由于在这个过程中创建了大量的特征向量，使用文本数据的计算成本可能相当高；在最后一节中，我们学习了如何利用核心外学习或增量学习来训练机器学习算法，而不将整个数据集加载到组件中。乌特的记忆。

在下一章中，我们将使用文档分类器并学习如何将其嵌入到Web应用程序中。

[250]

嵌入机器

Web应用程序中的学习模型

在前面的章节中，您学习了许多不同的机器学习概念和算法，这些概念和算法可以帮助我们做出更好、更高效的决策。但是，机器学习技术不仅限于离线应用程序和分析，而且可以作为Web服务的预测引擎。例如，在Web应用程序中，机器学习模型的流行和有用的应用程序包括提交表单中的垃圾邮件检测、搜索引擎、媒体或购物门户的推荐系统等等。

在本章中，您将学习如何将机器学习模型嵌入到Web应用程序中，该应用程序不仅可以进行分类，还可以实时从数据中学习。我们将讨论的主题如下：

保存经过训练的机器学习模型的当前状态

使用sqlite数据库进行数据存储

使用流行的flask Web框架开发Web应用程序

将机器学习应用程序部署到公共Web服务器

【251】

串联合适的SciKit学习估计量

训练机器学习模型在计算上相当昂贵，正如我们在第8章中看到的，将机器学习应用于情绪分析。当然，我们不想每次关闭python解释器并想进行新的预测或重新加载Web应用程序时都训练我们的模型？模型持久性的一个选项是python内置的pickle模块（），它允许我们序列化和反序列化python对象结构以压缩字节代码，这样我们就可以在当前状态下保存分类器，并在不需要学习再次从培训数据中建模。在执行以下代码之前，请确保您已经培训了第8章最后一节中的核心外逻辑回归模型，将机器学习应用于情绪分析，并在当前的python会话中做好准备：

>>>进口泡菜

>>>导入操作系统

>>>dest=os.path.join（'movieClassifier'，'pkl\_objects'）>>>如果不是os.path.exists（dest）：

……os.makedirs（目标）

>>>pickle.dump（停止，

……open（os.path.join（dest，'stopwords.pkl'），'wb'），

……协议=4）

>>>泡菜。倾倒（CLF，

……open（os.path.join（dest，'classifier.pkl'），'wb'），

……协议=4）

使用前面的代码，我们创建了一个movieclassifier目录，稍后将在其中存储Web应用程序的文件和数据。在这个movieclassifier目录中，我们创建了一个pkl\_对象子目录，将序列化的python对象保存到本地驱动器。然后，通过pickle的dump方法，我们序列化了经过训练的逻辑回归模型以及NLTK库中的停止词集，这样我们就不必在服务器上安装NLTK词汇表。dump方法将要pickle的对象作为其第一个参数，而对于第二个参数，我们提供了一个打开的文件对象，python对象将被写入其中。通过open函数内的wb参数，我们以二进制模式打开了pickle文件，并设置protocol=4以选择添加到python 3.4中的最新、最有效的pickle协议。（如果您在使用协议4时遇到问题，请检查您是否在使用最新的python 3版本安装。或者，您可以考虑选择较低的协议号）

[252]

我们的逻辑回归模型包含几个numpy数组，例如权重向量，而序列化numpy数组的一个更有效的方法是使用替代的joblib库。为了确保与我们将在后面的部分中使用的服务器环境兼容，我们将使用标准pickle方法。如果您感兴趣，可以在

我们不需要酸洗hashingvectorizer，因为它不需要安装。相反，我们可以创建一个新的python脚本文件，从中我们可以将向量器导入到当前的python会话中。现在，复制以下代码并将其另存为movieclassifier目录中的vectorizer.py：

从sklearn.feature\_extraction.text import hashingvectorizer import re import os import pickle

cur\_dir=os.path.dirname（uu file\_uuuu）stop=pickle.load（打开（

os.path.join（当前目录，

'pkl\_对象'，

'stopwords.pkl'），'rb'））

def标记器（文本）：

text=re.sub（'<[^>]\*>'，''，text）

表情符号=re.findall（'（？）：：=）（？：-）？（？：\）\（d p）’，text.lower（））text=re.sub（'[\w]+’，''，text.lower（））.\

+'.join（表情符号）。replace（'-'，''）tokenized=[w for w in text.split（）if w not in stop]返回tokenized

vect=hashingvectorizer（解码错误='ignore'，n\_功能=2\*\*21，预处理器=none，标记器=tokenizer）

[253]

在pickle了python对象并创建了vectorizer.py文件之后，现在最好重新启动我们的python解释器或ipython笔记本内核来测试我们是否可以无错误地反序列化这些对象。但是，请注意，从不受信任的源中取出数据可能存在潜在的安全风险，因为pickle模块对恶意代码不安全。从终端导航到movieclassifier目录，启动新的python会话并执行以下代码，以验证您是否可以导入矢量器并取消选取分类器：

>>>进口泡菜

>>>导入RE

>>>导入操作系统

>>>从矢量器导入矢量

>>>CLF=pickle.load（打开（

……os.path.join（'pkl\_对象'，

…'classifier.pkl'），'rb'））

在成功加载矢量器并取消选取分类器之后，我们现在可以使用这些对象预处理文档样本并对其情感进行预测：

>>>导入numpy为np

>>>label=0：'负'，1：'正'

>>>示例=[“我喜欢这部电影”]

>>>X=vect.transform（示例）

>>>print（'预测：%s\n性能：%.2f%%'%'%\

……（标签[clf.predict（x）[0]，

……np.max（clf.预测概率（x））\*100）

预测：阳性

概率：91.56%

由于我们的分类器将类标签返回为整数，因此我们定义了一个简单的

python字典将这些整数映射到它们的情感。然后我们使用

最后，我们使用逻辑回归分类器的预测方法对类标签进行预测，并使用预测概率法返回相应的预测概率。请注意，Predict\_Proba方法调用返回一个数组，该数组具有每个唯一类标签的概率值。由于具有最大概率的类标签对应于预测调用返回的类标签，因此我们使用np.max函数返回预测类的概率。

【254】

为数据存储设置sqlite数据库

在本节中，我们将建立一个简单的sqlite数据库，以收集来自Web应用程序用户的关于预测的可选反馈。我们可以使用此反馈更新分类模型。sqlite是一个开放源代码的SQL数据库引擎，不需要单独的服务器来操作，这使得它非常适合小型项目和简单的Web应用程序。实际上，sqlite数据库可以理解为一个独立的数据库文件，它允许我们直接访问存储文件。此外，sqlite不需要任何特定于系统的配置，并且由所有常见的操作系统支持。它因被谷歌、Mozilla、Adobe、苹果、微软等知名公司所使用而获得了非常可靠的声誉。如果你想了解更多关于sqlite的信息，我建议你访问官方网站

幸运的是，遵循包含python电池的理念，python标准库sqlite3中已经有了一个API，它允许我们使用sqlite数据库（有关sqlite3的更多信息，请访问）。

通过执行以下代码，我们将在movieclassifier目录中创建一个新的sqlite数据库，并存储两个电影评论示例：

>>>导入sqlite 3

>>>导入操作系统

>>>conn=sqlite3.connect（'reviews.sqlite'）

>>>C=conn.cursor（）。

>>>C.Execute（'创建表审阅\u db''\

…'（回顾文本、情感整数、日期文本）'

>>>示例1='我喜欢这部电影'

>>>C.Execute（“插入到review\u db”，\

“…”（回顾、情感、日期）价值观“，\

“…”（？，？，datetime（“now”）“，（示例1，1））

>>>示例2='我不喜欢这部电影'

>>>C.Execute（“插入到review\u db”，\

“…”（回顾、情感、日期）价值观“，\

“…”（？，？，datetime（“now”）“，（示例2，0））

>>>连接提交（）

>>>连接关闭（）

[255]

按照前面的代码示例，我们通过调用sqlite3的connect方法创建了一个到sqlite数据库文件的连接（conn），该方法在movieclassifier目录中创建了新的数据库文件reviews.sqlite（如果它还不存在）。请注意，sqlite不为现有表实现替换功能；如果要再次执行代码，则需要从文件浏览器中手动删除数据库文件。接下来，我们通过cursor方法创建了一个光标，它允许我们使用强大的SQL语法遍历数据库记录。通过第一次执行调用，我们随后创建了一个新的数据库表，review\_db。我们使用它来存储和访问数据库条目。除了review\_db，我们还在此数据库表中创建了三列：review、feelion和date。我们使用这些来存储两个示例电影评论和相应的类标签（情感）。使用SQL命令datetime（“now”），我们还将日期和时间戳添加到条目中。除了时间戳之外，我们还使用了问号符号（？）将电影评论文本（example1和example2）和相应的类标签（1和0）作为位置参数传递给作为元组成员的execute方法。最后，我们调用commit方法来保存对数据库所做的更改，并通过close方法关闭连接。

为了检查条目是否正确存储在数据库表中，我们现在将重新打开与数据库的连接，并使用SQL Select命令获取数据库表中在2015年初到今天之间已提交的所有行：

>>>conn=sqlite3.connect（'reviews.sqlite'）

>>>C=conn.cursor（）。

>>>c.execute（“select\*from review\_db where date”，\

“…”在“2015-01-01 00:00:00”和日期时间（“现在”）之间）

>>>结果=c.fetchall（）

>>>连接关闭（）

>>>打印（结果）

[我喜欢这部电影，1，'2015-06-02 16:02:12'），（'我不喜欢这部电影'，0，'2015-06-02 16:02:12'）]

或者，我们也可以使用免费的firefox浏览器插件sqlite manager（可从获取），它为使用sqlite数据库提供了一个很好的GUI界面，如下图所示：

[256]

使用flask开发Web应用程序

在我们准备好前一小节中对电影评论进行分类的代码之后，让我们讨论一下开发我们的Web应用程序的flask Web框架的基础。在Armin Ronacher于2010年首次发布flask之后，该框架在过去的几年中获得了巨大的欢迎，并且使用flask的流行应用实例包括LinkedIn和Pinterest。因为flask是用python编写的，所以它为我们的python程序员提供了一个嵌入现有python代码（如电影分类器）的方便接口。

烧瓶也被称为微框架，这意味着它的核心是保持瘦和简单，但可以很容易地与其他图书馆扩展。虽然轻量级flask API的学习曲线并没有其他流行的python web框架陡峭，

像姜戈，我鼓励你看看官方的烧瓶

有关其功能的详细信息，请参阅。

如果flask库还没有安装在当前的python环境中，您可以从终端通过pip安装它（在编写时，最新的稳定版本是0.10.1版）：pip安装flask

【257】

我们的第一个flask web应用程序

在本小节中，我们将开发一个非常简单的Web应用程序，以便在实现电影分类器之前更熟悉flask API。首先，我们创建一个目录树：

1st\_flask\_app\_1/app.py模板/first\_app.html

py文件将包含由python解释器执行以运行flask web应用程序的主代码。templates目录是flask将在其中查找静态HTML文件以在Web浏览器中呈现的目录。现在让我们来看看app.py的内容：从flask import flask，render\_template app=flask（uu name\_uuuuu）

@app.route（'/'）def index（）：返回render\_template（'first\_app.html'）

如果uu name\_uu='uu main\_uuuuu'：app.run（）

在本例中，我们将应用程序作为单个模块运行，因此我们初始化了一个新的

flask实例，带有参数“name”，让flask知道它可以在它所在的同一目录中找到HTML模板文件夹（模板）。接下来，我们使用route decorator（@app.route（'/'））来指定应该触发索引函数执行的URL。这里，我们的index函数只是先呈现html文件，它位于templates文件夹中。最后，我们使用run函数仅在该脚本由python解释器直接执行时在服务器上运行应用程序，我们确保使用带有uuu name\_uuu='uuu main\_uuuuuuuuuu'的if语句。

[第258页]

现在，让我们看一下第一个\app.html文件的内容。如果您还不熟悉HTML语法，我建议您访问了解HTML基础知识的有用教程。

<！文档类型HTML>

<html>

<head>

<title>第一个应用程序</title>

头部>

<body>

嗨，这是我的第一个flask web应用程序！<div>

</body>

</html>

这里，我们简单地用一个DIV元素（一个块级元素）填充了一个空的HTML模板文件，其中包含一句话：嗨，这是我的第一个flask web应用程序！.flask允许我们在本地运行应用程序，这对于在公共Web服务器上部署Web应用程序之前开发和测试Web应用程序非常有用。现在，让我们通过从第一个flask\_app\_1目录中的终端执行命令来启动我们的web应用程序：

python3应用程序.py

我们现在应该看到终端中显示的一行如下：

\*运行于http://127.0.0.1:5000/

这一行包含本地服务器的地址。现在，我们可以在Web浏览器中输入此地址，以查看正在运行的Web应用程序。如果一切执行正确，我们现在应该看到一个简单的网站内容：嗨，这是我的第一个烧瓶网络应用程序！.

表单验证和呈现

在本小节中，我们将使用HTML表单元素扩展我们的简单flask Web应用程序，以了解如何使用wtforms库（可以通过pip:pip安装wtforms）从用户收集数据。

[259]

此Web应用程序将提示用户在文本字段中键入自己的姓名，如下屏幕截图所示：

单击提交按钮（打招呼）并验证表单后，将呈现一个新的HTML页面以显示用户名。

我们需要为此应用程序设置的新目录结构如下所示：

1st\_flask\_app\_2/app.py static/style.css模板/

\_ formhelpers.html first\_app.html你好.html

以下是我们修改后的app.py文件的内容：

从flask import flask、render\_template、从wtforms import form、textAreafield、validator请求

【260】

app=烧瓶（uu name\_uuuu）

类HelloForm（Form）：sayHello=TextAreaField（“”，[validators.dataRequired（）]）

@app.route（'/'）def index（）：form=helloForm（request.form）

返回render\_template（'first\_app.html'，form=form）

@app.route（'/hello'，methods=['post']）def hello（）：form=helloForm（request.form）if request.method='post'and form.validate（）：name=request.form['sayhello']return render\_template（'hello.html'，name=name）return render\_template（'first\_app.html'，form=form）

如果uu name\_uuu='uu main\_uuuuu'：app.run（debug=true）

使用wtforms，我们用一个文本字段扩展了index函数，这个文本字段将使用textAreaField类嵌入到起始页中，它会自动检查用户是否提供了有效的输入文本。此外，我们定义了一个新的函数hello，如果表单经过验证，它将呈现一个html页面hello.html。这里，我们使用post方法将表单数据传输到消息体中的服务器。最后，通过在app.run方法中设置debug=true参数，我们进一步激活了flask的调试器。这是开发新Web应用程序的一个有用功能。

现在，我们将通过jinja2模板引擎在文件formhelpers.html中实现一个通用宏，稍后我们将在第一个app.html文件中导入该宏以呈现文本字段：

%宏呈现\_字段（字段）%

<dt>field.label

<dd>field（\*kwargs）safe

%如果field.errors%

<ul class=errors>

%表示字段中的错误。错误%

<li>错误</li>

%endfor%

<ul>

%endif%

<dd>

%endmacro%

[261]

关于Jinja2模板语言的深入讨论超出了本书的范围。但是，您可以在以下位置找到有关jinja2语法的全面文档：

接下来，我们设置一个简单的级联样式表（css）文件style.css，以演示如何修改HTML文档的外观和感觉。我们必须将下面的css文件保存在名为static的子目录中，该子目录是flask查找静态文件（如css）的默认目录，它的字体大小仅为HTML正文元素的两倍。代码如下：

正文字号：2em；

以下是修改后的第一个\u app.html文件的内容，该文件现在将呈现一个文本表单，用户可以在其中输入名称：

<！文档类型HTML>

<html>

<head>

<title>第一个应用程序</title>

<link rel=“stylesheet”href=“url\_for（'static'，filename='style.css'）”>

头部>

<body>%来自“\_formhelpers.html”导入呈现\_field%

你叫什么名字？<div>

<form method=post action=“/hello”>

<dl>

渲染\_字段（form.sayhello）

<dl>

<input type=submit value='say hello'name='submit\_ktn'>

<form>

</body>

</html>

[262]

在first\_app.html的header部分，我们加载了css文件。它现在应该改变HTML正文中所有文本元素的大小。在html body部分，我们从formhelpers.html导入了表单宏，并呈现了我们在app.py文件中指定的sayhello表单。此外，我们在同一表单元素中添加了一个按钮，这样用户就可以提交文本字段条目。

最后，我们创建一个hello.html文件，该文件将通过hello函数内的行返回呈现模板（“hello.html”，name=name）呈现，我们在app.py脚本中定义了该文件，以显示用户通过文本字段提交的文本。代码如下：

<！文档类型HTML>

<html>

<head>

<title>第一个应用程序</title>

<link rel=“stylesheet”href=“url\_for（'static'，filename='style.css'）”>

头部>

<body>

<div>hello name.<div>

</body>

</html>

设置了修改后的flask web应用程序后，我们可以通过从应用程序的主目录执行以下命令在本地运行它，我们可以在Web浏览器中查看结果，网址为http://127.0.0.1:5000/：python3 app.py

如果您是Web开发新手，那么这些概念中的一些乍一看可能非常复杂。在这种情况下，我建议您只需在硬盘上的目录中设置前面的文件，然后仔细检查它们。您将看到flask web框架实际上非常简单，比最初可能出现的要简单得多！另外，为了获得更多帮助，请不要忘记查看优秀的烧瓶文档和示例。

[263]

将电影分类器转换为Web应用程序

既然我们对flask web开发的基础有些熟悉，那么让我们进入下一步，将电影分类器实现到web应用程序中。

在本节中，我们将开发一个Web应用程序，首先提示用户

提交审核后，用户将看到一个新页面，显示预测类标签和预测的概率。此外，用户可以通过单击正确或不正确的按钮来提供有关此预测的反馈，如以下屏幕截图所示：

【264】

如果用户单击了正确或不正确的按钮，我们的分类模型将根据用户的反馈进行更新。此外，我们还将把用户提供的电影评论文本以及从按钮单击中推断出来的建议类标签存储在sqlite数据库中，以供将来参考。单击其中一个反馈按钮后，用户将看到的第三个页面是一个简单的感谢屏幕，其中有一个提交另一个审阅按钮，可将用户重定向回起始页。如下屏幕截图所示：

在我们更深入地了解这个Web应用程序的代码实现之前，我建议您先看看我上传的实时演示。

为了更好地理解我们在本节中要实现的目标。

[265]

首先，让我们来看看我们要为这个电影分类应用程序创建的目录树，如下所示：

在本章的前一节中，我们已经创建了vectorizer.py文件、sqlite数据库reviews.sqlite以及带有pickled python对象的pkl\_对象子目录。

主目录中的app.py文件是包含flask代码的python脚本，我们将使用review.sqlite数据库文件（我们在本章前面创建的）来存储提交到我们的web应用程序的电影评论。templates子目录包含将由flask呈现并显示在浏览器中的HTML模板，静态子目录将包含一个简单的css文件，用于调整呈现的HTML代码的外观。

由于app.py文件相当长，我们将分两步征服它。app.py的第一部分导入我们将需要的python模块和对象，以及要取消拾取和设置分类模型的代码：

从flask import flask、render\_template、从wtforms import form、textAreafield、validator import pickle import sqlite3 import os import numpy as np请求

[266]

#从矢量器的本地目录导入hashingvectorizer import vect app=flask（uu name\_uuuuu）

########准备分类器cur\_dir=os.path.dirname（uu file\_uuf=pickle.load（open（os.path.join）（cur\_dir，

'pkl\_objects/classifier.pkl'），'rb'））db=os.path.join（cur\_dir，'reviews.sqlite'）

定义分类（文档）：

label=0:'负'，1:'正'

x=vect.transform（[文档]）y=clf.predict（x）[0]proba=np.max（clf.predict\_Proba（x））返回标签[y]，proba

DEF系列（文件，Y）：

x=vect.transform（[文档]）clf.partial-fit（x，[y]）

def sqlite\_entry（路径，文档，Y）：conn=sqlite3.connect（路径）c=conn.cursor（）

C.执行（“插入审查数据库（审查、情绪、日期）”。\

“值（？”，？，datetime（“now”）“，（document，y））conn.commit（）conn.close（）。

到目前为止，app.py脚本的第一部分对我们来说应该非常熟悉。我们只需导入hashingvectorizer并取消逻辑回归分类器。接下来，我们定义了一个分类函数来返回预测的类标签以及给定文本文档的相应概率预测。如果提供了文档和类标签，那么train函数可以用来更新分类器。使用sqlite\_entry函数，我们可以在sqlite数据库中存储提交的电影评论及其类标签和个人记录的时间戳。请注意，如果重新启动Web应用程序，CLF对象将重置为其原始的pickled状态。在本章的最后，您将学习如何使用我们在sqlite数据库中收集的数据来永久更新分类器。

【267】

app.py脚本第二部分中的概念对我们来说也应该非常熟悉：

app=flask（uu name\_uuu）class reviewform（表单）：

moviereView=文本区域字段（“”，

[validators.dataRequired（），validators.length（min=15）]）

@app.route（'/'）def index（）：

Form=ReviewForm（请求.Form）

返回render\_template（'reviewform.html'，form=form）

@app.route（'/results'，methods=['post']）def results（）：

form=reviewForm（request.form）if request.method='post'and form.validate（）：

review=request.form['moviereview']y，proba=classify（review）返回render\_template（'results.html'，content=review，prediction=y，

probability=round（proba\*100，2））返回render\_template（'reviewform.html'，form=form）

@app.route（'/Thanks'，methods=['post']）def feedback（）：

feedback=request.form['feedback\_button']review=request.form['review']prediction=request.form['prediction']

inv label=negative'：0，'positive'：1 y=inv label[预测]如果反馈='不正确'：

y=int（非（y））train（review，y）sqlite\_entry（db，review，y）return render\_template（'thanks.html'）

如果uu name\_uuu='uu main\_uuuuu'：app.run（debug=true）

[第268页]

我们定义了一个reviewform类，该类实例化一个文本区域字段，该字段将在reviewform.html模板文件（我们的Web应用程序的登录页）中呈现。这反过来又由index函数呈现。用验证器。长度（min=15）参数，我们要求用户输入至少包含15个字符的评论。在results函数中，我们获取提交的web表单的内容并将其传递给分类器，以预测电影分类器的情感，然后将其显示在rendered results.html模板中。

反馈功能乍一看可能有点复杂。如果用户单击了正确或错误的反馈按钮，它实质上从results.html模板中获取预测类标签，并将预测情绪转换回整数类标签，该标签将用于通过train函数更新分类器，我们将其实现在app.py脚本的第一部分中。此外，如果提供了反馈，则将通过sqlite\_entry函数对sqlite数据库进行新的输入，并最终呈现thank.html模板以感谢用户的反馈。

接下来，让我们看一下reviewform.html模板，它构成了应用程序的起始页：

<！文档类型HTML>

<html>

<head>

<title>Movie Classification<title>

头部>

<body>

<h2>请输入电影评论：</h2>

%来自“\_formhelpers.html”导入呈现\_field%

<form method=post action=“/results”>

<dl>

渲染字段（form.moviereview，cols='30'，rows='10'）

<dl>

<div>

<input type=submit value='submit review'name='submit\_ktn'>

<div>

<form>

</body>

</html>

【269】

这里，我们简单地导入了我们在本章前面的表单验证和呈现部分中定义的相同的FormHelpers.html模板。此宏的render\_field函数用于呈现文本区域字段，用户可以通过页面底部显示的提交审阅按钮提供电影审阅并提交。此TextAreaField宽30列，高10行。

下一个模板results.html看起来更有趣：

<！文档类型HTML>

<html>

<head>

<title>Movie Classification<title>

<link rel=“stylesheet”href=“url\_for（'static'，filename='style.css'）”>

头部>

<body>

<h3>你的电影评论：</h3>

<div>content.<div>

<h3>预测：</h3>

<div>This movie review is<strong>prediction->

（概率：概率%）.<div>

<div id='button'>

<Form action=“/Thanks”method=“post”>

<input type=submit value='correct'name='feedback'u button'>

<input type=submit value='incorrect'name='feedback'u button'>

<input type=hidden value=''prediction'name='prediction'>

<input type=hidden value=''content'name='review'>

<form>

<div>

<div id='button'>

<form action=“/”>

<input type=submit value='submit another review'>

<form>

<div>

</body>

</html>

【270】

首先，我们在相应的字段内容，预测，和概率中插入提交的评审以及预测结果。您可能会注意到，我们在包含正确和错误按钮的表单中再次使用了内容和预测占位符变量。这是一个解决方法，可以将这些值发布回服务器，以更新分类器，并在用户单击这两个按钮之一时存储评审。此外，我们在results.html文件的开头导入了一个css文件（style.css）。此文件的设置非常简单；它将此Web应用程序内容的宽度限制为600像素，并将标有DIV ID按钮的错误和正确按钮向下移动20像素：

本体宽度：600px；

}

#按钮{

填充顶部：20px；

这个CSS文件只是一个占位符，所以请随意调整它，以调整Web应用程序的外观和感觉，使之符合您的喜好。

我们将为Web应用程序实现的最后一个HTML文件是thank.html模板。顾名思义，它只是在通过正确或错误的按钮提供反馈后，向用户提供了一条很好的感谢信息。此外，我们在这个页面的底部放置了一个Submit-anotherReview按钮，它将把用户重定向到起始页面。thanks.html文件的内容如下：

<！文档类型HTML>

<html>

<head>

<title>Movie Classification<title>

头部>

<body>

<h3>感谢您的反馈！</h3>

<div id='button'>

<form action=“/”>

<input type=submit value='submit another review'>

<form>

<div>

</body>

</html>

[271]

现在，在进入下一小节并将其部署到公共Web服务器之前，最好通过以下命令从终端本地启动Web应用程序：python3 app.py

测试完应用程序后，我们也不应该忘记在app.py脚本的app.run（）命令中删除debug=true参数。

将Web应用程序部署到公共服务器

在本地测试了Web应用程序之后，我们现在准备将Web应用程序部署到公共Web服务器上。在本教程中，我们将使用pythonanywhere Web托管服务，该服务专门用于托管python Web应用程序，并使其非常简单和轻松。此外，pythonanywhere还提供了一个初学者帐户选项，让我们可以免费运行单个Web应用程序。

要创建新的pythonanywhere帐户，我们访问网站，然后单击位于右上角的定价和注册链接。接下来，我们单击创建初学者帐户按钮，在这里我们需要提供用户名、密码和有效的电子邮件地址。在我们阅读并同意条款和条件后，我们应该有一个新的帐户。

不幸的是，免费的初学者帐户不允许我们从命令行终端通过ssh协议访问远程服务器。因此，我们需要使用pythonanywhere Web界面来管理我们的Web应用程序。但在我们可以将本地应用程序文件上载到服务器之前，需要为我们的pythonanywhere帐户创建一个新的Web应用程序。单击右上角的仪表板按钮后，我们可以访问显示在页面顶部的控制面板。接下来，我们单击网页顶部现在可见的Web选项卡。我们通过单击左侧的“添加一个新的Web应用程序”按钮继续，该按钮允许我们创建一个名为movieclassifier的新的python 3.4 flask Web应用程序。

在为我们的pythonanywhere帐户创建了一个新的应用程序之后，我们会前往“文件”选项卡，使用pythonanywhere Web界面从本地movieclassifier目录上载文件。上传我们在本地计算机上创建的Web应用程序文件后，我们的pythonanywhere帐户中应该有一个movieclassifier目录。它包含与本地movieclassifier目录相同的目录和文件，如下屏幕截图所示：

[272]

最后，我们再次转到Web选项卡，单击reload<username>.pythonanywhere.com按钮来传播更改并刷新我们的Web应用程序。最后，我们的Web应用程序现在应该可以通过地址<username>.pythonanywhere.com启动和运行并公开使用。

不幸的是，Web服务器对我们的Web应用程序中最微小的问题非常敏感。如果您在pythonanywhere上运行Web应用程序时遇到问题，并且在浏览器中收到错误消息，则可以检查服务器和错误日志，这些日志可以从pythonanywhere帐户的“Web”选项卡访问，以更好地诊断问题。

【273】

更新电影评论分类器

当用户提供有关分类的反馈时，我们的预测模型会即时更新，而当Web服务器崩溃或重新启动时，对CLF对象的更新将被重置。如果我们重新加载Web应用程序，CLF对象将从classifier.pkl pickle文件重新初始化。永久应用更新的一个选项是在每次更新之后再次pickle clf对象。但是，随着用户数量的增加，计算效率会变得非常低，并且如果用户同时提供反馈，可能会损坏pickle文件。另一种解决方案是根据SQLite数据库中收集的反馈数据更新预测模型。一个选项是从pythonanywhere服务器下载sqlite数据库，在我们的计算机上本地更新clf对象，并将新的pickle文件上载到pythonanywhere。为了在本地更新我们计算机上的分类器，我们在movieclassifier目录中创建了一个update.py脚本文件，内容如下：

导入pickle import sqlite3 import numpy as np import os

#从矢量器的本地目录导入hashingvectorizer导入vect def update\_model（db\_path，model，batch\_size=10000）：

conn=sqlite3.connect（db\_path）c=conn.cursor（）。

c.execute（“从审阅中选择\*”）

结果=c.fetchmany（批量大小），而结果：data=np.array（结果）

x=data[：，0]y=data[：，1].astype（int）

classes=np.array（[0，1]）x\_train=vect.transform（x）

clf.部分匹配（x轴列，y，类=类）结果=c.fetchmany（批量大小）

conn.close（）返回无

[274]

cur\_dir=os.path.dirname（uu file\_uuuu）

clf=pickle.load（打开（os.path.join）（cur\_dir，

'pkl\_对象'，

'classifier.pkl'），'rb'））db=os.path.join（cur\_dir，'reviews.sqlite'）更新\_模型（db\_path=db，model=clf，batch\_size=10000）

#如果您确信

#您希望永久更新classifier.pkl文件。

#pickle.dump（clf，open（os.path.join）（cur\_dir，

#'pkl\_objects'、'classifier.pkl'）、'wb'）

#，协议=4）

update\_model函数将一次从sqlite数据库中分批提取10000个条目，除非数据库包含的条目更少。或者，我们也可以通过使用fetch one而不是fetchmany一次获取一个条目，这在计算上效率非常低。如果我们处理的大型数据集超过了计算机或服务器的内存容量，那么使用替代的fetchall方法可能是一个问题。

现在我们已经创建了update.py脚本，我们还可以将其上载到pythonanywhere上的movieclassifier目录，并在主应用程序脚本app.py中导入update\_model函数，以便每次重新启动Web应用程序时从sqlite数据库更新分类器。为此，我们只需添加一行代码即可从app.py顶部的update.py脚本导入update\_model函数：

#从本地目录导入更新函数从更新导入更新\模型

然后我们需要调用主应用程序主体中的update\_model函数：

…如果uuu name\_uuu='uuu main\_uuu'：

更新\_model（filepath=db，model=clf，batch\_size=10000）…

【275】

总结

在本章中，您学习了许多有用的和实用的主题，这些主题扩展了我们对机器学习理论的知识。您学习了如何在培训后序列化模型，以及如何为以后的用例加载模型。此外，我们还创建了一个用于高效数据存储的sqlite数据库，并创建了一个Web应用程序，使我们的电影分类器可供外部世界使用。

在这本书中，我们确实讨论了很多关于机器学习的概念、最佳实践和分类的监督模型。在下一章中，我们将看看另一个子类别监督学习，回归分析，这让我们可以预测一个连续规模的结果变量，相比之下，分类模型的分类类标签，我们一直在研究。

【276】

预测连续

目标变量

回归分析

在前面的章节中，您学习了很多关于监督学习背后的主要概念，并为分类任务培训了许多不同的模型，以预测组成员身份或分类变量。在这一章中，我们将深入探讨另一个子类别的监督学习：回归分析。

回归模型被用来连续地预测目标变量，这使得它们对于解决科学中的许多问题以及工业中的应用具有吸引力，例如理解变量之间的关系、评估趋势或做出预测。一个例子是预测一家公司未来几个月的销售额。

在本章中，我们将讨论回归模型的主要概念，并涵盖以下主题：

探索和可视化数据集

寻找实现线性回归模型的不同方法

训练对异常值具有鲁棒性的回归模型

评估回归模型并诊断常见问题•将回归模型拟合到非线性数据

【277】

引入一个简单的线性回归模型

简单（单变量）线性回归的目标是建立单个特征（解释变量x）和连续值响应（目标变量y）之间的关系模型。一个解释变量的线性模型方程定义如下：y=w0+w1 x

这里，权重w0代表y轴截距，w1是解释变量的系数。我们的目标是学习线性方程的权重，以描述解释变量和目标变量之间的关系，然后可用于预测不属于训练数据集的新解释变量的响应。

根据我们之前定义的线性方程，线性回归可以理解为通过采样点找到最佳拟合直线，如下图所示：

这种最佳拟合线也称回归线，从回归线到采样点的垂直线就是所谓的偏移或残差——我们的预测误差。

[278]

一个解释变量的特殊情况也称为简单线性回归，但我们当然也可以将线性回归模型推广到多个解释变量。因此，这个过程称为多重线性回归：

ny＝W0x0+W1x1+…+WMXM＝Sig.Wi-X= WT x

i=0

这里，w0是x 0=1的y轴截距。

探索住房数据集

在我们实施第一个线性回归模型之前，我们将引入一个新的数据集，即住房数据集，它包含D.Harrison和D.L.Rubinfeld于1978年收集的波士顿郊区住房信息。住房数据集已免费提供，可从UCI机器学习库下载。

506个样本的特征可以总结为数据集描述的摘录：

克里姆：这是按城镇划分的人均犯罪率。

锌：这是超过25000平方英尺的住宅用地的比例。

印度工业：这是每个城镇非零售商业用地的比例。

chas：这是charles river虚拟变量（如果tract bounds river，则等于1；否则为0）

氮氧化物：这是一氧化氮浓度（百万分之一）

这是每个住宅的平均房间数。

年龄：这是1940年以前建造的业主单位的比例。

这是到波士顿五个就业中心的加权距离

rad：这是辐射状公路的可达性指标。

税：这是每10000美元的全额物业税税率。

这是按城镇划分的师生比。

B：按1000（bk-0.63）^2计算，其中bk是按城镇划分的非裔美国人的比例。

LStat：这是人口地位较低的百分比

medv：这是1000美元的自有住房的中值。

[279]

在本章的其余部分，我们将把房价（medv）作为我们的目标变量，即我们希望使用13个解释变量中的一个或多个来预测的变量。在我们进一步研究这个数据集之前，让我们先将它从UCI存储库提取到熊猫数据帧中：

>>>将熊猫作为PD导入

>>>df=pd.read\_csv（'https://archive.ics.uci.edu/ml/machine-learning-

数据库/住房/住房.数据'，

……header=none，sep='\s+'）

>>>df.columns=['crim'、'zn'、'indus'、'chas'，

…'nox'、'rm'、'age'、'dis'、'rad'，

…'税'、'ptratio'、'b'、'lstat'、'medv']

>>>测向头（）

为了确认数据集加载成功，我们显示了数据集的前五行，如下屏幕截图所示：

可视化数据集的重要特性

探索性数据分析（EDA）是机器学习模型培训前的重要和推荐的第一步。在本节的其余部分中，我们将使用图形EDA工具箱中的一些简单但有用的技术，这些技术可以帮助我们直观地检测异常值的存在、数据的分布以及特性之间的关系。

首先，我们将创建一个散点图矩阵，它允许我们在一个地方可视化这个数据集中不同特性之间的成对相关性。为了绘制散点图矩阵，我们将使用Seaborn库中的pairplot函数（http://stanford.edu/~mwaskom/software/seaborn/），这是一个用于根据matplotlib绘制统计图的python库：

>>>导入matplotlib.pyplot为plt

>>>导入Seaborn作为sns>>sns.set（style='whitegrid'，context='notebook'）

[280]

>>>cols=['lstat'、'indus'、'nox'、'rm'、'medv']

>>>sns.pairplot（df[cols]，大小=2.5）；

>>>plt.show（）。

如下图所示，散点图矩阵为我们提供了数据集中关系的有用图形摘要：

[281]

由于空间限制和可读性，我们只绘制了数据集中的五列：lstat、indus、nox、rm和medv。但是，我们鼓励您创建整个数据帧的散点图矩阵，以进一步探索数据。

使用这个散点图矩阵，我们现在可以快速观察数据是如何分布的，以及它是否包含异常值。例如，我们可以看到，rm与住房价格medv（第四行第五列）之间存在线性关系。此外，我们可以在柱状图（散点图矩阵中右下角的子图）中看到，medv变量似乎是正态分布的，但包含几个异常值。

注意，与一般的看法不同，训练线性回归模型不需要解释变量或目标变量是正态分布的。正态性假设只是本书范围之外的某些统计检验和假设检验的一个要求（蒙哥马利，D.C.，佩克，E.A.，维宁，G.线性回归分析导论）。John Wiley and Sons，2012年，第318-319页）。

为了量化特征之间的线性关系，我们现在将创建一个相关矩阵。相关矩阵与协方差矩阵密切相关，我们已经在第4章关于主成分分析（PCA）的章节中看到了这一点，即建立良好的训练集-数据预处理。直观地，我们可以将相关矩阵解释为协方差矩阵的重新调整版本。事实上，相关矩阵与从标准化数据计算出的协方差矩阵是相同的。

相关矩阵是一个包含皮尔逊积矩相关系数（通常简称皮尔逊r）的平方矩阵，用于测量特征对之间的线性相关性。相关系数的范围是-1和1。如果r=1，两个特征具有完全正相关；如果r=0，两个特征没有相关性；如果r=-1，两个特征具有完全负相关。如前所述，Pearson的相关系数可以简单地计算为两个特征x和y（分子）之间的协方差除以它们的标准差（分母）的乘积：

n r=∑i=1（x（i）−x）（y（i）−y）=σxy

∑in（x（i）−μx）2∑in=1（y（i）−μy）2σσx y

=1个

【282】

这里，μ表示相应特征的样本均值，σx y是特征x和y之间的协方差，σx和σy分别是特征的标准差。

我们可以证明标准化特征之间的协方差实际上等于它们的线性相关系数。

让我们首先标准化特征x和y，以获得它们的z分数，我们将分别表示为x′和y′：

x′=x−μx，y′=y−yσxσy

记住，我们计算两个特性之间的（总体）协方差如下：

σxy=−

由于标准化将特征变量集中在平均值0处，我们现在可以计算缩放特征之间的协方差，如下所示：σx y′=1n∑ni（x'−0）（y'−0）

通过恢复，我们得到如下结果：

1N∑Ni X Y

1（一）（一）

N x Y

我们可以简化如下：

σxy

σ'x y=σσx y

[283]

在下面的代码示例中，我们将在之前在散点图矩阵中显示的五个特征列上使用numpy的corrcoef函数，并使用seaborn的heat map函数将相关矩阵数组绘制为热图：

>>>导入numpy为np

>>>cm=np.corrcef（df[cols].values.t）

>>>sns.set（字体比例=1.5）

>>>hm=sns.heatmap（厘米，

……cbar=真，

……annot=真，

……平方=真，

……fmt=.2f'，

……annot\_kws=尺寸：15，

……Yticklabels=cols，

……xtickLabels=列）

>>>plt.show（）。

正如我们在结果图中所看到的，相关矩阵为我们提供了另一个有用的摘要图形，可以帮助我们根据其各自的线性相关性选择特征：

为了拟合线性回归模型，我们对那些与我们的目标变量medv具有高度相关性的特征感兴趣。查看前面的相关矩阵，我们发现目标变量medv与lstat变量的相关性最大（-0.74）。然而，正如您从散点图矩阵中所记得的，lstat和medv之间存在明显的非线性关系。另一方面，rm和medv之间的相关性也相对较高（0.70），考虑到我们在散点图中观察到的这两个变量之间的线性关系，rm似乎是探索变量引入简单线性回归m概念的一个好选择。以下部分中的模型。

[284]

普通最小二乘线性回归模型的实现

在本章的开头，我们讨论了线性回归可以理解为通过我们的训练数据的样本点找到最佳拟合直线。然而，我们既没有定义“最佳拟合”这一术语，也没有讨论拟合此类模型的不同技术。在下面的小节中，我们将使用普通最小二乘法（OLS）来估计回归线的参数，以最小化到采样点的垂直距离平方和（残差或误差）。

用梯度下降法求解回归参数的回归

考虑我们在第2章“训练机器学习分类算法”中实现的自适应线性神经元（adaline）；我们记得，人工神经元使用线性激活函数，我们定义了一个成本函数。

j（），通过优化算法，如梯度下降（gd）和随机梯度下降（sgd），我们将其最小化以学习权重。Adaline中的成本函数是平方误差（SSE）的总和。这与我们定义的OLS成本函数相同：

JW公司

这里，y\_是预测值y\_=wt x（注意术语1/2只是为了方便推导gd的更新规则）。本质上，OLS线性回归可以理解为没有单位阶跃函数的adaline，这样我们就可以获得连续的目标值，而不是类标签-1和1。为了证明相似性，让我们从第2章“训练机器学习算法进行分类”中学习adaline的gd实现，并去掉单位步函数来实现我们的第一个线性回归模型：类线性回归gd（object）：

def uu init\_uuuuu（self，eta=0.001，n\_u iter=20）：self.eta=eta

[285]

self.n？iter=n？iter

def fit（self，x，y）：self.w\_=np.zeros（1+x.shape[1]）self.cost=[]

对于范围内的i（self.n\_iter）：输出=self.net\_输入（x）错误=（y-输出）

self.w\_[1:+=self.eta\*x.t.dot（错误）self.w\_[0]+=self.eta\*errors.sum（）cost=（错误\*\*2）.sum（）/2.0 self.cost\_u.append（cost）返回self

def net\_输入（self，x）：

返回np.dot（x，self.w\_[1:）+self.w\_[0]

def predict（self，x）：返回self.net\_输入（x）

如果您需要更新权重的更新方式，在梯度的相反方向上迈出一步，请重新访问第2章“培训机器学习分类算法”中的adaline部分。

为了观察我们的线性回归器的作用，让我们使用住房数据集中的rm（房间数）变量作为解释变量，来训练一个可以预测medv（住房价格）的模型。此外，我们将标准化变量，以更好地收敛GD算法。代码如下：

>>>X=df['rm']]值

>>>Y=df['medv']值

>>>来自sklearn.preprocessing import standardscaler

>>>sc\_x=标准缩放器（）

>>>sc\_y=标准缩放器（）

>>>x\_std=sc\_x.fit\_transform（x）

>>>y\_std=sc\_y.fit\_transform（y）

>>>lr=线性回归gd（）

>>>左后配合（X&U标准，Y&U标准）

【286】

我们在第2章“训练机器学习分类算法”中讨论了，当我们使用优化算法（如梯度下降）检查收敛性时，最好将成本绘制为周期数（通过训练数据集）的函数。为了缩短长话短说，让我们将成本与时间段数进行比较，以检查线性回归是否收敛：

>>>plt.plot（范围（1，lr.n\_Iter+1），lr.cost\_uuu）

>>>plt.ylabel（'sse'）

>>>plt.xlabel（'epoch'）

>>>plt.show（）。

如下图所示，gd算法在第五个历元后收敛：

接下来，让我们想象一下线性回归线是否适合训练数据。为此，我们将定义一个简单的助手函数，该函数将绘制训练样本的散点图，并添加回归线：

>>>Def Lin\_Regplot（X，Y，型号）：

……散点图（x，y，c='blue'）

……plt.plot（x，model.predict（x），color='red'）

……无返回

现在，我们将使用这个lin-regplot函数根据房价绘制房间数量：

>>>Lin\_Regplot（x\_Std，y\_Std，lr）

>>>plt.xlabel（'平均房间数[rm]（标准化）'

>>>plt.ylabel（'1000美元[medv]价格（标准化）'

>>>plt.show（）。

[287]

如下图所示，线性回归线反映了房价随房间数量的增加而增加的总体趋势：

虽然这种观察是直观的，但数据也告诉我们，在许多情况下，房间的数量并不能很好地解释房价。在本章后面，我们将讨论如何量化回归模型的性能。有趣的是，我们还观察到一条奇怪的线y=3，这表明价格可能已经被削减了。在某些应用中，以原始比例报告预测结果变量也很重要。为了将预测的价格结果按1000美元的坐标轴重新调整，我们可以简单地应用标准缩放器的逆变换方法：

>>>num\_rooms\_std=sc\_x.transform（[5.0]）

>>>price\_std=lr.predict（num\_rooms\_std）

>>>打印（“1000美元价格：%.3f”%）\

……逆变换（价格标准）

1000美元价格：10.840

在前面的代码示例中，我们使用之前训练过的线性回归模型来预测一个有五个房间的房子的价格。根据我们的模型，这样的房子价值10840美元。

【288】

另一方面，值得一提的是，如果我们使用标准化变量，技术上不需要更新截距的权重，因为在这些情况下，Y轴截距始终为0。我们可以通过打印重量快速确认这一点：

>>>打印（'坡度：%.3f'%lr.w\_[1]）

坡度：0.695

>>>print（'截距：%.3f'%lr.w\_[0]）

截距：-0.000

通过Scikit学习估计回归模型的系数

在前一节中，我们实现了回归分析的工作模型。然而，在实际应用中，我们可能会对更有效的实现感兴趣，例如，Scikit Learn的线性回归对象，它利用liblinear库和高级优化算法，更好地处理非标准化变量。这有时对某些应用是可取的：

>>>来自sklearn.linear\_model import linearrelession

>>>SLR=线性回归（）

>>>SLR.fit（x，y）安装

>>>print（'坡度：%.3f'%slr.coef[0]）

坡度：9.102

>>>print（'截距：%.3f'%slr.intercept uu）

截距：-34.671

我们可以通过执行上述代码看到，Scikit Learn的线性回归模型与非标准化的rm和medv变量相匹配，产生了不同的模型系数。让我们将其与我们自己的GD实现进行比较，通过绘制MEDV与RM：

>>>林瑞格图（X，Y，SLR）

>>>plt.xlabel（'平均房间数[rm]（标准化）'

>>>plt.ylabel（'1000美元[medv]价格（标准化）'

>>>plt.show（）。

【289】

现在，当我们通过执行上面的代码来绘制训练数据和我们的拟合模型时，我们可以看到总体结果与我们的gd实现相同：

作为使用机器学习库的一种替代方法，有一个封闭式解决方案，用于求解涉及线性方程系统的OLS，这可以在大多数统计学入门教材中找到：

w1=（x t x）−1 x t y

W0=μY−μYμY\_

这里，μy是真实目标值的平均值，而μy\_是预测响应的平均值。

该方法的优点是保证了分析求解的最优解。但是，如果我们处理的是非常大的数据集，那么用这个公式（有时也称为正态方程）求逆矩阵的计算代价可能太高，或者样本矩阵可能是奇异的（不可逆的），这就是为什么在某些情况下我们更喜欢迭代法。

如果您对如何获得正态方程的更多信息感兴趣，我建议您看一看斯蒂芬·波洛克博士的章节，这是他在莱斯特大学讲授的经典线性回归模型，可从免费获得。

【290】

使用拟合稳健回归模型

RANSAC公司

异常值的存在会严重影响线性回归模型。在某些情况下，我们数据的一个很小的子集可能对估计的模型系数产生很大的影响。有许多统计测试可以用来检测异常值，这超出了本书的范围。然而，删除异常值总是需要我们自己作为数据科学家的判断，以及我们的领域知识。

作为剔除异常值的一种替代方法，我们将研究一种使用随机样本一致性（RANSAC）算法的稳健回归方法，该算法将回归模型与数据子集（所谓的输入值）相匹配。

我们可以将迭代RANSAC算法总结如下：

选择一个随机数的样本作为输入并拟合模型。

根据拟合模型测试所有其他数据点，并将这些点添加到用户给定的入口公差范围内。

使用所有入口重新调整模型。

估计拟合模型相对于进口的误差。

如果性能满足某个用户定义的阈值或达到固定的迭代次数，则终止算法；否则返回步骤1。

现在，让我们用scikit learn的ransacregressor对象将线性模型包装到ransac算法中：

>>>来自sklearn.linear\_model import transacrevisitor

>>>ransac=ransacregressor（linearegression（），

……最大试验次数=100，

……最小样本数=50，

……剩余\_metric=lambda x:np.sum（np.abs（x），轴=1）

……剩余阈值=5.0，

……随机状态=0）

>>>Ransac.fit（X，Y）

[291]

我们将transacregressor的最大迭代次数设置为100，并且使用min\_samples=50，我们将随机选择的样本的最小数量设置为至少50。利用残差度量参数，我们提供了一个可调用lambda函数，它只计算拟合线和采样点之间的绝对垂直距离。通过将“剩余阈值”参数设置为5.0，我们只允许在与拟合线的垂直距离在5个距离单位内的情况下，将样本包括在更高的集合中，这对这个特定的数据集很有效。默认情况下，Scikit Learn使用MAD估计来选择入口阈值，其中MAD表示目标值y的中值绝对偏差。但是，选择适当的入口阈值是特定于问题的，这是RANSAC的一个缺点。近年来，为了自动选择一个好的阈值，人们开发了许多不同的方法。您可以在R.Toldo和A.Fusello's.中找到详细的讨论。在鲁棒多结构拟合中，自动估计入口阈值（图像分析和处理-ICIAP 2009，第123-131页）。斯普林格，2009年）。

在我们拟合了ransac模型之后，让我们从拟合的ransac线性回归模型中获得入口和出口，并将它们与线性拟合一起绘制：

>>>inlier\_mask=ransac.inlier\_mask\_

>>>离群值\_mask=np.logical\_not（inlier\_mask）

>>>测线\_x=np.arange（3，10，1）

>>>line\_y\_ransac=ransac.predict（line\_x[：，np.newaxis]）

>>>PLT.散射（X[inlier\_mask]，Y[inlier\_mask]，

……c='blue'，marker='o'，label='inliers'）

>>>PLT.散射（X[离群值屏蔽]，Y[离群值屏蔽]，

……c='lightgreen'，marker='s'，label='outliers'）

>>>plt.plot（测线x，测线ransac，color='red'）

>>>plt.xlabel（“平均客房数[rm]”）

>>>plt.ylabel（'价格为1000美元[medv]'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

[292]

正如我们在下面的散点图中所看到的，线性回归模型被拟合到检测到的一组入口上，如圆所示：

当我们打印执行以下代码的模型的斜率和截距时，我们可以看到线性回归线与我们在没有RANSAC的情况下在上一节中获得的拟合略有不同：

>>>print（'坡度：%.3f'%ransac.estimator\_u.coef\_[0]）

坡度：9.621

>>>print（'截距：%.3f'%ransac.estimator\_u.intercept\_uu）

截距：-37.137

使用RANSAC，我们降低了这个数据集中异常值的潜在影响，但是我们不知道这种方法是否对未知数据的预测性能有积极的影响。因此，在下一节中，我们将讨论如何评估不同方法的回归模型，这是构建预测建模系统的关键部分。

[293]

评价线性回归模型的性能

在前一节中，我们讨论了如何根据训练数据拟合回归模型。但是，您在前面的章节中了解到，在培训过程中没有看到的数据上测试模型，以获得对其性能的无偏估计是至关重要的。

正如我们在第6章“学习模型评估和超参数调整的最佳实践”中所记得的，我们希望将数据集分成单独的训练和测试数据集，在这些数据集中，我们使用前者来适应模型，而后者来评估其性能，以归纳为未发现的数据。现在，我们不再使用简单的回归模型，而是使用数据集中的所有变量，并训练一个多重回归模型：

>>>来自sklearn.cross\_验证导入火车\_测试\_拆分

>>>X=df.iloc[：，：-1].值>>>Y=df['medv'].值

>>>X\_train，X\_test，Y\_train，Y\_test=列车\_test\_split（

……x，y，test\_size=0.3，random\_state=0）>>>slr=linearegression（）。

>>>SLR.Fit（X\_火车，Y\_火车）

>>>Y\_train\_pred=slr.predict（X\_train）>>>Y\_test\_pred=slr.predict（X\_test）

由于我们的模型使用多个解释变量，因此我们无法在二维图中显示线性回归线（或超平面，精确地说），但我们可以将残差（实际值和预测值之间的差异或垂直距离）与预测值绘制到DI。使我们的回归模型不成立。这些残差图是诊断回归模型的常用图形分析，用于检测非线性和异常值，并检查误差是否随机分布。

使用下面的代码，我们现在将绘制一个残差图，我们只需从预测的响应中减去真正的目标变量：

>>>请分散（Y火车火车火车火车，

……c='blue'，marker='o'，label='training data'）>>>plt.scatter（y\_test\_pred，y\_test\_pred-y\_test，y\_测试）

……c='lightgreen'，marker='s'，label='test data'）

>>>plt.xlabel（'预测值'）>>>plt.ylabel（'残差'）

>>>Plt.Legend（loc='左上角'）

>>>plt.hlines（y=0，xmin=-10，xmax=50，lw=2，color='red'）

>>>plt.xlim（[-10，50]）>>>plt.show（）。

【294】

在执行代码之后，我们应该看到一个带有一条线穿过x轴原点的残差图，如下所示：

在完美预测的情况下，残差将完全为零，这在现实和实际应用中可能永远不会遇到。然而，对于一个好的回归模型，我们期望误差是随机分布的，残差应该随机分布在中心线上。如果我们在残差图中看到模式，这意味着我们的模型无法捕获一些解释性信息，这些信息会泄漏到残差中，正如我们在前面的残差图中看到的那样。此外，我们还可以使用残差图来检测异常值，这些异常值由偏离中心线较大的点表示。

对模型性能的另一个有用的定量度量是所谓的均方误差（mse），它只是我们最小化以适应线性回归模型的SSE成本函数的平均值。MSE对于比较不同的回归模型或通过网格搜索和交叉验证调整其参数非常有用：

毫秒Y

执行以下代码：

>>>来自sklearn.metrics import mean\_squared\_错误

>>>print（'mse-train:%.3f，test:%.3f'%（均方误差（y\_-train，y\_-train\_-pred），均方误差（y\_-test，y\_-test\_-pred）））

【295】

我们将看到训练集上的MSE为19.96，测试集上的MSE更大，值为27.20，这是我们的模型过度拟合训练数据的一个指标。

有时报告确定系数（r2）可能更有用，可以理解为MSE的标准化版本，以便更好地解释模型性能。换句话说，r2是模型捕获的响应方差的分数。r2值定义如下：

r2=1−SSE

不锈钢

这里，sse是误差平方和，sst是平方和。

换句话说，这只是反应的变化。

让我们快速展示一下，r2实际上只是MSE的重新调整版本：

r2=1−SSE

不锈钢

1−毫秒

变量（Y）

对于训练数据集，r2在0和1之间有界，但对于测试集，它可以变为负数。如果r2=1，则模型完全符合相应的mse=0的数据。

根据训练数据评估，我们的模型的r2是0.765，听起来还不错。但是，测试数据集上的r2仅为0.673，我们可以通过执行以下代码进行计算：

>>>从sklearn.metrics导入r2\_分数

>>>打印（‘R^2列车：%.3f，测试：%.3f’%

[296]

……（r2\_分数（y\_train，y\_train\_pred）

……r2\_分数（y\_测验、y\_测验\_pred）））

使用正则化回归方法

正如我们在第3章中讨论的，机器学习分类器使用scikit-learn，正则化是通过添加附加信息来解决过度拟合问题的一种方法，从而缩小模型的参数值，从而对复杂性产生惩罚。最常用的正则化线性回归方法是岭回归、最小绝对收缩选择算子（lasso）和弹性网络法。

岭回归是一个二级惩罚模型，我们只需将权重的平方和添加到最小二乘成本函数：

∑n（（i）−y\_（i））2

J（W）脊=Y

I=1

在这里：

米

L2：宽WJ2

J=1

通过增加超参数λ的值，增加了模型的正则化强度，并缩小了模型的权重。请注意，我们没有将截距项w0正规化。

另一种可以导致稀疏模型的方法是套索。根据正则化强度，某些权重可以变为零，这使得套索作为一种受监督的特征选择技术也很有用：

∑n（（i）−y\_（i））2+λ‖w 1

j（w）套索=y

I=1

在这里：

米

l1：λ瓒瓒w 1=λ∑w j

J=1

[公元297年]

然而，lasso的一个限制是，如果m>n，它最多选择n个变量。脊回归和套索之间的折衷是弹性网，弹性网有一个l1惩罚来产生稀疏度，而l2惩罚来克服套索的一些限制，例如所选变量的数量。

j（w）弹性网络=∑n（y（i）−y\_（i））2+λ∑m w 2+λ∑m wj

I=1 J=1

这些正则化回归模型都可以通过Scikit Learn获得，其用法与正则回归模型类似，只是我们必须通过参数λ指定正则化强度，例如，通过k-折叠交叉验证优化。

岭回归模型可以初始化如下：

>>>来自sklearn.linear\_model import ridge

>>>岭=岭（alpha=1.0）

注意，正则化强度是调节的α，它类似于参数λ。同样，我们可以从线性\_模型子模块初始化lasso回归器：

>>>来自sklearn.linear\_model import lasso

>>>套索=套索（alpha=1.0）

最后，Elasticnet实现允许我们改变l1到l2的比率：

>>>来自sklearn.linear\_model import elasticnet

>>>套索=弹性网（alpha=1.0，l1\_比率=0.5）

例如，如果我们将l1\_比率设置为1.0，那么Elasticnet回归器将等于Lasso回归。有关线性回归的不同实现的更多详细信息，请参见

将线性回归模型转化为曲线-多项式回归

在前面的章节中，我们假设解释变量和响应变量之间存在线性关系。一种解释违反线性假设的方法是通过添加多项式项来使用多项式回归模型：y=w0+w1x+w2 x2x2+…+wd xd

[298]

这里，d表示多项式的度数。虽然我们可以用多项式回归来建立非线性关系模型，但由于线性回归系数w的存在，它仍然被认为是一个多元线性回归模型。

现在，我们将讨论如何使用Scikit的多项式特征Transformer类，学习将一个二次项（d=2）添加到具有一个解释变量的简单回归问题中，并将多项式与线性拟合进行比较。步骤如下：

1.添加二次多项式项：

从sklearn.preprocessing导入多项式特征

>>>X=np.数组（[258.0，270.0，294.0，

…320.0，342.0，368.0，

…396.0，446.0，480.0，

…586.0]）[：，np.newaxis]

>>>Y=np.数组（[236.4，234.4，252.8，

…298.6，314.2，342.2，

…360.8，368.0，391.2，

…390.8]）

>>>LR=线性回归（）

>>>pr=线性回归（）

>>>二次=多项式特征（度=2）

>>>X\_Quad=Squardinate.Fit\_Transform（X）

拟合简单线性回归模型进行比较：

>>>左后配合（X，Y）

>>>x\_fit=np.arange（250600,10）[：，np.newaxis]

>>>Y洇lin\_fit=lr.predict（x\_fit）

在多项式回归的变换特征上拟合多元回归模型：

>>>pr.fit（x\_Quad，y）

>>>y\_quad\_fit=pr.predict（squared.fit\_transform（x\_fit））绘制结果：

>>>plt.scatter（x，y，label='training points'）

>>>Plt.Plot（X轴拟合，Y轴拟合，

……label='linear fit'，linestyle='--'）

>>>plt.plot（x\_fit，y\_quad\_fit，y\_quad\_fit，请点击

……label='二次拟合'）

>>>Plt.Legend（loc='左上角'）

>>>plt.show（）。

【299】

在结果图中，我们可以看到多项式拟合比线性拟合更好地捕捉响应和解释变量之间的关系：

>>>y\_lin\_pred=lr.predict（x）

>>>y\_quad\_pred=pr.predict（x\_quad）

>>>打印（'training mse linear:%.3f，secondary:%.3f'%'（

……平均平方误差

……均方误差（y，y\_quad\_pred）））

训练MSE线性：569.780，二次型：61.330

>>>打印（'training r^2 linear:%.3f，secondary:%.3f'%（

……r2\_分数（y，y\_lin\_pred）

……r2\_分数（y，y\_quad\_pred）））

训练r^2线性：0.832，二次：0.982

正如我们在执行前面的代码之后看到的，MSE从570减少了。

（线性拟合）到61（二次拟合），并且确定系数反映了与二次模型（r2=0.982）的更接近拟合，而不是在这个特殊的玩具问题中的线性拟合（r2=0.832）。

住房数据集中的非线性关系建模

在我们讨论了如何构造多项式特征以适应玩具问题中的非线性关系之后，现在让我们来看一个更具体的例子，并将这些概念应用到房屋数据集中的数据中。通过执行以下代码，我们将使用二次（二次）和三次（三次）多项式对房价和LSTAT（人口地位下降百分比）之间的关系进行建模，并将其与线性拟合进行比较。

[300]

代码如下：

>>>X=df['lstat']]值

>>>Y=df['medv']值

>>>regr=线性回归（）

#创建多项式特征

>>>二次=多项式特征（度=2）

>>>立方=多项式特征（度=3）

>>>X\_Quad=Squardinate.Fit\_Transform（X）

>>>X\_cubic=cubic.fit\_transform（X）

#线性拟合

>>>x\_fit=np.arange（x.min（），x.max（），1）[：，np.newaxis]

>>>regr=regr.fit（x，y）

>>>y\_lin\_fit=regr.predict（x\_fit）

>>>线性\_r2=r2\_分数（y，regr.predict（x））。

#二次拟合

>>>regr=regr.fit（x\_quad，y）

>>>y\_quad\_fit=regr.predict（squared.fit\_transform（x\_fit））。

>>>二次方r2=r2\_分数（y，regr.predict（x\_quad））。

#立方拟合

>>>regr=regr.fit（x\_cubic，y）

>>>y\_cubic\_fit=regr.predict（cubic.fit\_transform（x\_fit））。

>>>立方\r2=r2\_分数（y，regr.predict（x\_cubic））。

#绘图结果

>>>请分散（X，Y，

……label='培训点'，

……color='lightgray'）

>>>Plt.Plot（X轴拟合，Y轴拟合，

……label='linear（d=1），$R^2=%.2f$'

……%线性\_r2，

……color='blue'，

……Lw=2，

……lineStyle=''）

>>>plt.plot（x\_fit，y\_quad\_fit，y\_quad\_fit，请点击

……label='squared（d=2），$R^2=%.2f$'

……%二次曲线

……color='red'，

……Lw=2，

……%立方英尺2，

……color='green'，

……Lw=2，

……lineStyle=--'）

>>>plt.xlabel（“%lower status of the population[lstat]”）

>>>plt.ylabel（'价格为1000美元[medv]'）

>>>Plt.Legend（loc='upper right'）

>>>plt.show（）。

正如我们在结果图中看到的那样，立方拟合比线性拟合和二次拟合更好地捕捉了房价和LStat之间的关系。但是，我们应该知道，添加越来越多的多项式特征会增加模型的复杂性，从而增加过度拟合的机会。因此，在实践中，建议您在单独的测试数据集上评估模型的性能，以估计泛化性能：

此外，多项式特征并不总是非线性关系建模的最佳选择。例如，我们可以通过查看medv-lstat散点图，提出lstat特征变量和medv平方根的对数转换可以将数据投影到适合线性回归拟合的线性特征空间。让我们通过执行以下代码来测试这个假设：

#转换功能

>>>x\_log=np.log（x）

【302】

>>>y\_sqrt=np.sqrt（y）

#调整功能

>>>x\_fit=np.arange（x\_log.min（）-1，

……x\_log.max（）+1，1）[：，np.newaxis]

>>>regr=regr.fit（x\_log，y\_sqrt）

>>>y\_lin\_fit=regr.predict（x\_fit）

>>>线性\_r2=r2\_分数（y\_sqrt，regr.predict（x\_log））。

#绘图结果

>>>请分散（x\_log，y\_sqrt，

……label='培训点'，

……color='lightgray'）

>>>Plt.Plot（X轴拟合，Y轴拟合，

……label='linear（d=1），$r^2=%.2f$'%linear\_r2，

……color='blue'，

……Lw=2）

>>>plt.xlabel（'日志（总体状态较低的百分比[lstat]）'）

>>>plt.ylabel（'$\sqrt price\；in\；\$1000\'s[medv]$'）

>>>Plt.Legend（loc='Lower Left'）

>>>plt.show（）。

在将解释转换成对数空间并取目标变量的平方根后，我们能够用一条线性回归线捕获两个变量之间的关系，该线性回归线似乎比任何多项式特征转换都更适合数据（r2=0.69）。ONS之前：

【303】

用随机森林处理非线性关系

在本节中，我们将介绍随机森林回归，这在概念上不同于本章前面的回归模型。随机森林是多个决策树的集合，可以理解为分段线性函数的总和，与前面讨论的全局线性和多项式回归模型不同。换句话说，通过决策树算法，我们将输入空间细分为更小的区域，从而更易于管理。

决策树回归

决策树算法的一个优点是，如果我们处理非线性数据，它不需要对特征进行任何转换。我们记得从第3章，使用scikit-learn的机器学习分类器之旅，我们通过迭代地分割决策树的节点，直到叶子是纯的或满足停止条件。当我们使用决策树进行分类时，我们将熵定义为杂质的度量，以确定哪个特征分割最大化了信息增益（ig），对于二进制分割，可以定义如下：

ig（dp，x）=i（dp）−n 1p i

这里，x是执行拆分的功能，n p是父节点中的样本数，i是杂质函数，dp是父节点中的训练样本子集，d和d是拆分后左、右子节点中的训练样本子集。记住，我们的目标是找到最大化信息获取的特征分割，换句话说，我们希望找到减少子节点中杂质的特征分割。在第三章，机器学习分类器使用Scikit学习，我们使用熵作为杂质的测量，这是一个有用的分类标准。为了使用决策树进行回归，我们将用MSE替换熵作为节点t的杂质度量：

【304】

其中，nt为t节点训练样本数，dt为t节点训练子集，y（i）为真目标值，y\_t为预测目标值（样本均值）：

y\_t=n1 i∑∈d t y（i）

在决策树回归的背景下，MSE通常也被称为节点内方差，这就是为什么拆分标准也被更好地了解的原因。

作为方差减少。为了了解决策树的线匹配情况，让我们使用SciKit中实现的决策树回归器，学习建模MEDV和LSTAT变量之间的非线性关系：

>>>从sklearn.tree导入decisiontreeregressor

>>>X=df['lstat']]值

>>>Y=df['medv']值

>>>tree=decisiontreeregressor（max\_depth=3）

>>>树.拟合（X，Y）

>>>sort\_idx=x.flatten（）.argsort（）。

>>>Lin\_Regplot（X[排序\_IDX]，Y[排序\_IDX]，树）

>>>plt.xlabel（“%lower status of the population[lstat]”）

>>>plt.ylabel（'价格为1000美元[medv]'）

>>>plt.show（）。

正如我们从结果图中看到的，决策树捕获了数据中的一般趋势。然而，该模型的一个局限性是它不能捕获所需预测的连续性和可微性。此外，我们需要谨慎地为树的深度选择适当的值，以避免数据过拟合或过拟合；这里，深度3似乎是一个不错的选择：

【305】

在下一节中，我们将介绍一种更健壮的回归树拟合方法：随机森林。

随机森林回归

正如我们在第3章中讨论的，使用scikit learn的机器学习分类器之旅，随机森林算法是一种结合多个决策树的集成技术。由于随机性有助于减少模型方差，随机森林通常比单个决策树具有更好的泛化性能。随机林的其他优点是，它们对数据集中的异常值不太敏感，并且不需要太多的参数调整。在随机森林中，我们通常需要试验的唯一参数是集合中的树数。回归的基本随机森林算法几乎与我们在第3章，使用SciKit学习的机器学习分类器教程中讨论的用于分类的随机森林算法相同。唯一的区别是，我们使用MSE准则来生长单个决策树，并且预测的目标变量被计算为所有决策树的平均预测。

现在，让我们使用住房数据集中的所有特性，在60%的样本上拟合一个随机森林回归模型，并在剩余的40%上评估它的性能。代码如下：

>>>X=df.iloc[：，：-1].值

>>>Y=df['medv']值

>>>X轴、X轴、Y轴、Y轴测试=\

……列车测试分割（x，y，

……测试尺寸=0.4，

……随机状态=1）

>>>来自sklearn.ensegle导入RandomForest回归器

>>>森林=RandomForest回归器（..

.n\_估计数=1000，

……标准='mse'，

……随机状态=1，

……n\_作业=-1）

>>>Forest.Fit（X\_火车，Y\_火车）

>>>Y\_train\_pred=森林。预测（X\_train）

>>>Y\_test\_pred=森林。预测（X\_test）

[306]

>>>打印（'mse-train:%.3f，test:%.3f'%（

……均方误差（y\_train，y\_train\_pred）

……均方误差（y检验，y检验）

>>>打印（‘R^2列车：%.3f，测试：%.3f'%（

……r2\_分数（y\_train，y\_train\_pred）

……r2\_分数（y\_测验、y\_测验\_pred）））

MSE列车：3.235，试验：11.635

R^2列车：0.960，试验：0.871

不幸的是，我们发现随机森林倾向于过度匹配训练数据。但是，它仍然能够很好地解释目标变量和解释变量之间的关系（在测试数据集中，r2=0.871）。

最后，让我们来看一下预测的残差：

>>>请分散（y\_train\_pred，

……你的火车

……C='黑色'，

……marker='o'，

……S=35，

……α=0.5，

……label='培训数据'）

>>>PLT.散射（Y\_test\_pred，

……y测试pred-y测试，

……C='浅绿色'，

……marker='s'，

……S=35，

……α=0.7，

……label='测试数据'）

>>>plt.xlabel（'预测值'）

>>>plt.ylabel（'残值'）

>>>Plt.Legend（loc='左上角'）

>>>plt.hlines（y=0，xmin=-10，xmax=50，lw=2，color='red'）

>>>plt.xlim（[-10，50]）

>>>plt.show（）。

【307】

正如已经由r2系数总结的那样，我们可以看到模型比测试数据更适合训练数据，正如Y轴方向的异常值所示。此外，残差的分布似乎并非完全随机分布在零中心点附近，这表明该模型无法捕获所有的勘探信息。然而，残差图表明，与我们在本章前面绘制的线性模型残差图相比，有了很大的改进：

在第三章，机器学习分类器使用SciKit学习，我们还讨论了核心技巧，可以结合支持向量机（SVM）用于分类，这是有用的，如果我们处理非线性问题。尽管讨论超出了本书的范围，SVMS也可以用于非线性回归任务。感兴趣的读者可以在S.R.Gunn:S.R.Gunn等人的一篇优秀报告中找到更多关于回归支持向量机的信息。支持向量机进行分类和回归。（ISIS技术报告，1998年14月）。SVM回归量是

也在SciKit Learn中实现，有关其用法的更多信息，请参见。

【308】

总结

在本章开头，您学习了如何使用简单线性回归分析来建模单个解释变量和连续响应变量之间的关系。然后，我们讨论了一种有用的解释性数据分析技术来观察数据中的模式和异常，这是预测建模任务中的重要第一步。

我们通过使用基于梯度的优化方法实现线性回归，建立了我们的第一个模型。然后，我们看到了如何利用Scikit Learn的线性模型进行回归，并实现了一种鲁棒回归技术（ransac）作为处理异常值的方法。为了评估回归模型的预测性能，我们计算了平方误差的平均值和相关的r2指标。此外，我们还讨论了一种诊断回归模型问题的有用图形方法：残差图。

在讨论了如何将正则化应用于回归模型以降低模型复杂度和避免过度拟合后，我们还介绍了几种非线性关系的建模方法，包括多项式特征变换和随机森林回归器。

我们在前面的章节中详细讨论了监督学习、分类和回归分析。在下一章中，我们将讨论机器学习的另一个有趣的子领域：无监督学习。在下一章中，您将学习如何使用集群分析在缺少目标变量的情况下查找数据中的隐藏结构。

【309】

使用未标记的

数据-聚类分析

在前面的章节中，我们使用有监督的学习技术，使用答案已知的数据构建机器学习模型，课程标签已经在我们的培训数据中提供。在本章中，我们将切换档位并探索集群分析，这是一类无监督的学习技术，它允许我们发现数据中的隐藏结构，因为我们事先不知道正确的答案。集群的目标是在数据中找到一个自然的分组，使得同一集群中的项目比不同集群中的项目更相似。

考虑到集群的探索性，它是一个令人兴奋的主题，在本章中，您将了解以下概念，这些概念可以帮助您将数据组织成有意义的结构：

用常用的K均值算法求相似中心

使用自下而上的方法构建分层簇树

使用基于密度的聚类方法识别物体的任意形状

[311]

用k-均值相似度对物体进行分组

在本节中，我们将讨论一种最流行的聚类算法k-means，它在学术界和工业界都有广泛的应用。群集（或群集分析）是一种技术，它允许我们查找相似对象的组，这些对象彼此之间比其他组中的对象更相关。面向业务的集群应用程序的示例包括按不同主题对文档、音乐和电影进行分组，或者根据常见的购买行为查找具有相同兴趣的客户，作为推荐引擎的基础。

稍后我们将看到，k均值算法非常容易实现，但与其他聚类算法相比，计算效率也非常高，这可能解释了它的流行。K均值算法属于基于原型聚类的范畴。在本章后面，我们将讨论两种其他类型的集群，分级集群和基于密度的集群。基于原型的聚类是指每个聚类由一个原型表示，该原型可以是具有连续特征的相似点的形心（平均值），也可以是类别特征的medoid（最具代表性或最频繁发生的点）。虽然k-均值在识别球形簇方面很好，但是这种聚类算法的缺点之一是我们必须预先指定簇数k。对k的不适当选择可能导致较差的群集性能。在本章后面，我们将讨论肘部方法和轮廓图，这是评估聚类质量的有用技术，以帮助我们确定最佳的聚类数k。

尽管k-means聚类可以应用于更高维度的数据，但为了可视化的目的，我们将使用简单的二维数据集遍历以下示例：

>>>从sklearn.datasets导入生成blobs

>>>X，Y=制造斑点（n\_samples=150，

……n\_特征=2，

……中心=3，

……簇标准=0.5，

……随机播放=真，

……随机状态=0）

>>>导入matplotlib.pyplot为plt

>>>PLT.散射（X[：，0]，

……X[：，1]，

……C='白色'，

[312]

……marker='o'，

……S=50）

>>>plt.grid（）。

>>>plt.show（）。

我们刚刚创建的数据集由150个随机生成的点组成，这些点大致分为三个密度较高的区域，通过二维散点图进行可视化：

在集群的实际应用中，我们没有关于这些样本的任何基本事实类别信息；否则，它将属于有监督学习的类别。因此，我们的目标是根据样本的特征相似性对样本进行分组，我们可以使用k-均值算法来实现这一点，该算法可以概括为以下四个步骤：

从采样点随机选取K形心作为初始聚类中心。

将每个样本分配到最近的质心μ（j），j∈1，，k。

将质心移动到指定给它的样本的中心。

重复步骤2和3，直到集群分配没有更改，或者达到用户定义的公差或最大迭代次数。

[313]

下一个问题是，我们如何测量物体之间的相似性？我们可以将相似性定义为距离的反义词，对于具有连续特征的聚类样本，常用的距离是m维空间中x和y两点之间的欧几里德平方距离：

（）∑m（）2 2 d x，y 2=x j−y j=x−y 2

J=1

注意，在前面的等式中，索引j指的是样本点x和y的j维（特征列）。在本节的其余部分中，我们将使用上标i和j分别指的是样本索引和聚类索引。

基于欧几里得距离度量，我们可以将k-均值算法描述为一个简单的优化问题，一种迭代方法，用于最小化平方误差和（SSE），有时也称为簇惯性：∑∑n k（i，j）x（i）−µ（j）2。

SSE=W i=1 j=1 2

这里，μ（j）是集群j的代表点（质心），如果样本x（i）在集群j中，则w（i，j）=1；否则w（i，j）=0。

既然您已经了解了简单k-means算法的工作原理，那么让我们使用Scikit Learn的cluster模块中的kmeans类将其应用于示例数据集：

>>>从sklearn.cluster导入kmeans

>>>km=kmeans（n\_集群=3，

……init='随机'，

……n\_init=10，

……最大值=300，

……TOL=1e-04，

……随机状态=0）

>>>Y\_km=km.fit\_predict（x）

使用前面的代码，我们将所需集群的数量设置为3；预先指定集群的数量是k-means的限制之一。我们将n\_init=10设置为独立运行k均值聚类算法10次，使用不同的随机中心，选择最终模型作为SSE最低的模型。通过max-iter参数，我们指定了每次运行的最大迭代次数（这里是300）。请注意，如果在达到最大迭代次数之前收敛，SciKit学习中的k-means实现将提前停止。

[314]

然而，对于特定的运行，k-均值可能无法达到收敛，如果我们为max-iter选择相对较大的值，这可能是有问题的（计算上很昂贵）。处理收敛问题的一种方法是为tol选择较大的值，tol是一个参数，它控制关于簇内和平方误差变化的公差，以声明收敛。在前面的代码中，我们选择了1e-04（=0.0001）的公差。

K-表示++

到目前为止，我们讨论了经典的K均值算法，它使用随机种子来放置初始形心，如果初始形心选择不当，有时会导致不良的聚集或缓慢的收敛。解决这个问题的一种方法是在数据集上多次运行k-means算法，并根据SSE选择性能最佳的模型。另一种策略是通过k-means++算法使初始质心彼此远离，这比经典的k-means（D.Arthur和S.Vassilvitskii）得到更好、更一致的结果。K-means++：精心播种的优点。在第十八届年度ACM-SIAM离散算法研讨会论文集，第1027-1035页。工业和应用数学学会，2007年）。

k-means++中的初始化可以总结如下：

初始化一个空集合m以存储所选的k形心。

从输入样本中随机选择第一个质心μ（j），并将其分配给m。

对于不以m为单位的每个样本x（i），求到到m中任何质心的最小平方距离d（x（i），m）2。

要随机选择下一个质心μ（p），请使用加权概率d（μ（p），m）2

分布等于∑i d（x（i），m）2。

重复步骤2和3，直到选择k形心。

继续使用经典的k均值算法。

要将k-means++与scikit learn的kmeans对象一起使用，我们只需要将init参数设置为k-means++（默认设置），而不是随机设置。

[315]

k-means的另一个问题是一个或多个集群可以是空的。请注意，对于k-medoid或模糊c-means，这个问题不存在，我们将在下一小节中讨论这个算法。然而，这个问题在当前Scikit Learn中的k-means实现中得到了解释。如果集群为空，算法将搜索距离空集群形心最远的样本。然后它会将质心重新指定为这个最远的点。

当我们使用欧几里得距离度量将k-均值应用于实际数据时，我们希望确保在相同的尺度上测量特征，并在必要时应用z-得分标准化或最小-最大尺度。

在我们预测了集群标签y\_km并讨论了k均值算法的挑战之后，现在让我们将数据集中k均值识别的集群与集群形心一起可视化。它们存储在拟合的kmeans对象的centers\_uu属性下：

>>>PLT.散射（x[y\_km==0,0]，

……x[y\_km==0,1]，

……S=50，

……C='浅绿色'，

……marker='s'，

……label='cluster 1'）

>>>PLT.散射（x[y\_km==1,0]，

……x[y\_km==1,1]，

……S=50，

……C='橙色'，

……marker='o'，

……label='cluster 2'）

>>>PLT.散射（x[y\_km==2,0]，

……x[y\_km==2,1]，

……S=50，

……C='浅蓝色'，

……marker='v'，

……label='cluster 3'）

>>>PLT.散射（km.cluster\_u centers\_[：，0]，

……km.集群中心

……S=250，

……标记='\*'，

……C='红色'，

……label='centroids'）

>>>Plt.Legend（）。

>>>plt.grid（）。

>>>plt.show（）。

【316】

在下面的散点图中，我们可以看到k-means将三个质心放置在每个球体的中心，这看起来像是给定此数据集的合理分组：

虽然k-means在这个玩具数据集上工作得很好，但是我们需要注意k-means的一些主要挑战。k-means的一个缺点是，我们必须预先指定集群k的数量，这在现实应用中可能并不总是那么明显，特别是当我们使用无法可视化的更高维数据集时。k-means的其他属性是集群不重叠，也不具有层次性，我们还假设每个集群中至少有一个项目。

硬群集与软群集

硬聚类描述了一系列算法，其中数据集中的每个样本都被精确地分配给一个集群，就像我们在前一小节中讨论的k-means算法一样。相反，软聚类算法（有时也称为模糊聚类）将样本分配给一个或多个聚类。软聚类的一个常见例子是模糊C均值（FCM）算法（也称为软K均值或模糊K均值）。最初的想法可以追溯到20世纪70年代，约瑟夫·C·邓恩首先提出了早期版本的模糊聚类来改进k-均值（J·C·邓恩）。等数据处理的模糊相对论及其在紧密分离簇检测中的应用。1973年）。大约十年后，JamesC.Bedzek发表了他在模糊聚类算法改进方面的工作，模糊聚类算法现在被称为FCM算法（J.C.Bezdek）。用模糊目标函数算法进行模式识别。斯普林格科学与商业媒体，2013年）。

[317]

FCM程序与K-Means非常相似。然而，我们用属于每个集群的每个点的概率来替换硬集群分配。在k-means中，我们可以用二进制值的稀疏向量表示样本x的集群成员：

（1）→0\_

\_（2）→1\_\_\_（3）→0\_\_

这里，值为1的索引位置表示样本分配给的簇形心μ（j）（假设k=3，j∈1，2，3）。相反，FCM中的成员向量可以表示为：

（1）→0.1\_

（2）→0.85

\_

（3）→0.05\_\_

\_

在这里，每个值都在范围[0，1]内，并表示属于各自簇形心的概率。给定样本的成员资格之和等于1。与k均值算法类似，我们可以将fcm算法概括为四个关键步骤：

指定k形心的数目，并随机为每个点指定簇成员身份。

计算簇形心μ（j），j 1，，k。

更新每个点的群集成员身份。

重复第2步和第3步，直到成员关系系数不变或达到用户定义的公差或最大迭代次数。

fcm的目标函数我们用jm来缩写它-看起来非常类似于我们用k-平均值最小化的簇内和平方误差：

∑∑n k m（i，j）x（i）−（j）2，m∈[1，∞）

jm=i j w 2

=1=1

[318]

但是，请注意，成员性指标w（i，j）不是k-平均值w（i，j）中的二进制值∈0,1），而是表示集群成员性概率的实值（w（i，j）∈[0,1]）。你可能也注意到我们在w（i，j）上增加了一个额外的指数；指数m，任何大于或等于1的数字（通常m=2），是控制模糊程度的所谓模糊系数（或简单的模糊系数）。m值越大，集群成员w（i，j）越小，导致集群更加模糊。集群成员概率本身计算如下：

W K X

（i，j）=∑p=1 x（（ii））-（（pj））2 m2−1−1

2

例如，如果我们选择三个集群中心作为前一个k-均值示例，我们可以计算属于μ（j）集群的x（i）样本的成员身份，如下所示：

2 2−1

（i）-（j）m-1 x（i）-（j）m-1 x（i）-（j）m-1

2

集群本身的中心μ（j）计算为集群中所有样本的平均值，该平均值由属于自己集群的成员度加权：

wm（i，j）x（i）w

仅仅通过查看计算集群成员资格的方程，就可以直观地说，FCM中的每个迭代都比K均值中的迭代更昂贵。

然而，FCM通常需要较少的迭代才能达到收敛。不幸的是，FCM算法目前没有在SciKit学习中实现。然而，实践中发现，k-means和fcm产生非常相似的聚类输出，如soumi ghosh和sanjay K.dubey（S.ghosh和S.K.dubey）的研究所述。K均值与模糊C均值算法的比较分析。Ijacsa，2013年4:35–38）。

[319]

用肘形法求最佳簇数

无监督学习的主要挑战之一是我们不知道确切的答案。我们的数据集中没有基本真理类标签，这些标签允许我们应用第6章“学习模型评估和超参数优化的最佳实践”中使用的技术，以评估受监控模型的性能。因此，为了量化集群的质量，我们需要使用本章前面讨论的内在度量，如集群内SSE（失真），来比较不同k均值集群的性能。方便的是，我们不需要显式计算集群内的SSE，因为在拟合一个kmeans模型后，它已经可以通过惯性\_u属性访问：

>>>打印（'失真度：%.2f'%km.惯性uu）

畸变：72.48

基于内簇SSE，我们可以使用一个图形化工具，即所谓的肘形法，来估计给定任务中簇k的最佳数目。直观地讲，我们可以说，如果K增加，失真将减少。这是因为样本将更接近它们所分配的质心。肘形法背后的思想是确定K值，其中畸变开始快速增加，如果我们为不同的K值绘制畸变，这将变得更加清晰：

>>>畸变=[]>>对于范围（1，11）中的i：

……km=kmeans（n\_簇=i，

……init='k-means++'，

……n\_init=10，

……最大值=300，

……随机状态=0）

>>>公里.适合（X）

>>>畸变。附加（公里惯性）

>>>plt.plot（范围（1,11），畸变，marker='o'）

>>>plt.xlabel（'簇数'）

>>>plt.ylabel（'失真'）

>>>plt.show（）。

[320]

如下图所示，肘部位于k=3处，这证明k=3确实是该数据集的一个好选择：

通过轮廓图量化聚类质量

另一个评估聚类质量的内在度量是轮廓分析，它也可以应用于聚类算法，而不是本章稍后将讨论的k-均值。轮廓分析可以用作图形工具来绘制集群中样本的紧密分组程度。要计算数据集中单个样本的轮廓系数，我们可以应用以下三个步骤：

计算集群内聚力a（i）作为样本x（i）与同一集群中所有其他点之间的平均距离。

根据样本x（i）与最近集群中所有样本之间的平均距离，计算下一个最近集群中的集群分离b（i）。

将轮廓S（i）计算为簇内聚力和分离之间的差异除以两者中的较大值，如下所示：

S（i）=MABX（i）B−（ia），（ai）（i）

[321]

轮廓系数的范围为-1到1。根据上述公式，当聚类分离度和凝聚力相等（b（i）=a（i））时，轮廓系数为0。此外，如果b（i）>>a（i），我们接近理想轮廓系数1，因为b（i）量化了样本与其他集群的不同程度，a（i）分别告诉我们它与自身集群中的其他样本的相似程度。

剪影系数可作为Scikit Learn公制模块的剪影样本提供，也可选择导入剪影分数。这将计算所有样本的平均轮廓系数，相当于numpy.mean（轮廓样本（…）。通过执行以下代码，我们现在将为k=3的k均值聚类创建轮廓系数图：

>>>km=kmeans（n\_集群=3，

……init='k-means++'，

……n\_init=10，

……最大值=300，

……TOL=1e-04，

……随机状态=0）

>>>Y\_km=km.fit\_predict（x）

>>>导入numpy为np

>>>从Matplotlib导入CM

>>>从sklearn.metrics导入架构\u示例

>>>cluster\_labels=np.unique（y\_km）

>>>n\_clusters=cluster\_labels.shape[0]

>>>Sighture\_vals=Sighture\_示例（x，

……是公里，

……metric='eucliden'）

>>>Y轴下部，Y轴上部=0，0

>>>Yticks=[]>>对于枚举（cluster\_labels）中的I、C：

……C\_Sighture\_vals=轮廓\_vals[y\_km==c]

……剪影效果。排序（）

……Y轴上部+=长度（C轴轮廓）

……颜色=cm.jet（i/n\_簇）

……plt.barh（范围（Y轴下部，Y轴上部）

……剪影壁纸，

……高度=1.0，

……edgecolor='none'，

……颜色=颜色）

……yticks.append（（y\_ax\_lower+y\_ax\_upper）/2）

……Y轴下部+长度（C轴轮廓）

>>>剪影平均值=np.平均值（剪影值）

>>>PLT.AXVLine（轮廓\_avg，

……color=“红色”，

……linestyle=“--”）

>>>plt.yticks（yticks，cluster\_labels+1）

[322]

>>>plt.ylabel（'集群'）

>>>plt.xlabel（'轮廓系数'）

>>>plt.show（）。

通过对轮廓图的目视检查，我们可以快速检查不同集群的大小，并识别包含异常值的集群：

正如我们在前面的轮廓图中看到的，我们的轮廓系数甚至不接近0，这可以作为一个良好的聚类指标。此外，为了总结聚类的优点，我们在图中添加了平均轮廓系数（虚线）。

要了解轮廓图如何寻找相对较差的聚类，让我们在k-means算法中植入两个质心：

>>>km=kmeans（n\_集群=2，

……init='k-means++'，

……n\_init=10，

……最大值=300，

……TOL=1e-04，

……随机状态=0）

>>>Y\_km=km.fit\_predict（x）

>>>PLT.散射（x[y\_km==0,0]，

……x[y\_km==0,1]，

……s=50，c='lightgreen'，

[323]

……marker='s'，

……label='cluster 1'）

>>>PLT.散射（x[y\_km==1,0]，

……x[y\_km==1,1]，

……S=50，

……C='橙色'，

……marker='o'，

……label='cluster 2'）

>>>PLT.散射（km.cluster\_u centers\_[：，0]，

……km.集群中心

……S=250，

……标记='\*'，

……C='红色'，

……label='centroids'）

>>>Plt.Legend（）。

>>>plt.grid（）。

>>>plt.show（）。

正如我们在下面的散点图中看到的，其中一个质心落在采样点的三个球面组中的两个之间。虽然集群看起来并不完全糟糕，但它是次优的。

接下来，我们创建轮廓图来评估结果。请记住，在现实世界问题中，我们通常没有在二维散点图中显示数据集的奢侈，因为我们通常使用更高维度的数据：

>>>cluster\_labels=np.unique（y\_km）

>>>n\_clusters=cluster\_labels.shape[0]

>>>Sighture\_vals=Sighture\_示例（x，

【324】

……是公里，

……metric='eucliden'）

>>>Y轴下部，Y轴上部=0，0

Yticks=[]

>>>对于枚举中的I、C（群集标签）：

……C\_Sighture\_vals=轮廓\_vals[y\_km==c]

……剪影效果。排序（）

……Y轴上部+=长度（C轴轮廓）

……颜色=cm.jet（i/n\_簇）

……plt.barh（范围（Y轴下部，Y轴上部）

……剪影壁纸，

……高度=1.0，

……edgecolor='none'，

……颜色=颜色）

……yticks.append（（y\_ax\_lower+y\_ax\_upper）/2）

……Y轴下部+长度（C轴轮廓）

>>>剪影平均值=np.平均值（剪影值）

>>>plt.axvline（轮廓\_avg，color=“red”，linestyle=“--”）

>>>plt.yticks（yticks，cluster\_labels+1）

>>>plt.ylabel（'集群'）

>>>plt.xlabel（'轮廓系数'）

>>>plt.show（）。

正如我们在结果图中看到的，现在轮廓的长度和宽度明显不同，这为次优聚类提供了进一步的证据：

【325】

将集群组织为层次树

在本节中，我们将介绍一种基于原型的集群的替代方法：层次集群。层次聚类算法的一个优点是，它允许我们绘制树状图（二元层次聚类的可视化），这有助于通过创建有意义的分类法来解释结果。这种分层方法的另一个有用的优点是，我们不需要预先指定集群的数量。

层次聚类的两种主要方法是聚类和分层次聚类。在可分割的层次集群中，我们从一个包含所有样本的集群开始，然后迭代地将集群拆分为较小的集群，直到每个集群只包含一个样本。在本节中，我们将重点讨论聚集性集群，它采用了相反的方法。我们从每个样本作为一个单独的集群开始，合并最近的集群对，直到只剩下一个集群。

聚类层次聚类的两种标准算法是单连杆和完全连杆。使用单连杆，我们计算每对簇的最相似成员之间的距离，并合并两个簇，其中最相似成员之间的距离最小。完全链接方法类似于单个链接，但是，我们不比较每对集群中最相似的成员，而是比较最不同的成员来执行合并。如下图所示：

[326]

其他常用的聚类算法包括平均聚类和沃德聚类。在平均连接中，我们根据两个集群中所有组成员之间的最小平均距离合并集群对。在Ward的方法中，将导致SSE集群内总量最小增加的两个集群合并。

在本节中，我们将重点讨论使用完全链接方法的聚集性集群。这是一个迭代过程，可以通过以下步骤进行总结：

计算所有样本的距离矩阵。

将每个数据点表示为单例集群。

根据最不相似（远）成员的距离合并两个最近的群集。

更新相似矩阵。

重复步骤2到4，直到保留一个集群。

现在我们将讨论如何计算距离矩阵（步骤1）。但是首先，让我们生成一些随机的样本数据。行代表不同的观察结果（ID 0到4），列是这些示例的不同特征（X、Y、Z）：

>>>将熊猫作为PD导入

>>>导入numpy为np

>>>np.random.seed（123）

>>>变量=['X'、'Y'、'Z']

>>>labels=['id\_0'、'id\_1'、'id\_2'、'id\_3'、'id\_4']

>>>X=np.random.random\_sample（[5,3]）\*10

>>>df=pd.dataframe（x，columns=variables，index=labels）

>>>测向

[327]

在执行上述代码之后，我们现在应该看到以下距离矩阵：

在距离矩阵上执行分层聚类

为了计算作为分层聚类算法输入的距离矩阵，我们将使用Scipy的Spatial.Distance子模块中的pdist函数：

>>>来自scipy.space.Distance import pdist，squareform

>>>row\_dist=pd.dataframe（squareform（

……pdist（df，metric='eucledian'）），

……列=标签，索引=标签）

>>>行距离

使用前面的代码，我们根据特征x、y和z计算数据集中每对采样点之间的欧几里得距离。我们提供了pdist返回的压缩距离矩阵作为squareform函数的输入，以创建成对dis的对称矩阵。如图所示：

[328]

接下来，我们将使用scipy的cluster.hierarchy子模块中的链接函数将完整的链接聚合应用到集群，该子模块返回所谓的链接矩阵。

但是，在调用链接函数之前，让我们仔细查看函数文档：

>>>来自scipy.cluster.hierarchy导入链接

>>>帮助（链接）

[…]

参数：Y:N阵列

压缩或冗余的距离矩阵。压缩距离矩阵是包含距离矩阵上三角的平面阵列。这是pdist返回的表单。或者，一组n维的m观测向量可以作为m乘n数组传递。

方法：str，可选链接算法使用。有关完整说明，请参阅下面的“链接方法”部分。

公制：str，可选要使用的距离公制。有关有效距离度量值的列表，请参见Distance.pdist函数。

返回：

z:ndarray层次聚类编码为链接矩阵。

[…]

在函数描述的基础上，我们得出结论，我们可以使用pdist函数的压缩距离矩阵（上三角形）作为输入属性。或者，我们也可以提供初始数据数组，并使用欧几里德度量作为链接中的函数参数。但是，我们不应该使用前面定义的平方距离矩阵，因为它会产生与预期值不同的距离值。综上所述，这里列出了三种可能的情况：

不正确的方法：在这种方法中，我们使用平方距离矩阵。代码如下：

>>>来自scipy.cluster.hierarchy导入链接

>>>row\_clusters=联动（row\_dist，

……method='完成'，

……metric='eucliden'）

[329]

正确的方法：在这种方法中，我们使用压缩距离矩阵。代码如下：

>>>row\_clusters=链接（pdist（df，metric='eucledian'），

……方法='完成'）

正确的方法：在这种方法中，我们使用输入样本矩阵。代码如下：

>>>row\_clusters=链接（df.values，

……method='完成'，

……metric='eucliden'）

为了更仔细地观察聚类结果，我们可以将它们转换为熊猫数据帧（最好在ipython笔记本中查看），如下所示：

>>>PD.数据帧（行\群集，

……columns=['行标签1'，

…'行标签2'，

…'距离'，

…'包括在内的项目数量。']

……index=['cluster%d'%（i+1）for i in

……范围（row\_clusters.shape[0]））

如下表所示，链接矩阵由几行组成，其中每行表示一个合并。第一列和第二列表示每个集群中最不同的成员，第三行报告这些成员之间的距离。最后一列返回每个集群中成员的计数。

现在我们已经计算了链接矩阵，我们可以将结果以树形图的形式可视化：

>>>从scipy.cluster.hierarchy导入树形图

#使树木图变黑（第1/2部分）

#从scipy.cluster.hierarchy导入集链接调色板

#设置“链接”调色板（[“黑色”]）

【330】

>>>row\_dendr=树木图（row\_clusters，

……labels=标签，

……#使树木图变黑（第2/2部分）

……#颜色阈值=np.inf

…）

>>>PLT.Tight\_布局（）

>>>plt.ylabel（'欧几里得距离'）

>>>plt.show（）。

如果您正在执行前面的代码或阅读本书的电子书版本，您将注意到结果树形图中的分支以不同的颜色显示。着色方案是从树形图中为距离阈值循环的matplotlib颜色列表中派生出来的。例如，要以黑色显示树形图，可以取消对前面代码中插入的各个部分的注释。

这样的树形图总结了聚集层次聚类过程中形成的不同聚类；例如，我们可以看到，基于欧几里得距离度量，样本id\_0和id\_4，后跟id\_1和id\_2是最相似的。

[331]

将树木图附加到热图上

在实际应用中，层次聚类树图通常与热图结合使用，这使得我们可以用颜色代码表示样本矩阵中的单个值。在本节中，我们将讨论如何将树形图附加到热图图中，并相应地对热图中的行进行排序。

但是，将树木图附加到热图上可能有点困难，因此让我们一步一步地完成此过程：

我们创建一个新的图形对象，并通过“添加轴”属性定义树形图的X轴位置、Y轴位置、宽度和高度。

此外，我们将树木图逆时针旋转90度。代码如下：

>>>图=PLT.图（图尺寸=（8,8））

>>>axd=图添加轴（[0.09,0.1,0.2,0.6]）

>>>row\_dendr=dendrogram（row\_clusters，orientation='right'）

接下来，我们根据可以通过leaves键从树图对象（本质上是一个python字典）访问的集群标签对初始数据框中的数据重新排序。代码如下：

>>>df\_rowclust=df.ix[行\_dendr['leaves'][：-1]]

现在，我们从重新排序的数据帧构建热图，并将其放置在树枝图旁边：

>>>axm=图添加轴（[0.23,0.1,0.6,0.6]）

>>>CAX=axm.matshow（df\_rowclust，

……interpolation='nearest'，cmap='hot\_r'）

最后，我们将通过删除轴刻度和隐藏轴尖刺来修改热图的美观性。此外，我们将添加一个颜色条，并分别将特征和示例名称分配给X和Y轴刻度标签。代码如下：

>>>axd.set\_xticks（[]）

>>>axd.set\_yticks（[]）>>>对于axd.spines.values（）中的i：

……i.设置“可见”（假）

>>>图颜色条（CAX）

>>>axm.set\_xtickLabels（['']+list（df\_rowclust.columns））。

>>>axm.set\_ytickLabels（['']+list（df\_rowclust.index））。

>>>plt.show（）。

【332】

在执行上述步骤后，应显示热图，并附上树木图：

如我们所见，热量图中的行顺序反映了树木图中样本的聚类。除了一个简单的树形图，每个样本的颜色编码值和热图中的特征为我们提供了一个很好的数据集总结。

【333】

通过Scikit学习应用聚集聚类

在这一部分中，我们看到了如何使用scipy执行聚集层次集群。然而，Scikit Learn中也有一个聚合集群实现，它允许我们选择要返回的集群数量。如果我们想要修剪层次集群树，这是很有用的。通过将n\_cluster参数设置为2，我们现在将使用基于欧几里得距离度量的相同完整链接方法将样本分为两组：

>>>来自sklearn.cluster导入聚合集群

>>>AC=聚集群集（n\_群集=2，

……亲和力='欧几里得'，

……链接='完成'）

>>>labels=ac.fit\_predict（x）

>>>print（'群集标签：%s'%标签）

群集标签：[0 1 1 0]

观察预测的聚类标签，我们可以看到第一个、第四个和第五个样本（ID\_0、ID\_3和ID\_4）被分配到一个聚类（0），而样本ID\_1和ID\_2被分配到第二个聚类（1），这与我们在树形图中观察到的结果一致。

通过定位高密度区域

数据库扫描

尽管在本章中我们不能涵盖大量不同的聚类算法，但我们至少要介绍一种聚类方法：基于密度的应用程序空间聚类（DBSCAN）。dbscan中的密度概念定义为在指定半径ε内的点数。

在DBSCAN中，使用以下标准为每个样本（点）分配一个特殊标签：

如果相邻点的至少一个指定数量（minpts）落在指定半径ε内，则该点被视为核心点。

边界点是在ε范围内相邻点少于minpts，但位于核心点的ε半径内的点。

所有非核心点和边界点的其他点均视为噪声点。

【334】

将这些点标记为核心点、边界点或噪声点后，DBSCAN算法可以概括为两个简单步骤：

为每个核心点或一组相连的核心点组成一个单独的集群（如果核心点距离不超过ε，则连接核心点）。

将每个边界点分配给其对应核心点的集群。

为了更好地了解DBSCAN的结果，在跳到实现之前，我们将在下图中总结您所了解的核心点、边界点和噪声点：

使用dbscan的一个主要优点是，它不假定簇具有K-均值中的球形。此外，DBSCAN不同于K均值聚类和层次聚类，因为它不一定将每个点分配给一个聚类，但能够去除噪声点。

为了更具说明性的示例，让我们创建一个半月形结构的新数据集，比较k-均值聚类、层次聚类和dbscan：

>>>从sklearn.datasets导入make\u moons

>>>x，y=制造月球（n\_samples=200，

……噪声=0.05，

……随机状态=0）

>>>PLT.散射（X[：，0]，X[：，1]）

>>>plt.show（）。

[335]

正如我们在结果图中看到的，有两个可见的半月形组，每个组由100个采样点组成：

我们将从使用k均值算法和完整的连接聚类开始，看看之前讨论过的其中一种聚类算法是否能够成功地将半月形状识别为单独的星团。代码如下：

>>>F，（ax1，ax2）=plt.子批次（1，2，figsize=（8，3））

>>>km=kmeans（n\_集群=2，

……随机状态=0）

>>>Y\_km=km.fit\_predict（x）

>>>ax1.散射（x[y\_km==0,0]，

……x[y\_km==0,1]，

……C='浅蓝色'，

……marker='o'，

……S=40，

……label='cluster 1'）

>>>ax1.散射（x[y\_km==1,0]，

……x[y\_km==1,1]，

……C='红色'，

……marker='s'，

……S=40，

……label='cluster 2'）

>>>ax1.set\_title（'k-表示集群'）

>>>AC=聚集群集（n\_群集=2，

[336]

……亲和力='欧几里得'，

……链接='完成'）

>>>y\_ac=ac.fit\_predict（x）

>>>ax2.散射（x[y\_ac==0,0]，

……x[y\_ac==0,1]，

……C='浅蓝色'，

……marker='o'，

……S=40，

……label='cluster 1'）

>>>ax2.散射（x[y\_ac==1,0]，

……x[y\_ac==1,1]，

……C='红色'，

……marker='s'，

……S=40，

……label='cluster 2'）

>>>ax2.set\_title（'聚集集群'）

>>>Plt.Legend（）。

>>>plt.show（）。

从可视化聚类结果可以看出，K均值算法无法将两个聚类分离，而分层聚类算法受到了这些复杂形状的挑战：

最后，让我们在这个数据集上尝试DBSCAN算法，看看它是否可以使用基于密度的方法找到两个半月形的星团：

>>>从sklearn.cluster导入dbscan

>>>db=dbscan（eps=0.2，

……最小样本数=5，

……metric='eucliden'）

>>>y\_db=db.fit\_predict（x）

【337】

>>>PLT.散射（x[y\_db==0,0]，

……x[y\_db==0,1]，

……C='浅蓝色'，

……marker='o'，

……S=40，

……label='cluster 1'）

>>>PLT.散射（x[y\_db==1,0]，

……x[y\_db==1,1]，

……C='红色'，

……marker='s'，

……S=40，

……label='cluster 2'）

>>>Plt.Legend（）。

>>>plt.show（）。

DBSCAN算法可以成功地检测出半月形状，这突出了DBSCAN（任意形状的聚类数据）的优点之一。

[第338条]

但是，我们也应该注意到DBSCAN的一些缺点。随着数据集中特征数量的增加，给定一个固定大小的训练集，维度诅咒的负面影响会增加。如果我们使用欧几里得距离度量，这尤其是一个问题。然而，维数诅咒的问题并不是DBSCAN所独有的；它也影响到其他使用欧几里得距离度量的聚类算法，例如k-均值和层次聚类算法。此外，我们在dbscan中有两个超参数（minpts和ε），需要对它们进行优化以获得良好的聚类结果。如果数据集中的密度差异相对较大，则找到minpts和ε的良好组合可能会有问题。

到目前为止，我们看到了三种最基本的聚类算法：基于原型的K均值聚类、聚集层次聚类和基于密度的DBSCAN聚类。然而，我还想提到第四类更高级的聚类算法，我们在本章中没有涉及到：基于图的聚类。基于图的聚类家族中最突出的成员可能是光谱聚类算法。尽管光谱聚类有许多不同的实现，但它们都有共同点，即它们使用相似矩阵的特征向量来推导聚类关系。由于光谱聚类超出了本书的范围，您可以阅读Ulrike von Luxburg的优秀教程，了解更多关于这个主题的信息（U.von Luxburg）。光谱聚类教程。统计与计算，17（4）：395–416，2007）。可从ARXIV免费获得

请注意，在实践中，对于给定的数据集，哪种算法执行得最好并不总是显而易见的，尤其是当数据具有多个维度，使其难以或不可能可视化时。此外，必须强调的是，成功的聚类不仅依赖于算法及其超参数。相反，选择一个合适的距离度量和使用有助于指导实验设置的领域知识可能更为重要。

【339】

总结

在本章中，您学习了三种不同的聚类算法，它们可以帮助我们发现数据中隐藏的结构或信息。我们从一个基于原型的方法k-means开始本章，该方法基于特定数量的簇形心将样本簇成球形。由于聚类是一种无监督的方法，因此我们不喜欢使用地面真值标签来评估模型的性能。因此，我们研究了一些有用的内在性能指标，例如肘部方法或轮廓分析，试图量化集群的质量。

然后，我们研究了一种不同的集群方法：集群层次集群。层次集群不需要预先指定集群的数量，结果可以用树状图表示法可视化，这有助于解释结果。我们在本章中看到的最后一个聚类算法是DBSCAN，它是一种基于局部密度对点进行分组的算法，能够处理异常值并识别非小叶形状。

在深入到无监督学习领域之后，现在是时候介绍一些最令人兴奋的机器学习算法了：多层人工神经网络。在最近的复苏之后，神经网络再次成为机器学习研究中最热门的话题。由于最近开发的深度学习算法，神经网络被认为是许多复杂任务的最先进技术，如图像分类和语音识别。在第12章，训练人工神经网络进行图像识别，我们将从头开始构建我们自己的多层神经网络。在第13章中，我们将介绍一个强大的库，它可以帮助我们最有效地训练复杂的网络结构。

【340】

训练人工神经

图像网络

承认

如你所知，深度学习正受到广泛的关注，无疑是机器学习领域最热门的话题。深度学习可以理解为一组算法，它们被开发用来最有效地训练多层人工神经网络。在本章中，您将学习人工神经网络的基本概念，这样您将有足够的能力进一步探索机器学习领域最令人兴奋的研究领域，以及目前正在开发的基于Python的高级深度学习库。

我们将讨论的主题如下：

对多层神经网络的概念理解

训练神经网络进行图像分类

实现强大的反向传播算法•调试神经网络实现

[341]

复杂函数的人工神经网络建模

在这本书的开始，我们开始了我们的旅程，通过机器学习算法与人工神经元在第2章，训练机器学习算法分类。人工神经元是多层人工神经网络的组成部分，我们将在本章中讨论。人工神经网络背后的基本概念建立在人类大脑如何工作以解决复杂问题任务的假设和模型之上。尽管人工神经网络近年来得到了广泛的应用，但早期的神经网络研究可以追溯到20世纪40年代，当时沃伦·麦卡洛克和沃尔特·皮特首次描述了神经元如何工作。然而，在20世纪50年代罗森布拉特的感知器麦卡洛奇-皮特神经元模型首次实施之后的几十年里，许多研究人员和机器学习实践者慢慢开始对神经网络失去兴趣，因为没有人有一个好的方法来训练神经网络。有多层的。最终，当D.E.Rumelhart、G.E.Hinton和R.J.Williams参与（重新）发现和推广反向传播算法以更有效地训练神经网络时，对神经网络的兴趣在1986年重新点燃，我们将在本章后面更详细地讨论。ter（Rumelhart，David E.；Hinton，Geoffrey E.；Williams，Ronald J.（1986）。通过反向传播错误来学习表示。《自然》323（6088）：533–536）。

在过去的十年中，许多更重要的突破导致了我们现在所说的深度学习算法，它可以用来创建特征检测器，从未标记的数据到预先训练由许多层组成的深度神经网络。神经网络不仅是学术研究领域的热门话题，也是Facebook、微软和谷歌等大型科技公司的热门话题，这些公司大量投资于人工神经网络和深度学习研究。到目前为止，在图像和语音识别等复杂问题的解决中，由深度学习算法支持的复杂神经网络被认为是最先进的。在我们的日常生活中，由深度学习提供动力的产品的流行例子是谷歌的图像搜索和谷歌翻译，这是一个智能手机应用程序，可以自动识别图像中的文本，以便实时翻译成20种语言（）。

【342】

许多更令人兴奋的深层神经网络应用正在主要科技公司积极开发中，例如Facebook的DeepFace（Y.Taigman、M.Yang、M.Ranzato和L.Wolf）。DeepFace：在面对面验证中缩小与人的绩效差距。在计算机视觉和模式识别cvpr，2014年IEEE会议，第1701-1708页）和百度

Deepspeech能够处理普通话语音查询（A.Hannun、C.Case、J.Casper、B.Catanzaro、G.Diamos、E.Elsen、R.Prenger、S.Satheesh、S.Sengupta、A.Coates等。Deepspeech：扩展端到端语音识别。arxiv预印arxiv:1412.5567，2014）。此外，制药行业最近开始使用深度学习技术进行药物发现和毒性预测，研究表明，这些新技术大大超过了传统虚拟筛选方法（T.Unterthiner，A.Mayr，G.Kla）的性能。mbauer和S.Hochreiter。使用深度学习进行毒性预测。arxiv预印arxiv:1503.01445，2015）。

单层神经网络综述

本章主要介绍多层神经网络，它们如何工作，以及如何训练它们解决复杂问题。然而，在深入研究一个特定的多层神经网络结构之前，让我们简单地重复一下我们在第2章中介绍的单层神经网络的一些概念，即用于分类的训练机器学习算法，即自适应线性神经元（adaline）如下图所示的算法：

[343]

在第二章，训练机器学习算法进行分类，我们实现了ADALINE算法进行二值分类，并使用梯度下降优化算法来学习模型的权重系数。在每个历代（通过训练集），我们使用以下更新规则更新权重向量w：w：=w+∆w，其中∆w=−ηj（w）

换言之，我们根据整个训练集计算梯度，并通过向梯度j（w）的相反方向迈出一步来更新模型的权重。为了找到模型的最优权重，我们优化了一个目标函数，定义为平方误差和（SSE）成本函数。

J（宽）。此外，我们将梯度乘以一个系数，即学习率η，我们仔细选择该系数来平衡学习速度与超出成本函数全局最小值的风险。

在梯度下降优化中，我们在每个历元后同时更新所有权重，并定义了权重向量中每个权重wj的偏导数。

W如下：

j w j（w）=∑i（y（i）−a（i））x（ji）

这里y（i）是特定样本x（i）的目标类标签，a（i）是神经元的激活，在Adaline的特殊情况下，这是一个线性函数。此外，我们定义了激活函数φ（）如下：φ（z）=z=a

这里，净输入是连接输入到输出层的权重的线性组合：z=∑j wj xj=wt x

当我们使用激活φ（z）计算梯度更新时，我们实现了一个阈值函数（重边函数），将连续值输出压缩成二进制类标签进行预测：

如果g（z）≥0，y=1

−1否则

【344】

请注意，尽管Adaline由两层组成，一个输入层和一个输出层，但由于它在输入层和输出层之间具有单一链接，因此它被称为单层网络。

介绍多层神经网络结构

在本节中，我们将看到如何将多个单个神经元连接到多层前馈神经网络；这种特殊类型的网络也被称为多层感知器（MLP）。下图解释了由三层组成的MLP的概念：一个输入层、一个隐藏层和一个输出层。隐藏层中的单元完全连接到输入层，输出层分别完全连接到隐藏层。如果这样一个网络有不止一个隐藏层，我们也叫它深层人工神经网络。

我们可以向MLP添加任意数量的隐藏层，以创建更深层次的网络架构。实际上，我们可以将神经网络中的层和单元的数量视为额外的超参数，我们希望使用第6章“学习模型评估和超参数调整的最佳实践”中讨论的交叉验证来优化给定问题任务。

然而，随着更多的层被添加到网络中，我们稍后将通过反向传播计算的误差梯度将变得越来越小。这种消失梯度问题使得模型学习更具挑战性。因此，针对这类深层神经网络结构的预处理，开发了一种特殊的算法，称为深层学习。

[345]

如上图所示，我们将l th层中的第i个激活单元表示为ai（l），激活单元a0（1）和a0（2）分别是偏差单位，我们将其设置为1。输入层中单位的激活只是输入加上偏差单位：

a0（1）1（i）a1（1）=（1）=x1

am（1）xm（i）

层L中的每个单元通过权重系数连接到层L+1中的所有单元。例如，层L中的k th单元与层L+1中的j th单元之间的连接将写为w（j l，）k。请注意，xm（i）中的上标i代表

第i个样本，而不是第i个层。在下面的段落中，为了清楚起见，我们经常省略上标i。

虽然输出层中的一个单元足以完成二进制分类任务，但我们在上图中看到了一种更通用的神经网络形式，它允许我们通过一个vsall（ova）技术的泛化来执行多类分类。为了更好地理解这是如何工作的，请记住我们在第4章中介绍的一个分类变量的热表示，即构建良好的训练集——数据预处理。例如，我们将把熟悉的IRIS数据集中的三个类标签（0=setosa，1=versicolor，2=virginica）编码如下：

1 0 0

0=0，1=1，2=0 0 0 1

这个热向量表示允许我们使用训练集中存在的任意数量的唯一类标签来处理分类任务。

[346]

如果您不熟悉神经网络表示，那么索引（下标和上标）周围的术语一开始可能看起来有点混乱。你可能想知道为什么我们写w（j l，）k而不是wk（l，）j是指把第l层的k个单位连接到第l+1层的j个单位的重量系数。看起来有点奇怪

首先，当我们对神经网络表示进行矢量化时，在后面的章节中将更有意义。例如，我们将通过矩阵w总结连接输入层和隐藏层的权重，其中h是隐藏单元数，m+1是隐藏单元数加上偏移单元。既然按照本章后面的概念内化这个符号是很重要的，那么让我们总结一下我们刚才在一个简化的3-4-3多层感知器的描述性说明中讨论的内容：

通过正向传播激活神经网络

在本节中，我们将描述正向传播的过程，以计算MLP模型的输出。为了了解它如何适应学习MLP模型的背景，让我们用三个简单的步骤来总结MLP学习过程：

从输入层开始，我们通过网络转发训练数据的模式以生成输出。

基于网络的输出，我们使用稍后将要描述的成本函数计算要最小化的错误。

我们对误差进行了反向传播，找到了它对网络中每个权重的导数，并对模型进行了更新。

【347】

最后，在重复多个时期的步骤并学习MLP的权重后，我们使用正向传播计算网络输出，并应用阈值函数获得一个热表示中的预测类标签，我们在前一节中描述了这一点。

现在，让我们通过前向传播的各个步骤，从训练数据中的模式生成输出。由于隐藏单元中的每个单元都连接到输入层中的所有单元，因此我们首先计算激活A1（2），如下所示：

z1（2）=a0（1）w1（，10）+a1（1）w1（，11）++am（1）w1（，1米）

A1（2）=φ（Z1（2））

这里，z1（2）是净输入，而φ（）是激活函数，它必须是可区分的，才能通过基于梯度的方法学习连接神经元的权重。为了能够解决图像分类等复杂问题，我们需要在MLP模型中使用非线性激活函数，例如，我们在第3章逻辑回归中使用的Sigmoid（Logistic）激活函数，使用SciKit学习的机器学习分类器之旅：

φ（z）=1+1e−z

我们可以记住，sigmoid函数是一条S形曲线，它将净输入映射到0到1范围内的逻辑分布，该分布通过z=0.5处的原点，如下图所示：

【348】

MLP是前馈人工神经网络的典型例子。“前馈”一词指的是，每一层都作为没有回路的下一层的输入，这与我们将在本章后面讨论的递归神经网络不同。多层感知器这个术语听起来有点令人困惑，因为这种网络结构中的人工神经元通常是乙状体，而不是感知器。直观地说，我们可以将MLP中的神经元视为逻辑回归单位，其返回值在0到1之间的连续范围内。

为了提高代码效率和可读性，我们现在将使用基本线性代数的概念以更紧凑的形式编写激活，这将允许我们通过numpy向量化代码实现，而不是为循环编写多个嵌套且昂贵的python：

Z（2）=W（1）A（1）A（2）=φ（Z（2））

【349】

这里，a（1）是样本x（i）的[m+1]×1维特征向量加上偏差单位。w（1）是一个h×[m+1]维权矩阵，其中h是我们神经网络中隐藏单元的数目。在矩阵向量相乘后，我们得到H×1维净输入向量Z（2），以计算活化a（2）（其中a）。

此外，我们可以将此计算归纳为训练集中的所有n个样本：

Z（2）=W（1）A（1）T

这里，A（1）现在是一个n×[m+1]矩阵，矩阵矩阵乘法将产生一个h×n维净输入矩阵z（2）。最后，我们将激活函数φ（）应用于网络输入矩阵中的每个值，得到下一层（这里是输出层）的H×N激活矩阵A（2）：

A（2）=φ（Z（2））

同样，我们可以用矢量化的形式重写输出层的激活：

Z（3）=W（2）A（2）

这里，我们将t×h矩阵w（2）（t是输出单元数）乘以h×n维矩阵a（2），得到t×n维矩阵z（3）（该矩阵中的列表示每个样本的输出）。

最后，我们应用乙状结肠激活函数来获得我们网络的连续值输出：

a（3）=φ（z（3）），a（3）∈t×n

手写数字分类

在前一节中，我们介绍了很多关于神经网络的理论，如果您是这个主题的新手，这些理论可能会有点压倒性。在我们继续讨论学习MLP模型权重的算法backpropagation之前，让我们先从理论上稍作休息，看看神经网络的作用。

[350]

神经网络理论可能非常复杂，因此我想推荐两种额外的资源，涵盖我们在本章中详细讨论的一些概念：

T.Hastie、J.Friedman和R.Tibshirani。统计学习要素，第2卷。斯普林格，2009年。

C.M.Bishop等人模式识别和机器学习，第1卷。斯普林格纽约，2006年。

在这一部分中，我们将训练我们的第一个多层神经网络，从Yann Lecun等人构建的流行的mnist数据集（国家标准与技术研究所混合数据库的缩写）中对手写数字进行分类。作为机器学习算法（Y.Lecun、L.Bottou、Y.Bengio和P.Haffner）的流行基准数据集。基于梯度学习的文档识别方法。《IEEE会议录》，86（11）：2278-2324，1998年11月）。

获取mnist数据集

mnist数据集在上公开，由以下四部分组成：

训练集图像：train-images-idx3-ubyte.gz（9.9 MB，47 MB解压缩，60000个样本）

训练集标签：train-labels-idx1-ubyte.gz（29kb，60kb解压，60000个标签）

测试集图像：t10k-images-idx3-ubyte.gz（1.6 MB，7.8 MB，解压缩，10000个样本）

测试集标签：t10k-labels-idx1-ubyte.gz（5 kb，10 kb解压，10000个标签）

mnist数据集由美国国家标准技术研究所（NIST）的两个数据集构成。该培训集由250名不同人群的手写数字、50%的高中生和50%的人口普查局雇员组成。请注意，测试集包含来自同一拆分后不同人员的手写数字。

下载完文件后，我建议从命令行终端使用unix/linux gzip工具解压文件，以便在本地mnist下载目录中使用以下命令提高效率：gzip\*ubyte.gz-d

[351]

或者，如果您使用的是运行在Microsoft Windows上的计算机，则可以使用您最喜欢的解压缩工具。图像以字节格式存储，我们将它们读取到numpy数组中，用于培训和测试MLP实现：

导入操作系统导入结构导入编号为NP

def load mnist（path，kind='train'）：从'path`'标签中加载mnist数据\u path=os.path.join（path，

“%s-labels-idx1-ubyte”

%kind）images\_path=os.path.join（路径，

“%s-images-idx3-ubyte”

%种类）

以open（labels\_path，'rb'）作为lbpath:magic，n=struct.unpack（'>ii'，lbpath.read（8））labels=np.fromfile（lbpath，dtype=np.uint8）

以open（images\_path，'rb'）作为imgpath：

magic，num，rows，cols=struct.unpack（“>iii”，imgpath.read（16））images=np.fromfile（imgpath，

dtype=np.uint8）.remaze（len（labels），784）

返回图像、标签

LOAD\_mnist函数返回两个数组，第一个是N×M维numpy数组（图像），其中N是样本数，M是特征数。培训数据集由60000个培训数字组成，测试集分别包含10000个样本。mnist数据集中的图像包括

28×28像素，每个像素用灰度强度值表示。这里，我们将28×28像素展开为一维行向量，表示图像数组中的行（每行或图像784个）。load\_mnist函数返回的第二个数组（标签）包含相应的目标变量，即手写数字的类标签（整数0-9）。

[352]

我们在图像中的阅读方式一开始可能有点奇怪：

magic，n=struct.unpack（'>ii'，lbpath.read（8））labels=np.fromfile（lbpath，dtype=np.int8）

为了了解这两行代码是如何工作的，让我们看看mnist网站上的数据集描述：

[偏移量][类型][值][描述]

0000 32位整数0x00000801（2049）幻数（最高位优先）

0004 32位整数60000个项目数

0008无符号字节？？标签0009无符号字节？？标签

……

XXXX无符号字节？？标签

使用前面代码的两行，我们首先读取magic number，它是文件协议的描述，以及文件缓冲区中的项数（n），然后使用from file方法将以下字节读取到numpy数组中。我们作为参数传递给struct.unpack的fmt参数值>ii包含两部分：

>：这是big-endian（定义存储字节序列的顺序）；如果您不熟悉big-endian和small-endian这两个术语，可以在维基百科上找到一篇关于endianness的优秀文章（。

这是一个无符号整数。

通过执行以下代码，我们现在将从mnist目录中加载60000个培训实例以及10000个测试样本，在该目录中我们解压缩mnist数据集：

>>>X\_train，Y\_train=装载\_mnist（‘mnist’，kind='train'）

>>>打印（'行：%d，列：%d'

……%（X诳火车形状[0]，X诳火车形状[1]））

行：60000，列：784

>>>X\_test，Y\_test=负载\_mnist（'mnist'，kind='t10k'）

>>>打印（'行：%d，列：%d'

……%（X轴测试形状[0]，X轴测试形状[1]））

行：10000，列：784

[353]

为了了解mnist中的图像是什么样子的，让我们将特征矩阵中的784个像素向量重新整形为原始的28×28图像后的数字0-9的示例可视化，我们可以通过matplotlib的imshow函数进行绘图：

>>>导入matplotlib.pyplot为plt

>>>图，ax=plt.subflots（nrows=2，ncols=5，sharex=true，sharey=true，）>>>ax=ax.flatten（）>>>对于范围（10）中的i：

……img=x\_列[y\_列==i][0].重塑（28，28）

……ax[i].imshow（img，cmap='greys'，interpolation='nearest'）

>>>ax[0].设置xticks（[]）

>>>AX[0].设置时钟（[]）

>>>PLT.Tight\_布局（）

>>>plt.show（）我们现在应该看到2×5子图的绘图，显示每个唯一数字的代表性图像：

此外，我们还可以绘制同一数字的多个示例，以了解这些手写示例的实际差异：

>>>图，ax=plt.子批次（nrows=5，

……ncols=5，

……sharex=真，

……shary=真，）

>>>AX=AX.Flatten（）>>>对于范围（25）内的I：

……img=x\_列[y\_列==7][i]。重塑（28，28）

……ax[i].imshow（img，cmap='greys'，interpolation='nearest'）

>>>ax[0].设置xticks（[]）

>>>AX[0].设置时钟（[]）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

[354]

在执行代码之后，我们现在应该看到数字7的前25个变体。

或者，我们可以将mnist图像数据和标签保存为csv文件，以便在不支持其特殊字节格式的程序中打开它们。但是，我们应该知道，csv文件格式将占用您本地驱动器上更多的空间，如下所示：

列车img.csv:109.5 MB

训练\u labels.csv:120 kb

测试\img.csv:18.3 MB

测试标签：20KB

如果我们决定保存这些csv文件，我们可以在将mnist数据加载到numpy数组后在python会话中执行以下代码：

>>>np.savetxt（'train\_img.csv'，x\_train，

……fmt='i'，分隔符=''）

>>>np.savetxt（'train\_labels.csv'，y\_train，

……fmt='i'，分隔符=''）

>>>np.savetxt（'test img.csv'，x\_test，

……fmt='i'，分隔符=''）

>>>np.savetxt（'test\_labels.csv'，y\_test，

……fmt='i'，分隔符=''）

[355]

保存完csv文件后，我们可以使用numpy的genfromtxt函数将其加载回python：

>>>x\_train=np.genfromtxt（'train\_img.csv'，

……dtype=int，分隔符为'，）

>>>Y\_train=np.genfromtxt（'train\_labels.csv'，

……dtype=int，分隔符为'，）

>>>x\_test=np.genfromtxt（'test\_img.csv'，

……dtype=int，分隔符为'，）

>>>Y\_test=np.genfromtxt（'test\_labels.csv'，

……dtype=int，分隔符为'，）

但是，从csv文件加载mnist数据需要更长的时间，因此我建议您尽可能使用原始字节格式。

实现多层感知器

在本小节中，我们现在将实现MLP的代码，其中一个输入层、一个隐藏层和一个输出层用于对mnist数据集中的图像进行分类。我已经尽力使代码尽可能简单。然而，起初看起来可能有点复杂，我鼓励您从packt发布网站下载本章的示例代码，您可以在该网站上找到用注释和语法突出显示注释的MLP实现，以提高可读性。如果您没有运行随附的i python笔记本中的代码，我建议您将其复制到当前工作目录中的python脚本文件中，例如neuralnet.py，然后可以通过以下命令导入到当前的python会话中：

从NeuralNet进口NeuralNetMLP

代码将包含我们尚未讨论的部分，例如反向传播算法，但是根据第2章中的adaline实现、用于分类的培训机器学习算法以及关于正向传播的讨论，大多数代码对您来说应该很熟悉。在前面的章节中。如果不是所有的代码都对您有直接意义，请不要担心；我们将在本章后面的部分中跟进某些部分。然而，在这个阶段检查代码可以使以后更容易地遵循这个理论。

从scipy.special import expit import sys导入numpy as np

类NeuralNetMLP（对象）：

def uu init\_uuuu（自我，n\_输出，n\_功能，n\_隐藏=30，l1=0.0，l2=0.0，epochs=500，eta=0.001，

[356]

alpha=0.0，decrease\_const=0.0，shuffle=true，minibatches=1，random\_state=none）：

np.random.seed（随机状态）self.n\_output=n\_output self.n\_features=n\_features self.n\_hidden=n\_hidden

self.w1，self.w2=self.\_initialize\_weights（）self.l1=l1 self.l2=l2 self.epochs=epochs self.eta=eta self.alpha=alpha self.decrease\_const=decrease\_const self.shuffle=shuffle self.minibatches=小批次

定义编码标签（self、y、k）：

onehot=np.zeros（（k，y.shape[0]），对于idx，枚举中的val（y）：onehot[val，idx]=1.0返回onehot

定义初始化权重（自身）：

w1=np.random.uniform（-1.0，1.0，size=self.n\_hidden\*（self.n\_features+1））w1=w1.remote（self.n\_hidden，self.n\_features+1）w2=np.random.uniform（-1.0，1.0，size=self.n\_output\*（self.n\_hidden+1））w2=w2.remote（self.n\_output，self.n\_hidden+1）返回w1，w2

定义乙状结肠（Self，Z）：

#expit等于1.0/（1.0+np.exp（-z））返回expit（z）

定义乙状结肠梯度（自，Z）：

sg=自乙状结肠（Z）返回sg\*（1-sg）

def \_add\_bias\_unity（self，x，how='column'）：如果how='column'：

x\_new=np.ones（（x.shape[0]，x.shape[1]+1））

x\_new[：，1:]=x elif how='行'：

[357]

x\_new=np.ones（（x.shape[0]+1，x.shape[1]））

x\_new[1:，：]=x其他：

引发attributeError（“`how` must be`column` or`row`”）返回x\u new

定义前馈（self，x，w1，w2）：

a1=self.\_add\_bias\_unity（x，how='column'）z2=w1.dot（a1.t）a2=self.\_sigmoid（z2）a2=self.\_add\_bias\_unity（a2，how='row'）z3=w2.dot（a2）a3=self.\_sigmoid（z3）return a1，z2，a2，z3，a3

def \_l2\_reg（self，lambda\_u，w1，w2）：返回（lambda\_2.0）\*（np.sum（w1[：，1:]\*\*2）\+np.sum（w2[：，1:]\*\*2））

def \_l1\_reg（self，lambda\_u，w1，w2）：返回（lambda\_2.0）\*（np.abs（w1[：，1:）.sum（）\+np.abs（w2[：，1:）.sum（））

def \_get\_cost（self，y\_enc，output，w1，w2）：term1=-y\_enc\*（np.log（output））term2=（1-y\_enc）\*np.log（1-output）cost=np.sum（term1-term2）

l1\_term=self.\_l1\_reg（self.l1，w1，w2）l2\_term=self.\_l2\_reg（self.l2，w1，w2）cost=cost+l1\_term+l2\_term return cost

定义获取渐变（self，a1，a2，a3，z2，y\_enc，w1，w2）：

#反向传播Sigma3=a3-y\_Enc

z2=self.\_添加\_bias\_Unit（z2，how='row'）sigma2=w2.t.dot（sigma3）\*self.\_sigmoid\_gradient（z2）sigma2=sigma2[1:，：]grad1=sigma2.dot（a1）grad2=sigma3.dot（a2.t）

#正规化

等级1[：，1:+=（w1[：，1:]\*（self.l1+self.l2））

[358]

grad2[：，1:+=（w2[：，1:]\*（self.l1+self.l2））返回grad1，grad2

def预测（self，x）：

a1，z2，a2，z3，a3=自前馈（x，self.w1，self.w2）y\_pred=np.argmax（z3，axis=0）返回y\_pred

def fit（self，x，y，print\_progress=false）：self.cost\_=[]

x\_data，y\_data=x.copy（），y.copy（）y\_enc=self.\_encode\_标签（y，self.n\_输出）

delta\_w1\_prev=np.zeros（self.w1.shape）delta\_w2\_prev=np.zeros（self.w2.shape）for i in range（self.epochs）：

#自适应学习率self.eta/=（1+self.decrease常数\*i）

如果打印进度：sys.stderr.write（

'\repoch:%d/%d'%（i+1，self.epochs））sys.stderr.flush（）。

如果self.shuffle：

idx=np.random.permutation（y\_data.shape[0]）x\_data，y\_data=x\_data[idx]，y\_data[idx]

mini=np.数组分割（范围（

y\_data.shape[0]），self.minibatches）用于mini中的idx：

#前馈

a1，z2，a2，z3，a3=自我前馈（

x[idx]，self.w1，self.w2）cost=self.\u get\_cost（y\_enc=y\_enc[：，idx]，output=a3，w1=self.w1，w2=self.w2）self.cost\_u.append（cost）

[359]

#更新权重

delta\_w1，delta\_w2=self.eta\*grad1，self.eta\*grad2 self.w1-=（delta\_w1+（self.alpha\*delta\_w1\_prev））self.w2-=（delta\_w2+（self.alpha\*delta\_w2\_prev））delta\_w1\_prev，delta\_w2\_prev=delta\_w1，delta\_w2返回自我

现在，让我们初始化一个新的784-50-10 MLP，一个具有784个输入单元（n\_特性）、50个隐藏单元（n\_隐藏）和10个输出单元（n\_输出）的神经网络：

>>>n n=neuralnetmlp（n\_输出=10，

……n\_features=x\_train.shape[1]，

……n\_隐藏=50，

……l2=0.1，

……l1=0.0，

……epoch=1000，

……eta=0.001，

……α=0.001，

……减少常数=0.00001，

……随机播放=真，

……小批量=50，

……随机状态=1）

正如您可能已经注意到的，通过回顾前面的MLP实现，我们还实现了一些附加功能，这些功能在这里总结如下：

l2：l2正则化的λ参数，以减小过拟合程度；等于l1是l1正则化的λ参数。

epochs：通过训练集的次数。

[360]

eta：学习率η。

α：动量学习的一个参数，将前一个梯度的系数添加到权重更新中，以便更快地学习∆wt=ηj（wt）+α∆wt−1（其中t是当前的时间步或历元）。

减少常数：自适应学习速率n的减少常数d，随着时间的推移减少，以便更好地收敛η/1+t×d。

洗牌：在每个历代之前对训练集进行洗牌，以防止算法陷入循环。

小批量：在每个时期将训练数据拆分为k个小批量。为了更快地学习，将分别计算每个小批次的梯度，而不是整个训练数据。

接下来，我们使用已经洗牌的mnist训练数据集中的60000个样本来训练MLP。在执行以下代码之前，请注意，在标准台式计算机硬件上培训神经网络可能需要10-30分钟：

>>>nn.fit（x\_train，y\_train，print\_progress=true）

时代：1000/1000

与我们以前的Adaline实现类似，我们将每个时期的成本保存在我们现在可以看到的成本清单中，确保优化算法达到收敛。在这里，我们仅每50步绘制一次，以说明50个小批次（50个小批次×1000个周期）。代码如下：

>>>plt.plot（range（len（nn.cost\_uuu）），nn.cost\_uuu）

>>>plt.ylim（[0，2000]）

>>>plt.ylabel（成本）

>>>plt.xlabel（'epochs\*50'）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

[361]

正如我们在下面的图表中看到的，成本函数的图表看起来非常嘈杂。这是因为我们用小批量学习训练我们的神经网络，这是随机梯度下降的一个变种。

虽然我们已经在图中看到优化算法在大约800个周期（40000/50=800）后收敛，但是让我们通过在小批量间隔内求平均值，根据周期数绘制一个更平滑的成本函数版本。代码如下：

>>>批次=np.array\_split（range（len（nn.cost\_uuu）），1000）

>>>成本=np.array（nn.cost）

>>>cost\_avgs=[np.mean（cost\_ary[i]）for i in batches]

>>>plt.plot（范围（len（cost\_avgs）），

……成本平均值，

……color='red'）

>>>plt.ylim（[0，2000]）

>>>plt.ylabel（成本）

>>>plt.xlabel（'epochs'）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

[362]

下面的图表给我们提供了一幅更清晰的图片，表明训练算法在第800个世纪后不久收敛：

现在，让我们通过计算预测精度来评估模型的性能：

>>>y\_train\_pred=nn.predict（x\_train）

>>>acc=np.sum（y\_train==y\_train\_pred，axis=0）/x\_train.shape[0]

>>>打印（‘培训精度：%.2f%%'%（acc\*100））

培训精度：97.74%

正如我们所看到的，模型对大多数训练数字进行了正确的分类，但是它如何将其归纳为以前从未见过的数据呢？让我们计算测试数据集中10000个图像的精度：

>>>y\_test\_pred=nn.predict（x\_test）

>>>acc=np.sum（y\_test==y\_test\_pred，axis=0）/x\_test.shape[0]

>>>打印（‘培训精度：%.2f%%'%（acc\*100））

测试精度：96.18%

基于训练和测试精度之间的微小差异，我们可以得出结论：模型只略微超出了训练数据。为了进一步对模型进行微调，我们可以使用第6章“学习模型评估的最佳实践”和“Hyperpa”中讨论的技术来更改隐藏单元的数量、正则化参数的值、学习速率、减少常数的值或自适应学习。Rameter调谐（这是留给读者的练习）。

[363]

现在，让我们来看看我们的MLP所面临的一些图像：

>>>Miscl\_img=x\_测试[y\_测试！=Y\_测试\_pred][：25]

>>>正确的测试！=Y\_测试\_pred][：25]

>>>Miscl\_lab=y\_test\_pred[y\_test！=Y\_测试\_pred][：25]

>>>图，ax=plt.子批次（nrows=5，

……ncols=5，

……sharex=真，

……shary=真，）

>>>AX=AX.Flatten（）>>>对于范围（25）内的I：

……img=miscl\_img[i].整形（28，28）

……ax[i].显示（img，

……cmap='greys'，

……插值='最近'）

……ax[i].设置标题（“%d）t:%d p:%d”

……%（I+1，正确的实验室[I]，其他实验室[I]））

>>>ax[0].设置xticks（[]）

>>>AX[0].设置时钟（[]）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

我们现在应该看到一个5×5的子块矩阵，其中副标题中的第一个数字表示绘图索引，第二个数字表示真正的类标签（t），第三个数字表示预测的类标签（p）。

[364]

正如我们在前面的图中所看到的，其中一些图像甚至对我们人类正确分类具有挑战性。例如，我们可以看到，如果数字的下半部分具有钩状曲率（子段3、16和17），则数字9被分类为3或8。

训练人工神经网络

既然我们已经看到了一个正在运行的神经网络，并且通过查看代码已经基本了解了它是如何工作的，那么让我们更深入地挖掘一些概念，例如逻辑成本函数和我们为学习权重而实现的反向传播算法。

计算物流成本函数

我们作为“获取成本”方法实现的物流成本函数实际上非常简单，因为它与我们在第3章“使用SciKit学习的机器学习分类器之旅”的逻辑回归部分中描述的成本函数相同。

j（w）=-∑n y（i）log（a（i））+（1−y（i））log（1−a（i））。

I=1

这里，a（i）是我们在正向传播步骤中计算的一个层中第i个单元的乙状结肠激活：a（i）=φ（z（i））

现在，让我们添加一个正则化项，它允许我们减少过度拟合的程度。正如您在前面几章中提到的，l2和l1正则化术语定义如下（请记住，我们不规范化偏差单位）：

百万

l2=λw=λw2j和l1=λw=λwj

j=1 j=1

[365]

虽然我们的MLP实现支持L1和L2正则化，但为了简单起见，我们现在只关注L2正则化术语。然而，同样的概念也适用于l1正则化项。通过在物流成本函数中加入L2正则化项，我们得到如下方程：

j（w）=∑i=n1 y（i）log（a（i））+（1−y（i））log（1−a（i））+λ2 w 2 2

由于我们实现了一个多类分类的MLP，这将返回一个T元素的输出向量，我们需要在一个热编码表示中与T×1维目标向量进行比较。例如，特定示例的第三层和目标类（此处为类2）的激活可能如下所示：

0.1 0 A（3）=0.9，Y=1

  

   

0.3 0\_

因此，我们需要将物流成本函数推广到我们网络中的所有激活单元j。因此，我们的成本函数（没有正则化项）变成：

不适用

路虎1号

i=1 k=1

这里，上标i是我们训练集中特定样本的索引。

下面的广义正则化项一开始可能看起来有点复杂，但这里我们只是计算我们添加到第一列的一个层l的所有权重之和（不含偏差项）：

J（W）=-N t对数（φ（z（i））J）+（1−Y（ji））对数（1−φ（z（i））J）

Ij\_

[366]

下式表示二级罚款：

记住，我们的目标是最小化成本函数j（w）。因此，我们需要计算矩阵w相对于网络中每个层的每个权重的偏导数：

\_

在下一节中，我们将讨论反向传播算法，它允许我们计算这些偏导数以最小化成本函数。

注意w由多个矩阵组成。在具有一个隐藏单元的多层感知器中，我们有将输入连接到隐藏层的权重矩阵w（1）和将隐藏层连接到输出层的权重矩阵w（2）。矩阵w的直观可视化如下图所示：

在这个简化的图中，似乎w（1）和w（2）的行数和列数都相同，这通常不是这样，除非我们用相同数量的隐藏单元、输出单元和输入功能初始化MLP。

[367]

如果这听起来有点令人困惑，请继续关注下一节，我们将在后向传播算法的上下文中更详细地讨论w（1）和w（2）的维数。

通过反向传播训练神经网络

在这一部分中，我们将通过反向传播的数学来了解如何非常有效地学习神经网络中的权重。根据您对数学表示的舒适程度，以下方程式起初可能看起来相对复杂。许多人喜欢自下而上的方法，并且喜欢一步一步地检查方程，以发展算法的直觉。但是，如果您喜欢自上而下的方法，并且希望在不使用所有数学符号的情况下了解反向传播，那么我建议您先阅读下一节，发展您对反向传播的直觉，稍后再重新阅读本节。

在上一节中，我们看到了如何将成本计算为激活最后一层和目标类标签之间的差异。现在，我们将看到反向传播算法如何更新MLP模型中的权重，这是我们在\_get\_渐变方法中实现的。正如我们从本章开始所回忆的，我们首先需要应用正向传播来获得输出层的激活，我们将其表述如下：

Z（2）=W（1）A（1）T（隐藏层的净输入）

a（2）=φ（z（2））（激活隐藏层）

Z（3）=Z（2）A（2）（输出层的净输入）

A（3）=φ（Z（3））（输出层激活）

[第368页]

简而言之，我们只是通过网络中的连接转发输入特性，如下所示：

在反向传播中，我们将错误从右向左传播。我们首先计算输出层的误差矢量：

δ（3）=a（3）−y

这里，y是真正类标签的向量。

接下来，我们计算隐藏层的误差项：

这里，是简单的乙状结肠激活函数的导数，我们将其作为“乙状结肠梯度”来实现：

请注意，星号符号（）在此上下文中表示元素乘法。

[369]

尽管，遵循下一个方程并不重要，你可能会好奇我是如何得到激活函数的导数的。我在这里逐步总结了推导过程：

φ′（Z）=Z 1+1e−Z

=（E−Z Z）2

1+E

=（11++e e−z z）2−1+1 e−z 2

=（1+1e−z）−1+1e−z 2

=φ（z）−（φ（z））2

=φ（z）（1−φ（z））。

=A（1−A）

为了更好地理解我们如何计算δ（2）项，让我们更详细地讨论一下。在前面的方程中，我们将t×h维矩阵w（2）的转置（w（2））t相乘；t是输出类标签的个数，h是隐藏单元的个数）。现在，（w（2））t变成一个带有δ（2）的h×t维矩阵，它是一个维向量。然后我们进行了一对乘法

（w（2））δ（3）和（a（2）（1−a（2）），也是t×1维矢量。最后，在得到δ项后，我们现在可以写出成本函数的推导如下：

\_

W

[370]

接下来，我们需要累积L层中每个j个节点的偏导数和L+1层中每个j个节点的i个误差：

∆i（，l）j：=∆i（，l）j+a（jl）δi（l+1）

记住，我们需要为训练集中的每个样本计算∆（i，l）j。因此，更容易将其作为矢量化版本实现，就像前面的MLP代码实现一样：

∆（l）=∆（l）+δ（l+1）（a（l））t

在我们积累了偏导数之后，我们可以加上正则化项，如下所示：

∆（l）：=∆（l）+λ（l）（除最后一项外）

最后，在我们计算了梯度之后，我们现在可以通过对梯度采取相反的步骤来更新权重：

W（L）：=W（L）−η∆（L）

为了将所有内容整合到一起，让我们在下图中总结一下反向传播：

[371]

培养你的直觉来进行反向传播

虽然反传播在近30年前被重新发现和推广，但它仍然是最广泛使用的人工神经网络培训非常有效的算法之一。在这一节中，我们将看到一个更直观的总结，以及这个迷人的算法如何工作的更大画面。

从本质上讲，反向传播只是计算复杂成本函数导数的一种非常有效的方法。我们的目标是使用这些导数来学习权重系数，以便参数化多层人工神经网络。神经网络参数化的挑战在于，我们通常在高维特征空间中处理大量的权重系数。与我们在前几章中看到的其他成本函数相比，神经网络成本函数的误差面不是凸的或光滑的。为了找到成本函数的全局最小值，我们必须克服这个高维成本曲面（局部极小值）中的许多障碍。

你可以从微积分入门课程中回忆起链式法则的概念。

链规则是一种派生复杂嵌套函数的方法，例如，

f（g（x））=y，分解为基本成分：

Y=F G X G X

在计算机代数的背景下，已经发展出一套非常有效地解决这类问题的技术，也称为自动微分。如果您有兴趣了解更多关于机器学习应用程序中自动微分的知识，我建议您参考以下资源：A.G.Baydin和B.A.Pearlmutter。机器学习算法的自动微分。ARXIV预印ARXIV:1404.74562014，可在ARXIV上免费获得。

[372]

自动微分有两种模式，分别是前进模式和后退模式。反向传播只是反向模式自动微分的一个特例。关键是在正向模式下应用链规则可能非常昂贵，因为我们必须为每一层（Jacobian）乘以大型矩阵，最终我们将其乘以向量以获得输出。反向模式的诀窍是我们从右到左开始：我们用一个向量乘一个矩阵，这个向量产生另一个向量，这个向量被下一个矩阵相乘，依此类推。矩阵向量乘法的计算量要比矩阵矩阵乘法便宜得多，这就是为什么反向传播算法是神经网络训练中最常用的算法之一。

基于梯度检验的神经网络调试

人工神经网络的实现可能非常复杂，手动检查是否正确地实现了反向传播总是一个好主意。在本节中，我们将讨论一个称为梯度检查的简单程序，它本质上是我们在网络中的分析梯度和数值梯度之间的比较。梯度检验并不是针对前馈神经网络的，但可以应用于任何其他使用基于梯度优化的神经网络结构。即使您计划使用基于梯度的优化（如线性回归、逻辑回归和支持向量机）来实现更简单的算法，检查梯度计算是否正确也是一个不错的主意。

在前面的章节中，我们定义了一个成本函数j（w），其中w是人工网络权重系数的矩阵。请注意，j（w）大致上是一个“叠加”矩阵，由多层感知器中的矩阵w（1）和w（2）组成，其中有一个隐藏单元。我们将w（1）定义为将输入层连接到隐藏层的h×[m+1]-维矩阵，其中h是隐藏单元数，m是特征数（输入单元）。将隐藏层连接到输出层的矩阵w（2）的尺寸为t×h，其中t是输出单元的数量。然后，我们计算重量wi（，lj）的成本函数导数，如下所示：

\_

[373]

记住，我们是通过向梯度方向迈出相反的一步来更新权重。在梯度检查中，我们将此解析解与数值近似梯度进行比较：

J·W

J（W）≈

ε

这里，ε通常是一个非常小的数字，例如1e-5（注意，1e-5只是0.00001的一个更方便的符号）。直观地说，我们可以将这种有限差分近似看作连接两个权重w和w+ε的成本函数点的割线的斜率（两者都是标量值），如

一个更好的方法是计算由两点公式给出的对称（或中心）差商，从而得出更精确的梯度近似值：

J·W

ε

[374]

通常，数值梯度jn'和解析梯度ja'之间的近似差被计算为l2向量范数。出于实际原因，我们将计算出的梯度矩阵展开为平面向量，以便更方便地计算误差（梯度向量之间的差）：误差=j'n−j'a 2

一个问题是，误差不是尺度不变的（如果权重向量范数也很小，则小误差更为显著）。因此，建议计算归一化差：

J'N−J'A 2相对误差=

J'N 2+J'A 2

现在，我们希望数值梯度和分析梯度之间的相对误差尽可能小。在实现梯度检查之前，我们需要再详细讨论一个问题：通过梯度检查的可接受错误阈值是什么？通过梯度检查的相对误差阈值取决于网络结构的复杂性。根据经验法则，如果正确实现了反向传播，我们添加的隐藏层越多，数值梯度和分析梯度之间的差异就越大。由于在本章中我们已经实现了一个相对简单的神经网络结构，因此我们希望对阈值有相当严格的要求，并定义以下规则：

相对误差<=1e-7表示一切正常！

相对误差<=1e-4意味着情况有问题，我们应该调查一下。

相对错误>1e-4意味着代码中可能有错误。

现在我们已经建立了这些基本规则，让我们实现梯度检查。为此，我们可以简单地使用我们以前实现的NeuralNetMLP类，并将以下方法添加到类体中：

def \_gradient\_checking（self，x，y\_enc，w1，w2，epsilon，grad1，grad2）：“”应用渐变检查（仅用于调试）

退换商品

--------

[375]

相对误差：数值近似梯度和反向传播梯度之间的浮动相对误差。

“”“

num\_grad1=np.zeros（np.shape（w1））epsilon\_ary1=np.zeros（np.shape（w1））for i in range（w1.shape[0]）：for j in range（w1.shape[1]）：epsilon\_ary1[i，j]=epsilon a1，z2，a2，z3，a3=self.\_前馈（

x，w1-epsilon\_Ary1，w2）cost1=自我。GeT\_Cost（y\_Enc，a3，w1-epsilon\_Ary1，w2）a1，z2，a2，z3，a3=自我前馈（

x，w1+epsilon\_Ary1，w2）cost2=自我。GeT\_Cost（y\_Enc，a3，w1+epsilon\_Ary1，w2）

num\_grad1[i，j]=（cost2-cost1）/（2\*epsilon）epsilon\_ary1[i，j]=0

num\_grad2=np.zeros（np.shape（w2））epsilon\_ary2=np.zeros（np.shape（w2））for i in range（w2.shape[0]）：for j in range（w2.shape[1]）：epsilon\_ary2[i，j]=epsilon A1，z2，a2，z3，a3=self.\_前馈（

x，w1，w2-epsilon\_Ary2）cost1=自我获得成本（y\_Enc，a3，w1，w2-epsilon\_Ary2）a1，z2，a2，z3，a3=自我前馈（

[376]

x，w1，w2+epsilon\_Ary2）cost2=自身。\_get\_Cost（y\_Enc，a3，w1，w2+epsilon\_Ary2）num\_ grad2[i，j]=（cost2-cost1）/（2\*epsilon）epsilon\_ary2[i，j]=0

num\_grad=np.hstack（（num\_grad1.flatten（），num\_grad2.flatten（））grad=np.hstack（（grad1.flatten（），grad2.flatten（））norm1=np.linalg.norm（num\_grad-grad）norm2=np.linalg.norm（num\_grad）norm3=np.linalg.norm（grad）relative\_error=norm1/（norm2）+norm3）返回相对误差

梯度检查代码看起来相当简单。不过，我个人的建议是尽可能简单。我们的目标是重复检查梯度计算，因此我们希望通过编写高效但复杂的代码来确保在梯度检查中不会引入任何额外的错误。接下来，我们只需要对fit方法做一个小的修改。在下面的代码中，为了清晰起见，我省略了fit函数开头的代码，并且只需要在注释开始渐变检查和结束渐变检查之间实现我们需要添加到方法中的行：

类mlpgradientcheck（object）：[…]def fit（self，x，y，print\_progress=false）：

[…]

#通过后向传播梯度1计算梯度，梯度2=自身。\_得到梯度（a1=a1，a2=a2，a3=a3，z2=z2，y\_enc=y\_enc[：，idx]，w1=self.w1，w2=self.w2）

##开始渐变检查

[377]

Grad\_Diff=自我检查（

x=x[idx]，y\_enc=y\_enc[：，idx]，w1=self.w1，w2=self.w2，epsilon=1e-5，grad1=grad1，Grad2=Grad2）如果Grad\_diff<=1e-7：

print（“确定：%s”“%grad\_diff）elif grad\_diff<=1e-4:print（“警告：%s”“%grad\_diff”）否则：

print（'问题：%s'%grad\_diff）

##末端坡度检查

#更新权重；[alpha\*delta\_w\_prev]

#对于动量学习，delta\_w1=self.eta\*grad1 delta\_w2=self.eta\*grad2 self.w1-=（delta\_w1+\

（self.alpha\*delta\_w1\_prev）上）

self.w2-=（delta\_w2+\

（self.alpha\*delta\_w2\_prev）上）

delta\_w1\_prev=delta\_w1 delta\_w2\_prev=delta\_w2返回self

假设我们将修改后的多层感知器类命名为

MLPGradientCheck，我们现在可以用10个隐藏层初始化新的MLP。此外，我们禁用正则化、自适应学习和动量学习。此外，我们使用规则的梯度下降，将小批量设置为1。代码如下：

>>>n n\_check=mlpgradientcheck（n\_output=10，n\_features=x\_train.shape[1]，n\_hidden=10，l2=0.0，l1=0.0，epochs=10，eta=0.001，alpha=0.0，decrease const=0.0，minibatches=1，random\_state=1）

[第378页]

梯度检查的一个缺点是它的计算量非常非常昂贵。训练一个启用梯度检查的神经网络是如此缓慢，以至于我们真的只想将它用于调试目的。因此，只对少数训练样本（这里，我们选择5个）进行梯度检查并不少见。代码如下：

>>>nn\_check.fit（x\_train[：5]，y\_train[：5]，print\_progress=false）

好的：2.56712936241E-10

好的：2.94603251069E-10

好的：2.37615620231E-10

好的：2.43469423226E-10

好的：3.37872073158E-10

好的：3.63466384861E-10

好的：2.22472120785E-10

好的：2.33163708438E-10

好的：3.44653686551E-10

好的：2.17161707211E-10

正如我们从代码输出中看到的，我们的多层感知器通过了这个测试，并取得了很好的结果。

神经网络的收敛性

你可能想知道为什么我们不使用规则的梯度下降，而是使用小批量学习来训练我们的神经网络进行手写数字分类。你可能还记得我们讨论过的随机梯度下降，我们曾用来实现在线学习。在在线学习中，我们根据单个训练示例（k=1）一次计算梯度，以执行权重更新。虽然这是一种随机方法，但它通常会导致非常精确的解，其收敛速度比常规梯度下降快得多。小批量学习是随机梯度下降的一种特殊形式，我们根据子集计算梯度。

n个训练样本中的k，1<k<n。小批量学习有优势

通过在线学习，我们可以利用我们的矢量化实现来提高计算效率。但是，我们可以比常规梯度下降更快地更新权重。直观地说，你可以把小批量学习看作是通过一项民意测验预测总统选举的投票率，只询问代表性人群的一部分，而不是询问整个人群。

[379]

此外，我们还添加了更多的调整参数，例如减少常数和自适应学习速率的参数。原因是神经网络比简单的算法（如adaline、逻辑回归或支持向量机）更难训练。在多层神经网络中，我们通常有数百、数千甚至数十亿个权重需要优化。不幸的是，输出函数有一个粗糙的曲面，优化算法很容易陷入局部极小，如下图所示：

请注意，由于我们的神经网络有许多维度，因此这种表示非常简化；它使人的眼睛无法看到实际的成本表面。这里，我们只显示X轴上单个重量的成本曲面。然而，主要的信息是我们不希望我们的算法陷入局部极小。通过提高学习速度，我们可以更容易地摆脱这种局部极小值。另一方面，如果学习率太大，我们也会增加超越全局最优的机会。因为我们是随机初始化权重的，所以我们从一个通常是无可救药的错误的优化问题的解决方案开始。我们在前面定义了一个减少常数，它可以帮助我们在开始时更快地从成本表上爬下来，自适应学习速率可以使我们更好地适应到全局最小值。

【380】

其他神经网络结构

在本章中，我们讨论了一种最流行的前馈神经网络表示，多层感知器。神经网络是目前机器学习领域最活跃的研究课题之一，还有许多其他的神经网络结构远远超出了本书的范围。如果你有兴趣了解更多关于神经网络和算法的深入学习，我建议阅读引言和概述；Y.Bengio。深入学习

人工智能架构。机器学习的基础和趋势，2（1）：1-1272009。Yoshua Bengio的书目前可在免费获得。

尽管神经网络确实是另一本书的主题，但让我们至少简单地看看另外两种流行的结构，卷积神经网络和循环神经网络。

卷积神经网络

卷积神经网络（CNN或ConvNets）因其在图像分类任务中的优异性能而在计算机视觉中得到广泛应用。到目前为止，CNN是深度学习中最流行的神经网络结构之一。卷积神经网络的关键思想是建立多层特征检测器，以考虑输入图像中像素的空间排列。注意，CNN有许多不同的变体。在本节中，我们将只讨论这个体系结构背后的一般概念。如果你有兴趣了解更多关于CNN的信息，我建议你看一下Yann Lecun（）的出版物，他是CNN的共同发明人之一。特别是，我可以推荐以下关于CNN入门的文献：

Y.Lecun、L.Bottou、Y.Bengio和P.Haffner。基于梯度学习的文档识别方法。《IEEE会议录》，86（11）：2278–2324，1998年。

P.Y.Simard、D.Steinkraus和J.C.Platt。卷积神经网络应用于可视文档分析的最佳实践。IEEE，2003年，第958页。

[381]

正如您将从我们的多层感知器实现中回忆的那样，我们将图像展开为特征向量，并且这些输入完全连接到隐藏层，空间信息在此网络架构中没有编码。在CNN中，我们使用接收字段将输入层连接到功能图。这些接收字段可以理解为重叠窗口，我们在输入图像的像素上滑动以创建一个功能图。窗口滑动的跨距长度以及窗口大小是我们需要定义先验的模型的附加超参数。创建特征图的过程也称为卷积。这种卷积层的一个例子，即将输入像素连接到特征图中每个单元的层，如下图所示：

需要注意的是，特征检测器是重复的，这意味着将特征映射到下一层中的单位的接收字段共享相同的权重。这里，关键的想法是，如果一个特征检测器在图像的一部分有用，那么它在另一部分也可能有用。这种方法的好的副作用是它大大减少了需要学习的参数的数量。由于我们允许图像的不同补丁以不同的方式表示，CNN尤其擅长识别图像中不同大小和不同位置的对象。我们不需要像在mnist中那样担心图像的重新缩放和居中。

在CNN中，卷积层后面跟着一个池层（有时也称为子采样）。在池中，我们总结了相邻的特征检测器，以减少下一层的特征数量。池可以理解为一种简单的特征提取方法，我们取一个相邻特征块的平均值或最大值，然后将其传递到下一层。为了创建一个深度卷积神经网络，我们在将卷积层和汇集层连接到多层感知器进行分类之前，将多层交替堆叠在一起。如下图所示：

[382]

循环神经网络

递归神经网络（RNN）可以看作是一种具有反馈回路或随时间反向传播的前馈神经网络。在RNN中，神经元在（暂时）失活之前只在有限的时间内放电。反过来，这些神经元激活其他在稍后时间点开火的神经元。基本上，我们可以将递归神经网络视为具有附加时间变量的MLP。时间组件和动态结构不仅允许网络使用当前输入，还允许网络使用之前遇到的输入。

[383]

尽管RNN在语音识别、语言翻译和连接手写识别方面取得了显著的效果，但这些网络结构通常难以训练。这是因为我们不能简单地逐层反向传播误差；我们必须考虑额外的时间分量，它放大了消失和爆炸梯度问题。1997年，Juergen Schmidhuber及其同事推出了所谓的长期短期记忆单元来解决这个问题：长期短期记忆（LSTM）单元；S.Hochreiter和J.Schmidhuber。长期短期记忆。神经计算，9（8）：1735-1780，1997。

但是，我们应该注意到RNN有许多不同的变体，详细的讨论超出了本书的范围。

关于神经网络实现的最后几句话

您可能想知道为什么我们只是为了实现一个简单的多层人工网络，它可以对手写数字进行分类，而不是使用开源的Python机器学习库。一个原因是，在撰写本书时，Scikit Learn没有MLP实现。更重要的是，我们（机器学习实践者）应该至少对我们正在使用的算法有一个基本的了解，以便正确和成功地应用机器学习技术。

既然我们知道了前馈神经网络是如何工作的，那么我们就可以探索建立在numpy之上的更复杂的python库了，比如theano（http://deeplearning.net/software/theano/），这使我们能够更有效地构建神经网络。我们将在第13章中看到这一点，将神经网络训练与theano并行。在过去的几年里，Theano在机器学习研究人员中得到了广泛的欢迎，他们使用它来构建深度神经网络，因为它能够利用图形处理单元（GPU）优化用于多维数组计算的数学表达式。

可以在中找到大量Theano教程

还有许多有趣的库正在积极开发，用于培训Theano中的神经网络，您应该随时关注这些库：

学习2（

千层面（）

角膜（）

【384】

总结

在本章中，您已经了解了多层人工神经网络背后最重要的概念，这是目前机器学习研究中最热门的话题。在第二章，分类训练机器学习算法中，我们从简单的单层神经网络结构开始了我们的旅程，现在我们已经将多个神经元连接到强大的神经网络结构，以解决手写数字识别等复杂问题。我们对目前流行的反向传播算法进行了解释，它是许多用于深度学习的神经网络模型的组成部分之一。在学习了反向传播算法之后，我们能够更新这样一个复杂神经网络的权重。我们还添加了一些有用的修改，例如小批量学习和自适应学习速率，使我们能够更有效地训练神经网络。

【385】

并行神经网络

与Theano一起训练

在前一章中，我们讨论了许多数学概念，以了解前馈人工神经网络和多层感知器是如何工作的。首先，了解机器学习算法的数学基础非常重要，因为它可以帮助我们最有效和正确地使用这些强大的算法。在前面的章节中，您花费了大量时间学习机器学习的最佳实践，甚至自己从头开始练习实现算法。在这一章中，你可以稍微向后靠一点，好好休息，我想让你享受这段激动人心的旅程，通过一个最强大的库，机器学习研究人员使用它来实验深层神经网络，并非常有效地训练它们。大多数现代机器学习研究使用具有强大图形处理单元（GPU）的计算机。如果你对深入学习感兴趣，这是目前机器学习研究中最热门的话题，这一章绝对适合你。但是，如果您没有访问GPU的权限，请不要担心；在本章中，使用GPU是可选的，而不是必需的。

在我们开始之前，让我简单概述一下我们将在本章中介绍的主题：

用theano编写优化的机器学习代码

人工神经网络激活函数的选择

使用KRAS深度学习库进行快速简便实验

【387】

使用theano构建、编译和运行表达式

在本节中，我们将探讨功能强大的Theano工具，它被设计为最有效地使用Python培训机器学习模型。Theano的开发始于2008年，由Yoshua Bengio领导的Lisa实验室（Laboratoire d'Informatique des System\_mes adaptifs（）的缩写）。

在我们讨论Theano是什么以及它可以为我们加速机器学习任务做些什么之前，让我们先讨论一下当我们在硬件上运行昂贵的计算时所面临的一些挑战。幸运的是，多年来，计算机处理器的性能一直在不断提高，这使我们能够训练更强大、更复杂的学习系统，以提高机器学习模型的预测性能。即使是现在最便宜的台式计算机硬件也有多核处理器。在前面的章节中，我们看到SciKit学习中的许多函数允许我们将计算分散到多个处理单元上。但是，默认情况下，由于全局解释器锁（gil），Python仅限于在一个核心上执行。然而，尽管我们利用它的多处理库将计算分布在多个核心上，但我们必须考虑即使是高级桌面硬件也很少有超过8或16个这样的核心。

如果回顾上一章，我们实现了一个非常简单的多层感知器，其中只有一个由50个单元组成的隐藏层，那么我们已经需要优化大约1000个权重来学习一个非常简单的图像分类任务的模型。mnist中的图像相当小（28 x 28像素），如果我们想添加额外的隐藏层或使用具有更高像素密度的图像，我们只能想象参数数量的爆炸。对于单个处理单元来说，这样的任务很快就会变得不可行。现在的问题是，我们如何才能更有效地解决这些问题？这个问题的明显解决方案是使用GPU。GPU是真正的权力之马。您可以将图形卡视为机器内的小型计算机集群。另一个优势是，与最先进的CPU相比，现代GPU相对便宜，如下面的概述所示：

【388】

可在以下网站上找到此信息的来源：

（日期：2015年8月20日）

以现代CPU的70%的价格，我们可以得到一个拥有450倍多核的GPU，并且每秒能够进行大约15倍的浮点计算。那么，是什么阻碍了我们利用GPU完成机器学习任务呢？挑战在于，向目标GPU写入代码并不像执行代码那么简单

我们的解释器中的python代码。有一些特别的套餐，如CUDA和

允许我们瞄准GPU的opencl。然而，在CUDA或OpenCL中编写代码可能不是实现和运行机器学习算法最方便的环境。好消息是，这就是Theano开发的目的！

【389】

西亚诺是什么？

Theano究竟是什么——一种编程语言、编译器或Python库？事实证明，它符合所有这些描述。Theano的开发是为了非常有效地实现、编译和评估数学表达式，特别关注多维数组（张量）。它附带了一个在CPU上运行代码的选项。然而，它的真正威力来自于利用GPU来利用大内存带宽和浮点数学的强大功能。使用theano，我们也可以轻松地在共享内存上并行运行代码。2010年，当代码在CPU上运行时，Theano的开发人员报告的性能比Numpy快1.8倍，如果Theano以GPU为目标，它甚至比Numpy快11倍（J.Bergstra、O.Breuleux、F.Bastien、P.Lamblin、R.Pascanu、G.Desjardins、J.Turian、D.Warde Farley和Y.Bengio.theano:python中的CPU和GPU数学编译器。进行中。科学会议上的第9条巨蟒，第1-7页，2010年）。现在，请记住，这个基准是从2010年开始的，多年来，Theano有了显著的改进，现代图形卡的功能也有了很大提高。

那么，Theano和numpy有什么关系？theano是在numpy之上构建的，它有一个非常相似的语法，这使得已经熟悉numpy的人使用起来非常方便。公平地说，Theano不仅像许多人描述的那样“对类固醇麻木”，而且它还与sympy（用于符号计算（或符号代数）的python包）有一些相似之处。正如我们在前几章中看到的，在numpy中，我们描述了变量是什么，以及如何组合它们；然后，代码逐行执行。然而，在Theano中，我们首先写下这个问题以及我们想要如何分析它的描述。然后，如果我们想在GPU上运行，TeaNo使用C/C++或CUDA/OpenCL对代码进行优化和编译。为了为我们生成优化的代码，Theano需要知道问题的范围；将其视为操作树（或符号表达式图）。请注意，Theano仍在积极开发中，添加了许多新功能，并定期进行改进。在这一章中，我们将探讨该软件背后的基本概念，并学习如何将其用于机器学习任务。由于Theano是一个拥有许多高级功能的大型图书馆，因此不可能在本书中涵盖所有这些功能。但是，我将提供指向优秀在线文档的有用链接（如果您想了解有关此库的更多信息的话）。

【390】

有theano的第一步

在本节中，我们将对Theano采取第一步。根据系统的设置方式，通常只需使用pip安装程序，并通过命令行终端执行以下命令，从pypi安装theano：

PIP安装Theano

如果您在安装过程中遇到问题，我建议您阅读有关系统和平台特定建议的更多信息，请注意，本章中的所有代码都可以在您的CPU上运行；使用GPU是完全可选的，但如果您完全使用了享受天野之福。如果您有支持CUDA或OpenCL的图形卡，请参阅中的最新教程以进行适当设置。

在其核心，theano是围绕所谓的张量来计算符号数学表达式的。张量可以理解为标量、向量、矩阵等的泛化。更具体地说，标量可以定义为秩0张量，向量可以定义为秩1张量，矩阵可以定义为秩2张量，矩阵可以定义为秩3张量。作为一个热身练习，我们将从无张量模块中的简单标量开始，计算一维数据集中一个样本点x的净输入z，其权重为w1，偏差为w0:z=x1×w1+w0。

代码如下：

>>>导入theano

>>>来自无导入张量as t

#初始化

>>>x1=t.scalar（）。

>>>W1=T.scalar（）。

>>>W0=T.scalar（）。

>>>Z1=W1\*X1+W 0

#编译

>>>net\_input=theano.函数（inputs=[w1，x1，w0]，

……输出=Z1）

#执行

>>>打印（‘净输入：%.2f’%净输入（2.0，1.0，0.5））

净投入：2.50

[391]

这很简单，对吧？如果我们用theano编写代码，我们只需要遵循三个简单的步骤：定义符号（变量对象），编译代码，然后执行它。在初始化步骤中，我们定义了三个符号x1、w1和w0来计算z1。然后，我们编译了一个函数net\_input来计算net input z1。

但是，如果编写no代码，有一个特别的细节需要特别注意：变量的类型（dtype）。把它看作是一种福祉或负担，但在Theano中，我们需要选择是使用64位还是32位整数还是浮点，这将极大地影响代码的性能。让我们在下一节中更详细地讨论这些变量类型。

配置电话号码

现在，不管我们是运行Mac OS X、Linux还是Microsoft Windows，我们主要使用64位内存地址的软件和应用程序。然而，如果我们想要加速GPU上数学表达式的计算，我们仍然经常依赖旧的32位内存地址。目前，这是Theano中唯一支持的计算体系结构。在本节中，我们将看到如何适当地配置theano。如果您对Theano配置的更多详细信息感兴趣，请参阅上的联机文档。

当我们实现机器学习算法时，我们主要使用浮点数。默认情况下，numpy和theano都使用双精度浮点格式（float64）。但是，当我们开发用于在CPU上进行原型化和在GPU上执行的no代码时，来回切换float64（CPU）和float32（GPU）将非常有用。例如，要访问ano的float变量的默认设置，我们可以在python解释器中执行以下代码：

>>>打印（theano.config.floatx）float64

如果在安装该否之后未修改任何设置，则浮点默认值应为float64。但是，在当前的python会话中，我们只需通过以下代码将其更改为float32：

>>>theano.config.floatx='float32'

请注意，尽管当前的GPU使用率在ANO中需要float32类型，但我们可以在CPU上同时使用float64和float32。因此，如果要全局更改默认设置，可以通过命令行（bash）终端更改theano\_flags变量中的设置：export theano\_flags=floatx=float32

[392]

或者，您也可以通过如下方式运行特定的python脚本，将这些设置仅应用于该脚本：

theano\_flags=floatx=float32 python your\_script.py

到目前为止，我们讨论了如何设置默认的浮点类型，以便在使用theano的GPU上获得最佳的buck。接下来，让我们讨论在CPU和GPU执行之间切换的选项。如果我们执行以下代码，我们可以检查是否使用CPU或GPU：

>>>打印（theano.config.device）CPU

我个人的建议是使用CPU作为默认值，这使得原型设计和代码调试更加容易。例如，您可以通过执行脚本（如从命令行终端）在CPU上运行no代码：

theano\_flags=device=cpu，floatx=float64 python your\_script.py

但是，一旦我们实现了代码并希望使用我们的GPU硬件最有效地运行它，我们就可以通过以下代码运行它，而不需要对原始代码进行额外的修改：

theano\_flags=device=gpu，floatx=float32 python your\_script.py

在主目录中创建一个.theanoc文件也很方便，可以使这些配置永久化。例如，要始终使用float32和gpu，可以创建一个包含这些设置的.theanoc文件。命令如下：

echo-e“\n[全局]\nfloatx=float32\ndevice=gpu\n”>>~/.theanoc

如果您不在MacOS X或Linux终端上操作，可以使用您最喜欢的文本编辑器手动创建.theanoc文件，并添加以下内容：

[全局]floatX=float32设备=gpu

既然我们知道了如何针对可用的硬件适当地配置theano，那么我们可以在下一节中讨论如何使用更复杂的数组结构。

[393]

使用数组结构

在这一节中，我们将讨论如何使用它的张量模块在theano中使用数组结构。通过执行以下代码，我们将创建一个简单的2 x 3矩阵，并使用Theano优化的张量表达式计算列和：

>>>导入numpy为np

#初始化

>>>x=t.fmatrix（name='x'）

>>>x\_sum=t.sum（x，axis=0）

#编译

>>>calc\_sum=theano.函数（inputs=[x]，outputs=x\_sum）

#执行（python列表）

>>>ARY=[[1，2，3]，[1，2，3]]

>>>print（'列和：'，计算和（ary））

列总和：[2。第四章。6.]

#执行（numpy数组）

>>>ary=np.array（[[1，2，3]，[1，2，3]]，

……dtype=theano.config.floatx）

>>>print（'列和：'，计算和（ary））

列总和：[2。第四章。6.]

正如我们前面看到的，在使用theano时，我们只需要遵循三个基本步骤：定义变量、编译代码和执行它。前面的示例显示，theano可以同时使用python和numpy类型：list和numpy.ndarray。

注意，我们在创建fmatrix tensorvariable时使用了可选的name参数（这里是x），这有助于调试代码或打印theano图。例如，如果我们不给fmatrix符号x命名就打印它，那么print函数将返回其tensortype:

>>>打印（X）

<tensortype（float32，matrix）>

但是，如果像前面的示例一样，用名称参数x初始化了tensorvariable，则print函数将返回它：

>>>打印（X）X

可以通过type方法访问tensortype：

>>>打印（x.type（））

<tensortype（float32，matrix）>

[394]

Theano还拥有一个非常智能的内存管理系统，可以重新利用内存使其快速运行。更具体地说，它将内存空间分布在多个设备、CPU和GPU上；要跟踪内存空间中的变化，它会给各个缓冲区起别名。接下来，我们将看看共享变量，它允许我们分布大型对象（数组），并授予多个函数读写访问权，这样我们还可以在编译后对这些对象执行更新。关于theano中内存处理的详细描述超出了本书的范围。因此，我鼓励您在以下网站上继续了解关于ano和memory management的最新信息：

#初始化

>>>X=T.Fmatrix（'X'）

>>>W=theano.shared（np.asarray（[[0.0，0.0，0.0]]，dtype=theano.config.floatx））。

>>>Z=X.DOT（宽T）

>>>更新=[[w，w+1.0]]

#编译

>>>net\_input=theano.函数（inputs=[x]，

……updates=更新，

……输出=z）

#执行

>>>数据=np.数组（[[1，2，3]]，

……dtype=theano.config.floatx）>>>对于范围（5）中的i：

……print（'z%d:%i，net\_input（data））z0:[0.]z1:[6.]z2:[12.]z3:[18.]z4:[24.]

如您所见，通过theano共享内存非常容易：在前面的示例中，我们定义了一个更新变量，在其中声明希望在for循环中的每个迭代之后，通过值1.0更新数组w。在定义了要更新的对象以及如何更新之后，我们将这些信息传递给no.function编译器的update参数。

[395]

Theano的另一个巧妙的技巧是在编译之前使用givens变量将值插入到图中。使用这种方法，我们可以减少从CPU上的RAM到GPU的传输数量，以加速使用共享变量的学习算法。如果我们在no.函数中使用inputs参数，那么数据会多次从CPU传输到GPU，例如，如果我们在渐变下降期间多次迭代数据集（epoch）。使用givens，我们可以将数据集保存在GPU上，前提是它适合它的内存（例如，如果我们使用小批量学习）。代码如下：

#初始化

>>>数据=np.数组（[[1，2，3]]，

……dtype=theano.config.floatx）

>>>X=T.Fmatrix（'X'）

>>>W=theano.shared（np.asarray（[[0.0，0.0，0.0]]，

……dtype=theano.config.floatx）类型）

>>>Z=X.DOT（宽T）

>>>更新=[[w，w+1.0]]

#编译

>>>net\_input=theano.函数（inputs=[]，

……updates=更新，

……givens=x:数据，

……输出=z）

#对范围（5）中的i执行>>>：

……print（'z'，net\_input（））z0:[0.]z1:[6.]z2:[12.]z3:[18.]z4:[24.]

在前面的代码示例中，我们还看到givens属性是一个python字典，它将变量名映射到实际的python对象。这里，我们在定义fmatrix时设置了这个名称。

[396]

总结事情–线性回归示例

现在我们已经熟悉了theano，让我们来看一个实际的例子，并实现普通最小二乘（ols）回归。有关回归分析的快速更新，请参阅第10章，用回归分析预测连续目标变量。

让我们先用五个训练样本创建一个小的一维玩具数据集：

>>>x\_train=np.asarray（[[0.0]，[1.0]，

……[2.0]，[3.0]，

……[4.0]，[5.0]，

……[6.0]，[7.0]，

……[8.0]，[9.0]，

……dtype=theano.config.floatx）

>>>Y\_train=NP.Asarray（[1.0，1.3，

……3.1，2.0，

……5.0，6.3，

……6.6，7.4，

……8.0，9.0]，

……dtype=theano.config.floatx）

请注意，在构造numpy数组时，我们使用的是no.config.floatx，因此如果需要，我们可以选择在CPU和GPU之间来回切换。

接下来，让我们使用平方误差和成本函数实现一个训练函数来学习线性回归模型的权重。注意，w0是偏压单位（y轴在x=0处的截距）。代码如下：

导入Theano

从输入张量为t的输入张量为np def train\_linreg（x\_train，y\_train，eta，epoch）：

成本=[]

#初始化数组eta0=t.fscalar（'eta0'）y=t.fvector（name='y'）x=t.fmatrix（name='x'）

【397】

W=共享的电话号码（np.零（

shape=（x\_train.shape[1]+1），dtype=theano.config.floatx，name='w'）

#计算成本

净投入=t.dot（x，w[1:）+w[0]误差=y-净投入成本=t.sum（t.pow（误差，2））

#执行渐变更新渐变=t.grad（cost，wrt=w）update=[（w，w-eta0\*渐变）]

#编译模型

train=theano.函数（inputs=[eta0]，outputs=cost，updates=update，givens=x:x\_train，y:y\_train，）用于范围内（epoch）：

附加费用（列车（ETA）

退货成本，w

Theano中一个非常好的特性是我们在前面的代码示例中使用的Grad函数。Grad函数会自动计算表达式对其参数的导数，我们将其作为wrt参数传递给函数。

在我们实现了训练函数之后，让我们训练我们的线性回归模型，并查看平方误差和（SSE）成本函数的值，以检查它是否收敛：

>>>导入matplotlib.pyplot为plt

>>>成本，w=列车linreg（x列车，y列车，eta=0.001，epochs=10）

>>>plt.plot（范围（1，长度（成本）+1），成本）

>>>PLT.Tight\_布局（）

>>>plt.xlabel（'epoch'）

>>>plt.ylabel（成本）

>>>plt.show（）。

[398]

如下图所示，学习算法在第五个时期后已经收敛：

到目前为止还不错；通过查看成本函数，我们似乎从这个特定的数据集构建了一个有效的回归模型。现在，让我们编译一个新函数，根据输入特性进行预测：

def predict\_linreg（x，w）：x t=t.matrix（name='x'）net\_input=t.dot（xt，w[1:）+w[0]predict=theano.function（inputs=[xt]，givens=w:w，outputs=net\_input）返回predict（x）

按照theano的三个步骤（define、compile和execute）实现预测函数非常简单。接下来，让我们根据训练数据绘制线性回归拟合：

>>>请分散（X\_火车，

……你的火车，

……marker='s'，

……S=50）

>>>plt.plot（范围（x\_train.shape[0]），

【399】

……预测线性（x列，w）

……color='gray'，

……marker='o'，

……markersize=4，

……线条宽度=3）

>>>plt.xlabel（'x'）

>>>请选择依拉贝尔（‘Y’）

>>>plt.show（）。

正如我们在结果图中看到的，我们的模型适合数据点：

实现一个简单的回归模型是熟悉TheanoAPI的一个很好的练习。然而，我们的最终目标是发挥theano的优势，即实现强大的人工神经网络。我们现在应该配备所有的工具，我们将需要实现多层感知器从第12章，训练人工神经网络图像识别，在Theano。不过，这会很无聊，对吧？因此，我们将看看我最喜欢的一个在Theano基础上构建的深度学习库，以尽可能方便地使用神经网络进行实验。然而，在介绍Keras库之前，我们先在下一节中讨论神经网络中激活函数的不同选择。

[400]

前馈神经网络激活函数的选择

为了简单起见，我们到目前为止只讨论了多层前馈神经网络背景下的乙状结肠激活函数；我们在第12章“训练用于图像重编码的人工神经网络”的多层感知器实现中使用了隐藏层和输出层。版本。尽管我们将这种激活功能称为乙状结肠功能，正如文献中通常所说的那样，更精确的定义是逻辑函数或负对数似然函数。在下面的小节中，您将了解更多关于可选的乙状结肠功能的信息，这些功能对于实现多层神经网络很有用。

从技术上讲，我们可以在多层神经网络中使用任何函数作为激活函数，只要它是可微的。我们甚至可以使用线性激活函数，如adaline（第2章，分类的训练机器学习算法）。然而，在实践中，对隐藏层和输出层使用线性激活函数并不是很有用，因为我们希望在典型的人工神经网络中引入非线性，以便能够处理复杂的问题任务。线性函数之和最终会产生一个线性函数。

我们在上一章中使用的逻辑激活功能可能最接近地模仿了大脑中一个神经元的概念：我们可以把它看作是一个神经元是否开火的概率。然而，如果我们有高度负输入，逻辑激活功能可能会有问题，因为在这种情况下，乙状结肠功能的输出将接近于零。如果sigmoid函数返回接近于零的输出，神经网络学习将非常缓慢，并且在训练期间更可能陷入局部极小。这就是为什么人们通常喜欢使用双曲正切作为隐藏层中的激活函数。在我们讨论双曲正切是什么样子之前，让我们简单地回顾一下逻辑函数的一些基础知识，并看一个泛化，它对于多类分类任务更有用。

【401】

后勤功能概述

正如我们在本节的引言中提到的，逻辑函数，通常称为乙状结肠函数，实际上是乙状结肠函数的一个特例。我们从第3章的逻辑回归部分回顾了使用SciKit Learn的机器学习分类器，我们可以使用逻辑函数来模拟二进制分类任务中样本x属于正类（1类）的概率：

φlogistic（z）=1+1e−z

这里，标量变量z定义为净输入：

m z=w0x0++wmxm=∑xjwj=wt x

J=0

注意，w0是偏压单位（y轴截距，x0=1）。为了提供一个更具体的例子，让我们假设一个二维数据点x的模型和一个分配给向量w的权重系数如下的模型：

>>>X=np.array（[[1，1.4，1.5]]）

>>>W=np.数组（[0.0，0.2，0.4]）

>>>def net\_输入（x，w）：

……Z=X.dot（宽）

……返回Z

>>>定义逻辑（Z）：

……返回1.0/（1.0+np.exp（-z））

>>>DEF物流激活（X，W）：

……Z=净输入（x，w）

……退货物流（Z）

>>>打印（'P（Y=1\_X）=%.3f'

……%逻辑激活（x，w）[0]）

P（Y=1\_X）=0.707

【402】

如果我们计算净输入并用它激活具有这些特定特征值和权重系数的逻辑神经元，我们得到一个0.707的值，我们可以解释为这个特定样本x属于正类的概率为70.7%。在第12章，训练人工神经网络进行图像识别，我们使用一个热编码技术来计算由多个逻辑激活单元组成的输出层的值。但是，正如我们将通过以下代码示例演示的那样，由多个逻辑激活单元组成的输出层不会产生有意义的、可解释的概率值：

#W：数组，shape=[n\_output\_units，n\_hidden\_units+1]隐藏层的权重矩阵->输出层。

#注意，第一列（a[：][0]=1）是偏差单位。

>>>W=np.array（[[1.1，1.2，1.3，0.5]，

……[0.1，0.2，0.4，0.1]，

……[0.2、0.5、2.1、1.9]）

#A:数组，shape=[n\_hidden+1，n\_samples]激活隐藏层。

#注意，第一个元素（a[0][0]=1）是偏差单位。

>>>A=np.array（[[1.0]，

……[0.1]，

……[0.3]，

……[0.7]]）

#z:数组，shape=[n\_output\_units，n\_samples]输出层的净输入。

>>>Z=W.DOT（A）

>>>Y ou Probas=后勤（Z）

>>>print（'概率：\n'，y概率）概率：

[0.87653295]

[0.57688526]

[0.90114393]]

正如我们在输出中看到的，特定样本属于第一类的概率几乎是88%，特定样本属于第二类的概率几乎是58%，特定样本属于第三类的概率是90%。分别是NT。这显然令人困惑，因为我们都知道一个百分比应该直观地表示为100的分数。然而，如果我们只使用我们的模型来预测类标签，而不是类成员的概率，这实际上不是一个大问题。

>>>Y\_class=np.argmax（Z，axis=0）

>>>打印（‘预测类标签：%d’%y\_类[0]）预测类标签：2

[第403页]

然而，在某些情况下，为多类预测返回有意义的类概率是有用的。在下一节中，我们将介绍一个对逻辑函数SoftMax函数的概括，它可以帮助我们完成这项任务。

用SoftMax函数估计多类分类的概率

SoftMax函数是逻辑函数的一个推广，它允许我们在多类设置中计算有意义的类概率（多项式逻辑回归）。在SoftMax中，净输入z属于第i类的特定样本的概率可以用分母中的一个标准化项计算，分母是所有m个线性函数的和：

p（y=i\_z）=φSoftmax（z）=eiz em z

要查看SoftMax的运行情况，让我们用python对其进行编码：

>>>def softmax（z）：

……返回np.exp（z）/np.sum（np.exp（z））

>>>def softmax\_激活（x，w）：

……Z=净输入（x，w）

……返回乙状结肠（Z）

>>>Y\_Probas=软最大值（Z）

>>>print（'概率：\n'，y概率）概率：

[0.40386493]

[0.07756222]

[0.51857284]]

>>>Y圮Probas.sum（）。

1.0分

【404】

正如我们所看到的，正如我们所期望的，预测的类概率现在加起来是一个。值得注意的是，第二类的概率接近于零，因为z1和max（z）之间有很大的差距。但是，请注意，预测的类标签与logistic函数中的类标签相同。直观地说，它可能有助于将SoftMax函数视为一个标准化的逻辑函数，它有助于在多类设置中获得有意义的类成员预测。

>>>Y\_class=np.argmax（Z，axis=0）>>>print（'预测类标签：

……%d“%y\_class[0]）预测类标签：2

用双曲正切扩大输出谱

另一个常用于人工神经网络隐藏层的乙状结肠函数是双曲正切（tanh），它可以解释为逻辑函数的重新缩放版本。

EZ−E−Z型

φtanh（z）=2×φlogistic（2×z）−1=ez+e−z

φlogistic（z）=1+1e−z

逻辑（2×Z）×2−1

【405】

双曲正切比逻辑函数的优点在于它具有更宽的输出光谱和范围的开放区间（-1，1），这可以提高反向传播算法的收敛性（C.M.Bishop）。用于模式识别的神经网络。牛津大学出版社，1995年，第500-501页）。相反，logistic函数返回一个输出信号，该信号范围为打开间隔（0，1）。为了直观地比较逻辑函数和双曲正切，让我们在一维空间中绘制两个乙状结肠函数：

>>>导入matplotlib.pyplot为plt

>>>定义TANH（Z）：

……e\_p=np.exp（z）

……e\_m=np.exp（-z）

……返回（e\_p-e\_m）/（e\_p+e\_m）

>>>Z=np.arange（-5，5，0.005）

>>>LOG U ACT=后勤（Z）

>>>Tanh\_act=Tanh（Z）

>>>plt.ylim（[-1.5，1.5]）

>>>plt.xlabel（'净输入$z$'）

>>>plt.ylabel（'激活$\phi（z）$'）

>>>plt.axhline（1，color='black'，linestyle=--'）

>>>plt.axhline（0.5，color='black'，linestyle=--'）

>>>plt.axhline（0，color='black'，linestyle=--'）

>>>plt.axhline（-1，color='black'，linestyle=--'）

>>>Plt.Plot（Z，Tanh\_Act，

……线条宽度=2，

……color='black'，

……label='tanh'）

>>>Plt.Plot（Z，Log\_Act，

……线条宽度=2，

……color='lightgreen'，

……label='logistic'）

>>>Plt.Legend（loc='右下'）

>>>PLT.Tight\_布局（）

>>>plt.show（）。

[406]

如我们所见，两条乙状曲线的形状非常相似；但是，tanh函数的输出空间比logistic函数大2倍：

注意，为了便于说明，我们口头实现了logistic和tanh函数。在实践中，我们可以使用numpy的tanh函数来获得相同的结果：

>>>tanh\_act=np.tanh（z）

此外，在Scipy的特殊模块中还提供后勤功能：

>>>来自scipy.special import expit

>>>log\_act=expit（z）

[第407页]

既然我们对人工神经网络中常用的不同激活函数有了更多的了解，那么让我们以本书中遇到的不同激活函数的概述来结束本节。

利用角膜有效训练神经网络

在本节中，我们将介绍Keras，一个最近开发的促进神经网络培训的库。Keras的开发始于2015年的头几个月；到今天为止，它已经发展成为一个最受欢迎和使用最广泛的图书馆，建立在Theano之上，并允许我们利用我们的GPU加速神经网络培训。它的一个显著特点是它是一个非常直观的API，它允许我们只在几行代码中实现神经网络。一旦安装了theano，就可以通过从终端命令行执行以下命令从pypi安装keras:pip install keras

【408】

有关Keras的更多信息，请访问官方网站

为了了解通过Keras进行的神经网络培训是什么样子的，让我们实现一个多层感知器，从mnist数据集中对手写数字进行分类，我们在上一章中介绍了这一点。mnist数据集可以从以下四个部分下载：

train-images-idx3-ubyte.gz：这些是训练集图像（9912422字节）

train-labels-idx1-ubyte.gz：这些是训练集标签（28881字节）

t10k-images-idx3-ubyte.gz：这些是测试集图像（1648877字节）

t10k-labels-idx1-ubyte.gz：这些是测试集标签（4542字节）

在下载和解压档案后，我们将文件放在当前工作目录的目录mnist中，这样我们就可以使用以下功能加载培训和测试数据集：

import os import struct import numpy as np def load mnist（path，kind='train'）：“从'path`'加载mnist数据”labels\_path=os.path.join（path，

“%s-labels-idx1-ubyte”

%kind）images\_path=os.path.join（路径，

“%s-images-idx3-ubyte”

%种类）

以open（labels\_path，'rb'）作为lbpath:magic，n=struct.unpack（'>ii'，lbpath.read（8））labels=np.fromfile（lbpath，dtype=np.uint8）

以open（images\_path，'rb'）作为imgpath：

magic，num，rows，cols=struct.unpack（“>iii”，imgpath.read（16））images=np.fromfile（imgpath，

dtype=np.uint8）.remaze（len（labels），784）

[409]

返回图像、标签

x\_train，y\_train=载荷\_mnist（“mnist”，kind='train'）

print（'行数：%d，列数：%d'%（x\_train.shape[0]，x\_train.shape[1]））

行：60000，列：784

x\_测试，y\_测试=负载\_mnist（'mnist'，kind='t10k'）

print（'行：%d，列：%d'%（x\_test.shape[0]，x\_test.shape[1]））行：10000，列：784

在下面的页面中，我们将逐步介绍使用keras的代码示例，您可以直接从Python解释器执行这些代码。但是，如果您有兴趣在GPU上训练神经网络，您可以将其放入Python脚本，或者从packt发布网站下载相应的代码。要在GPU上运行python脚本，请从mnist\_keras\_mlp.py文件所在的目录中执行以下命令：

no\_flags=mode=fast\_run，device=gpu，floatx=float32 python mnist\_ukeras\_mlp.py

要继续准备培训数据，让我们将mnist图像数组转换为32位格式：

>>>导入theano

>>>theano.config.floatx='float32'

>>>x\_train=x\_train.astype（theano.config.floatx）

>>>x\_test=x\_test.astype（theano.config.floatx）

接下来，我们需要将类标签（整数0-9）转换为一种热格式。幸运的是，Keras提供了一个方便的工具：

>>>从keras.utils导入np-utils

>>>打印（“前3个标签：”，Y\_train[：3]）

前3个标签：[5 0 4]

>>>y\_train\_ohe=np\_utils.to\_categorial（y\_train）

>>>打印（“\n前3个标签（一个热）：\n”，y\_train\_ohe[：3]）前3个标签（一个热）：

[0.0.0.0.0.1.0.0.0.0。]

[1.0.0.0.0.0.0.0.0.0。]

[0.0.0.0.1.0.0.0.0.0。]]

【410】

现在，我们可以进入有趣的部分，实现一个神经网络。在这里，我们将使用与第12章“训练人工神经网络用于图像识别”相同的架构。但是，我们将用双曲正切激活函数替换隐藏层中的逻辑单元，用SoftMax替换输出层中的逻辑函数，并添加一个额外的隐藏层。正如您在下面的代码实现中看到的，KERA使这些任务非常简单：

>>>从keras.models导入序列

>>>来自keras.layers.core import close

>>>从keras.optimizers导入sgd

>>>NP.随机。种子（1）

>>>型号=顺序（）

>>>模型.添加（密集（input\_dim=x\_train.shape[1]，

……输出尺寸=50，

……init='uniform'，

……激活='tanh'））

>>>型号.添加（密集（输入尺寸=50，

……输出尺寸=50，

……init='uniform'，

……激活='tanh'））

>>>型号.添加（密集（输入尺寸=50，

……输出\_dim=y\_train\_ohe.shape[1]，

……init='uniform'，

……activation='softmax'））

>>>sgd=sgd（lr=0.001，衰减=1e-7，动量=0.9）

>>>model.compile（loss='categorial\_cross熵'，optimizer=sgd）

首先，我们使用序列类初始化一个新的模型来实现前馈神经网络。然后，我们可以添加任意多的层。但是，由于我们添加的第一层是输入层，所以我们必须确保input\_dim属性与训练集中的特性（列）数量（此处为768）匹配。此外，我们必须确保两个连续层的输出单元（output\_dim）和输入单元（input\_dim）的数量匹配。在前面的示例中，我们添加了两个隐藏层，每个隐藏层有50个隐藏单元加上1个偏移单元。请注意，在Keras的完全连接网络中，偏差单位初始化为0。这与第12章“为图像识别训练人工神经网络”中的MLP实现不同，我们将偏差单位初始化为1，这是一种更常见（不一定更好）的约定。

[411]

最后，输出层中的单元数应该等于唯一类标签数，即一个热编码类标签数组中的列数。在我们编译模型之前，我们还必须定义一个优化器。在前面的例子中，我们从前面的章节中选择了一个我们已经熟悉的随机梯度下降优化。此外，我们还可以设置权重衰减常数和动量学习的值，以调整每个时期的学习速率，如第12章“训练用于图像识别的人工神经网络”所述。最后，我们将成本（或损失）函数设置为分类交叉熵。二元交叉熵是逻辑回归中成本函数的技术术语，分类交叉熵是通过SoftMax对多类预测的推广。在编译模型之后，我们现在可以通过调用fit方法来训练它。这里，我们使用的是每批300个训练样本的小批量随机梯度。我们对MLP进行了50多个阶段的培训，在培训过程中，我们可以通过设置verbose=1来跟踪成本函数的优化。验证分割参数特别方便，因为它将保留10%的训练数据（这里是6000个样本），以便在每个时期后进行验证，以便我们可以在训练期间检查模型是否过度拟合。

>>>型号.fit（x\_train，

……你的火车

……nb\_epoch=50，

……批量大小=300，

……详细=1，

……验证分割=0.1，

……显示\准确度=真）

培训54000个样本，验证6000个样本

时代0

54000/54000[======================]-1s-损失：2.2290-acc:0.3592-val\_损失：2.1094-val\_acc:0.5342

时代1

54000/54000[======================]-1s-损耗：1.8850-acc:0.5279-val\_损耗：1.6098-val\_acc:0.5617

时代2

54000/54000[======================]-1s-损失：1.3903-acc:0.5884-val\_损失：1.1666-val\_acc:0.6707

时代3

54000/54000[======================]-1s-损耗：1.0592-acc:0.6936-val\_损耗：0.8961-val\_acc:0.7615

[…]

第49世纪

54000/54000[======================]-1s-损耗：0.1907-加速度：0.9432-Val\_损耗：0.1749-Val\_加速度：0.9482

[412]

在训练过程中，打印成本函数的值非常有用，因为我们可以快速发现训练过程中成本是否在下降，如果不这样做，可以提前停止算法来调整超参数值。

为了预测类标签，我们可以使用Predict\_Classes方法将类标签直接返回为整数：

>>>Y\_train\_pred=model.predict\_classes（x\_train，verbose=0）

>>>print（'前3个预测：'，y\_train\_pred[：3]）

>>>前3个预测：[5 0 4]

最后，让我们在培训和测试集上打印模型精度：

>>>列车加速度=NP.SUM（

……y\_train==y\_train\_pred，axis=0）/x\_train.shape[0]

>>>打印（‘训练精度：%.2f%%'%（train\_acc\*100））

培训精度：94.51%

>>>Y\_test\_pred=model.predict\_classes（x\_test，verbose=0）

>>>测试\_acc=np.sum（y\_test==y\_test\_pred，

……轴=0）/x\_test.shape[0]print（'测试精度：%.2f%%'%（测试\_acc\*100））测试精度：94.39%

注意，这只是一个非常简单的神经网络，没有优化的调节参数。如果你有兴趣玩更多的喀拉斯，请随时进一步调整学习率，动量，重量衰减，和隐藏单位的数量。

虽然keras是实现和试验神经网络的伟大库，但还有许多其他的theano包装器

值得一提的图书馆。一个突出的例子是

Pyrearn2（），已在蒙特利尔的Lisa实验室开发。此外，千层面（）可能是

如果您更喜欢一个更简单但可扩展的库，它提供了对底层theano代码的更多控制，那么您会感兴趣。

【413】

总结

我希望你喜欢机器学习的最后一章。在这本书中，我们涵盖了这个领域必须提供的所有基本主题，现在您应该有充分的准备将这些技术付诸行动，以解决现实世界中的问题。

我们从简要概述不同类型的学习任务开始了我们的旅程：监督学习、强化学习和无监督学习。我们讨论了几种可用于分类的不同学习算法，从第2章“分类的训练机器学习算法”中简单的单层神经网络开始。然后，我们在第3章“使用SciKit Learn的机器学习分类器之旅”中讨论了更高级的分类算法，您在第4章“构建良好的训练集-数据预处理”和第5章“压缩”通过降维来生成数据。记住，即使是最先进的算法也会受到训练数据中的信息的限制。在第6章，学习模型评估和超参数调整的最佳实践中，您学习了构建和评估预测模型的最佳实践，这是机器学习应用程序的另一个重要方面。如果一个单一的学习算法不能达到我们所期望的性能，那么创建一个专家组来进行预测有时会很有帮助。我们在第7章中讨论了这一点，结合不同的集成学习模型。在第8章，将机器学习应用于情感分析中，我们将机器学习应用于分析由互联网上的社交媒体平台主导的现代时代可能最有趣的数据形式：文本文档。然而，机器学习技术并不局限于离线数据分析，在第9章，将机器学习模型嵌入到Web应用程序中，我们看到了如何将机器学习模型嵌入到Web应用程序中，以便与外部世界共享。在很大程度上，我们的重点是分类算法，可能是机器学习最流行的应用。然而，这不是它的终点！在第10章，用回归分析预测连续目标变量中，我们探索了几种用于回归分析的预测连续值输出值的算法。机器学习的另一个令人兴奋的子领域是聚类分析，它可以帮助我们找到数据中隐藏的结构，即使我们的培训数据没有提供正确的答案来学习。我们在第11章中讨论了这一点，使用未标记的数据——聚类分析。

【414】

在本书的最后两章中，我们看到了整个机器学习领域中最漂亮和最令人兴奋的算法：人工神经网络。虽然深入学习真的超出了这本书的范围，但我希望我至少能激起你对这一领域最新进展的兴趣。如果你正在考虑做一个机器学习研究者的职业，或者即使你只是想跟上这个领域的最新进展，我也可以推荐你去跟随这个领域的顶尖专家的作品，比如Geoff Hinton（），Andrew Ng（Yann LeCun）（Juergen Schmidhub）。Er（，和Yoshua Bengio（

仅举几个例子。此外，请毫不犹豫地加入Scikit Learn、Theano和Keras邮件列表，参与有关这些库的有趣讨论，以及一般的机器学习。我很期待在那里见到你！如果您对本书有任何疑问或需要一些有关机器学习的一般提示，欢迎随时与我联系。

我希望这段经历机器学习不同方面的旅程真的是值得的，并且你学习了许多新的有用的技能来推动你的事业，并将它们应用到现实世界的问题解决中。

【415】

符号

5x2交叉验证188

7-zip网址234

A

前馈神经网络的准确度（ACC）191激活函数

logistic函数reap 402-404输出谱，用双曲正切405-407加宽

概率，多类估计

通过SoftMax功能404405分类

选择401

自适应提升

弱学习者，通过224-231自适应线性神经元（Adaline）利用33285个自适应线性神经元

大约33个成本函数，用梯度最小化

下降34-36

在python 36-42中实现大规模机器学习42-47随机梯度下降42-47

凝聚聚类

关于326申请，通过SciKit学习334

算法

调试，学习和验证

曲线179

索引

算法选择

具有嵌套交叉验证187-189

曲线下面积（AUC）193人工神经网络

物流成本函数，计算365-367神经网络，通过

反向传播368-371

培训365

人工神经元18平均连接327

B

反向传播368、369

直觉，发展372

袋装218-220字袋模型

定义236个文档，处理成标记24243个文本数据，清理240241个词汇表，创建236个单词相关性，通过术语进行评估

反文件频率238-240

文字，转化为特征

向量236237

基本术语8提升224引导聚合220边界点334

威斯康星州乳腺癌数据集加载170

C

层叠样式表（CSS）262分类数据类标签，编码105，106，处理104个热编码，在名义上执行

特征106、107

序数特征，映射104、105

分类算法选择49，50

分类误差82类概率，逻辑回归建模

大约56

逻辑回归直觉和条件概率56-59

逻辑回归模型，训练

Scikit学习62-65

过度安装，通过

正规化65-68

物流成本函数的权重59-61

簇惯性314簇

组织，作为层次树326327

完成连接326复杂函数，用人工神经网络建模

关于342多层神经网络

建筑345-347

神经网络，通过正向传播激活347-350

单层神经网络回顾343344

计算研究库（CORR）

URL 246混淆矩阵

读数190、191

收敛性，在神经网络中379380卷积382卷积层382卷积神经网络

（cnns或convnets）381382核心点334 csv（逗号分隔值）100维度诅咒96

D

数据集

分区、培训和

测试集108、109

数据存储

sqlite数据库，设置为255、256

数据库扫描

约334个缺点339个高密度区域，通过335-339定位

决策区域53决策树学习

大约80，81决策树，建立88，89信息获取，最大化82-86弱到强的学习者，结合VIA

随机森林90-92

决策树回归304，305决策树304决策树分类器80深学习341树图约326附，到热图332333

基于密度的有噪声应用的空间聚类。

见DBSCAN深度参数185降维118距离矩阵

分层聚类，

在328-331上表演

分层次聚类326文档分类逻辑回归模型，

244-246培训

虚拟特征107

E

弹性网法297肘法

大约312320个，用于寻找

集群320

整体分类器评估213-218调谐213-218

集合方法199分类器建筑集合，来自引导程序

样品219-224

整体学习199-202

熵82 epoch 344误差（err）191勘探数据分析（eda）280

F

假阳性率（fpr）192个特征检测器342，381个特征提取118个特征重要性评估，随机森林124-126

功能图382功能缩放

约110表示110，111

特征选择大约112，118稀疏解决方案，

有L1正则化112-117

适配SciKit学习估计量

序列化252-254

Flask Web应用程序定义258、259，开发257表单验证259-263呈现259-263

花卉数据集50正向传播

神经网络，通过347-350激活

模糊系数319模糊系数319模糊聚类317模糊C均值（FCM）算法317模糊K均值317

克

高斯核152

基尼指数82

全球口译员锁（GIL）388

谷歌开发者门户

URL 241渐变检查

大约373个神经网络，用373-379进行调试

梯度下降优化算法344

石墨烯

URL 89网格搜索

大约185个超参数，通过186个机器学习模型进行调整，

通过185进行微调

H

手写数字分类350

硬聚类

关于317与软聚类317-319

热图

约332张树木图，附332333

隐藏层345分层和基于密度的聚类312

层次聚类

大约326在距离矩阵328-331上执行

高密度区域

定位，通过DBSCAN 334-339

保持交叉验证173保持方法关于173缺点174

住房数据集关于279个特征280-284探索279、280个特征279 URL 279

HTML基础

url 259双曲正切（sigmoid）内核152双曲正切（tanh）405超参数

约173345调优，通过网格搜索186 i

IMDB电影评论数据集获取233-235

内置pickle模块url 252信息增益（ig）304实例学习93智能机器构建，将数据转换为知识2互联网电影数据库（imdb）234反向文档频率238

ipython笔记本URL 25

IRIS数据集8、9、50、210

鸢尾51

鸢尾51、210、弗吉尼亚鸢尾51、210

J

jinja2语法

URL 262作业库

网址253

K

keras大约使用408 url 409，用于训练神经网络408-413内核函数148-151内核主成分分析，在python 154中使用155，用于非线性映射148

核主成分分析，实例

同心圆，分离159-161个半月形状，分离155-158个新数据点，投影162-165

核心主成分分析，Scikit Learn 166

kernel svm 75 kernel trick 148-151 k-fold交叉验证关于173-178维持方法173用于评估模型性能173

K-表示

大约312个，用于按

相似性312-315 k-均值+315-317 k-最近邻分类器（knn）92 k-最近邻92 knn算法93-96

我

L1正则化稀疏解112-117

L2正则化66，112

兰开斯特干酪243

千层面

网址413

潜在Dirichlet分配249懒惰学习者92 lda，通过SciKit学习146、147学习曲线

大约179个偏差和方差问题，

用180-182诊断

学习率344最小绝对收缩与选择二元化243

liblinear网址74

伦敦银行同业拆借市场

网址74

线性回归模型性能，评估294-296转，变成曲线298-300

联动矩阵329

Lisa实验室参考388

logistic function 57 logistic regression 56，348 logistic regression model training，for document.文档

分类244-246

逻辑功能56

长期短期记忆（LSTM）384

米

机器学习历史18-24 python，用于13强化学习2监督学习2无监督学习2

机器学习模型微调，通过网格搜索185

宏观平均法197多数票90多数票原则200保证金69保证金分类替代实施，在SciKit学习74

最大裕度直觉70，71非线性可分离情况，

处理71、72

Matplotlib URL 25 McCulloch-Pitt神经元模型342平均插补102均方误差（mse）295中位绝对偏差（mad）292公制参数参考96

微平均法197缺失数据，处理约99100个特征，消除101个缺失值，输入102个样本，消除101个SciKit学习估计量API 102

mnist数据集关于351个多层感知器，

实施356-365

获取351-356套图像，测试351套图像，培训351套标签，测试351

设置标签，培训351

URL 351型号性能

评估，使用k倍交叉验证173

模型持久性252模型选择173电影分类器

转换为Web应用程序264-271

电影评论分类器更新274275

电影评论数据集

URL 234多层前馈神经网络345

多层感知器345多元线性回归279

杂音哈希3功能URL 247

N号

自然语言处理（NLP）233嵌套交叉验证

用于算法选择187-189

神经网络体系结构

关于381卷积神经网络

（CNN或ConvNets）381382

复发性神经

网络（RNNS）383、384神经网络实现384神经网络

收敛379380展开，梯度检查373-379

培训，Keras使用408-413

N克237

NLTK公司

URL 242噪声点334标称特征104非空类82非线性映射核主成分分析，用于148

非线性问题，用

内核支持向量机

大约7576个内核技巧，用于寻找分离

超平面77-80

非线性关系

处理，随机森林使用304建模，住房数据集300-303

非参数模型93正态方程290标准化110符号8，9

麻木

网址25

O

物体

按相似性分组，

K-表示使用312-315

优势比56抵消278一个热编码107一个热表示346一对所有（ova）28一对其余（ovr）28定义246-249的在线算法

意见挖掘233序数特征104正最小二乘线性回归模型约285

系数，通过Scikit Learn估算289、290

执行285回归，求解回归

梯度下降参数285-289

普通最小二乘回归397

核心外学习定义246-249

过度安装53、65、112

P

熊猫

URL 25参数模型93

皮尔逊积矩相关系数282

感知器50感知器学习算法

在python 24-27中实现

感知器模型

培训，关于IRIS数据集27-32

关于189个混淆矩阵的绩效评估指标，阅读190、191个指标，多类分类得分197、198

分类模型的精确性和召回，

优化191193

接收操作员特征（ROC）

图表，绘制193-197

花瓣长度51，210花瓣宽度51管道

变压器和估计器，结合171

工作流，使用169简化

多元表决200多项式核152多项式回归298-300池层382波特-斯坦默算法242精度（pre）192精度召回曲线194主成分分析（pca）282

[第422页]

主要成分分析，Scikit Learn 135-137

基于原型的集群312公共服务器Web应用程序，部署到27273

学习2

网址413

皮普林

网址234

蟒蛇

关于13核主成分分析，

在154155年实施

软件包，安装13-15参考14使用，用于机器学习13

pythonanywhere帐户URL 272

问

聚类质量

量化，通过轮廓图321-324

R

径向基函数（RBF）约152实现152153

随机森林回归304-308随机森林90

随机样本一致性（RANSAC）算法291

原始术语频率237召回（REC）192接收字段382

复发性神经

网络（RNNS）383，384回归线278正则表达式（regex）240正则化365正则化参数67，185正则化方法使用，用于回归297，298强化学习

残差图294残差278岭回归297路线图，用于机器学习系统

约10个模型，评估13个预测模型，选择12个预测模型，训练12个预处理11个未看到的数据实例，预测13个

稳健回归模型

配件，使用的RANSAC 291-293

曲线下ROC面积（ROC AUC）210

S

散射图矩阵280场景，距离值正确接近330错误接近329

Scikit学习

大约50个聚集性集群，通过334感知器应用，通过50-55参考链接训练167

Scikit Learn Estimator API 102、103 Scikit Learn在线文档

URL 55情绪分析233 Sepal宽度210

顺序向后选择（SBS）118顺序特征选择算法118-123

乙状结肠功能57乙状结肠（logistic）激活功能348轮廓分析321轮廓系数321轮廓图

大约312个集群的质量，

通过321-324进行量化

简单线性回归模型278279简单多数票分类器不同算法，结合多数票210-212

单连杆326雪球杆243软聚类

约317与硬聚类317-319

Soft K-Means 317 SoftMax函数404稀疏236光谱聚类算法339

sqlite数据库设置，用于数据存储255、256

平方欧氏距离314 S形（乙状）曲线58叠加218标准化110，169随机梯度下降246随机梯度下降（SGD）285停止字删除243强学习者90次抽样382

平方误差和（SSE）285、398、344监督数据压缩，通过线性判别分析

大约138-140线性判别法，选择新的

特征子空间143-145个样本，投射到新的

功能空间145

散射矩阵，计算140-142

监督学习

关于3个分类，用于预测类标签3，4个预测，用3个回归进行，用于预测连续

结果4，5

支持向量机（SVM）69、148、186、308

支持向量69

症状

关于390 URL 390

T

术语频率238术语频率反转文档频率（tf idf）238

西雅娜

关于390阵列结构，使用394-396配置392393线性回归示例397-400参考390使用391392

阈值函数344变压器102级变压器和估计器组合，管道171真正率（TPR）192

u

通过主成分分析，在未监督的情况下，将65个单克模型237降维。

关于128，129解释了方差130-133特征转换133-135总方差130-133

无监督学习6维约简，

对于数据压缩7、8

隐藏结构，6个子群发现，聚类7技术发现311

五

验证曲线

约179过拟合和欠拟合，寻址

有183、185

验证数据集121矢量化27

W

沃德联系327名弱势学生

约90224杠杆，通过自适应增压224-231

Web应用程序

部署到公共服务器27273开发中，使用flask 257实现，url 265电影分类器，转化为264-271电影评论分类器，更新274775功能109色调类221

网址108 word2vec

大约249

URL 249词干242工作流

流线型，带管道169

wtforms库URL 259

葡萄酒数据集关于108221级酒精221

【425】

谢谢你的购买

python机器学习

关于packt发布

packt，发音为“packed”，于2004年4月出版了其第一本书《掌握phpmyadmin以实现有效的mysql管理》，随后继续专门出版关于特定技术和解决方案的高度集中的书籍。

我们的书籍和出版物分享了您的IT专业人员在调整和定制当今系统、应用程序和框架方面的经验。我们基于解决方案的书籍为您提供了定制软件和技术以完成工作的知识和能力。与过去的IT书籍相比，packt书籍更具体，更不一般。我们独特的业务模式使我们能够为您提供更为集中的信息，让您了解更多需要了解的内容，减少不需要了解的内容。

Packt是一家现代化但独特的出版公司，专注于为开发者、管理者和新手等社区制作高质量、尖端的书籍。有关更多信息，请访问我们的网站。

关于packt开源

2010年，Packt推出了两个新品牌：Packt开源和Packt Enterprise，以继续专注于专业化。这本书是packt开源品牌的一部分，是围绕开源许可证构建的软件上出版的书籍的发源地，并向任何人提供信息，从高级开发人员到初露头角的网页设计师。开源品牌还运行着Packt的开源版税计划，通过该计划，Packt向每个开源项目提供版税，说明其软件书籍的销售情况。

为packt写作

我们欢迎对创作感兴趣的人的所有查询。书籍建议应发送至author@packtpub.com。如果你的书的想法还处于早期阶段，你想在写正式的书建议之前先讨论一下，那么请联系我们；我们的一个委托编辑会与你联系。

我们不仅仅是在寻找已出版的作者；如果你有很强的技术能力但没有写作经验，我们经验丰富的编辑可以帮助你发展写作事业，或者仅仅是为你的专业知识获得一些额外的奖励。

用python构建机器学习系统

ISBN:978-1-78439-277-2平装本：326页

通过使用python创建实用的机器学习系统，从数据中获得更多信息

构建自己的基于python的机器学习系统，以解决任何问题。

了解Python如何为创建机器学习系统提供多上下文解决方案。

使用关键的python机器学习库在项目中成功实现的实际场景。

用Scikit学习掌握机器学习

ISBN:978-1-78398-836-5平装本：238页

使用SciKit Learn将有效的学习算法应用于实际问题

为常见任务（包括回归、分类和聚类）设计和故障排除机器学习系统。

熟悉常用的机器学习算法，包括决策树、逻辑回归和支持向量机。

一个基于实例的实用指南，帮助您在使用Scikit Learn实现和评估机器学习系统方面获得专业知识。

请访问www.packtpub.com了解我们的标题信息。

文件分类。

用python构建机器学习系统

ISBN:978-1-78216-140-0平装本：290页

掌握用python进行机器学习的艺术，并用这个深入的实践指南构建有效的机器学习系统。

掌握使用大量Python库的机器学习，并开始构建自己的基于Python的ML系统。

在扩展应用程序的同时，了解模块化和代码组织的最佳实践。

包括分类、回归、特性工程，以及更多由实际例子指导的内容。

请访问www.packtpub.com了解我们的标题信息。