Table of Contents

概率图模型R语言学习

Familiarize yourself with probabilistic graphical models through real-world problems and illustrative code examples in R David Bellot BIRMINGHAM - MUMBAI

# 概率图模型R语言学习

Copyright © 2016 Packt Publishing All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews. Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book. Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information. First published: April 2016 Production reference: 1270416 Published by Packt Publishing Ltd. Livery Place 35 Livery Street Birmingham B3 2PB, UK. ISBN 978-1-78439-205-5 www.packtpub.com

# 致谢

Author David Bellot Reviewers Mzabalazo Z. Ngwenya Prabhanjan Tattar Acquisition Editor Divya Poojari Content Development Editor Trusha Shriyan Technical Editor Vivek Arora Copy Editor Stephen Copestake Project Coordinator Kinjal Bari Proofreader Sa s Editing Indexer Mariammal Chettiyar Graphics Abhinash Sahu Production Coordinator Nilesh Mohite Cover Work Nilesh Mohite

# 关于作者

David Bellot is a PhD graduate in computer science from INRIA, France, with a focus on Bayesian machine learning. He was a postdoctoral fellow at the University of California, Berkeley, and worked for companies such as Intel, Orange, and Barclays Bank. He currently works in the nancial industry, where he develops nancial market prediction algorithms using machine learning. He is also a contributor to open source projects such as the Boost C++ library.

# 关于审稿

Mzabalazo Z. Ngwenya holds a postgraduate degree in mathematical statistics from the University of Cape Town. He has worked extensively in the eld of statistical consulting and has considerable experience working with R. Areas of interest to him are primarily centered around statistical computing. Previously, he has been involved in reviewing the following Packt Publishing titles: Learning RStudio for R Statistical Computing, Mark P.J. van der Loo and Edwin de Jonge; R Statistical Application Development by Example Beginner‘s Guide, Prabhanjan Narayanachar Tattar; Machine Learning with R, Brett Lantz; R Graph Essentials, David Alexandra Lillis; R Object-oriented Programming, Kelly Black; Mastering Scienti c Computing with R, Paul Gerrard and Radia Johnson; and Mastering Data Analysis with R, Gergely Darócz.

Prabhanjan Tattar is currently working as a senior data scientist at Fractal Analytics, Inc. He has 8 years of experience as a statistical analyst. Survival analysis and statistical inference are his main areas of research/interest. He has published several research papers in peer-reviewed journals and authored two books on R: R Statistical Application Development by Example, Packt Publishing; and A Course in Statistics with R, Wiley. The R packages gpk, RSADBE, and ACSWR are also maintained by him.

# www.PacktPub.com

eBooks, discount offers, and more Did you know that Packt offers eBook versions of every book published, with PDF and ePub les available? You can upgrade to the eBook version at www.PacktPub. com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub.com](mailto:customercare@packtpub.com) for more details. At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks. TM <https://www2.packtpub.com/books/subscription/packtlib> Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books. Why subscribe?

* Fully searchable across every book published by Packt
* Copy and paste, print, and bookmark content
* On demand and accessible via a web browser

# 前言

概率图模型是机器学习领域，表示现实世界带有概率信息的数据和模型的最先进的技术之一。在许多场景中，概率图模型使用贝叶斯方法来描述算法，以便可以从带有噪声和不确定性的现实世界中得出结论。

本书介绍了一些相关话题，例如推断（自动推理和学习），可以自动从原始数据中构建模型。它解释了算法是如何逐步运行的，并使用诸多示例展示了即时可用的R语言解决方案。介绍完概率和贝叶斯公式的基本原理之后，本书给出了概率图模型（Probabilistic Graphical Models，PGMs），以及几种类型的推断和学习算法。读者会从算法设计过渡到到模型的自动拟合。

然后，本书关注了在解决数据科学问题上有成功案例的有用模型，例如贝叶斯分类器，混合模型，贝叶斯线性回归，以及用于构建复杂模型的基本模型组件。

## 主要内容

*第1章，概率推理*，介绍了概率论和概率图模型的基本概念，并通过贝叶斯公式的表示，为概率模型提供一种易用，高效，简单的建模方法。

*第2章，精确推断*, 介绍了如何通过简单图形的组合和模型查询构建概率图模型。该查询使用一种叫做联结树算法的精确推断算法。

*第3章，学习参数*, 包括从数据集中使用最大似然法，拟合和学习概率图模型。

*第4章，贝叶斯建模——基础模型*, 介绍了简单而强大的贝叶斯模型，其可以作为更加复杂模型的基础模块，以及如何使用自适应算法来拟合和查询贝叶斯模型。

*第5章，近似推断*, 介绍了概率图模型上使用采用算法的第二种推断方法，同时介绍了主要的采样算法，例如马尔科夫链蒙特卡洛（MCMC）。

*第6章，贝叶斯建模——线性模型*, 介绍了标准线性回归算的更高级贝叶斯视角，并给出了解决过拟合问题的解决方案。

*第7章，概率混合模型*, 介绍了更加复杂的概率模型，其中的数据来自于几种简单模型的混合。

*附录*，介绍了本书所引用的所有书籍和文献。

## 环境准备

All the examples in this book can be used with R version 3 or above on any platform and operating system supporting R.

## 本书受众

本书面向需要处理海量数据，并从中得出结论的读者，尤其是当数据有噪声或者存在不确定性的读者。数据科学家，机器学习爱好者，工程师和其他对机器学习最新技术感兴趣的人会觉得概率图模型很有意思。

## 行文约定

在本书中，你会发现几种文本风格，来区分不同信息类型。下面是这些风格的示例，和相应的含义。

文中代码，数据库表名，文件夹名，文件名，文件扩展名，路径名，虚拟URL，用户输入和推特账户做如下展示：“我们也可以提及arm程序包，它提供了glm()和polr()的贝叶斯版本，并实现了层次模型。”

所有命令行输入或者输出做如下展示：

pred\_sigma <- sqrt(sigma^2 + apply((T %\*% posterior\_sigma) \* T, MARGIN = 1,   
 FUN = sum))  
upper\_bound <- T %\*% posterior\_beta + qnorm(0.95) \* pred\_sigma  
lower\_bound <- T %\*% posterior\_beta - qnorm(0.95) \* pred\_sigma

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

## 读者反馈

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

## 客户支持

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## 下载示例代码

You can download the example code les for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the les e-mailed directly to you.

You can download the code les by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the SUPPORT tab at the top.
3. Click on Code Downloads & Errata.
4. Enter the name of the book in the Search box.
5. Select the book for which you're looking to download the code les.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on Code Download.

You can also download the code les by clicking on the Code Files button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the Search box. Please note that you need to be logged in to your Packt account. Once the le is downloaded, please make sure that you unzip or extract the folder using the latest version of:

* WinRAR / 7-Zip for Windows
* Zipeg / iZip / UnRarX for Mac
* 7-Zip / PeaZip for Linux

## 勘误

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you nd a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you nd any errata, please report them by visiting <http://www.packtpub>. com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details of your errata. Once your errata are veri ed, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title. To view the previously submitted errata, go to <https://www.packtpub.com/books/> content/support and enter the name of the book in the search eld. The required information will appear under the Errata section.

### 隐私

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy. Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material. We appreciate your help in protecting our authors and our ability to bring you valuable content.

## 问题

If you have a problem with any aspect of this book, you can contact us at [questions@packtpub.com](mailto:questions@packtpub.com), and we will do our best to address the problem.

# 第1章 概率推理

在2000年以后所有的预测中，最不希望的一个也许是我们需要每天收集世界上任何地方，关于任何事情的海量数据。近几年来，人们见证了关于世界，生活和技术方面难以置信的数据爆炸，这也是我们确信引发变革的源动力。虽然我们生活在信息时代，但是仅仅收集数据而不发掘价值和抽取知识是没有任何意义的。

在20世纪开始的时候，随着统计学的诞生，世界都在收集数据和生成统计。那个时候，唯一可靠的工具是铅笔和纸张，当然还有观察者的眼睛和耳朵。虽然在19世纪取得了长足的发展，但是科学观察依然处在新生阶段。

100多年后，我们有了计算机，电子感应器，以及大规模数据存储。我们不但可以持续的保存物理世界的数据，还可以通过社交网络，因特网和移动电话保存我们的生活数据。而且，存储技术水准的极大提高也使得以很小的容量存储月度数据成为可能，甚至可以放进手掌中。

但是存储数据不是获取知识。存储数据只是把数据放在某个地方以便后用。同样，随着存储容量的快速演化，现代计算机的容量甚至在以难以置信的速度提升。在博士期间，我记得当我收到一个崭新耀眼的全功能PC机来开展科研工作时，我在实验室是多么的骄傲。而今天，我口袋里老旧的智能手机，还要比当时快20倍。

在本书中，你会学到把数据转化为知识的最先进的技术之一：**机器学习**。这项技术用在当今生活的方方面面，从搜索引擎到股市预测，从语音识别到自动驾驶。而且，机器学习还用在了人们深信不疑的领域，从产品链的质量保障到移动手机网络的天线阵列优化。

机器学习是计算机科学，概率论和统计学相互融合的领域。机器学习的核心问题是推断问题或者说是如何使用数据和例子生成知识或预测。这也给我们里带来了机器学习的两个基础问题：从大量数据中抽取模式和高层级知识的算法设计，和使用这些知识的算法设计——或者用科学命名方式：学习和推断。

皮埃尔-西蒙拉普拉斯（Pierre-Simon Laplace，1749-1827），法国数学家，也是有史以来最伟大的科学家之一，被认为是第一批理解数据收集重要性的人：数据不可靠，有不确定性，也就是今天说的有噪声。他也是第一个研究使用概率来处理不确定性等问题，并表示事件或信息信念度的人。

在他的论文《概率的哲学》（*Essai philosophique sur les probabilités*，1814）中，拉普拉斯给出了最初的支持新老数据推理的数学系统，其中的用户信念会在新数据可用的时候得到更新和改进。今天我们称之为贝叶斯推理。事实上，托马斯贝叶斯确实是第一个，早在18世纪末就发现这个定理的人。没有贝叶斯工作的铺垫，皮埃尔-西蒙拉普拉斯需要重新发现同一个定理，并形成贝叶斯理论的现代形式。有意思的是，拉普拉斯最终发现了贝叶斯过世之后的发表文章，并承认了贝叶斯是第一个描述归纳推理系统原理的人。今天，我们会提及拉普拉斯推理，而不是贝叶斯推理，并称之为贝叶斯-普莱斯-拉普拉斯定理（Bayes-Price-Laplace theorem）。

一个多世纪以后，这项数学技术多亏了计算概率论的新发现得到重生，并诞生了机器学习中一个最重要，最常用的技术：概率图模型。

从此刻开始，我们需要记住，概率图模型中的术语**图**指的是图论，也就是带有边和点的数学对象，而不是图片或者图画。众所周知，当你想给别人解释不同对象或者实体之间的关系时，你需要拿纸画出带有连线或箭头的方框。这是一种简明易懂的方法，可以来介绍任何不同元素之间的关系。

**概率图模型**（Probabilistic Graphical Models，简称PGM）确切是指：你想描述不通变量之间的关系。但是，你又对这些变量不太确定，只有一定程度的相信或者一些不确定的知识。现在我们知道，概率是表示和处理不确定性的严密的数学方法。

概率图模型是使用概率来表示关于事实和事件的信念和不确定知识的一种工具。它也是现在最先进的机器学习技术之一，并有很多行业成功的案例。

概率图模型可以处理关于世界的不完整的知识，因为我们知识总是有限的。我们不可能观察到所有的事情，不可能用一台计算机表示整个宇宙。 和计算机相比，我们作为人类从根本上是受限的。有了概率图模型，我们可以构建简单的学习算法，或者复杂的专家系统。有了新的数据，我们可以改进这些模型，尽全力优化模型，也可以对未知的局势和事件作出推断或预测。

在第1章中，你会学到关于概率图模型的基础知识，也就是概率知识和积分简单规则。我们会有一个概率图模型能力概览，以及相关的R程序包。这些程序包都很成功，我们只需要探讨最重要的R程序包。

我们会看到如何一步一步的开发简单模型，就像方块游戏一样，以及如何把这行模型连接在一起开发出更加复杂的专家系统。我们会介绍下列概念和应用。每一部分都包含几个可以直接用R上手的示例：

* 机器学习
* 使用概率表示不确定性
* 概率专家系统的记号表示
* 使用图来表示知识
* 概率图模型
* 示例和应用

## 机器学习

本书是关于机器学习领域的书籍，或者更广义的叫做人工智能。为了完成任务，或者从数据中得出结论，计算机以及其他生物需要观察和处理自然世界的各种信息。从长期来看，我们一直在设计和发明各种算法和系统，来非常精准的并以非凡的速度解决问题。但是所有的算法都受限于所面向的具体任务本身。另外，一般生物和人类（以及许多其他动物）展现了在通过经验，错误和对世界的观察等方式取得适应和进化方面令人不可思议的能力。

试图理解如何从经验中学习，并适应变化的环境一直是科学界的伟大课题。自从计算机发明之后，一个主要的目标是在机器上重复生成这些技能。

机器学习是关于从数据和观察中学习和适应的算法研究，并实现推理和借助学到的模型和算法来执行任务。由于我们生活的世界本身就是不确定的，从这个意义上讲，即便是最简单的观察，例如天空的颜色也不可能绝对的确定。我们需要一套理论来解决这些不确定性。最自然的方法是概率论，它也是本书的数学基础。

但是当数据量逐渐增长为非常大的数据集时，即便是最简单的概率问题也会变得棘手。我们需要一套框架，支持面向现实世界问题必要复杂度的模型和算法的便捷开发。

说的现实世界的问题，我们可以设想一些人类可以完成的任务，例如理解人类语言，开车，股票交易，识别画中的人脸，或者完成医疗诊断等。

在人工智能的早期，构建这样的模型和算法是一项非常复杂的任务。每次，新算法产生，实现和规划总是带着内在的错误和偏差。本书给出的框架，叫做概率图模型，旨在区分模型设计任务和算法实现任务。因为，这项技术基于概率论和图论，因此它拥有坚实的数学基础。但是同时，这种框架也不需要实践者一直编写或者重写算法，因为算法是针对非常原生的问题而设计的，并且已经存在了。

同时，概率图模型基于机器学习技术，它有利于实践人员从数据中以最简单的方式创造新的模型。

概率图模型中的算法可以从数据中学到新的模型，并使用这些数据和模型回答相关问题，当然也可以在有新数据的时候改进模型。

在本书中，我们也会看到概率图模型是我们熟知的许多标准和经典模型的数学泛化，并允许我们复用，混合和修改这些模型

本章的其他部分会介绍概率论和图论所需的概念，帮助你理解和使用R语言概率图模型。

最后关于书名《概率图模型R语言学习》的一些介绍。事实上，这个书名有两层含义：你会学到如何构建概率图模型，以及如何让计算机学习概率图模型。这就是机器学习！

## 使用概率表示不确定性

概率图模型，从数学的角度看，是一种表示几个变量概率分布的方法，也叫做联合概率分布。换句话说，它是一种表示几个变量共同出现的数值信念的工具。基于这种理解，概率图模型看起来很简单，但是概率图模型强调的是对于许多变量概率分布的表示。在某些情况下，许多意味着大量，比如几千个到几百万个。在这一部分里，我们会回顾概率图模型的基本概念和R语言的基本实现。 如果你对这些内容很熟悉，你可以跳过这一部分。我们首先研究为什么概率是表示人们对于事实和事件信念的优良工具，然后我们会介绍概率积分的基本概念。接着，我们会介绍贝叶斯模型的基础构建模块，并做一些简单而有意思的计算。最后，我们会讨论本书的主要话题：贝叶斯推断。

我之前说过贝叶斯推断是本书的主要话题吗？确实，概率图模型也是执行贝叶斯推断，或者换句话说，从先前信念和新数据中计算新的事实和结论的前沿技术。

更新概率模型的原理首先是由托马斯贝叶斯发现的，并由他的朋友普莱斯于1763年发表在著名的论文*《论机会主义下的问题解决》（ An Essay toward solving a Problem in the Doctrine of Chances）*中。

### 信念和不确定性的概率表示

Probability theory is nothing but common sense reduced to calculation  
Théorie analytique des probabilités,   
1821. Pierre-Simon, marquis de Laplace

正如皮埃尔-西蒙拉普拉斯所说，概率是一种量化常识推理和信念程度的工具。有意思的是，在机器学习的背景下，信念这一概念已经被不知不觉的扩展到机器上，也就是计算机上。同时使用算法，计算机会对确定的事实和事件，通过概率表示自己的信念。

让我们举一个众人熟知的例子：掷硬币游戏。硬币正面或者反面向上的概率或机会是多少？大家都应该回答是50%的机会或者0.5的概率（记住，概率是0和1之间的数）。

这个简单的记法有两种理解。一种是**频率派**解释，另一种是**贝叶斯派**解释。第一种，频率派，意思是如果我们投掷多次，长期来看一半次数正面向上，另一半次数反面向上。使用数字的话，硬币有50%的机会一面朝上，或者概率为0.5。然而，频率派的思想，正如他的名字，只在实验可以重复非常多的次数时才有效。如果只观察到一两次事实，讨论频率就没有意义了。相反，贝叶斯派的理解把因素或事件的不确定性通过指认（0和1之间，或者0%和100%之间）数值来量化。如果你投掷一枚硬币，即使在投掷之前，你也肯定会给每个面指认50%的机会。如果你观看10匹马的赛马，而且对马匹和骑手一无所知，你也肯定会给每匹马指认0.1（或者10%）的概率。

投掷硬币是一类可以重复多次，甚至上千次或任意次的试验。然而，赛马并不是可以重复多次的实验。你最喜欢的团队赢得下次球赛的概率是多少？这也不是可以重复多次的实验：事实上，你只可以实验一次，因为只有一次比赛。但是由于你非常相信你的团队是今年最厉害的，你会指认一个概率，例如0.9，来确信你的团队会拿下下一次比赛。

贝叶斯派思想的主要优势是它不需要长期频率或者同一个实验的重复。

在机器学习中，概率是大部分系统和算法的基础部件。你可能想知道收到的邮件是垃圾邮件的概率。你可能想知道在线网站下一个客户购买上一个客户同一个商品的概率（以及你的网站是否应该立刻给它打广告的概率）。你也想知道下个月，你的商铺拥有和这个月同样多客户的概率。

从这些例子可以看出，完全频率派和完全贝叶斯派之间的界限远远不够清晰。好消息是不论你选择哪一种理解，概率积分的规则是完全相同的。

### 条件概率

机器学习尤其是概率图模型的核心是条件概率的思想。事实上，准确的说，概率图模型都是条件概率的思想。让我们回到赛马的例子。我们说，如果你对骑手和马匹一无所知，你可以给每一匹马指认0.1的概率（假定有10匹马）。现在，你知道这个国家最好的骑手也参加了这项赛事。你还会给这些骑手指认相同的机会吗？当然不能！因此这个骑手获胜的概率可能是19%，进而所有其他骑手获胜的概率只有9%。这就是条件概率：也就是基于已知其他事件的结果，当前事件的概率。这种概率的思想可以完美的解释改变直觉认识或者（更技术性的描述）给定新的信息来更新信念。概率图模型就是关注这些技术，只是放在了更加复杂的场景中。

### 概率积分和随机变量

在之前的部分，我们看到了为什么概率是表示不确定性或者信念，以及事件或事实频率的优良工具。我们也提到了不管是贝叶斯派还是频率派，他们使用的概率积分规则是相同的。在本部分中，我们首先回顾概率积分规则，并介绍随机变量的概念。它是贝叶斯推理和概率图模型的核心概念。

#### 样本空间，事件和概率

在这一部分中，我们会介绍本书概率论中使用的基本概念和语言。如果你已经知道了这些概念，你可以跳过这一部分。

一个**样本空间**是一个实验所有可能输出的集合。在这个集合中，我们称中的一个点，为一个**实现**。我们称的一个子集为一个**事件**。

例如，如果我们投掷一枚硬币一次，我们可以得到正面（H）或者反面（T）。我们说样本空间是。一个事件可以是我得到了正面（H）。如果我们投掷一枚硬币两次，样本空间变得更大，我们可以记录所有的可能。一个事件可以是我们首先得到了正面。因此我的事件是。

更复杂的例子可以是某人身高的米数度量[[1]](#footnote-53)。样本空间是所有从0.0到10.9的正数。 你的朋友很有可能都没有10.9米高，但是这并不会破坏我们的理论。

一个事件可以是所有的篮球运动员，也就是高于2米的的人。其数学记法写作，相对区间，。

一个**概率**是指派给每一个事件E的一个实数。概率必须满足下列三个公理。在给出它们之前，我们需要回顾为什么需要使用这些公理。如果你还记得我们之前说的，不论我们对概率做何理解（频率派或贝叶斯派），控制概率积分的规则是一样的：

* 对于任意事件E，：我们说概率永远为正。
* ，意味着包含所有可能事件的概率为1。因此，从公理1和2看到，任何概率都在0和1之间。
* 如果有独立事件，，...，那么。

#### 随机变量和概率积分

在计算机程序中，变量是与计算机内存中一部分存储空间相关联的名称或者标记。因此一个程序变量可以通过它的位置（许多语言中的类型）来定义，并保存有且仅有一个取值。这个取值可以很复杂，例如数组或者数据结构。最重要的是，这个取值是已知的，并且除非有人特意改变，它也保持不变。换句话说，取值只能在算法确定要改变它的时候才会发生变化。

而随机变量有点不同：它是从样本空间到实数的函数映射。例如，在一些试验中，随机变量被隐式的使用：

• 当投掷两颗骰子的时候，两个点数之和是一个随机变量。 • 当投掷一枚硬币N次时，正面向上的次数是一个随机变量。

对于每一个可能的事件，我们可以关联一个概率。所有这些概率的集合是随机变量的**概率分布**。

让我们看一个例子：考虑投掷一枚硬币3次的实验。（样本空间中的）样本点是3次投掷的结果。例如，HHT，两次正面向上和一次背面向上是一个样本点。

因此我们可以很容易的列举所有可能的输出，并找出样本空间：

假设为第i次投掷正面向上的事件。例如：

如果我们给每个事件指认1/8的概率，那么使用列举的方法，我们可以看到。

在这个概率模型中，事件,,是相互独立的。要验证这个结论，我们首先有：

我们还必须验证每一对乘积。例如：

对于另外两对也需要同样的验证。所以,,是相互独立的。通常，我们把两个独立事件的概率写作他们独自概率的乘积：。我们把两个不相干独立事件的概率写作他们独立概率的和：。

如果我们考虑不同的结果，我们可以定义另外一种概率分布。例如，假设我们依然投掷3次骰子。这次随机变量是完成3次投掷后，正面向上的总次数。

使用列举方法我们可以得到和之前一样的样本空间：

但是这次我们考虑正面向上的次数，随机变量会把样本空间映射到下列数值：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | HHH | HHT | HTH | THH | TTH | THT | HTT | TTT |
| X(s) | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 0 |

因此随机变量的取值范围是。和之前一样，如果我们假设所有点都有相同的概率1/8，我们可以推出取值范围的概率函数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | 0 | 1 | 2 | 3 |
| P(X=x) | 1/8 | 3/8 | 3/8 | 1/8 |

### 联合概率分布

让我们回到第一个游戏，同时得到2次正面向上和一次6点，低概率的获胜游戏。我们可以给硬币投掷实验关联一个随机变量，它是2次投掷后获得正面的次数。这个随机变量可以很好的刻画我们的实验，取0，1和2。因此，我们不说对两次正面向上的事件感兴趣，而等价的说我们对事件感兴趣。这种表述方便我们查看其它事件，例如只有1次正面（HT或TH），甚至0次正面（TT）。我们说，给每个取值指派概率的函数叫做概率分布。另一个随机变量是，表述投掷骰子之后的点数。

当我们同时考虑两个实验（投掷硬币2次和投掷一个骰子）的时候，我们对同时获得0，1或2的概率以及1，2，3，4，5或6的点数概率更感兴趣。这两个同时考虑的随机变量的概率分布写作，称作**联合概率分布** 。

如果一直加入越来越多的实验和变量，我们可以写出一个很长很复杂的联合概率分布。例如，我们可能对明天下雨的概率，股市上涨的概率，以及明天上班路上高速堵车的概率感兴趣。这是一个复杂的例子但是没有实际意义。我们几乎可以确定股市和天气不会有依赖关系。然而，交通状况和天气状况是密切关联的。我可以写出分布——天气，股市，交通——但是它似乎有点过于复杂了。而事实并非如此，这也是本书要一直探讨的话题。

一个概率图模型就是一个联合概率分布。除此以外，并无他物。

联合概率分布的最后一个重要概念是**边缘化**（marginalization）。当你考察几个随机变量的概率分布时，即联合概率分布时，你也许想从分布中消除一些变量，得到较少变量的分布。这个操作很重要。联合分布的边缘分布可以通过下列操作获得：

，其中我们按照所有可能的取值汇总概率。通过这个操作，你可以从消除。作为练习，可以考虑一下这个概率与之前看到的两个不相干事件概率之间的关系。

对于数学见长的读者，当是连续值时，边缘化可以写作。

这个操作非常重要，但对于概率图模型也很难计算。几乎所有的概率图模型都试图提出有效的算法，来解决这个问题。多亏了这些算法，我们可以处理现实世界里包含许多变量的复杂而有效的模型。

### 贝叶斯规则

让我们继续探讨概率图模型的一些基本概念。我们看到了边缘化的概念，它很重要因为当有一个复杂模型的时候，你可能希望从一个或者少数变量中抽取信息。此时就用上边缘化了。

但是最重要的两个概念是条件概率和贝叶斯规则。

**条件概率**是指在知道其他事件发生的条件下当前事件的概率。很明显，两个事件必须某种程度的依赖，否则一个事件的发生不会改变另一个事件：

* 明天下雨的概率是多少？明天路上拥堵的概率是多少？
* 知道明天要下雨的话，路上拥堵的概率又是多少？它应该比没有下雨知识的情况下要高。

这就是条件概率。更形式化的，我们可以给出下列公式：

从这两个等式我们可以轻松的推导出贝叶斯公式：

这个公式是最重要的公式，它可以帮助我们转换概率关系。这也是拉普拉斯生涯的杰作，也是现代科学中最重要的公式。然而它也很简单。

在这个公式中，我们把叫做是给定下的后验分布。因此，我们也把叫做先验分布。我们也把叫做似然率，叫做归一化因子。

我们再解释一下归一化因子。回忆一下。而且我们有，即旨在消除（移出）联合概率分布中单个变量的边缘化。

因此基于上述理解，我们可以有。

借助简单的代数技巧，我们可以把贝叶斯公式改写成一般的形式，也是最方便使用的形式：

这个公式之美，以至于我们只需要给定和使用和，也就是先验和似然率。虽然形式简单，分母中的和正如以后所见，可能是一个棘手的问题，复杂的问题也需要先进的技术。

#### 理解贝叶斯公式

现在我们有和两个随机变量的贝叶斯公式，让我们改写成另外另个变量的形式。毕竟，用什么字母并不重要，但是它可以给出公式背后的自然理解：

这些概念背后的直觉逻辑如下：

* **先验分布**是指我们在知道其他信息之前对的认识——我的初始信念。
* 给定值下的\_\_似然率\_\_，是指我可以\_生成\_什么样的数据。换句话说，对于所有的，D的概率是多少。
* **后验概率**，是指观察到之后，对的新信念。

这个公式也给出了更新变量信念的前向过程。使用贝叶斯规则可以计算新的分布。如果又收到了新的信息，我们可以一次又一次更新信念。

#### 贝叶斯规则的第一个例子

在这一部分中，我们会看到第一个R语言的贝叶斯程序。我们会定义**离散随机变量**，也就是随机变量只能取预定义数量的数值。假设我们有一个制作灯泡的机器。你想知道机器是否正常工作还是有问题。为了得到答案你可以测试每一个灯泡，但是灯泡的数量可能很多。使用少量样本和贝叶斯规则，你可以估计机器是否在正常的工作。

在构建贝叶斯模型的时候，我们总是需要建立两个部件：

* 先验分布
* 似然率

在这个例子中，我们不需要特殊的程序包；我们只需要编写一个简单的函数来实现贝叶斯规则的简单形式。

先验分布是我们关于机器工作状态的初始信念。我们确定了第一个刻画机器状态的随机变量。这个随机变量有两个状态。我们相信机器是好的，是可以正常工作的。所以先验分布如下：

简单的说，我们对于机器正常工作的信念度很高，即99%的正常和1%的有问题。很明显，我们在使用概率的贝叶斯思想，因为我们并没有很多机器，而只有一台机器。我们也可以询问机器供应商，得到生产正常机器的频率信息。我们也可以使用他们提供的数字，这种情况下，概率就有了频率派的解释。但是，贝叶斯规则在所有理解下都适用。

第二个变量是，是机器生产的灯泡。灯泡可能是好的，也可能是坏的。所以这个随机变量包含两个状态。

同样，我们需要给出灯泡变量的先验分布：在贝叶斯公式中，我们需要给出先验分布和似然率分布。在这个例子中，似然率是，而不是。

这里我们事实上需要定义两个概率分布：一个是机器正常时的概率，一个是机器损坏时的概率。我们需要回答两遍：

* 当机器正常的时候，生产出好的灯泡或者坏的灯泡的可能性是多少？
* 当机器不正常的时候，生产出好的灯泡或者坏的灯泡的可能性是多少？

让我们给出最可能的猜测，不管是支持贝叶斯派还是频率派，因为我们有下列统计：

我们相信，如果机器正常，生产100个灯泡只会有一个是坏的，这比之前说的还要高些。但是在这个例子中，我们知道机器工作正常，我们期望非常高的良品率。但是，如果机器坏掉，我们认为至少40%的灯泡都是坏的。现在，我们已经完整的刻画了模型，并可以使用它了。

使用贝叶斯模型是要在新的事实可用时计算后验分布。在我们的例子中，我们想知道，在已知最后一个灯泡是坏的情况下机器是否可以正常工作。所以，我们想计算。我们只需要给出和，最后只需用一下贝叶斯公式来转换概率分布。

例如，假设最后生成的灯泡是坏的，即。使用贝叶斯公式我们有：

正如所见，机器正常工作的概率是71%。这个值比较低，但是符合机器依然正常的直观感觉。尽管我们收到了一个坏灯泡，但也仅此一个，也许下一个就好了。

让我们重新计算同样的问题，其中机器正常与否先验的概率相等：50%的机器工作正常，50%的机器工作不正常。结果变成：

机器有2.4%的概率正常工作。这就很低了。确实，给定机器质量后，正如建模成似然率，机器似乎要生产出坏灯泡。在这个例子中，我们并没有做有关机器正常的任何假设。生成出一个坏灯泡可以看做是出问题的迹象。

#### 贝叶斯规则的第一个R语言例子

看了之前的例子，有人会问第一个有意义的问题，如果观察多个坏灯泡我们需要怎么办？只看到一个坏灯泡就说机器需要维修，这似乎有些不合情理。贝叶斯派的做法是使用后验概率作为新的概率，并在序列中更新后验分布。然后，徒手做起来会很繁重，我们会编写第一个R语言贝叶斯程序。

下列代码是一个函数，计算给定先验分布，似然率和观察数据序列后的后验概率。这个函数有3个变量：先验分布，似然率和数据序列。prior和data是向量，likelihood是矩阵：

prior <- c(working = 0.99, broken = 0.01)  
likelihood <- rbind(working = c(good = 0.99, bad = 0.01), broken = c(good = 0.6,   
 bad = 0.4))  
data <- c("bad", "bad", "bad", "bad")

所以我们定义了3个变量，包含工作状态working和broken的prior， 刻画每个机器状态（working和broken）的likelihood，灯泡变量上的distribution。因此一共有4个值，R矩阵确实类似与之前定义的条件概率：

likelihood  
 good bad  
working 0.99 0.01  
broken 0.60 0.40

data变量包含观察到的，用于测试机器和计算后验概率的灯泡序列。因此，我们可以定义如下贝叶斯更新函数：

bayes <- function(prior, likelihood, data) {  
 posterior <- matrix(0, nrow = length(data), ncol = length(prior))  
 dimnames(posterior) <- list(data, names(prior))  
 initial\_prior <- prior  
 for (i in 1:length(data)) {  
 posterior[i, ] <- prior \* likelihood[, data[i]]/sum(prior \* likelihood[,   
 data[i]])  
 prior <- posterior[i, ]  
 }  
 return(rbind(initial\_prior, posterior))  
}

这个函数做了下列事情：

* 创建一个矩阵，存储后验分布的连续计算结果。
* 然后对于每一个数据，给定当前先验概率计算后验概率：和之前的一样，你可以看到贝叶斯公式的R代码。
* 最后，新的先验概率是当前的后验概率，而且同样的过程可以迭代。

最终，函数返回了一个矩阵，包含初始先验概率和所有后续后验概率。

让我们多运行几次，理解一下工作原理。我们使用函数matplot绘出两个分布的演化情况。一个是机器正常（绿色线）的后验概率，一个是机器故障（红色线）的后验概率：

matplot(bayes(prior, likelihood, data), t = "b", lty = 1, pch = 20, col = c(3,   
 2))

结果可以从图中看到：随着坏灯泡的增多，机器正常的概率快速下降（实线或绿色线）[[2]](#footnote-60)

我们原本希望100只灯泡中只有1个坏灯泡，不要太多就好。所以这个机器现在需要维护了。红色线或虚线表示机器有问题。

如果先验概率不同，我们可以看到不同的演化。例如，假设我们不知道机器是否可以正常工作，我们为每一种情况指认相同的概率：

prior <- c(working = 0.5, broken = 0.5)

再次运行代码：

matplot(bayes(prior, likelihood, data), t = "b", lty = 1, pch = 20, col = c(3,   
 2))

我们又得到了一个快速收敛的曲线，其中机器有问题的概率很高。这对于给定一批坏灯泡的情形来说，并不意外：

如果一直变换数据，我们可以看到不同的行为。例如，假设机器正常工作的概率是99%。我们观察10个灯泡，其中第一个灯泡是坏的。我没有R代码：

prior = c(working = 0.99, broken = 0.01)  
data = c("bad", "good", "good", "good", "good", "good", "good", "good", "good",   
 "good")  
matplot(bayes(prior, likelihood, data), t = "b", pch = 20, col = c(3, 2))

结果如下图所示：

算法在第一个灯泡处犹豫了一下。因为这么好的机器，不大可能生产出一个坏灯泡。但是然后它又收敛到很高的概率，因为好灯泡的序列不会预示任何问题。

我们的第一个R语言贝叶斯模型就完成了。本章的其他部分，会介绍如何创建带有多于两个随机变量现实世界的模型，以及如何解决两个重要问题：

* 推断的问题，即收到新数据时计算后验概率的问题。
* 学习的问题，即数据集里先验概率的确定问题。

细心的读者也许会问：刚才看到的这个简单的算法可以解决推断问题吗？它确实可以，但是只能在有两个离散变量的时候。这有些过于简单，而无法捕捉现实世界的复杂性。我们会介绍本书的核心内容和执行贝叶斯推理的主流工具：概率图模型。

## 概率图模型

在本章的最后一部分，我们会介绍概率图模型，作为原生框架支持通过简单的模块生成复杂的概率模型。这些复杂模型通常对于要解决的复杂任务是必须的。而复杂并不意味着混乱，简单的事情是最好最有效的。复杂是指为了表示和解决拥有很多输入，部件或者数据的任务，我们需要一个不完全平凡的模型，但是要满足足够的复杂度。

这个复杂的模型可以分解成几个相互交互的简单问题。最终，最简单的构建模块是一个变量。这个变量有一个随机值，或者像之前部分看到的带有不确定性的一个值。

### 概率模型

如果你还记得，我们看到使用概率分布表示复杂概念是有可能的。当我们有许多随机变量时，我们把这个分布叫做联合分布。有时拿到几百个甚至上千个更多的随机变量并非不可能。表示这么庞大的分布是非常困难的，在大多数情况下也是不可能的。

例如，在医学诊断中，每一个变量表示一个症状。我们可以拿到许多这样的变量。其他变量可以表示病人的年龄，性别，体温，血压等等。 我们可以使用许多不同的变量表示病人状态。我们也可以加入其他信息，例如最近的天气条件，病人的年龄和饮食状况。

从这个复杂的系统中，我们想解决两个问题：

* 从病人的数据库中，我们希望评估和发现所有概率分布，以及相关参数。这当然是自动的过程。
* 我们希望把问题放入模型中，例如，“如果我们观察到了一系列症状，我们病人是否还健康？”。类似的，“如果我改变病人的饮食，并开了这个药，我的病人是否会恢复？”。

然而，还有一个重要的问题：在这个模型中，我们想利用其他重要的知识，甚至是最重要的知识之一：不同模型部件之间的交互。换句话说，不同随机变量之间的依赖。例如，症状和疾病之间有明显的依赖关系。另外，饮食和症状之间的依赖关系比较遥远，或者通过其他变量例如年龄性别有所依赖。

最终，在这个模型中完成的所有推理都天然的带有概率的性质。从对变量的观察，我们想推出其他变量的后验分布，得到它们的概率而不是简单的是或不是的回答。有了这个概率，我们可以拿到比二元响应更丰富的回答。

### 图和条件独立

让我们做一个简单的计算。假设我们有两个二元随机变量，比如一个是在本章上一节看到的变量。我们把它们命名为和。这两个变量的联合概率分布是。它们是二元变量，因此我们可以为每一个取值，简便起见称之为，和，。

我们需要给定多少概率值？一共有四个，即，，，和。

假设我们不止有两个二元随机变量，而是十个。这还是一个非常简单的模型，对吧？我们把这些变量叫做，，，，，，，，，。这种情况下，我们需要提供个值来确定我们的联合概率分布。如果我们还有10个变量，也就是一共20个变量该怎么办？这还是一个非常小模型。但是我们需要给定个值。这已经超过了一百万个值了。因此对于这么简单的模型，建模任务已经变得几乎不可能了！

概率图模型正是简洁的描述这类模型的框架，并支持有效的模型构建和使用。事实上，使用概率图模型处理上千个变量并不罕见。当然，计算机模型并不会存储几十亿个值，但是计算机会使用条件独立，以便模型可以在内存中易于处理和表示。而且，条件独立给模型添加了结构知识。这类知识给模型带来了巨大的不同。

在一个概率图模型中，变量之间的知识可以用图表示。这里有一个医学例子：如何诊断感冒。这只是一个示例，不代表任何医学建议。为了简单，这个例子做了极大地精简。我们有如下几个随机变量：

* : 年内季节
* : 鼻子堵塞
* : 病人头痛
* : 病人经常打喷嚏
* : 病人咳嗽
* : 病人感冒

因为每一个症状都有不同的程度，所以我们很自然的使用随机变量来表示这些症状。例如，如果病人的鼻子有点堵塞，我们会给这个变量指派，例如60%。即和。

在这例子中，概率分布一共需要个值（4个季节，每一个随机变量取2个值）。这已经很多了。坦白来讲，这已经很难确定诸如“鼻子不堵塞的概率”，“病人头痛和打喷嚏等的概率”。

但是，我们可以说头痛与咳嗽或鼻子堵塞并不是直接相关，除非病人得了感冒。事实上，病人头痛有很多其他原因。

而且，我们可以说**季节**对**打喷嚏**，**鼻子阻塞**有非常直接的影响，或者**咳嗽**对于**头痛**的影响很少或没有。在概率图模型中，我们会用图表示这些依赖关系。如下所示，每一个随机变量都是图中的节点，每一个关系都是两个节点间的箭头：

如上图所示：概率图模型中的每一个节点间都存在有向关系，即箭头。我们可以使用这种方式来简化联合概率分布，以便概率可以追踪。

使用图作为模型来简化复杂（或者甚至混乱）的分布有诸多好处：

* 首先，可以从上个例子中看到，通常我们建模一个问题的时候，随机变量只与其他随机变量的小规模子集直接交互。因此，使用图可以使得模型更加紧凑和易于处理。
* 图中的知识和依赖易于理解和沟通。
* 图模型引出了联合概率分布的紧凑表示，并且易于计算。
* 执行推断和学习的算法可以使用图论和相关算法来改进和推动所有推断和学习：与初始的联合概率分布相比，使用概率图模型会以几个级数速度的加速计算。

### 分解分布

在之前的普通感冒诊断的例子中，我们定义了一个简单的模型，包含变量，，，，和。我们看到，对于这样一个简单的专家系统，我们就需要128个参数！

我们还看到，我们可以基于常识或者简单的知识做出几个独立假设。在以后的内容中，我们会看到如何从数据集中发现这些假设（也叫做**结构学习**）。

所有我们可以做出假设，重写联合概率分布：

在这分布中，我们进行了分解。也就是说，我们把原来的联合概率分布表示为一些因子的乘积。在这例子中，因子是更加简单的概率分布，例如，病人感冒的情况下咳嗽的概率。由于我们可以把所有的变量看作是二元的（除了季节，它有4个取值），每一个小的因子（分布）只需要确定少量的参数：。 我们只需要30个简单的参数，而不是128个！这是个巨大的改进。

我说过，参数非常容易确定，不管是通过手工还是根据数据。例如，我们不知道病人是否得了感冒，因此我们可以给变量Cold指派相同的概率，即。

类似的，我们也很容易确定，因为如果病人得了感冒（），他很有可能咳嗽。如果他没有感冒，病人咳嗽的概率很低，但是不是零，因为还有其他可能的原因。

### 有向模型

通常，有向概率图模型可以按照如下形式分解多个随机变量，，...，上的联合概率分布：

是图中定义的变量的父变量的子集。

图中的父变量很容易理解：当箭头从**A**指向**B**时，**A**就是**B**的父变量。一个节点可以有很多可能的子节点，也可以有可能多的父节点。

有向模型非常适合建模需要表示*因果*关系的问题。它也非常适合参数学习，因为每一个局部概率分布都很容易学习。

我们在本章中多次提到了概率图模型可以使用简单的模块进行构建，并组合出更大的模型。在有向模型中，模块指的是小的概率分布。

而且，如果我们想给模型扩展9个新的变量以及一些关系，我们只需简单扩展图形。有向概率图模型的算法适用于任何图形，不管什么样的规模。

尽管如此，并不是所有的概率分布都可以表示成有向概率图模型。有时，我们也有必要放松一些假设。

同时，注意到图必须是*无环*的很重要。这意味着，你不可能同时找到从A到B的箭头和从B到A的箭头，如下图所示：

事实上，这个图并不表示之前定义的分解过程。它可能意味着*A是B的原因，同时B也是A的原因*。这是矛盾的，也没有等价的数学表示。

当假设或者关系不是有向的，还存在第二种概率图模型的形式。它的边都是无向的。它也叫作无向概率图模型或者马尔科夫网络。

### 无向模型

无向概率图模型可以按照如下形式分解多个随机变量，，...，上的联合概率分布：

这个公式的解释如下：

* 左边的第一个项是通常的联合概率分布。
* 常数是归一化常数，确保右侧所有项的和是1，因为这是一个概率分布。
* 是变量子集上的因子，以便这个子集的每一个成员是一个极大团，也就是内部所有节点都相互连接的子图：

在上图中，我们有4个节点，并且函数定义在子集，也就是极大团和上。因此这里的概率分布一点也不复杂。这种类型的模型在计算机视觉，图像处理，财务和其他变量间关系遵循一定模式的领域都有广泛的应用。

### 示例和应用

现在来讨论一下概率图模型的应用。其实这些应用用几百页去讲述也很难涵盖其中的一部分。正如我们看到的，概率图模型是一种建模复杂概率模型的很有用的框架，可以使得概率易于理解和处理。

在这部分中，我们会使用之前的两个模型：灯泡机和感冒诊断。

回忆一下，感冒诊断模型有下列分解形式：

而灯泡机仅仅通过两个变量定义：和。分解形式也很简单。

对应分布的图模型也很简单：

为了表示概率图模型，我们会使用R程序包gRain。安装如下：

source("http://bioconductor.org/biocLite.R")  
biocLite()  
install.packages("gRain")

需要注意，这个安装过程可能会持续几分钟，因为这个程序包还依赖于许多其他的程序包（尤其是我们经常用到的gRbase程序包），而且提供了对图模型的一些基本操作函数。当程序包安装好后，你可以加载基程序包：

library("gRbase")

首先，我们想定义一个带有变量，，，，的简单无向图：

graph <- ug("A:B:E + C:E:D")  
class(graph)

我们定义了带有团，和以及另一个团，和的图模型。这形成了一个蝴蝶状的图。它的语法很简单：字符串的每一个团用+分开，每一个团使用冒号分隔的变量名定义。

接着我们需要安装图的可视化程序包。我们会使用流行的Rgraphviz。要安装可以输入：

install.packages("Rgraphviz")  
plot(graph)

你可以得到第一个无向图，如下所示：

接着，我们希望定义一个有向图。假设我们依然有变量：

dag <- dag("A + B:A + C:B + D:B + E:C:D")  
dag  
plot(dag)

语法依然很简单：没有父节点的节点单独表示例如A，否则父节点通过冒号分隔的节点列表刻画。

这个程序包提供了多种定义图模型的语法。你也可以按照节点的方式构建图模型。我们会在本书中用到几种表示法，以及一个非常著名的表示法：矩阵表示法。一个图模型可以等价地表示为一个方阵，其中每一行和每一列表示一个节点。如果节点间存在边，那么矩阵的系数是1，否则为0。如果图是无向的，矩阵会是对称的；否则可以是任何样式。

最终，通过第二个例子我们可以得到下列图模型：

现在我们想为灯泡机问题定义一个简单的图模型，并给出数值概率。我们再做一遍计算，看看结果是否一致。

首先，我们为每一个节点定义取值：

machine\_val <- c("working", "broken")  
light\_bulb\_val <- c("good", "bad")

然后为两个随机变量定义百分比数值：

machine\_prob <- c(99, 1)  
light\_bulb\_prob <- c(99, 1, 60, 40)

接着，使用gRain定义随机变量：

M <- cptable(~machine, values = machine\_prob, levels = machine\_val)  
L <- cptable(~light\_bulb | machine, values = light\_bulb\_prob, levels = light\_bulb\_val)

这里，cptable表示条件概率表：它是离散型随机变量概率分布的内存表示。我们会在第2章精确推断中再次讨论这个表示法。

最后，我们可以构建新的概率图模型。当我们在第2章精确推断中研究推断算法例如 **联结树算法（Junction Tree Algorithm）**时，这种表示法更加易于理解:

plist <- compileCPT(list(M, L))  
plist

打印网络的时候，结果如下：

CPTspec with probabilities:  
 P( machine )  
 P( light\_bulb | machine )

这里，可以清楚的看到之前定义的概率分布。如果我们打印出变量的分布，我们可以再次看到之前的结果：

plist$machine  
plist$light\_bulb

输出的结果如下：

> plist$machine  
machine  
working broken  
 0.99 0.01  
> plist$light\_bulb  
 machine  
light\_bulb working broken  
 good 0.99 0.6  
 bad 0.01 0.4

现在我们从模型中找出后验概率。首先，给模型输入证据（即我们观察到一个坏灯泡），操作如下：

net <- grain(plist)  
net2 <- setEvidence(net, evidence = list(light\_bulb = "bad"))  
querygrain(net2, nodes = c("machine"))

程序包会借助推断算法计算结果，并输出下列结果：

$machine  
machine  
 working broken  
0.7122302 0.2877698

这个结果与之前使用贝叶斯方法得到的结果完全相同。现在我们可以创建更加强大的模型，以及针对不同的问题应用不同的算法。这就是下一章关于图模型上精确推断的内容。

## 小结

在第1章中，我们学到了概率论的基础概念。

我们看到了如何以及为什么使用概率来表示数据和知识的不确定性，同时我们还介绍了贝叶斯公式。这是计算后验概率的最重要的公式。也就是说，当新的数据可用时，要更新关于一个事实的信念和知识。

我们看到了什么是联合概率分布，同时看到它会很快变得很复杂以至于难以处理。我们学到了概率图模型的基础知识，它是对概率模型进行易于处理，高效 和简单建模的原生框架。最后，我们介绍了概率图模型的不同类型，并学到如何使用R程序包来编写第一个模型。

在下一章中，我们会学到概率图模型上执行贝叶斯推断的一系列算法，即给模型提出问题和寻求答案。我们会介绍R程序包的新特性，同时我们会学到这些算法如何工作，以及高效的执行。

# 第2章 精确推断

完成构建概率图模型之后，一个主要任务是我们想给模型提出问题并找出答案。对于联合概率分布的图模型和表示法，有很多使用方式。例如，我们可以研究随机变量之间的交互。我们还可以看到模型是否捕捉到关联关系或因果关系。而且，由于控制随机变量的概率模型已经参数化了，变量的概率分布可以通过对部分数值参数的认识变得完全已知。我们还可能对其他参数已知的情况下，特定参数的取值感兴趣。

本章主要介绍通过使用模型和变量子集的观察结果，来发现另一个变量子集后验概率的算法。我们没有必要观察和查询所有的变量。事实上，本章所有我们即将看到的算法都可以用在任何被观察的子集和被查询的子集上。

主要有两种类型的查询：

* **概率查询**，其中，我们观察到变量的一个子集，并选择这些变量的一个实例，称之为证据。然后我们计算变量集的子集的后验概率分布：。
* **MAP查询**，指的是找出对拥有最大概率的变量子集的共同赋值。同时，如果我们把叫做是被观察变量的集合，是模型的其他变量，那么MAP赋值可以通过来定义。换句话说，我们要找出未被观察的变量的值，满足如果我们观察到赋值，那么未被观察的变量拥有最大的概率。

本章的目的就是介绍解决精确推断问题的主流算法，即回答上述查询的问题。通常，推断问题可以化解为通过贝叶斯规则找出后验概率的问题。用数学的语言讲，如果我们把叫做模型的所有变量的集合，是被观察变量（证据）的集合，是隐含变量或非观察变量的集合，那么计算图模型的推断可以使用：

例如，在医学问题中，给定一个观察到的症状集合，我们想知道所有可能的疾病。在语音识别系统中，我们想知道被记录的声音（即说话者的语音）中最可能的单词序列。在雷达跟踪系统中，我们想从雷达读数中知道跟踪物体位置的概率分布。在推荐系统中，在给出售卖网站上用户最近的点击数据后，我们想知道待售产品的后验概率分布，以便给客户提供最优的5个产品的排序和推荐。

所有这些问题，以及更多的问题，总是需要计算后验概率分布。为了解决这个复杂的问题，我们打算研究一下另一个不同的算法，这个算法以概率图模型作为基础图形来执行高效的计算。但是，在本章的第一部分，为了理解算法如何工作，我们会看到如何执行朴素计算。该计算并不十分高效，但是可以作为基于此改进效率的框架。这叫做变量消解，它会逐步的减少查询中用不到的变量。

接着我们会看到，我们可以重用之前的计算，使用第二个算法，**和积算法**来改进算法效率。我们会把这个算法应用到每不同类型的概率图模型，特别是带有树状层级的图形。这部分将引出最后一个也是最重要的算法，**联结树算法**。它可以接收任何图形，并把它们转换为树状结构，进而生成高效的计算序列。这个算法可以使用大部分R程序包实现。我们也会在本章中使用。

在本章中，你会学到如何执行简单推断，改进计算效率，并最终使用图模型。这个图模型可以根据现实世界问题的复杂程度的需要而构建。我们会介绍R语言的算法，以及R程序包，例如gRain，gR，和rHugin中的函数。

在开始所有数学和编程工作之前，我们会在第一节中介绍概率图模型的设计过程，并且介绍几个专家系统的例子。我们还会展示如何把遗产模型表示为图模型，并从中收益。

本章的组织结构如下：

* 构建图模型
* 变量消解
* 和积与信念更新
* 联结树算法

## 构建图模型

图模型的设计考虑到两个不同的方面。首先，我们需要确定模型中涉及的变量。变量指我们可以观察到或度量到的事实，例如温度，价格，距离，项的数量，时间段，或任何其他值。一个变量也可以表示一个或真或假的简单事实。

同时，这也是我们为什么要构建图模型的原因。变量可以捕捉问题的局部，而我们却无法直接度量或者估计这些与问题相关的变量。例如，一个外科大夫能够度量病人的一系列症状。但是，疾病并不是我们可以直接观察到的事实。它们只能通过几个症状的观察结果推论出来。以一般的感冒为例，当我们说一个人得了感冒，每个人都能理解这句话，这很自然。但是，并没有一个叫做感冒的东西。当某种类型的鼻病毒过分增殖时，病人却有严重的上呼吸道（鼻子）病毒感染。这是一个复杂过程，但是这只是一个普通的感冒

直接推出病人得了感冒几乎不可能，除非医生对粘液采样，并评估样本中鼻病毒的数量足够说明病人得了某种特定的疾病。另一种方法是从简单的症状例如头痛，流鼻涕中得出结论。在这种情况下，表示病人得感冒的变量不能直接观测到。

第二种方法是图。图表示了不同变量之间的依赖，它们彼此如何关联，如何直接或间接的交互。如果你之前学过统计，你就会使用相关性的概念。在图模型中，对两个变量之间的依赖理解的更加宽泛。事实上，相关性只表示变量之间的线性关系。

例如，表示症状的变量和表示疾病的变量可以连接起来，因为这两个变量之间有直接的关系。

### 随机变量的类型

在多数情况下，我们要使用的变量都是离散型的。原因是我们对或真或假，或取特定数量的事实感兴趣。另一个原因是在许多科学领域使用离散变量进行建模非常常见，而且离散变量的图模型背后的数学逻辑已于理解和实现。

一个离散随机变量是定义在有限样本空间上。离散随机变量的例子包括：

* 一个骰子有样本数据集
* 一枚银币定义在集合上
* 一个症状定义在值上
* 单词中的一个字母定义在集合上
* 一个单词定义在一个非常大的英语单词集合上，这个集合是有限的
* 大小可以定义在有限数据集上

一个连续随机变量是定义在一个连续的样本空间上，例如，，或者其他区间。当然，我们也可以把随机变量定义在多维空间上，例如上，但是要保证每一个维度都有相关的含义。有时，把维度分成个不同的定义在上的随机变量也是有意义的。连续型随机变量的例子有：

* 距离公里数
* 温度
* 价格
* 其他随机变量的平均值
* 其他随机变量的方差

当我们在考虑问题的贝叶斯方案时，最后两个例子很重要，而且可以导出机器学习问题的有用表示。确实，在贝叶斯方法中，所有的数量都看作是随机变量。因此，如果我们定义一个服从分布的随机变量，我们可以进一步把和理解为随机变量。

事实上，在图模型中，把许多参数当成是随机变量并在图中连接起来通常是很有用的。这些连接可以基于常识，因果交互，或者其他存在与两个变量之间的足够强的依赖。

### 构建图

连接变量的原因有很多，正如我们在本章会看到的，也有许多算法可以自动的从数据集中学习出这些连接。如果你读了一些科技文献，你会找到因果关系，稀疏模型或者分解的相关文献。所有这些原因都是图模型中足够好的连接变量的原因。在本节中，我们会构建这样的模型，并介绍当两个变量连接时，模型和信息流会发生什么变化。我们会使用一个重要的概念**d-分离**。

另一个生成图模型的方法是模块化。这是图模型最吸引人的特点之一，因为你可以通过简单的模块构建复杂的模型，而且可以通过扩展图形来扩展已知的模型。

**学习参数和查询模型可以化解为同样的学习和推断算法应用。**

让我们看一些图模型实际的和理论的例，以及他们捕捉的问题类型。

#### 概率专家系统

假设我们想进行肺结核医疗诊断。介绍这个例子之前，我们需要为读者声明一下：本书的作者并无任何医疗技术和知识，他只关注在本书和机器学习领域。因此，下面例子的唯一目的是展示如何构建一个简单的图模型，而不用于任何医疗目的。而且，下列例子出自 <http://en.wikipedia.org/wiki/Tuberculosis_diagnosis> 。

肺结核是由结核杆菌引起的。只有临床生物分析检验可以检测出这种病毒，确认是否得了肺结核。然而，物理检验可以揭示一些肺结核的线索，帮助外科医生判断是否需要全面的临床检验得出病人体内是否存在这种病毒。而且，完整的肺结核医疗评估必须包含医疗历史，物理检验，胸部透视，以及微生物检验。因此，如果我们想查看可能的症状和检验，我们也可以确定相应的模型中用到的随机变量：

* : 超过三周的咳嗽。可以为真，也可以为假。
* : 胸部疼痛。可以为真，也可以为假。
* : 咯血。这也是一个或真或假的二元变量。
* : 盗汗。也是一个二元变量。
* : 饭量减少。这个比较主观。我们可以按照三个值分级：，表示饭量减少的程度。
* 最后，正如我们所说的，只有微生物研究可以断定肺结核，而其他的症状只能假设它的存在。因此我们需要两个随机变量，一个有关微生物研究的二元变量叫做 ，表明是否发现病毒，另一个是判断病人是否断定，可能，假设患有肺结核，以及肺结核呈阴性。它是有四个值的随机变量。

为了生成图模型，我们需要做两件事：首先是需要图形连接随机变量，然后评估每个变量相互连接的先验概率，或者是图中的每一个节点。对于第二个任务，评估概率需要医疗专家知识。很显然这已经超出了本书的范围（也超出了作者当时的技能范围）。因此我们简单引入几个概率饿名称，例如, , 等。

症状通常是由疾病引起的，反过来则不成立。例如，盗汗并不是肺结核的原因。而反过来讲却是成立的。而且盗汗可能由其他原因引起，例如卧室的大功率加热器。但是，病毒可以引起疾病。事实上，如果病毒确实存在但是量很少，它可能也不会引发疾病。这个简单的推理启发我们设计图形的思路。

让我们首先从二元症状，，和开始。它们都是有疾病引起的。变量也可以按照同样的原则添加到模型中，因此图形如下所示：

我们看到变量之间的连接存在某种模式。在图模型中处理原因和结果的时候，这个模式很常见。如果我们把同样的思路用到微生物研究和疾病的关系中，我们会得到以下交互结果：

因此，当我们把之前的两个图形放在一起后，最终的图形如下：

我们完成的操作是概率图模型非常重要的方面：我们把两个子模型连接在一起得到一个更加复杂，而且可以更有效的捕捉信息的大模型。事实上，我们可以在同一个图中添加更多症状和疾病，以便可以区分诸如肺结核和肺炎以及其他包含类似症状的疾病。通过计算给定症状下每个疾病的后验概率，医生可以判断采取什么医疗方案来应对最可能的疾病。这种形式的概率图模型有时也叫作**概率专家系统**。

#### 概率图模型的基本结构

我们继续研究图模型中的结构和模式，包括结构的类型和属性。我们会通过多个R程序包实现和展示其中的结果和模式，进而结束这一部分。

如果我们对同一个事实有很多原因，这些原因会在图中指向事实。在变量数目不多的情况下这个结构非常常见。确实，假设我们有原因到，它们都是二元变量，以及事实，它也是二元变量。正如我们在第一章看到的，相应的（局部）概率分布是：。

注意到所有的变量都是二元的，我们希望表示成一个带有个值的表[[3]](#footnote-76)。如果，这个值事实上并不大，我们就需要2048个值！这就需要确定大量的概率。如果我们有31个原因，!!!

是的，你需要40亿个值里表示31个原因和1个事实。使用标准的双精度浮点值，这将会占用计算机34,359,738,368字节的内存，也就是32GB！对于这么小的一个模型，这已经过于庞大了。如果你的变量不仅拥有两个值，而是拥有k个值，你就需要个值，来辨识之前的条件概率。这个数字太大了！

下图展示了原因：

我们可以进一步在原因上推理，因为其中一些原因并没有直接的与事实关联，而是引起了其他的原因。在这种情况下，我们给出原因的层级，如下图所示：

在这个例子中，我们处理了8个原因，但是每一个局部条件概率，例如最多只涉及4个变量。这就容易处理了。

当我们查看变量的序列时，我们想到了另一个结构。这个结构不会捕捉因果关系而是捕捉变量在时间上的顺序。这也是一种非常常见的结构。假设我们有一个随机变量来表示系统在时间时的状态，并假设系统的当前状态可以预测下一刻的状态。因此，我们可以在给定上一个状态的情况下，其中和表示时间，回答当前系统状态的概率分布。

接着，假设在每一个时刻，设想的系统都会生成一个值，或者换句话说，我们可以间接的观察系统。这个观察结果不是系统的状态，而是对其有直接依赖的信息。因此确定概率也是合理的，其中是依赖于状态的观察结果。最后，我们把这些讨论放在一起，有下图：

根据随机变量和的类型不同，这个图有几个名字。当变量是离散的，正如之前所述，这个模型也叫作**隐马尔可夫模型（Hidden Markov Model）**，即状态不能被直接观察到（隐藏的）的**马尔可夫模型（Markov Model）**。马尔可夫模型是这样一种模型，它的当前状态只依赖于之前的状态。在这个图中，只依赖于，这一事实清楚的建模了马尔可夫的特性这个属性。当所有的变量服从高斯分布（而且不是离散的），这就是著名的**卡尔曼滤波器（Kalman filter）**！

概率图模型引人注目的地方是遗产模型也可以表示成图模型。

你一定还记得这样一个图，边都是有向的（箭头表示），而且有环。从哲学的角度讲，这意味着一个结果可能是原因的原因，这有点矛盾。这也说明你的表示概率分布的分解公式可能不完整，这在数学上是错误的。例如你不能写成。

在下一节中，我们会看到如何在给定任何图形中其他变量，计算后验概率。但是在此之前，让我们看最后一个图，它是之前图形的综合：

这个图形很有意思，它在同一个图中结合了两个隐马尔可夫模型。但是其中一个，，也是另一个模型的状态的原因。这是非常有力的结合。我们可以执行逆向训练和写出这个图形的联合概率分布：

## 变量消解

之前的例子令人印象深刻，而且似乎有些复杂。在这一节中，我们会看到如何处理复杂的问题，并在任一模型上执行推断。事实上，我们会看到事情并不像想象的那么完美，还是有一些限制。而且，正如我们在第一章看到的，当人们处理推断问题时，他需要面对一个NP-困难问题，即导致算法有指数级的时间复杂度。

然而我们有动态编程算法可以在需要推断问题中达到相当高的效率。

回忆一下，推断是指给定模型中变量子集的观测值，计算其他变量子集的后验概率。解决这个问题通常意味着我们可以选取任一不相交的子集。

设是图模型中所有变量的集合，和是两个不相交的变量子集， 。我们把当做查询子集，也就是需要知道其中变量的后验概率，把作为观测子集，也就是其中的变量都有观测值，也称为证据（所有记作）。

因此根据第一章，*概率推理*中贝叶斯理论，一个查询的一般形式是 。事实上，可以看做是上的函数，使得——即同时有和的概率。

最后，我们可以定义，即图模型中既不是查询变量又不是观测变量的变量子集。然后我们可以计算。如果我们沿进行边际化，我们只有。

如果使用同样的推理，我们也可以计算证据的概率，例如。

因此贝叶斯推理的一般机制是沿着不需要的和观测到的变量进行边际化，只剩下要查询的变量。

让我们看一下下图中的简单例子：

这个图模型编码了下列概率分布：

这是一个非常简单的推理链，可以用来展示变量消解算法。正如我们之前看到的，图中的每一个节点都关联了一个潜在的函数，这个函数在类似于此的有向图中就是，给定父节点：情况下的条件概率。如果可以直接从图中的关联函数中读出，就需要通过沿边际化计算得出：。

它看起来很简单，但是也可以变得计算量很大（好吧，也许没那么大）。如果 and （即，有k个可能的值，有m个可能的值），执行之前的求和需要次操作。为了理解这个论断，我们写出求和公式：

这个公式需要用到每一个的值，和的每一个m值。

完成这个操作后，我们边际化。我们可以说，得到了一个等价的图模型，如下图：

这里，的分布已经通过的信息进行了更新。如果我们想找出的边缘分布，我们可以使用相同的算法，获取。同样，我们可以获得。最终，为了获取，我们所做的一切有下列完整求和形式：

但是，因为每一次求和中，我们只需要关注特定的变量。我们可以重写求和公式：

这极大的简化了计算量，因为求和操作只需要使用局部分布。作为练习，我想让读者展示，对于给定的概率图模型，表示中个变量的链，然后计算复杂度只有。注意，大记号表示，计算时间的上界与括号中的函数成比例（也称作最差时间复杂度）。很明显，这个方案已经很有效。

这个例子中的主要思想是，我们可以对变量求和，并在下一步中重用之前的结果。理想情况下，我们希望对任何图形使用同样的思想，然后通过暂存中间结果来逐步消解变量。这是可以做到的，因为得益于图模型的结构，每一个求和步骤中的表示只依赖少数变量，我们可以把结果沿着图中的路径暂存起来。

## 和积与信念更新

在计算一个变量（或者变量子集）的分布时，主要操作是边际化。边际化通过在一个变量上求和（或者变量子集）来把变量从表达式中消解出去。如果我们把叫做联合概率分布分解中的一个因子，我们可以使用如下属性，像之前章节中看到的，来泛化和优化变量消解算法：

* 对称律：
* 结合律：
* 如果：

如果我们再次使用这些属性，处理之前章节中的联合分布，我们有：

最后，反复出现的主要表达式是在一个因子上的和积结果，可以写作$*Z* {} $。

因此，通常如果我们可以找到有向图模型中因子或变量的优质顺序，正如之前看到的，我们可以使用和积公式，逐步消解每一个变量直到得到想要的子集。

顺序必须可以边际化每一个包含待消除变量的因子，生成可以再次使用的新因子。

一种可能的执行方式是使用下列算法（《概率图模型（Probabilistic Graphical Models）》，D. Koller，N. Friedman，2009，MIT出版社），叫做**和积变量消解算法（sum-product variable elimination algorithm）**：

* ：因子集合
* ：要消解的变量集合
* ：上的序

1. 设是上的序，满足当且仅当:
2. for i=1,...,k
3. 返回

这个算法执行如下：当收到消解变量或因子的顺序后，对每一个变量（或因子）使用算法消解变量并使用这个函数（后面会定义）的结果缩小因子集合。然后乘以剩下的因子并返回结果。

子过程如下，目的是一次消除一个变量：

和积消解算法（：因子集合，：要消解的变量）

1. $\tau=\sum\_Z \varPsi$
2. 返回

第二个步骤就是完成了我们在之前的例子中逐步消解的行为。这个思想首先乘上变量出现时的潜在函数，然后边际化（第4行）消解变量。最后，算法返回因子集合，这集合已经去掉所有包含的因子（第2行）。新的和积因子通过对的边际化得到，并添加进来（第5行）。同时注意，第1行选取了包含所有待消解的变量的因子。

最后，当这个过程按变量顺序执行，我们可以逐个消解变量直到获得期望的子集。

让我们看看这个过程在下列例子上是如何工作的：

这是分解形式：$ P(ABCD) = P(A) P(B|A) P(C|B) P(D|B)$。条件概率分布由如下矩阵定义：

A = matrix(c(0.8, 0.2), 2, 1)  
B = matrix(c(0.6, 0.4, 0.3, 0.7), 2, 2)  
C = matrix(c(0.5, 0.5, 0.8, 0.8), 2, 2)  
D = matrix(c(0.3, 0.7, 0.4, 0.6), 2, 2)

条件概率分布是用矩阵中的列表示。例如，是：

[,1] [,2]  
[1,] 0.6 0.3  
[2,] 0.4 0.7

要消解的变量集合是D$的边缘概率分布。我们可以逐步使用算法：

1. 首先消除顺序中的，获得，因此我们需要边际化 ：
2. 执行同样的过程，复用之前的结果，继续消解获得。在第一个算法的第3行中，你可以看到通过调用SumProductEliminateVar的结果有指派给了。这里使用了之前步骤的结果：

3. 现在，我们只剩下两个变量和，我们需要使用第二个算法中的同样步骤消解：

在R中，你可以使用下列代码，迅速的得到结果

Bs = t(A) %\*% t(B)  
Cs = Bs %\*% t(C)  
Ds = Cs %\*% t(D)  
Ds  
 [,1] [,2]  
[1,] 0.3362 0.6638

最后，我们还有三个问题：

* 如果我们观察到一个变量，该如何计算其他变量子集的后验概率？
* 是否可能自动找出变量的最优（或者至少非常高效）序列？
* 如果存在这样的序列，我们是否可以应用到任何类型的图中，特别是带有回路（而非环，如之前所述）的图中？

第一个问题的答案很简单，我们只需要通过实例化替换每一个因子。但是如果我们使用之前的算法，我们会得到，其中是查询子集。所以我们还需要正则化，根据贝叶斯公式，获取期望的后验条件概率。

之前的算法可以扩展成如下形式：T

* ，其中 是之前计算的边缘分布

我们会在本章中，使用一种叫做联结树的算法回答第二和第三个问题。这个算法也是当今概率图模型中最基础的算法。它试图把任何类型的图形转换成具有变量聚类的树中。这样我们就可以使用之前的算法，同时保证最优顺序和最小化的计算成本。

## 联结树算法

在这一节中，我们会对概率图模型中的主要算法有个大概了解。它叫做联结树算法。这个名字是来源于下列事实：在执行数值计算之前，我们会把概率图模型转换为一个树，它的一些属性保证后验概率的高效计算。

算法的其中一个特点是，不仅计算查询中的后验概率分布，它还计算所有其他未被观察到的变量后验概率分布。因此，对于同样计算代价，我们可以得到任何变量的分布。

为了得到这样的结果，联结树算法结合了信念传播和之前和积算法的效率，以及变量消解过程的通用性。事实上，变量消解算法可以用在任何类型的树上（除了带有回路的图中），和积算法可以保存中间结果以便使计算高效。因为变量消解算法只能用在树中，我们需要把带有回路的图转换为表示等价分布分解形式的树。

联结树算法基于下列思想。让我们还以之前熟悉的例子为例，，对每一个因子使用贝叶斯规则：

这个公式非常有意思，因为我们把集合交集中的变量作为分母。这种初始分解的重参数化是转换图模型和在转换结果上进行推断的重要指标。和是上述集合之间分子的概率分布，也就是说，可用的聚类交集。

当然，这并非一直通用的方法，但是是从图模型构建树模型，并执行推断的有用观察结果。

联结树的构建经过四个步骤，最终把图模型转换成树：

1. 对图模块化，包括使用无向边连接每一个结点的对父节点：
2. 然后图形被转换成一个无向图，其中每一个箭头都被一般的边替代。前两步操作的结果是每一个变量（图中的结点）和父节点现在都在同一个团中，即所有的结点都相互连接的子图中。
3. 接着对图三角化：当图有回路时，变量消解的结果以及导出图的再表示等价于给两个属于同一个无向回路的变量添加边。我们之前看到一个简单的例子：消解变量得到新图。当图有回路时，这个消解步骤等价于给两个结点添加套索。我们需要首先在图中执行这个步骤。在下一个图中，虚线来源于三角化，而实线源自之前的两步：
4. 最后一步会把三角化的图转换为一个聚类树，其中每个结点表示变量子集中的因子。子集由图中的每一个团决定。在每一个聚类节点之间，我们还有另外一种节点，叫做分隔节点。回忆一下本节开始时第一个简单的例子，当时我们使用类似的技术重参数化了模型：这里也执行了同样的操作，但是是在任意类型的图形上。聚类树的计算如下：

* 找出三角化图中每一个团，并从这些团给单个节点加入新的节点。
* 计算图上的最大扩展树。联结树就是一个最大扩展树。

因此从得到的聚类树，或者说是联结树上，我们有两种类型的结点：聚类结点和分隔结点。更一般的是，类似于我们最开始的例子，联结树的赶驴分布等于：

其中是联结树每一个聚类的因子，是联结树每一个分隔的因子。让我们从《贝叶斯推理和机器学习》（D. Barber, 剑桥大学出版社，2012）的一个例子中看一下完整转换的过程。 The initial graph is as follows:

现在，基于初始图形的三角化的无向图如下：

最终，联结树如下：

联结树上的推断是通过从一个聚类传递信息给另一个聚类实现的，传递的路径有两种：自顶向下和自底向上。完成聚类之间的完整信息更新后，，每一个聚类都会包含自身变量的后验概率分布（例如，例子中顶层结点的）。最后，找出任意变量的后验概率都可以归结为对其中一个聚类使用贝叶斯规则，并边缘化我们不感兴趣的那些变量。

联结树的实现算法是一个复杂的任务，但是幸运的是，一些R程序包已经包含了完整的实现过程。你也用过它们了。在第一章中，我们看一些使用程序包进行贝叶斯推断的简单例子。推断算法就是联结树算法。

作为联系，我们会使用之前的一个例子构建一个实验。例子中包含变量。简单起见，我们会考虑每一个变量都是二元的，一边我们不用处理太多的值。我们会假定下列分解过程：

这个公式可以用下图表示：

我们首先给加载程序包：

library(gRain)

然后创建从A到F的随机变量集合：

val = c("true", "false")  
F = cptable(~F, values = c(10, 90), levels = val)  
C = cptable(~C | F, values = c(10, 90, 20, 80), levels = val)  
E = cptable(~E | F, values = c(50, 50, 30, 70), levels = val)  
A = cptable(~A | C, values = c(50, 50, 70, 30), levels = val)  
D = cptable(~D | E, values = c(60, 40, 70, 30), levels = val)  
B = cptable(~B | A:D, values = c(60, 40, 70, 30, 20, 80, 10, 90), levels = val)

也许你还记得，函数创建一个条件概率表，它是离散变量写因子。与每一个变量关联的概率只取决于和服务于当前的例子。

因为，我们在创建条件概率表的时候已经给出了每一个变量的父节点，我们也就完全定义了我们的图。因此，接下来就是计算联结树。在大多数程序包中，计算联结树都是通过调用一个函数完成的。算法会在一次运行中完成所有的事情：

这里，我们运行如下命令：

plist = compileCPT(list(F, E, C, A, D, B))  
plist

检查已经正确编译到概率图模型中的变量列表，从之前的代码中获取次列表：

CPTspec with probabilities:  
 P( F )  
 P( E | F )  
 P( C | F )  
 P( A | C )  
 P( D | E )  
 P( B | A D )

这里事实上是概率分布的分解过程，如前所述。如果我们想进一步检查，我们可以查看一些变量的条件概率：

print(plist$F)  
print(plist$B)

结果如同设想的一样，条件概率表如下：

F  
true false  
0.1 0.9  
, , D = true  
A  
B true false  
 true 0.6 0.7  
 false 0.4 0.3  
, , D = false  
A  
B true false  
 true 0.2 0.1  
 false 0.8 0.9

第二个输出有些复杂，但是如果你仔细看一下会发现过两个分布：和。它们比要易读。

最终，我们创建了图模型，并通过下列命令调用联结树算法：

jtree = grain(plist)

再次检查得到的结果：

jtree  
Independence network: Compiled: FALSE Propagated: FALSE  
 Nodes: chr [1:6] "F" "E" "C" "A" "D" "B"

现在，你会想，就这么多吗？是的。有了联结树的图表示，你就可以执行任何可能的推理。而且，你只需要计算联结树一次。所有的查询都可以使用同一个联结树。当然，如果你改变了的联结树，你就要重新计算。让我们执行几个查询：

querygrain(jtree, nodes=c("F"), type="marginal")  
$F  
F  
 true false  
0.1 0.9

如果你需要的边缘分布，你会拿到初始的条件概率表，因为没有父节点。至少我们知道这个过程是可行的！

querygrain(jtree, nodes=c("C"), type="marginal")  
$C  
C  
true false  
0.19 0.81

这更加有意思，因为当我们在给定的情况下，只声明的条件概率分布时，它计算了的边缘分。我们不需要复杂的算法，例如联结树算法来计算这么小的边缘分布。之前看到的变量消解算法也足够了。

但是如果你需要的边缘分布，变量消解就不够用了，因为图中有环。然而联结树算法可以给出下列结果：

querygrain(jtree, nodes=c("B"), type="marginal")  
$B  
B  
true false  
0.478564 0.521436

我们可以查询更加复杂的分布，例如和的联合概率分布：

querygrain(jtree, nodes=c("A","B"), type="joint")  
 B  
A true false  
 true 0.309272 0.352728  
 false 0.169292 0.168708

事实上，任何联合分布都可以给出：

querygrain(jtree, nodes=c("A","B","C"), type="joint")  
, , B = true  
A  
C true false  
 true 0.044420 0.047630  
 false 0.264852 0.121662  
, , B = false  
A  
C true false  
 true 0.050580 0.047370  
 false 0.302148 0.121338

现在我们想观察变量并计算后验分布。假设F=true，我们想把这个信息传播到网络中的其余部分：

jtree2 = setEvidence(jtree, evidence = list(F = "true"))

我们可以再次查询网络：

querygrain(jtree, nodes=c("F"), type="marginal")  
$F  
F  
 true false  
0.1 0.9  
querygrain(jtree2, nodes=c("F"), type="marginal")  
$F  
F  
 true false   
 1 0

这个查询最有意思：在jtree的第一个查询中，我们有的边缘分布，在jtree2的第二个查询中，我们有... P(F=true) = 1!!! 事实上，我们可以在网络中设置一个证据，即F=true。这样概率就是1了。更加有趣的是，我们可以查询联合分布或者边缘分布：

querygrain(jtree, nodes=c("A"), type="marginal")  
$A  
A  
 true false  
0.662 0.338  
querygrain(jtree2, nodes=c("A"), type="marginal")  
$A  
A  
 true false  
 0.68 0.32

这里我们看到F=true改变了上的边缘分布 （第二个查询再次使用jtree2，即带有证据的树）。

我们可以查询任何其他的变量（看看结果有什么不同）：

querygrain(jtree, nodes=c("B"), type="marginal")  
$B  
B  
 true false  
   
querygrain(jtree2, nodes=c("B"), type="marginal")  
$B  
B  
 true false  
0.4696 0.5304

最后，我们可以设置更多证据并在网络中把它们进行前向和后向传播，也可以计算逆概率：

jtree3 = setEvidence(jtree, evidence = list(F = "true", A = "false"))

这里我们说F=true，A=false并再次查询网络，看看设置证据前后结果的不同：

querygrain(jtree, nodes=c("C"), type="marginal")  
$C  
C  
 true false  
 0.19 0.81  
querygrain(jtree2, nodes=c("C"), type="marginal")  
$C  
C  
 true false  
0.0989819 0.9010181  
querygrain(jtree3, nodes=c("C"), type="marginal")  
$C  
C  
 true false  
0.15625 0.84375

正如期望的，知道和的值可以极大的改变的概率分布。作为练习，读者可以设置的证据（然后是），看看的后验概率的变化。

## 概率图模型示例

在最后一部分中，我们会给出几个概率图模型的例子。它们都是理解精确推断的有些示例。这一部分的目的是展示实际的但是简单的例子，给读者提供一些开发自己模型的思路。

### 洒水器例子

这是一个在很多书本中提到的久远的例子。它很简单，但是可以展示一些推理过程。

假如我们在照看花园，草地是湿的。我们想知道草地为什么是湿的。有两种可能：之前下过雨或者我们忘记关掉洒水器。而且，我们可以观察天空。如果是多云天气，就有可能之前下过雨。但是，如果是多云天气，我们很有可能不会打开洒水器。因此在这个例子中，我们更有可能相信，我们并非忘记关掉洒水器。

这是一个因果推理的简单例子，可以用概率图模型表示。我们可以确定4个随机变量：cloudy, sprinkler, rain和wetgrass。每一个都是二元变量。

我们可以给出所有的先验分布。例如，P(cloudy=true) = P(cloudy=false)=0.5。

对于其他的变量，我们可以设定条件概率表。例如变量rain可以按照如下定义：

|  |  |  |
| --- | --- | --- |
| **cloudy** | **P(rain=T cloudy)** | **P(rain=F cloudy)** |
| True | 0.8 | 0.2 |
| False | 0.2 | 0.8 |

读者可以想象一下其他概率表。

概率图模型如下：

### 医疗专家系统

表示医疗知识的一种方法是把症状和原因连接起来。背后的推理是证明这些原因可以导致可观测的症状。问题是我们有很多症状，而且它们中有许多的原因都是相同的。

用概率图模型表示医疗知识库的思想包括两层节点：一层是原因节点，一层是症状节点。

每个节点的条件概率表都会强化或弱化症状和原因之间的连接，以便更好的表示每个症状最可能的原因。

依据关联的复杂程度，模型可是优良的推理模型，也可能是欠佳的模型而不利于精确推断。

而且，表示大型概率表可能会是个问题。因为，有太多的参数需要确定。然而，使用事实数据库，我们可以学习参数。在下一章中，我们会看到如何学习参数。

概率图模型如下：

在这个模型中，我们看到**symptom 2**和**symptom 3**有3个父节点。在更实际的医疗模型中这样的父节点更多。例如，头痛症状可以由很多不同的原因引起。在这例子中，使用近似方案来表示与点关联的条件概率表也是可能的。常用的模型叫**Noisy-OR模型**。

### 多于两层的模型

不像之前的例子，多于两层的模型在许多应用中更有意义，它拥有更深的因果推理过程，而且每一个节点上都有相应的原因和结果。对于理解问题本身的结构也很自然。

在这一类例子中，模型的复杂度并没有理论限制，但是我们通常会建议保证节点间的关系简单化。例如，每个节点拥有不超过3个父节点是比较好的策略。如果确实是这样，稍微深度的研究一下这些关系还是值得，可以看看模型是否可能进一步分解。

例如，J. Binder, D. Koller, S. Russell和K. Kanazawa的文章《*Adaptive Probabilistic Networks with Hidden Variables*》，Machine Learning, 29(2-3):213-244, 1997, 介绍了一个模型可以用来估计一个汽车保险客户可能的声明损失。

在这模型中，采用更多层的模型来表示关于汽车保险的知识。下图给出了这个模型，隐含节点用阴影表示，输出节点使用粗框表示：

有时，模型可能会很复杂，但是依然可用。例如，S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck和F. Jensen,的书《*MUNIN - an Expert EMG Assistant. In Computer-Aided Electromyography and Expert Systems*》（Elsevier (Noth-Holland), 1989.）第12章设计了一个复杂的网络，

这里我们展示了一个非常大的概率图模型，它来自于R程序包bnlearn (<http://www.bnlearn.com/>)。

R程序包bnlearn可以通过CRAN程序库获取，按照其他程序包一样的方法安装。

下图展示了之前文章中提到的模型。这个模型有1041个节点和1397条边。

很明显，手动设定所有的参数是不可能的。这种类型的概率图模型需要从数据中学习得到。但是它确实是一个很有意思的复杂模型：

### 树结构

树结构的概率图模型是一种有意思的模型。它通常可以生成非常高效的推理。建模变量之间关系的思想很简单，每个节点都只有一个父节点但是可以有很多子节点。

因此对于模型中的任意变量，我们一直在表示那些可以用编码的简单关系。

下图给出了这样一个模型：

在这个模型中，由联结树算法生成的节点簇总是由两类节点构成：子节点和父节点。因此，这个模型可以保证联结树算法的复杂度较低，并支持快速推断。

当然，所有这些节点都可以连接在一起，如果问题需要的话可以形成更加复杂的模型。这些都只是例子，也鼓励读者开发自己的模型。读者可以首先理解关注节点之间因果关系是什么样的。

而且我们也可以挖掘领域的结构知识来设计新的模型。逐步方案通常是个不错的选择。读者可以从非常简单的仅有几个节点的模型开始，执行查询看看模型的表现如何，然后再扩展模型。

给这样的模型设定参数是个困难的事情，在下一章中，我们会研究算法来从数据中学习参数，使有效的概率图模型开发任务更加简单。

## 小结

在第2章中，我们介绍了推断的基础知识，并看到计算后验概率的最重要的算法：变量消解和联结树算法。我们学习了如何考虑因果关系，时序关系以及确定变量之间的模式来构建图模型。我们接触了图模型的一些基础特征，支持组合图形构建更加复杂的模型。我们学习了如何在R中使用联结树算法执行推断，并看到同样的联结树可以用到边缘分布和联合分布任何类型的查询。在最后一节中，我们看到几个现实世界的概率图模型例子。概率图模型通常是精确推断的优良备选方案。

在这一章里，我们在定义一个新的图模型时遇到了一个问题：确定参数很繁琐。事实上，即使在小型例子中，这个事情也很复杂。在下一章中，我们会学习如何从数据集中自动的找出参数。我们会介绍**EM（期望最大化，Expectation Maximization）**算法，并尝试解决复杂问题：学习图本身的结构。我们会看到推断是所有学习机器算法中最重要子任务，因此很有必要设计诸如联结树这种高效的算法。

# 第3章 学习参数

构建概率图模型大致需要三个步骤：定义随机变量，即图中的节点；定义图的结构；以及定义每个局部分布的数值参数。到目前为止，最后一步已经通过人工解决，我们可以手动的给每个局部概率分布指定数值。在很多情型中，我们可以获取到大量数据，并使用叫做**参数学习（parameter learning）**的方法找出这些参数的取值。在其他领域中，这种方法也叫做**参数拟合（parameter fitting ）**或者**模型校准（model calibration）**。

参数学习是机器学习中的重要课题。在这一章中我们会看到如何使用数据集为给定的图模型学习参数。我们会从一个简单但是常见的例子开始，其中的数据完全可观测，然后进入一个复杂的例子，其中的数据部分可观测，需要跟多先进的技术。

参数学习可以通过多个手段完成，问题本身没有终极解决方案，因为问题依赖于模型使用者的最终目的。尽管如此，人们还是经常是用最大似然率的思想，并最大化后验概率。既然已经熟悉了先验概率和后验概率的分布，那么读者对最大化后验概率也应该有一些认识。

在这一章中，我们会使用数据集。当模型中有许多变量时，我们在任何时候都可以观测到这些变量的取值。所有变量同一时刻的观察结果表示一个数据集。例如，我们有一个关于某位学生在大学中表现的模型。在这个模型中，我们有几个随机变量，例如年龄，课程，分数，性别和年份等。一个观察结果可以是{21, Statistics, B+, female, 2nd year}。一个数据集就是这些观测结果的大型集合。

在整个这一章中，我们会做一个假设，即数据集是的，**独立同分布（independently and identically distributed）**的缩写。这意味着每个变量都假设服从同样的概率分布，且每个观测又独立于数据集中其他的变量。对于刚才学生的例子，这也很自然。但是如果我们考虑时间序列数据集，例如一个国家的GDP，那么数据集就不是的，相应的参数学习算法也不同。事实上，的数据集已经能够涵盖大量的应用了。

借助所有可能的方案，我们可以进一步讨论本章中的主要话题了。设为数据集，为图模型的参数，似然率函数为，换句话说，即给定参数下观测到（或者生成）数据集的概率。这就是为什么概率图模型有时也叫做**生成模型（generative models）**。

最大似然估计目的是要找出参数的值，最大化似然率。也可以写作。这是一个优化问题，即找到的最优值来最大化。

如果想更准确的刻画，我们可以采用贝叶斯方法，也给出参数的先验概率分布。在这例子中，找出参数值可以分解为找出的最大值。这个过程叫做**最大化后验概率（maximum a posteriori）**。

在这一章中，我们首先研究一下使用最大化似然率进行参数估计的简单例子，并给出R语言实现。然后，我们会看到概率图模型上的最大似然估计。最后，我们会研究更复杂的估计问题。这类问题可以包括数据缺失，可以是随机缺失，也可以是参数包括隐变量。这就需要我们介绍机器学习中最重要的一个算法，EM算法。 **EM**意思是**期望最大化**。

本章结构如下：

* 一个简单例子的引入
* 作为推断学习参数
* 最大似然率
* 期望最大化算法

## 引言

在这一章中，我们会学到如何让计算机学习模型的参数。我们的例子会使用多个自己构建的数据集，也可能是从其他网站下载的数据集。网络上有很多可用的数据集，我们会使用来自UCI机器学习库的数据。加州大学（UCI）尔湾分校的机器学习和智能系统中心提供了相关链接。

鸢尾花照片来自 <https://en.wikipedia.org/wiki/File:Iris_germanica_%28Purple_bearded_Iris%29,_Wakehurst_Place,_UK_-_Diliff.jpg>

鸢尾花照片来自 <https://en.wikipedia.org/wiki/File:Iris_germanica_%28Purple_bearded_Iris%29,_Wakehurst_Place,_UK_-_Diliff.jpg>

例如，最重要的一个数据集是鸢尾花数据集，其中的每一个数据点都表示一种鸢尾花的特点。不同的属性用来表示花萼的长度和宽度，以及花瓣的长度和宽度。

这个数据集可以下载到本地，并存在R的数据框data.frame中。每一个变量都是一列，我们会使用数据（或者假设数据按照顺序分布）来简化微积分和计算。

让我们下载数据集：

x=read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data",col.names=c("sepal\_length","sepal\_width","petal\_length","petal\_width","class"))  
head(x)  
 sepal\_length sepal\_width petal\_length petal\_width class  
1 4.9 3.0 1.4 0.2 Iris-setosa  
2 4.7 3.2 1.3 0.2 Iris-setosa  
3 4.6 3.1 1.5 0.2 Iris-setosa  
4 5.0 3.6 1.4 0.2 Iris-setosa  
5 5.4 3.9 1.7 0.4 Iris-setosa  
6 4.6 3.4 1.4 0.3 Iris-setosa

可以看到，数据集的每一次观察都是数据记得一行。使用data.frame来简化参数计算会非常有用。

我们可以使用数据集做一些简单的估计。例如我们只考虑第一个变量sepal\_length，并假设这个变量服从高斯分布，那么高斯分布中两个参数（平均值和方差）的最大似然估计可以简单的通过计算经验平均值和经验方差得到。在R中，几行代码如下：

mean(x$sepal\_length)  
[1] 5.848322  
var(x$sepal\_length)  
[1] 0.6865681

如果我们想处理离散变量，正如本章中的大多数情形，我们可以使用著名的程序包plyr来简化计算：

library(plyr)

现在，我们计算变量class上的分布。在data.frame中，可以执行：

y = daply(x,.(class),nrow) / nrow(x)  
y  
 Iris-setosa Iris-versicolor Iris-virginica  
 0.3288591 0.3355705 0.3355705

有意思的是，可以看到每种类型的分布大概是33%。我们只是把data.frame中class列的每个值的出现计了数，并除以值的总数。这样的操作给出了分布值，也可以用来作为每一类的先验概率。在这例子中，我们的分布基本上是均匀分布。

进一步，我们可以看一下给定一个类别下其他变量的分布。假设sepal\_length服从平均值为，方差为的高斯分布。一个简单的联合分布由以下分解给出：

。

计算条件概率等价于计算class变量中每个值对应的平均值和方差。执行如下：

daply(x,.(class), function(n) mean(n$sepal\_length))  
 Iris-setosa Iris-versicolor Iris-virginica  
 5.004082 5.936000 6.588000

类似的，变量class下每一个分布的方差如下：

daply(x,.(class), function(n) var(n$sepal\_length))  
 Iris-setosa Iris-versicolor Iris-virginica  
 0.1266497 0.2664327 0.4043429

使用R函数可以非常容易的计算条件建概率分布。如果想在离散分布上执行同样的计算，我们可以使用下列代码。首先，对变量sepal\_width离散化转换成离散值。它表示宽度，因此（简化起见）我们可以设定三个不同的值: {small, medium, large}。我们可以使用下列代码自动完成：

q <- quantile(x$sepal\_width, seq(0, 1, 0.33))

我们找出了变量sepal\_width的33%和66%分位数。33%以下的值对应small，33%和66%之间的值对应medium，大于66%的值对应large。

q  
 0% 33% 66% 99%  
2.000 2.900 3.200 4.152

然后我们在data.frame中创建一个新的变量，sepal\_width的离散版本。执行下列代码：

x$dsw[ x$sepal\_ width < q['33%']] = "small"  
x$dsw[ x$sepal\_ width >= q['33%'] & x$sepal\_width < q['66%'] ] = "medium"  
x$dsw[ x$sepal\_ width >= q['66%'] ] = "large"

对于分位数定义的每一个区间，我们都关联上small, medium或large值，放在x中新的列dsw（即discrete sepal width）中。

最终，我们可以通过下列代码学到条件概率分布:

p1 <- daply(x,.(dsw,class), function(n) nrow(n))  
p1  
 class  
dsw Iris-setosa Iris-versicolor Iris-virginica  
large 36 5 13  
medium 12 18 18  
small 1 27 29

这样在指定class值后，可以得到dsw中每一个值每次出现的计数。如果我们像把它们变成概率，只需要除以每个列的和。事实上，每个列都代表一个概率分布。可以执行下列代码：

p1 <- p1/colSums(p1)

最终结果是：

class  
dsw Iris-setosa Iris-versicolor Iris-virginica  
 large 0.7346939 0.1020408 0.2653061  
 medium 0.2400000 0.3600000 0.3600000  
 small 0.0200000 0.5400000 0.3800000

使用之前class上的分布，我们可以完全参数化模型，得到联合分布： 。

如果我们分析完成的过程，并试着抽取一条经验规则，可以说参数通过对class每一个值下sepal\_width值出现次数计数来找出了参数。我们还可以说我们分别发现了分布的每个因子的参数： 一个是，一个是。

在后面的内容中，我们会学习更加形式化的方法，解决如何泛化这一思想来学习带有离散变量的概率图模型，以及从理论的角度理解为什么这一思想通常是很有效的。

## 通过推断学习

在本章的引言部分，我们看到学习任务可以通过频率主义的计数法来完成。在多数情况下，这已经足够了，但是这还只是机器学习思想的狭窄认识。更普遍的讲，学习的过程是把数据与领域知识整合在一起的过程，以便创建新的模型或改进现有的模型。因此，学习可以看作是推断问题，人们需要更新已有的模型得到更好的模型。

让我们思考一个简单的问题：建模投掷硬币的结果。我们想检验硬币是不是公平的（没有做过手脚）。设为硬币正面朝上的概率。公平的投掷会满足0.5的概率。多次投掷硬币后，我们想估计这个概率。假设第i次投掷正面朝上，记输出是，否则为0。 我们也可以假设每一次投掷之间没有依赖。这意味着，观察是的。最终，我们把每一个投掷当做随机变量。投掷序列的联合分布是。每次投掷依赖于概率\theta，因此模型是：

在图模型中，可以表示成如下模型：

现在我们介绍一个新的图模型表示法：**平板表示法（the plate notation）**。左边的图是我们常用的表示法，其中只表示了的第一个和最后一个节点，以保证模型更简单。有时这有点令人费解，甚至在很多例子中节点一多就会有歧义和带来麻烦。右边的图表示了同样的图模型，其中方框的意思是节点重复N次。

在之前一节中，我们看到学习过程把数据整合到了模型中。在第一章中，我们看到，使用贝叶斯公式，我们可以基于给定的新信息更新概率分布。使用相同的方法，在现在问题里，我们想估计一下概率：

这也是贝叶斯公式的简单应用。

下一步，我们要确定公式的多个因子，首先是先验概率。直觉上讲，是一个连续变量，因为它可以取0到1之间的任意值。但是，我们会简化这个问题，采用离散化策略。设可以取3个不同的值——正面不公平， 反面不公平和公平；即，我们给出先验概率：

这个公式说明，我们相信硬币有75%的概率是公平的，有25%的概率趋向正面，5%的概率趋向反面朝上。接下来是估计的后验概率分布。需要注意，从现在开始我们会省略分母，并使用符号。它的含义是“正比于”，而不是完全等于（=）。

因此后验概率可以表示为：

这个公式并没有看上去复杂。首先，我们把替换为它的分解形式，如图中所示。然后，我们把替换为自己的分解，其中时取，时取。函数在括号内条件为真时等于1，否则为0。我们使用。

我们希望读者可以完成计算。最终我们有：

表达式中的求和是对实验中正面（或反面）朝上的简单计数， 以便确定硬币是否公平。如果我们把这两个计数记作和，就可以简化后验概率的表达，如下：

因此，我们最终可以给R环境输入这个公式，查看贝叶斯学习过程的结果：

posterior <- function(prob, nh, nt, Theta = c(0.2, 0.5, 0.8)) {  
 x = numeric(3)  
 for (i in 1:3) x[i] = prob[i] \* (Theta[i]^nh) \* ((1 - Theta[i])^nt)  
 norm = sum(x)  
 return(x/norm)  
}

在这个函数中，prob是每个取值的概率向量，nh和nt遵循之前的定义，Theta是可能值的向量。我们默认使用之前提到的值。这些代码还可以优化，但是在这里我们会尽量简单化。最重要的一行是实现后验概率公式的那一行。归一化因子是norm，最后返回后验概率分布的值。

让我尝试使用这个公式，看看不同的先验概率会带来什么样的结果

posterior(c(0.2,0.75,0.05),2,8)  
[1] 6.469319e-01 3.530287e-01 3.948559e-05  
posterior(c(0.2,0.75,0.05),8,2)  
[1] 0.0003067321 0.6855996202 0.3140936477  
posterior(c(0.2,0.75,0.05),5,5)  
[1] 0.027643708 0.965445364 0.006910927  
posterior(c(0.2,0.75,0.05),10,10)  
[1] 0.0030626872 0.9961716410 0.0007656718  
posterior(c(0.2,0.75,0.05),50,50)  
[1] 5.432096e-11 1.000000e+00 1.358024e-11

我们做下面几个实验：2次正面朝上和8次反面朝上，8次正面朝上和2次反面朝上，都是5次朝上，都是10次朝上和，都是50次朝上。注意，最后一个实验由于次数较多带来误差，其概率分布的和不等于1。读者应该时刻记得测试和调试自己的程序：概率分布的和应该等于1。在这个例子中，这意味着我们达到了机器精确度的上限，看到了误差。

分析结果，确定如何解决这些精确度问题：

* 2次正面朝上和8次反面朝上：硬币以65%的概率趋向于一个小的值 。这意味着硬币趋向于反面朝上。但是硬币的公平度还是35%，依然不小。
* 8次正面朝上和2次反面朝上：我们得到了相反的结果。但是由于先验概率（趋向于正面朝上）很低，, 结果依然以68%的概率说明硬币是公平的。
* 如果我们得到了相同的次数，那么结果充分说明硬币是公平的。这个概率也随着实验次数的增加而增加。

最后，我们给出一个处理概率计算时的必要技巧。当你需要把许多小数值相乘时，使用对数和加和，而不是原始值和乘法。因此新的算法会用到下等式： 。新算法进而变成下面计算的样子:

x[i] = exp(log(prob[i]) + nh \* log(Theta[i]) + nt \* log(1 - Theta[i]))

最后的检验是当先验分布是均匀分布，即硬币的每个面都赋予相等的概率的时候。使用下面的代码：

posterior(c(1/3,1/3,1/3),2,8,c(0.2,0.5,0.8))  
[1] 0.8727806225 0.1270062963 0.0002130812  
posterior(c(1/3,1/3,1/3),8,2,c(0.2,0.5,0.8))  
[1] 0.0002130812 0.1270062963 0.8727806225  
posterior(c(1/3,1/3,1/3),5,5,c(0.2,0.5,0.8))  
[1] 0.08839212 0.82321576 0.08839212

我们可以看到，每次实验的结论都靠近高的概率。

## 最大似然法

这一节会介绍一个简单的算法，来学习图模型中的所有参数。在第一节中，我们有了学习模型的初步经验。我们知道，参数可以通过每个变量的局部学习而得到。这就意味着，对于每一个拥有父节点的变量， 对于父节点的每一个组合，我们可以计算每一个值出现的频率。如果数据集足够完备，这会生成图模型的最大似然估计。

对于图模型中的每一个变量，以及其父节点的每一个组合:

* 抽取中对应值的所有数据点
* 计算值的的直方图
* 指定

这就可以了？没错，你只需完成这些步骤。相对困难的部分的是数据点的抽取，而这个任务可以使用R中的ddply或者aggregate函数解决。

为什么会如此简单？在看R算法之前，让我们看看这个算法为什么会有效。

### 经验分布和模型分布是如何关联的？

图模型表示一个变量集合的联合概率分布。但是并非每一个联合概率分布都可以表示成图模型。这里我们只对先前定义的有向概率图模型感兴趣。这个定义也可以看作是对期望表达的概率分布类型的一种约束，在这个例子中，约束可以表示为：

到目前为止，这是我们熟知的有向图模型的定义。

Empirical Distribution 设是数据点集合，表示变量的状态，那么经验分布会均匀分布在数据点上，而数据点外的值是0。 摘自：*Bayesian Reasoning and Machine Learning*, D. Barber 2012, 剑桥大学出版社。

假设中的点都是独立同分布，经验累积分布函数是。或者说，这种分布对于的每一个可能的状态，我们都会从数据集中关联上一个计算好的频率，如果数据集中没有该点，就给出0。

让我们想一下经验分布和模型分布的关系。

Kullback-Leibler散度（也叫作相对熵）是对两种概率分布和差异的非对称度量，记作。它给出了从中样本到中样本转换所需的比特数。直觉上讲，如果两个分布是一样的，那么Kullback-Leibler散度是0。

经验分布和模型分布之间的KL散度是

模型的对数似然率是，我们可以在之前的公式中看到，最右边的项就是模型在经验分布下的对数似然率。因此我们可以写作：

同时，因为项并不依赖，模型分布可以考虑改写为：

所以，从之前的公式我们看到最大化似然率等价于最小化对数似然率。假设第二项是个常数，最小化对数似然率也会最小化经验分布和模型分布之间的KL散度。这只是意味着，找出最大化似然参数等价于最小化经验分布和模型分布之间的KL散度。

如果上没有约束，那么答案就是。

但是回忆一下，我们确实有一些约束：必须是个图模型。因此，把真正的放到公示中看看有什么结果。

不要被两个求和符号吓坏，只需记得 ；即我们只用图模型概率分布的对数运算。由于外圈求和只依赖于内圈求和在变量上的每一项，因此这个大项可以简化为以下形式：

现在内部求和计算了在变量的子集限制下，分布的对数似然率。让我们再把常数加到公式里： Now let's add back the constant to this formula:

公式看着又很复杂，但是如果仔细观察括号内的部分，这次你会看到 and 之间的KL散度的公式。这个漂亮的结果意味着我们可以进一步简化公式：

我们在最后这个公式上的操作就是对KL散度带权求和。概率分布和KL散度都是正值，其他所有项也是正值，因此最小化求和就是对每一项最小化。同时，如前所述，最小化求和也意味着最大似然估计。但是如果我们仔细观察求和的内容，你会看到更多的KL散度，每一个都是与图中节点关联的一个小分布！我们需要最小化这些散度。 因此，这意味着，如果我们想最小化整个和之间的KL散度（并得到图模型中的最大似然估计，我们需要逐个的对每个节点分别做同样的处理。最小化这些KL散度，等价于计数和计算频度。因此，有向图模型的最大似然估计量可以通过选取父节点在中的数据点，并对图中每一个节点的数据点计数得到（这是计算频度）。

### 最大似然法和R语言实现

现在，我们可以写一个简单的R算法来学习图中的参数。在这一节中，我们会使用来自UCI的Nursery数据集(<https://archive.ics.uci.edu/ml/datasets/Nursery>)。这个算法不会使用任何图模型程序包，只会使用图程序包和常用的R包。

这个数据集有9个变量，与幼儿园入园申请有关。这个数据集保存在1980年代的斯洛文尼亚的卢布尔雅那，用于在申请量太高的情况下对其进行排序，以便构建一个专家系统来客观的解释为什么某个申请被接收或拒绝。所有变量都是类别型的，因此我们只需关注离散变量。

本节的目的是为了通过具体应用说明本章中所学的技能，因此我们不会试图构建一个完美的专家系统。基于这些理解，我们使用简单的图形来解释这个例子：

R代码如下，包括学习函数，其使用两个新程序包Tgraph和Rgraphviz。你会看到，为了易于理解，线都用数字标记（并不是代码的内容）：

1 library(graph)  
 2 library(Rgraphviz)  
 3 library(plyr)  
 4  
 5 data0 <- data.frame(  
 6 x=c("a","a","a","a","b","b","b","b"),  
 7 y=c("t","t","u","u","t","t","u","u"),  
 8 z=c("c","d","c","d","c","d","c","d"))  
 9  
 10 edges0 <- list(x=list(edges=2),y=list(edges=3),z=list())  
 11 g0 <- graphNEL(nodes=names(data0),edgeL=edges0,edgemod="directed")  
 12 plot(g0)  
 13  
 14 data1 <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/nursery/nursery.data",col.names=c("parents","has\_nurs","form","children","housing","finance","social","health","class"))  
 15 edges1 <- list( parents=list(), has\_nurs=list(), form=list(),children=list(),  
 16 housing=list(), finance=list(),social=list(),health=list(),  
 17 class=list(edges=1:8))  
 18 g1 <- graphNEL(nodes=names(data1), edgeL=edges1,edgemod="directed")  
 19 plot(g1)

从第1行到第3行，我们加载了必要的程序包。在第5行我们创建了一个简单的数据集来测试学习函数。这个数据集有3个变量，会对每一种组合生成50%的概率。

在第10行和第11行，我们创建了边和相应的图。在第12行，我们绘制了图，并得到下列图模型：

从第14行到第19行，我们绘制了第二个图模型，这一次使用Nursery数据集，绘图函数的输出是：

下面，我们有一个简单的函数来为每一带有（或不带有）父节点的变量计算条件概率表：

1 make\_cpt<-function(df,pa)   
 2{  
 3 prob <- nrow(df)  
 4 parents <- data.frame(df[1,pa])  
 5 names(parents) <- pa  
 6  
 7 data.frame(parents,prob) 8}

这个函数事实上会被后边的（plyr程序包）ddply函数调用，计算变量与其父节点的每一个组合的频率。这个计算通过第3行的nrow函数调用实现。

频率只是数据集中相同的组合出现的次数计数。因为这个函数在调用时组合是唯一的（得益于ddply）,所以我们可以在第4行中，只从第1行开始抽取所有父节点的值。

最终，主要的学习函数如下。代码并没有优化，但是非常简单明了。我们可以理解每一行的含义：

1 learn <- function(g,data)   
 2{  
 3 rg <- reverseEdgeDirections(g)  
 4 result <- list()  
 5  
 6 for(var in rg@nodes) 7{  
 8 pa <- unlist(adj(rg,var))  
 9 if(length(pa)>0)  
 10 {  
 11 X <- ddply(data, c(var,pa), make\_cpt, pa)  
 12 Y <- ddply(data, pa, make\_cpt, pa)  
 13 for(i in 1:nrow(Y))  
 14 {  
 15 c <- sapply(1:nrow(X), function(j) all(X[j,pa] ==  
Y[i,pa]))  
 16 c <- which(c)  
 17 X$prob[c] <- X$prob[c]/Y$prob[i]  
 18 }  
 19 }  
 20 else  
 21 {  
 22 X <- ddply(data,var, function(df) c(prob=nrow(df)))  
 23 X$prob <- X$prob/sum(X$prob)  
 24 }  
 25  
 26 result[[length(result)+1]] <- X  
 27 }  
 28  
 29 return(result)  
 30 }

这个函数有两个参数：图g和数据集data。在第3行中，它反转了所有边的方向，以便用程序包graph中的函数adj()找出每一个变量的父节点。理论上讲，反转图形并没有什么稀奇；这么做只是为了方便找出父节点。

在第6行中，函数开始分别学习每一个变量，正如我们在之前的章节看到的，我们处理了2个问题，一个是变量没有父节点（因此我们计算了变量的边缘分布）的问题，另一个是变量有父节点（因此我们计算了条件概率表）的问题。

在第11行中，对于每一个变量和父节点的值，我们计算P(var,pa(var))频度（更准确的说应该是计数）。在第12行中，我们对P(pa(var))做同样的操作。

最后，从第13行到第18行，我们使用贝叶斯公式得到条件概率表，并把计算转化为概率（对于本例，也是频度）。第22行和第23行执行同样的操作生成边缘概率表。

每个变量的结果都保存在名为result列中！

我们用这个函数，以及两个数据集，看到了同样的结果，并进行分析。

### 应用

首先，加载并运行之前的R代码，然后执行下列代码：

learn(g0, data0)

结果如下：

[[1]]  
 x prob  
1 a 0.5   
2 b 0.5  
[[2]]  
 y x prob  
1 t a 0.5  
2 t b 0.5  
3 u a 0.5  
4 u b 0.5  
[[3]]  
 z y prob  
1 c t 0.5  
2 c u 0.5  
3 d t 0.5  
4 d u 0.5

这个结果表明，。观察数据集，我们看到变量有相同数量的和。结果没问题！

另外的表是和。注意条件概率不是直接概率。这些表需要根据父节点的值进行理解。 例如，和。很明显，二者的和为1。

现在，让我们在Nursery数据集上使用learn函数，看看结果如何：

learn(g1, data1)

为了简化输出，我们只给出部分变量，首列为class:

class prob  
1 not\_recom 3.333591e-01  
2 priority 3.291921e-01  
3 recommend 7.716645e-05  
4 spec\_prior 3.120611e-01  
5 very\_recom 2.531059e-02

这是边缘概率表，和之前从图中得到的一样。总和是1，而且我们看到有些概率值比其它概率值要高。这个表会用在专家系统中，进而推出结论。当然，我们的模型非常简单，而更加实际的模型会拥有不同的图模型和值。我们鼓励读者修改图g1来测试不同的选择。

如果我们看一下finance变量，我们有下表：

finance class prob  
1 convenient not\_recom 0.5000000  
2 convenient priority 0.5260197  
3 convenient recommend 1.0000000  
4 convenient spec\_prior 0.4589515  
5 convenient very\_recom 0.6646341  
6 inconv not\_recom 0.5000000  
7 inconv priority 0.4739803  
8 inconv spec\_prior 0.5410485  
9 inconv very\_recom 0.3353659

这个表比之前看到的都要大，但是也正常有效。然而，有一个小问题，最大似然估计过程并不是贝叶斯过程，而只是一个频率主义的过程。它在多数情形下都会有效，但是有时我们可能会有问题。这里，第3行中，我们看到。

虽然概率等于1并不是一个问题，但是很烦人。这是因为我们在数据集中只有一个特定组合的例子，这会给出极端的结果。这个结果并不是期望的结果，因为我们希望可以覆盖所有可能的场景，而不要落入某一个概率为1的独特场景中。

我们会在后面看到，在许多情况下给模型的所有参数添加先验概率分布会很有意思。这样可以避免它们带有概率为0或1的情型，而且可以发掘出尽可能多的场景。

## 学习隐含变量——期望最大化算法

本章的最后一部分介绍一个最重要的算法。这个算法会在本书中多次使用。这是一个学习隐含变量（即有些变量观察不到）概率模型的非常通用的算法。包含隐含变量的模型有时也叫做隐变量模型（latent variable models）。 期望最大化算法是解决这一类问题的方案，并且在概率图模型上也有很好的表现。

多数情况下，当我们想学习模型的参数时，我们会写一个目标函数，例如似然函数，我们的目的是找出可以最大化函数的参数。通常，我们可以只用一个黑盒数值优化函数，计算给定函数的相关参数。然而，在多事情况下，这会比较难以驾驭，而且容易导致数值错误（因为CPU的内部估计误差）。所有这个通常不是一个很好的方案。

我们力图使用优化问题的特异性（除了图模型对联合概率分布所做的假设）来改善计算流程，使其更快更可靠。

对于找到图模型的最优参数问题，期望最大化算法是一种非常优雅的方案。它也可以用到许多类型的模型中。

### 隐变量

隐变量可以用在所有模型中，以便引入一个简化层，或者分离概念，或者给模型定义一些层级。例如，我们可以观察变量之间的特定关系，我们并不让这些变量相互依赖，相反，假定有其他的隐含变量导出它们，而且依赖关系通过更高层的变量实现。

这种自顶向下的方式可以生成更简单的模型，如下图所示。这个模型已经很复杂了，不是吗？

但是如果我们添加一下隐含变量（在下图中用希腊字母表示），模型会变得非常简单，且易于处理和理解。问题是我们并没有数据评估隐含变量的概率分布。这就需要期望最大化算法了。

我们还可以添加更多的隐含变量，以便使用不同的父节点对变量归类，如下图所示：

## 期望最大化的算法原理

因为隐变量观察不到，这类模型的似然函数是边缘分布。我们需要对隐含变量求和。边缘化会生成变量之间的依赖关系，是问题变得难以解决。

期望最大化算法通常会在给定分布的情况下，使用期望值填补缺失数据，来处理问题。当我们不断的迭代这一过程，它会收敛到最大似然函数。这个填补策略是通过给定当前变量集合以及可观测的变量，计算隐含变量的后验概率分布而实现。这就是E-步骤（E代表期望）。在M-步骤，M代表最大化，我们会调整模型参数，并在新的E-步骤中再次迭代。这个过程会一直持续，直到我们看到参数收敛，或者似然率的增长收敛。而且，期望最大化算法可以确定每一次EM步骤后模型的似然率不会下降。这意味着当似然率只增长了很小的量时，我们就可以说算法收敛到（局部）最大值，算法可以停止了。增长率的多少依赖于应用本身，但是当似然率的增长不超过或，算法也可以停止。这只是经验规则。针对具体的案例，读者可以尝试绘制似然率曲线来更好的理解期望最大化算法的行为。

### 期望最大化算法推导

假设我们有数据集，包含个（独立同分布）个点，图模型的参数为。现在，如果还不理解是什么，它其实是一个包含图模型中每个变量所有参数的变量。由于写出这个集合会很冗长，我们可以简单的把它看作是一个高维变量。

设，。图模型的似然率有如下的定义：

在完备情况下，即所有变量都可以被观测到，似然函数可以有如下分解:

$$
\begin{align}
L(\theta) & = log \; p (D \mid \theta) \notag\\
& = \sum\_{i=1}^N log \;p (x\_i \mid \theta) \notag\\
& = \sum\_{i=1}^N \sum\_{j=1}^K log \;p (x\_i^{(j)} \mid pa(x\_i^{(j)}), \theta\_i) \notag\\
& = \sum\_{i=1}^K L\_i(\theta\_i)\notag\\
\end{align}
$$

我们再次看到了之前对数似然率的结果。这个结果可以再次写成每个变量（即图中的节点）的局部对数似然率加和的形式。

但是，当存在隐含变量的时候，我们的结果就不再优美。设可观测变量为，隐含变量为。模型的对数似然率可以写成：

这里，是所有变量的集合。我们的主要问题是：函数内部的求和并不容易计算。事实上，为了获取似然函数，我们需要边缘化掉隐含变量。有了这个求和，我们就可能让内部的所有变量都相互依赖。因此，由于使用图模型，我们会舍弃优良分解的所有好处。最终，计算会变得困难。

但是如果使用隐含变量上任意分布, 它可以定义出对数似然函数的下界，让我们看看原因：

$$\begin{align}
L(\theta) & = log \;\sum\_y p(x,y \mid \theta) \notag\\
& = log \;\sum\_y q(y)\frac{p (x,y \mid \theta)}{q(y)} \notag\\
& \geq \sum\_y q(y) log \frac{p (x,y \mid \theta)}{q(y)} \notag\\
& = \sum\_y q(y) log \; p(x,y \mid \theta) -\sum\_y q(y) log \; q(y)\notag\\
& = F(q,\theta)\notag\\
\end{align}$$

再逐行解释一下其中的原因：

1. 这是上对数似然率的标准定义，其中我们边缘化掉隐变量。
2. 这里我们在分子和分母中引入，以便它们可以相互抵消。
3. 基于这个原因，我们可以使用詹森不等式来或者下界。公式的右边就是的下界。
4. 这个下界可以进一步简化。最右边的项独立于和。

最终，新的函数 是对数似然率的下界。

期望最大化算法的的运行方式是通过调整两个步骤中的优化来完成：

* **E-步骤**:
* **M-步骤**:

这个算法通常会用参数的随机集合来做初始化。

算法首先在给定当前参数集合下找出隐含变量的一个新的边缘分布，然后使用之前的分布找出参数的最大似然估计。事实上，在步骤1的E-步骤中，的最大值可通过设定得到。新增，下界变成了等式：

这个结果非常重要，因为它保证了似然率在每一步中只能增长或保持不变。因此这个结果意味着，使用当前的参数，我们可以推出给定其他观察变量下的分布。这个步骤可以使用之前的章节中的任何推理算法。而且，这个过程会创建给定当前参数下期望的观察结果全集。

M-步骤中的最大值可以通过最大化之前推导中第4行的第一项实现，即分布下期望的对数似然率。

所有在期望最大化算法的开始介绍中，我们有。E-步骤也不会更改。我们知道E-步骤不会降低似然率。因此对数似然率只会增长或保持不变。实际应用中，我们通常会看到对数似然率的收敛。当增长率很小时，我们可以停止算法，认为当前的方案就是优良方案。

### 对图模型使用期望最大化算法

正如已经看到的，我们会在实际问题中使用带有离散变量的图模型。例如，假设图中的某个地方有两个变量和，使得是的父节点。因此我们有节点的局部分布。

回忆一下，最大似然估计的计算如下：

这是我们在用R之前所看到的和实现了的。到目前为止，没有什么新的知识了。之前，我们使用ddply函数高效的在一次调用中执行计算。读者也可以使用aggregate函数来得到同样的结果。

但是这个公式只会在和完全可观测的情况下有效！这里二者并非如此。而使用期望最大化算法克服这个困难很简单。

M-步骤就是处理这种情况：

但是如何获取这两个概率分布？我们在E-步骤中使用观测变量和选定的推理算法得到它们。至于A和B中使用的参数，它们是期望最大化算法之前步骤的参数。

最后，我们回忆一下期望最大化算法的所有步骤：

1. 用随机参数初始化图形模型。确保分布的总和为1。随机参数似乎能够提供比均匀分布更好的结果，但这只是给你一个实用的提示。
2. 完成以下步骤，直到对数似然率收敛：
   * **E-步骤**: 使用选定的推理算法计算所有隐藏的变量的后验分布。这是分布。
   * **M-步骤**: 使用前面推断出的分布计算图模型的新参数集。
   * 更新对数似然率，检查是否收敛。通常需要检查当前似然率和上一步似然率的差异是否小于预定于的阈值。

因此M-步骤似乎可以使用期望的概率实现隐含变量可观测。

## 小结

在本章中，我们看到了如何使用最大似然估计计算的图模型的参数。读者应该看到，这种方法不是贝叶斯的，可以通过设置图模型参数的先验分布来改善。这个过程会用到更多的领域知识，并帮助获得更好的估计。

当数据无法完全观测到，有些变量是隐藏的，我们学会了如何使用非常强大的期望最大化算法。我们还看到了一个完全可观测的图形上，R语言学习算法的完整实现。

现在，我们鼓励读者使用本章中的思想，扩展和改进自己的学习算法。机器学习最重要的需求是专注于算法无法生效的场景。任何算法都会从数据集中提取一些信息。然而，当关注于算法的错误以及它不生效的地方，我们会发现数据中的价值。

在下一章中，我们将看几个简单的，但功能强大的，可以表示为图模型的贝叶斯模型。我们会看到，他们中的一些可以高度优化以便推理和学习。我们还将使用高斯混合模型，探讨期望最大化算法的应用，进而找出数据中的簇。

# 第4章 贝叶斯建模——基础模型

在学习完如何表示图模型，如何计算后验分布，如何用最大似然估计使用参数，以及如何在数据缺失和存在隐含变量下学习相同的模型时，我们要深入研究使用贝叶斯范式来进行建模的问题。在本章中，我们会看到一些简单的问题并不容易建模和计算，进而需要特定的解决方案。首先，推理是一个困难的问题，联结树算法只能解决特定的问题。其次，模型的表示目前都是基于离散变量的。

在本章中，我们将介绍简单，但功能强大的贝叶斯模型，并展示如何作为概率图模型表示它。我们会看到使用不同的技术，它们的参数可以有效地学习出来，以及如何以最有效的方式在这些模型上进行推理。我们将看到这些算法是可以适应这些模型，同时考虑到每个特异性。

首先，我们开始使用带有连续值的变量，即可以取任意数值的随机变量，而不仅仅是有限数量的离散数值。

我们会看到一些简单的模型，它们是复杂解决方案的基本构成。这些模型是基本的模型，我们将从非常简单的事情逐渐过渡到更复杂的问题，如高斯混合模型。所有这些模型都在被广泛使用，并有很好的贝叶斯表示。我们会在这一章逐步介绍。

更具体的，我们会对如下模型感兴趣：

* 朴素贝叶斯 模型及其扩展，主要用于分类
* Beta-二项分布模型，这也是最基础的模型
* 高斯混合模型，最常用的聚类模型之一

## 朴素贝叶斯模型

朴素贝叶斯模型机器学习中最出名的分类模型。虽然看上去很简单，但是这个模型非常强大而且只需要很少的精力就可以输出很好的结果。当然，在考虑分类问题的时候我们不应该只局限于一个模型，例如朴素贝叶斯模型，我们还要尝试更多的模型，看看对于特定的数据集哪一种模型是最好的。

分类是机器学习中的一类重要问题，它可以定义为一种关联观察结果和具体类别的任务。假设我们有包含个变量的数据集，并给每一个数据点指认一个类别。这个类别可以是，，或等。我们会看到有时考虑二元分类问题（问题只涉及两个分类）要更简单。但是大多数分类问题都有超过两种类别。

例如，给定生理特征，我们可以把动物分成哺乳动物或爬行动物。给定邮件中用到的 单词，我们可以把邮件分成垃圾邮件或正常邮件。给定信用记录和其他财务数据F，我们可以把客户分成贷款可信或者贷款不可信。

尝试接下来的小例子，看看分类的显式展示。

Sigma <- matrix(c(10, 3, 3, 2), 2, 2)  
x1 <- mvrnorm(100, c(1, 2), Sigma)  
x2 <- mvrnorm(100, c(-1, -2), Sigma)  
plot(x1, col = 2, xlim = c(-5, 5), ylim = c(-5, 5))  
points(x2, col = 3)

这个例子展示了两个变量的二元分类问题。两个变量通过轴和轴表示。问题似乎很明显，事实上并非如此，因为红色类别和绿色类别的分界面定义并不清楚。这就是典型的现实世界的例子。

在这个例子中，我们依然可以在中间画一条直线来分成两个类别。但是有时问题并不明显，一条线也不够用。当一条直线可以分出两种类别时，我们称之为**线性分类问题**。当我们需要一条曲线来区分时，我们称之为**非线性分类问题**。

我们评估分类器质量的方法是看错误率。我们希望最低的错误率，即每次分类器预测数据的类别都应该是正确的。然而，根据分类问题的不同，错误可能意味着不同的结果。例如，在医疗分类问题中，把病人分成得病类别要比分成健康类别或带有未知检出疾病通常要危险小些。

很明显 ，我们希望分类器尽可能准确，构建分类器的通用规则是完全关注于棘手的例子。

### 表示

朴素贝叶斯模型是一种概率分类模型，其中个随机变量作为特征，1个随机变量作为类别变量。模型的主要（强）假设是，**给定类别，特征都是独立的**。这个假设看上去很强，而且会给出非常好的结果。

朴素贝叶斯模型的联合概率分布是：

它可以使用下列图模型表示：

事实上，这是一个非常简单的模型，你可以从图中看到为什么知道类别就会使所有特征变量相互独立。

因此，借助贝叶斯规则，给定新的数据点，我们可以计算最有可能的类别，具体如下：

为了使问题简单，我们会把所有变量和类别变量当做二元变量。但是理论上多元变量在这个问题中处理起来也是一样的。实际上，如果你考虑连续变量，这个模型的理论也是类似的。例如，对于实值特征，我们可以考虑高斯分布，并有：

这里，表示高斯分布。

当特征是二元的，结果也是一样的，除了特征使用伯努利分布：

这里，取值为，是给定类别下的概率。

### 学习朴素贝叶斯模型

朴素贝叶斯模型的学习非常建简单。回忆一下第3章的内容，*参数学习*，我们很容易得到，对于每一个，在二元特征和二元分类变量下，，其中是当时变量的计数，是类别1的计数。

至于类别变量，跟家简单：，其中是数据点的总个数。

原因和上一章一样。为了更好的理解，我们需要写出最大似然估计模型。对于一个数据点，概率是：

知道在二元分类器中类别只可以取值，我们有：

因此对数似然率是：

为了最大化这个函数，我们看到可以分别优化每一个项，以便获得每一个参数的简单形式。因此，很自然的，它输出了和通常图模型一样的结果。

除了手动实现模型，我们会使用名为e1071的程序包。如果你还没有安装，可以使用下列代码来完成：

install.packages("e1071")  
library(e1071)

这个程序包提供了朴素贝叶斯模型的完整实现。我们可以加载数据并查看结果：

data(iris)  
model <- naiveBayes(Species~.,data=iris)  
Naive Bayes Classifier for Discrete Predictors  
Call:  
naiveBayes.default(x = X, y = Y, laplace = laplace)  
A-priori probabilities:  
Y  
 setosa versicolor virginica  
 0.3333333 0.3333333 0.3333333  
   
 Conditional probabilities:  
 Sepal.Length  
Y [,1] [,2]  
 setosa 5.006 0.3524897  
 versicolor 5.936 0.5161711  
 virginica 6.588 0.6358796  
 Sepal.Width  
Y [,1] [,2]  
 setosa 3.428 0.3790644  
 versicolor 2.770 0.3137983  
 virginica 2.974 0.3224966  
 Petal.Length  
Y [,1] [,2]  
 setosa 1.462 0.1736640  
 versicolor 4.260 0.4699110  
 virginica 5.552 0.5518947  
 Petal.Width  
Y [,1] [,2]  
 setosa 0.246 0.1053856  
 versicolor 1.326 0.1977527  
 virginica 2.026 0.2746501

我们再解释一下这个例子。参数laplace控制数据的拉普拉斯平滑，以便处理数据并不完美平衡或数据集有问题的情形。我们后边会再回到这个问题，但是这是大多数分类问题都必须处理的一个主要问题。

使用这个模型并尝试预测（或推断）类别，我们有：

p <- predict(model, iris)  
hitrate <- sum(p == iris$Species)/nrow(iris)

我们得到了 0.96的命中率，即96%的数据点都被正确分类。这还不错，但是需要记得我们使用了训练集得到这个百分比。你不能只是用训练集中的点来评估分类模型的真正能力。理想情况下，我们应该把数据集分成两部分，假设我们使用的数据进行测试，的数据进行模型训练。这种分割需要以随机的方式完成：

ni <- sample(1:nrow(iris), 2 \* nrow(iris)/3)  
no <- setdiff(1:nrow(iris), ni)  
model <- naiveBayes(Species ~ ., data = iris[ni, ])  
p <- predict(model, iris[no, ])

这里，ni和no是从初始数据集随机得到的数据点索引列表。

### 完全贝叶斯的朴素贝叶斯模型

尽管名字叫贝叶斯，但是这个模型并不是真正的贝叶斯模型。要使模型具有完整意义的贝叶斯属性，我们应该在参数上建立先验概率。 在朴素贝叶斯模型中，参数是指类别变量和特征变量。这些参数使用最大似然法进行估计。但是如果数据集不够平衡怎么办？如果针对特定的类别数据点不够多，参数如何处理？我们会得到很差的估计，最坏的情况是异常表示的参数权重都是0。这显然不是我们期望的，因为结果完全是错误的。我们给正常特征和分类分配了太多的权重，而对异常表示的参数没有分配权重。

这个问题叫做**过拟合**（over-fitting）。过拟合的解决可以使用贝叶斯方法，并在模型中包含额外信息，比如“如果数据没有表示出来，可以假设它们有很小的概率，而不是零概率”。

一种简便易行的方法是使用模型参数的先验概率，并使用贝叶斯的方式开发模型。让我们做一些假设来简化计算。首先，我们假设所有恶天恒办理 First of all, we will assume that all the feature variables have the same nite number values. We will call this number S. You can easily generalize this to any number of values for each feature, but in our presentation it makes things simpler. Then we will assume we can use a factored prior on the θ feature parameters, as follows: p(θ)=p(π)∏N ∏C p(θic) i=1 c=1 Here, θ represents all the parameters. In order to make it clear, we use the following notation:

* θ represents all the feature parameters and π the class parameters
* θi represents all the parameters of the variable i—that is, the parameters of the conditional distribution p(Xi | C)
* θic represents all the parameters of the variable i—that is, the parameter distribution p(Xi | C = c)
* θ(s) represents the parameter of the probability p(X = s| C = c)—that is, ic i because Xi is a discrete multinomial variable (in fact it's more accurate to say categorical than multinomial here), p ( X = s | C = c ) = θ (s ) i ic

And because we just mentioned multinomial distribution, it's important to note that the Dirichlet distribution is also the conjugate distribution for the multinomial (and the categorical) distribution. If we consider all the θ(s) to be random variables and ic no longer just simple parameters, we need to give them a probability distribution a priori. We will assume they are Dirichlet distributed for two reasons:

* The Dirichlet distribution is a distribution over a vector of values such that their sum is 1, which corresponds to the well-known constraint that ∑S θ(s) =1. So far, nothing new. s=1 ic
* The Dirichlet distribution being the conjugate prior for the multinomial distribution, this means that, if a data point has a categorical or multinomial distribution and the prior distribution on the parameters is a Dirichlet (as in our case), then the posterior distribution on the parameters is also a Dirichlet. And this will simplify all of our computations.

In fact, contumacy is a very powerful tool in Bayesian data analysis. In practice it works as follows:

* Let's say that α is the concentration parameter—that is, the parameter of the Dirichlet distribution Dir(⍺).
* So we assume that the θ's are distributed Dirichlet—that is p(θic | ⍺) = Dir(⍺).
* And, of course, we know that our feature variables have a categorical or multinomial distribution.
* Therefore the posterior probability of the parameters of the distribution of Xi after counting data (as we did before), will be a Dirichlet Dir(Ni +α), where Ni is the counts we did before! It's as simple as that, thanks to the conjugacy.

So nally, if we want to incorporate the Dirichlet prior into our computations, the posteriors of the parameters for the class variable are: πc=Nc+αc whereα0=∑αc. N +α0 c And the prior distribution of π is a Dir(⍺) where ⍺ = (⍺1, ...,⍺c). For the parameters of the feature variables, the solution is exactly the same: θ(s) = Nic +βs ic Nc +β0 where β0 =∑s βs [ 105 ]

Bayesian Modeling – Basic Models And the prior distribution of θic is a Dir(β), where β = (β1, ..., βs). Wait! Is it really as simple as this? Well, yes it is, thanks to the conjugacy in this Bayesian model. If you look carefully at the formulas, you will see that none of the π and θ(s) can be equal to zero now because of the values of α and β. Indeed, in c ic the de nition of the Dirichlet distribution, it is required that the parameters of the Dirichlet should be strictly positive. So the last problem we need to solve is choosing a value for α and β. One common choice is to take 1 for all of them. In terms of Dirichlet distributions, it means we choose a uniform prior for all the parameters of the class and feature variables. It means we will allow our parameters to take any value with equal probability except of course 0. Choosing different values for α and β will lead to different form results. We can try to promote certain values by pushing the Dirichlet distribution in one direction or another; or, on the contrary, we can try to keep all parameters with similar values. If you choose 1 for the Dirichlet parameter, you will obtain something called Laplace smoothing, which we saw before in the naiveBayes function of the e1071 R package. Sometimes, it is also called a pseudo-count because it could be seen as arti cially adding one example of any case to your dataset. But the Dirichlet prior is not the only possible prior we can use. In the case of binary variables, another distribution of interest is the Beta distribution. In the next section we will present more formally the Beta-Binomial model and see its relation to the Dirichlet-Multinomial model we just saw. We will see that the results are similar and also how to play with the parameters of the Beta distribution in order to describe different types of prior for our class and feature variables.

## Beta-Binomial

The Beta-Binomial prior is another example and a well-known model where we set a prior distribution on the parameter of the distribution of interest. Here we are going to use a binomial distribution with a θ parameter. The θ parameter can be seen as a probability that an event will occur or not, or a proportion of the positive events in a sequence of experiments. Therefore, the parameter θ takes values in [0,1]. Let's rst review the Binomial distribution with a simple example: let's say we have a coin and we want to know if the coin is impartial or not when we play the heads or tails game. The game is to toss the coin N times and try to estimate what is the probability θ of obtaining a head. This problem is very important because it is the basis of many other models. You can replace the game of heads or tails with the result of experimentation (positive or negative), the result of a poll (yes or no), or any other binary answer. [ 106 ]

Chapter 4 Historically, this model has been studied by Thomas Bayes and generalized by Laplace, thus giving birth to the Bayes rule (or more formally the Bayes-Laplace, as we saw in Chapter 1, Probabilistic Reasoning). In this problem we will follow again a Bayesian approach: we need a prior distribution on the parameter θ, a likelihood of the data given the parameter p(x | θ), and we will compute the posterior distribution p(θ | x). If we want to be complete, we can also compute the predictive posterior distribution p(x | θ, x), which is the distribution of a new data point (a new toss) given parameters and the previous experiments. When we assume that all the observations (that is, the results of each toss) are independently and identically distributed we can again write that: p(D|θ)=∏N p(xi |θ) i=1 Here, D = {x1, ..., xN} is the dataset. Is this assumption true? From a theoretical point of view, it's true because: (1) every time we toss a coin, the previous toss has no in uence on the new one and (2) we use the same coin for all the tosses, so the parameter θ doesn't change. Therefore the distribution of heads and tails is the same. But is it true in real life? If we assume that each toss can microscopically change the air in the room and that each toss will rip off a few atoms of metal from the coin, then the distribution is certainly not independent and certainly not identical. But in fact the acute reader will have understood that those effects are so negligible that they have absolutely no in uence on the results. Maybe one should toss the coin a few billion times to start seeing a difference. However, when designing such an experiment, one has to be careful about the conditions of the experimentation and be sure that the data is indeed identically and independently distributed. For example, if the experiment is a poll and one puts the same question to people next to each other, one after the other, it is very likely that the second person will be in uenced by the answer of the rst one. And therefore the data will not be iid anymore. Now we can star solving our Bayesian problem by assuming distributions for all the required components. [ 107 ]

Bayesian Modeling – Basic Models The Bernoulli distribution is a probability distribution that gives the probability θ to a random variable when it takes the value 1 and (1 –θ) if it takes the value 0. We say that x ~ Ber(θ), that is p(x | θ) = θx(1 – θ)1-x with x ε {0,1}. If we repeat the same Bernoulli experiment many times (that is, if we toss the coin many times), we obtain the dataset D = {x1, ..., xN} with this probability distribution: ∏N 1−x 1−x 1−x p(D|θ)= p(x |θ)=θx1 (1−θ) 1 θx2 (1−θ) 2 ...θxN (1−θ) N Because of the iid assumption and the fact that the product is commutative, and if we call N1 the number of heads and N0 the number of tails, we can rewrite this likelihood by: p(D|θ)=θN1 (1−θ)N0 From now on, we can take the log of this expression: LL(θ)=logθN1 (1−θ)N0 =N1logθ+N0log(1−θ) Before going further, it's interesting to understand why we are always using logarithms in our computations. The rst reason is historical: probabilities are numbers between 0 and 1. Multiplying two probabilities is essential when computing the likelihood of iid data. In fact, in such a likelihood, it's not unusual to multiply hundreds or thousands of those probabilities. On early computers, multiplication was very slow compared to addition. Because log(a.b)= log(a)+log(b), it was useful to rst transform all the data into logarithms and then add them. It was generally faster. Nowadays, this is not true anymore and usually multiplication can be (almost) as fast as addition. The cost of computing the logarithm can sometimes overwhelm the small gain of just doing additions instead of multiplications. The second reason is that, when we multiply numbers between 0 and 1, the result tends to decrease and be smaller and smaller. This time, even modern computers have a limited capacity to represent numbers and moreover, because real values are discretized (usually following the IEEE-754 norm), the accuracy of computations suffers enormously and errors are accumulated throughout the computations, leading to very inaccurate if not false results. However, with logarithms, numbers between -∞ and 0 are added together, making the computations more accurate. Small values are now contributing a lot to the results, making computations accurate. [ 108 ] i=1 i

Chapter 4 Usually we take the negative log-likelihood, and we only have to deal with positive numbers. The other reason is that, when one wants to maximize the likelihood, one can also minimize (towards zero) the negative log-likelihood. This is an equivalent problem. And a lot of optimization algorithms try to nd the zeros of functions. Therefore implementation is simpler. To illustrate this, just plot the following in R:

x <- seq(0, 1, 0.05)  
plot(x, -log(x), t = "b", col = 1)

After this parenthesis, we are back to the problem of a sequence of tosses. So we assumed that the random variable representing the result of a toss has a Bernoulli distribution and we calculated the likelihood of a sequence of N tosses. Now let's consider the problem from a different angle and let's assume we toss the coin N times, N being known in advance. The question is now: what is the probability of obtaining N1 heads out of N tosses. [ 109 ]

Bayesian Modeling – Basic Models On average, the answer will depend on how the coin is biased. And the bias of the coin is known too; it is the parameter θ. If θ = 0.5, then N1 should rather be N2 . We want to have a probability for each possible value that could take N1 in the function of θ and N. In this case, it is important to remark the following: how many positive cases do we have? By positive cases, we mean, How many sequences of N tosses can provide N1 heads? On a small example with N=3 and N1 =2, we can have {HHT, HTH, THH}. That is three possibilities. In general, we want to know the number of combinations of r values out of n events with replacement. And this is equal to p(N|θ,N)=⎛N1+N0⎞θN (1−θ)N0 10⎜N⎟1 ⎛ n ⎞ = n! ⎜r⎟ r!(n−r)! . Now remember that each sequence is independent of the others and that the probability of two independent events is the sum of probability ⎝⎠ P(A ⋁ B) = P(A) + P(B). So nally, the probability of N =⎛n⎞ independent Bernoulli 1 ⎜r⎟ events of parameter θ is: ⎝ ⎠ This probability distribution is very well-known and is called the Binomial distribution. In fact, the Binomial distribution is usually de ned with two parameters: N, the total number of tosses, and θ. It is commonly written as: p(n|θ,N)=⎛N⎞θn (1−θ)N−n ⎜n⎟ ⎝1⎠ ⎝⎠ In R, the binomial distribution is provided by default in the language and it can be used with the following functions:

* dbinom: Density
* pbinom: Cumulative
* qbinom: Quantile
* rbinom: Random generation

We can illustrate the distribution of the binomial in R with the little program that follows:

x <- seq(1, 20)  
plot(x, dbinom(x, 20, 0.5), t = "b", col = 1, ylim = c(0, 0.3))  
lines(x, dbinom(x, 20, 0.3), t = "b", col = 2)  
lines(x, dbinom(x, 20, 0.1), t = "b", col = 3)

We show three different distributions with the parameter θ varying from 0.1 to 0.5. When θ is small, obviously the probability of having many positive outcomes quickly decreases. When θ is 0.5, the black curve shows that the probability of 50% positive outcomes is obviously the highest.

### 先验分布

The next question that arises is, What prior distribution should we use as a prior on θ? The beta distribution is a very common choice and has the nice property of being a conjugate to the Binomial and Bernoulli distributions. In fact the beta distribution has a very nice form, similar to the Binomial and Bernoulli distributions, as follows: p(θ |α,β)∝θα−1 (1−θ)β−1 [ 111 ]

Bayesian Modeling – Basic Models We will add the normalization constant later. The good thing about the Beta is that its domain, that is, the value that θ can take, is [0,1]. And therefore θ from the Beta distribution can also be interpreted as a proportion or a probability and used as a parameter in the Binomial or Bernoulli distributions. This makes the Beta distribution a perfect candidate for the prior distribution. To complete the formula we have to realize that this distribution is a density and the integral over its domain must be one. Therefore it is usual to write: θα−1 (1−θ)β−1 p(θ |α,β)= ∫1 xα−1 (1−x)β−1 dx 0 And the integral at the denominator is known as the Beta function. In general we can simply write the density as: p(θ |α,β)= 1 θα−1 (1−θ)β−1 = Γ(α +β) θα−1 (1−θ)β−1 Beta(α,β) Γ(α)Γ(β) Here, the Gamma function is de ned as: Γ(x)= ∫∞ exp(−t)tx−1dt 0 When x is an integer Γ(x)=(c−1)!

### 带有共轭属性的后验分布

Now we need to combine the Binomial distribution with the Beta prior to obtain the posterior distribution. The posterior distribution is obtained by applying the Bayes rule as usual: p(θ|N,n,α,β)= p(n|N,θ)p(θ|N,α,β)∝p(n|N,θ)p(θ|N,α,β) p(n| N,α,β) [ 112 ]

Chapter 4 And nally, by replacing each distribution with its analytical form, we obtain: p(n| N,θ)p(θ | N,α,β)= N! θn (1−θ)N−n ×Γ(α)Γ(β)θα−1 (1−θ)β−1 n!(N −n)! Γ(α +β) And this is simply proportional to: θn (1−θ)N−n ×θα−1 (1−θ)β−1 =θn+α−1 (1−θ)N−n+β−1 And in fact the last form is exactly the same as the initial form of the Beta distribution. It means that our posterior distribution on θ is also Beta-distributed. Therefore, we have found our posterior and we can resume this calculus as follows: If n follows a Binomial distribution Binomial(θ, N) and the prior distribution over θ is Beta(α, β), then the posterior distribution on θ will also be a Beta distribution Beta(⍺ + n, β+ N – n). In this case, thanks to the conjugacy property, we've made a very ef cient posterior computation, which boils down to a few additions only. The notion of conjugacy is extremely important in Bayesian reasoning mainly for this property.

### 如何选取Beta参数的值

This will depend on what type of information we want to include in the model. For example, we might decide that every value of θ is a priori acceptable and we want to give the same importance to all of them. This is what we did with the Dirichlet distribution in the previous section when adding pseudo-counts of 1. With the Beta distribution, we can have a uniform distribution with the parameters Beta(1,1). But we can also try to give more importance to extreme values close to 0 or 1, with for example a Beta(0.5,0.5) distribution. On the other hand, to force θ to stay more centered around 0.5, we can use Beta(2,2), Beta(3,3). The higher the values, the more probability mass is given to the center. The next code in R shows different distributions with different values:

x <- seq(0, 1, length = 100)  
par(mfrow = c(2, 2))  
param <- list(list(c(2, 1), c(4, 2), c(6, 3), c(8, 4)), list(c(2, 2), c(3, 2),   
 c(4, 2), c(5, 2)), list(c(1, 1), c(2, 2), c(3, 3), c(4, 4)), list(c(0.5,   
 0.5), c(0.5, 1), c(0.8, 0.8)))  
for (p in param) {  
 c <- 1  
 leg <- character(0)  
 fill <- integer(0)  
 plot(0, 0, type = "n", xlim = c(0, 1), ylim = c(0, 4))  
 for (v in p) {  
 lines(x, dbeta(x, v[1], v[2]), col = c)  
 leg <- c(leg, paste0("Beta(", v[1], ",", v[2], ")"))  
 fill <- c(fill, c)  
 c <- c + 1  
 }  
 legend(0.65, 4, leg, fill, bty = "n")  
}

## 高斯混合模型

The Gaussian mixture model is the rst example of a latent variable model. Latent variables are also called hidden variables and are variables that are present in the model but are never observed. The notion of using unobserved variables can be surprising at rst because we might wonder how to estimate the parameters of the distribution of such a variable. In fact, we might wonder what the real meaning of such a latent variable is. For example, let's say we observe data represented by a group of random variables. This data tends to group into clusters, aggregating together depending on their underlying meaning. For example, we could observe physiological traits from animals and group those data points by species such as dogs, cats, cows, and so on. If we think in terms of generating models, then we can say that, by choosing a group such as for example pony, we will observe features that are speci c to this group and not to another group such as cats. However, none of the physiological variables carry an explicit reference to the fact that they come from the pony group or the cat group. This distinction only exists because we want to group things together but they're not part of the real world. However, it helps a lot to group data like this, by pony or cat features, to understand animals in general. This is the grouping we want to do with a latent variable. Using latent variables in a model can be problematic at rst because there is no data to estimate their distribution. However, we saw before that algorithms such as EM can be very helpful when it comes to solving this task. Moreover, we can simplify the model by introducing some conditional independence between features or a hierarchy between variables, thus making the model easier to interpret or easier to compute. The Gaussian mixture model is a latent variable mainly used for density estimation problems, where, roughly speaking, the main assumption is that a random process will, according to a multinomial distribution, choose a Gaussian at random and then (according to this selected Gaussian distribution) choose a data point at random. It is a very simple 2-step process. And it also simpli es the problem of estimating a complex dataset distribution. Instead of searching for a very complex distribution, the model estimates it with a set of simple Gaussian distributions, joined by the latent variable. It is similar to a divide-and-conquer approach.

### 定义

In this model we will call X the variable we can observe and we will call Z a multinomial random variable that is hidden. The model is de ned by the following probability distribution: p(X |Θ)=∑i p(Zi =1|πi )p(X |Zi =1,θi ) = ∑iπip(X|Zi =1,θi) Here, πi are the mixing proportions and p(X | Zi = 1, θi) are the mixture components. In this formula, because Z is a multinomial distribution, we have p(Zi | πi) = πi and Zi is the ith component of Z. Finally Θ is the set of all parameters of the model and θi are the parameters of the X variables. When X has a Gaussian distribution, we can write the preceding distribution as: p(X |Θ)=∑iπiN(X |θi =(μi,∑i )) Here, Σi are the covariance matrices of the variable X. If we expand the formula we obtain: graphical model and see the equivalence in it:

This represents the probability distribution P(X,Z) where, this time, the Z node is white to indicate that it is hidden or latent. If we marginalize out the variable Z from this model to obtain the distribution on X we will obtain exactly the previous formula. Also note that, in this model, X has a multivariate Gaussian distribution. [ 116 ] p(X |Θ)=∑iπ (2π) ∑ i m/2 1/2 ⎜ i 1 ⎛1 T∑−1 ⎞ i⎟ This result looks dense but it's time to draw the corresponding probabilistic exp⎝−2(x−μ ) i (x−μ )⎠ i

Chapter 4 From this distribution it is also simple to compute the posterior of Z given X. This is a value of interest: we want to know what the probability is of Z being in state i after observing Z; in other words, it gives us information about which distribution the observed variable comes from. Indeed, let's rst draw a mixture of three Gaussians in R with the following code. This time we will use the package called mixtools:

N <- 400  
X <- list(mvrnorm(N, c(1, 1), matrix(c(1, -0.5, -0.5, 1), 2, 2)/4), mvrnorm(N,   
 c(3, 3), matrix(c(2, 0.5, 0.5, 1), 2, 2)/4), mvrnorm(N, c(5, 5), matrix(c(1,   
 -0.5, -0.5, 4), 2, 2)/4))  
plot(0, 0, xlim = c(-1, 7), ylim = c(-1, 7), type = "n")  
for (i in 1:3) points(X[[i]], pch = 18 + i, col = 1 + i)

This little program simply generates three sets of data from a multivariate Gaussian distribution with two dimensions. It plots the three datasets on the same graph and as expected the points create three clusters of data, as shown in the next gure. If we regroup the three little datasets into a big one, one interesting problem would be to nd out the parameters of the three clusters. In this example, we have an ideal situation because we generated an equal number of points for each cluster. But in real applications, this will rarely be the case.

The posterior probability of the hidden variable Z can be written as p(Zi =1|X,Θ)= p(X|Zi =1,θi)p(Zi =1|πi) p(X |Θ) πiN(X,μi |∑i) ∑jπjN(X,μj |∑j) This is a simple application of the Bayes rule again. The next step is to estimate the parameters θ of the model, assuming again that the data is iid. If we call D = {xn} the dataset, we can, as before, write the log-likelihood of the model: = LL(Θ|D)=∑nlogp(xn |Θ) = nlog i πN(x |μ,∑) This log-likelihood is a bit hard to optimize and we will use an adequate optimization algorithm. As mentioned before, the fact that a variable is hidden will lead us to use the EM algorithm. From the previous example, we will assume that the variable Z has three states {z1,z2,z3} for each of the three Gaussian components. The only constraint in the Gaussian mixture model is that one has to assume the number of Gaussians beforehand. Other models exist where the number of Gaussian components is also a random variable and the learning algorithm will try to discover the most probable number of components while at the same time nding the mean and covariance matrix of each component. Here we use the same code as previously:

library(mixtools)  
N <- 400  
X <- list(mvrnorm(N, c(1, 1), matrix(c(1, -0.5, -0.5, 1), 2, 2)/4), mvrnorm(N,   
 c(3, 3), matrix(c(2, 0.5, 0.5, 1), 2, 2)/4), mvrnorm(N, c(5, 5), matrix(c(1,   
 -0.5, -0.5, 4), 2, 2)/4))  
x <- do.call(rbind, X) # transform X into a matrix  
model2 <- mvnormalmixEM(x, verb = TRUE)  
model3 <- mvnormalmixEM(x, k = 3, verb = TRUE)

Chapter 4 It will take some time to compute the results. The parameter verb=TRUE displays the result of each iteration of the EM algorithm. What is interesting to see is the log- likelihood. In the rst case (model2), the log-likelihood will go from approximately 3711 to -3684 in 27 steps. Your results might be different because remember that we generate the dataset at random using mvrnorm. The problem with model2 is that the number of components is taken by default to be 2: you can perform help(mvnormalmixEM) in R to see the parameter k. And we know we have three components in this mixture. However, model3 has a number of components k=3, closer to the real dataset, and obviously the log-likelihood will be closer to 3. It goes from 3,996 to only 3,305 in 41 iterations (again it might be slightly different on your computer). So it seems the convergence has been far better in the second case when we assume the correct number of components. We can now plot the log-likelihood evolution of the EM algorithm to understand the difference between the two models:

plot(model2, xlim = c(0, 50), ylim = c(-4000, -3000))  
par(new = T)  
plot(model, lt = 3, xlim = c(0, 50), ylim = c(-4000, -3000))

Note that, by xing the size of the graph, we can easily superimpose the two graphs. The dashed line is the graph corresponding to the model with three components. It is clear that the log-likelihood gets closer to zero during this algorithm. However, it takes more iterations to reach this result.

Bayesian Modeling – Basic Models By looking at the results in model3, we can have a better understanding of the model that has been found by the EM algorithm:

model3$lambda  
[1] 0.3358283 0.3342840 0.3298877

The proportions of each component are, as expected, very close to our initial proportions. You can change the number of points for each component and check again:

X <- list(mvrnorm(100, c(1, 1), matrix(c(1, -0.5, -0.5, 1), 2, 2)/4), mvrnorm(200,   
 c(3, 3), matrix(c(2, 0.5, 0.5, 1), 2, 2)/4), mvrnorm(300, c(5, 5), matrix(c(1,   
 -0.5, -0.5, 4), 2, 2)/4))  
x <- do.call(rbind, X)

And then let's rerun it with mixtools:

model3.2 <- mvnormalmixEM(x, k = 3, verb = TRUE)

We can see the log-likelihood going from approximately -1925 to -1691 in 84 iterations. But now the proportions are 0.3378457, 0.1651263, and 0.4970280 which indeed correspond to the proportions we initially set in our toy dataset. Again we can check the other parameters, and see they are similar to those we set up in our dataset. In a real-world application, we don't have any idea of the location and covariance of each component, of course. But this example shows that the EM algorithm usually converges to the desired values:

model3$mu  
[[1]]  
[1] 3.025684 3.031763  
[[2]]  
[1] 0.9854002 1.0289426  
[[3]]  
[1] 4.989129 5.076438

Now it's time to look at the result more graphically to really understand which components have been found by the EM algorithm. First of all, we plot model3 with the following command and display its three components with plot(model3, which=2):

Then we display, for comparison purposes, model2 and model3.2:

The following gure shows model3.2:

And now we conclude from our observations:

* model3 and model3.2 are extremely similar, as expected.
* model2, for which we chose to have only two components, seems to have made an acceptable choice with the components. Indeed the two bottom components have an almost similar orientation. So the algorithm converged toward a solution in which one Gaussian will embed the two bottom components, and another Gaussian will embed the top one, which has a different orientation. It is a good result.

## 小结

In this chapter we used the simple yet powerful Bayesian model, which has a representation as a probabilistic graphical model. We saw a Bayesian treatment of the over- tting problem with the use of priors, such as the Dirichlet-multinomial and the famous Beta-Binomial model. [ 122 ]

Chapter 4 The last section introduced another graphical model, which was around before the invention of probabilistic graphical models and is called the Gaussian mixture. It is a very important model to capture data coming from different subsets within the same model. And nally, we saw another application of the EM algorithm: learning such models and nding out the parameters of each Gaussian component. Of course, the Gaussian mixture is not the only latent variable model; in fact it represents a lot of Bayesian models and the probabilistic graphical model framework. In the next chapter, we will continue our study of inference algorithms for Bayesian models and probabilistic graphical models with the introduction of a new and very important family of algorithms: the sampling algorithm, also known as the Monte- Carlo algorithm. It is arguably one of the most important algorithms in the eld of machine learning because it allows the use of many types of model that were previously too complicated to use. [ 123 ]

# 第5章 近似推断

This chapter introduces a second class of inference algorithms, maybe the most important of all because of their versatility. The approach is completely different from what we have seen until now. Indeed, we saw two classes of algorithms, one based on a pure analytic resolution of the problem by calculating manually the posterior distribution and the other one by using message propagation in a graph. In both cases, the result was computed exactly. In the case of an analytic solution, computing the solution usually boils down to computing a function of the posterior distribution. In the case of a message-passing algorithm, computing the posterior distribution is done step-by-step by propagating messages on a graph. If the graph is not appropriate for this type of algorithm, the computations can be extremely long and often intractable. However, in many cases, we can trade a bit of accuracy for more speed. This is the main idea of approximate inference. Does it really matter if we are less accurate? Well, it appears that, in many applications, approximate inference is still very accurate. On the other hand, it allows us to deal with more complex models and with many types of distributions, something that is not completely possible with the other approaches. In this chapter, we will see one important class of algorithms, called sampling algorithms, also known as Monte-Carlo sampling. The main idea of this class of algorithms is to draw at random from the posterior distribution in order to replace complex computations with simple statistics. For example, if we want to compute the posterior mean of a random variable, we can draw many samples at random from its posterior distribution and simply compute the mean of the values we obtained. Monte-Carlo sampling is what really made the Bayesian revolution possible in science. Before, Bayesian models were hard to compute, if not impossible. [ 125 ]

Approximate Inference More speci cally, we will look at the following algorithms:

* Rejection and importance sampling, as a basis for many other methods
* Markov Chain Monte-Carlo and the Metropolis-Hastings algorithm

These two classes of algorithms will cover most of what we need to know about the Monte-Carlo method. However, many new algorithms are regularly being discovered.

## 从分布中采样

We have a big problem with probabilistic graphical models in general: they are intractable. They quickly become so complex that it is impossible to run anything in a reasonable amount of time. Not to mention learning them. Remember, for a simple algorithm such as EM, we need to compute a posterior distribution at each iteration. If the dataset is big, which is common now, if the model has a lot of dimensions, which is also common, it becomes totally prohibitive. Moreover, we limited ourselves to a small class of distributions, such as multinomial or Gaussian distributions. Even if they can cover a wide range of applications, it's not the case all the time. In this chapter, we consider a new class of algorithms based on the idea of sampling from a distribution. Sampling here means to draw values of the parameters at random, following a particular distribution. For example, if one throws a dice, one draws a sample from a multinomial distribution, such that six values are possible, with equal probability. The result is a number between 1 and 6. If the die is not fair (say 6 has a higher probability), then it is possible we will obtain more 6s than the other numbers. If we throw the die many times and then calculate the mean value of all the results, we will presumably see a number closer to 6 than 3. In many situations, we are more interested in some properties of the distribution rather than the distribution itself—for example, its mean or variance. It means that, in many cases, we want to compute an expectation of a function f(x) with respect to a probability distribution p(x). Here, x can be any random variable of any dimension we want. For example, p(x) can be a multivariate Gaussian distribution or a probabilistic graphical model. In the case of continuous variables, we want to solve the generic problem of evaluating the expectation:

E ( f ) = ∫ f ( x )p ( x ) dx

When x is discrete, the integral is replaced by a summation. In the example in the screenshot, the distribution is in red and the function in green. We immediately see there will be many problems sampling from such a complex distribution. The algorithms presented in this chapter will try to solve many of those problems. The main idea in sampling is to replace the evaluation of an integral with a simpler sum over a set of samples drawn independently from the distribution p(x). The previous expectation can then be approximated by a nite sum: ˆ 1∑L ((l)) f=Ll=1 fx [ 127 ]

Approximate Inference E f =E(f) If the samples are drawn from the distribution p(x) then ( ˆ ) and, similarly, the variance of the estimator is: var f =LE f−E(f) (ˆ)1(2) In this method, the problem is to obtain independent samples. It is not always the case and the number of effective samples might be fewer than the number of points drawn from the distribution. However, as seen in the previous formula, the variance of ˆ , the estimator does not depend on the dimension of x. It means that, even in f high-dimensional problem such as a graphical model, high accuracy can be obtained with a relatively small number of samples. But as we stated before, the main problem is in particular sampling from the distribution p(x). And it can be hard or even impossible to do it sometimes. When the distribution p(x) is a graphical model in a directed form (such as most of the models we have in this book), the technique to draw a sample is quite simple and called ancestral sampling. Given an ordering of the variables xi in the graph, from the top of the graph to the bottom, one can draw a sample from the graphical model by sampling successively each variable and then assigning the sampled values at the corresponding variables to sample from its descendant. For example, let's say we have the following graph:

We start by sampling from p(A) and p(B) independently, then assign the sampled values to A=a and B=b so that we sample from p(C | A=a, B=b). Finally, the last sample is drawn from p(D | C=c). If none of the variables are observed, the procedure is very simple. If, however, one variable is observed, one technique is to keep only the samples that agree with the value of the observed variables. If we have A=a1, for example, then we will keep only the samples in which we have been lucky enough to have A=a1. In this case, not all the samples are usable and the difference between the number of drawn samples and the number of useful samples can be big, because each time a set of samples disagrees with the observed values, it has to be discarded. The probability of a set of samples being accepted decreases with the number of observed nodes.

## 基本采样算法

We start our study of sampling algorithms by looking at some basic algorithms that can be used as subroutines in more advanced schemes. In all the algorithms from this book, we assume that we can generate random numbers uniformly from the interval [0,1]. The problem of generating random numbers on a computer is complex and vast. In most cases, random numbers will be generated by a deterministic algorithm, and they are called pseudo-random numbers. They are not random, but their distribution and properties are close enough to a real random generator that they can be used as real random numbers. Pseudo-random number generators are usually based on the estimation of chaotic functions that are extremely sensitive to their initial conditions, which are called the seed. Changing the value can generate a completely different sequence of numbers, even if the seed value is just a little bit different from the previous one. Nowadays, we also have electronic devices that can generate random numbers from physical phenomena such as thermal noise, photoelectric effects, or quantum phenomena.

### 标准分布

In R, it is possible to generate random numbers from standard distributions. However, for the sake of understanding, we will review simple techniques for how to generate random numbers from a uniform distribution only. In R, random numbers can be generated from a family of functions beginning with the letter r, such as runif, rnorm, rbeta, rbinom, rcauchy, rgamma, rgeom, rhyper, rlogis, and so on. In fact, density can be estimated using the functions beginning with d, and the cumulative distribution function with functions beginning with the letter p. [ 129 ]

Approximate Inference For example:

runif(1)  
[1] 0.593396

Here, the parameter of the function is the number of numbers one wants:

runif(10)  
[1] 0.7334754 0.2519494 0.7332522 0.9194623 0.5867712 0.3880692 0.2869559   
[8] 0.7379801 0.4886681 0.5329107

Of course, these are (pseudo) random numbers, so your results will be different from the examples shown earlier.

rnorm(1,10,1)  
[1] 9.319718

This generates a normally distributed number, with mean 10 and variance 1. If we generate many of these numbers and plot the running mean, we'll see the mean converging little by little to its true value. This is the main property we will use throughout this chapter and in sampling algorithms in general.

x <- rnorm(1000, 10, 1)  
y <- cumsum(x)/(1:1000)  
plot(y, t = "l")  
abline(h = 10)

Generating a random number from a simple distribution is the basis of all sampling algorithms. Here, we consider we know how to generate a random number from a uniform distribution. In R, we perform runif(1,0,1). By default, the min and max parameters are already 0 and 1. Therefore runif(1) will do. Suppose we transform the uniformly random values with a function f (.) such that y=f(x). The distribution of y will be: p(y)= p(x) dx dy We need functions f(x) such that the distributions of y are distributed according to the desired distribution p(y). Integrating p(y), we have: x = h(y)≡ ∫y p(yˆ)dyˆ −∞ Therefore, y = h−1 (x), which is the inverse function on the inde nite integral of the desired distribution. Let's take a simple example with the exponential distribution. This distribution is continuous with a density function p(x)=λexp(−λx) and a support on [0, +∞[. Integrating h(y) gives h(y)=1−exp(−λy), which is the cumulative distribution function of the exponential distribution. As a side note, the exponential distribution is useful to describe the lengths of inter- arrival times in a Poisson process. It can be viewed as the continuous version of the geometric distribution. Therefore, if we transform our uniformly distributed variable x with the function h−1 = −λ−1 ln(1− x), then y will be exponentially distributed. We can check it experimentally by plotting the distribution of our function and comparing it with the distribution of an exponential distribution. We take lambda=2 in this example:

x <- runif(20000)  
inv\_h <- function(x, lambda) -(1/lambda) \* log(1 - x)  
hist(inv\_h(x, 2), breaks = 100, freq = F)  
t <- seq(0, 4, 0.01)  
lines(t, dexp(t, 2), lw = 2)

We rst generate 20,000 points from a uniform U(0,1) distribution. Then, inv\_h is the function de ned earlier and we plot a histogram. Note the parameter freq=F to draw with the densities instead of the frequency. Finally, we draw the density function of an exponential distribution with the same parameter (the thick black line in the following graph) and see that the two distributions, the empirical one and the analytic one, are a very good match. The following screenshot shows the result:

The problem with this technique is the evaluation of the inde nite integral. In simple cases, this integral is readily available but this is not always so. In such a case, we need another strategy and the key is to use a simpler distribution to approximate the more complex distribution we can sample from. There are two basic techniques to do that. One is called rejection sampling, and it uses a simple distribution to draw a sample from and accept the sample at the same time as it falls into the more complex distribution. Otherwise it is rejected. The other technique is called importance sampling and, in this case, samples from the approximate distribution are corrected to take into account their difference with respect to the original distribution one wants to sample from. Both techniques are important and used as a basis for more advanced techniques, such as Markov Chain Monte-Carlo (MCMC), that we will see in the second part of this chapter. [ 132 ]

Chapter 5 In all cases, one of the main ideas is to use a proposal distribution, whose goal is to approximate, even roughly, the distribution we want to sample from. We will call q(x) the proposal distribution and p(x) the initial distribution in the following sections.

## 拒绝性采样

Suppose we want to sample from a distribution that is not a simple one. Let's call this distribution p(x) and let's assume we can evaluate p(x) for any given value x, up to a normalizing constant Z, that is: p(x)= 1 p (x) Zp In this context, p(x) is too complex to sample from but we have another simpler distribution q(x) from which we can draw samples. Next, we assume there exists a constant k such that kq(x)≥ p (x) for all values of x. The function kq(x) is the comparison function as shown in the following gure:

The distribution p(x) has been generated with a simple plot:

0.6 \* dnorm(x, 1) + 0.4 \* dnorm(x, 5)

Approximate Inference The rejection sampling algorithm is based on the following idea:

* Draw a sample z0 from q(z), the proposal distribution
* Draw a second u0 sample from a uniform distribution on [0, kq(z0)]
* If u0 > p (z0 ) then the sample is rejected otherwise u0 is accepted

In the following gure, the pair (z0, u0) is rejected if it lies in the gray area. The accepted pairs are a uniform distribution under the curve of p(z) and therefore the z values are distributed according to p(z):

The probability of such a pair being accepted is: p ( a c c e p t e d ) = 1 ∫ p ( z ) d z k Because it depends on k, it is necessary for the proposal distribution to be as close as possible to the real distribution, otherwise the algorithm will converge very slowly and would be practically useless. [ 134 ]

Chapter 5 This algorithm is very simple and can be easily implemented. However, it suffers from a drastic problem related to the dimension of the problem. In the case of a probabilistic graphical model, the dimension quickly becomes very large. Rejection sampling is usually a good idea in one or two dimensions, but the rejection rate grows exponentially with the number of dimensions. However, it can be used as a subroutine in more advanced algorithms to generate samples for simple probabilistic forms (for example, at the level of one node if needed).

### R语言实现

We consider the problem of estimating a mixture of Gaussian distribution with a normal distribution. The mixture of Gaussian and the proposal distributions are shown in the following gure, where the proposal distribution in red has been scaled with k=3.1. The mixture of Gaussian distribution is in black and has two modes:

In R, we de ne the proposal and target distributions as follows:

q <- function(x) dnorm(x, 0, 0.5)  
rq <- function(x) rnorm(1, 0, 0.5)  
p <- function(x) 0.6 \* dnorm(x, 0, 0.1) + 0.4 \* dnorm(x, 0.6, 0.2)

The parameters are arbitrary. We have a proposal distribution q centered on 0 with a standard deviation of 0.5. The target distribution is a mixture of Gaussian with two components. The rejection algorithm is as follows:

rejection <- function(N,k,p,q,rq)  
{  
 accept <- logical(N)  
 x <- numeric(N)  
 for(i in 1:N)  
 {  
 z0 <- rq() # draw one point from the proposal distribution  
 u0 <- runif(1,0,1) # drawn one point from the uniform  
 if(u0 < p(z0)/(k\*q(z0))) # rejection test  
 accept[i] <- TRUE  
 else accept[i] <- FALSE  
x[i] <- z0 }  
 data.frame(x=x,accept=accept)  
}

The parameters are as follows:

* N: This is the number of samples.
* k: This is the coefficient for the proposal distribution.
* p: This is the distribution to estimate. You must pass a function that takes one parameter.
* q: This is the proposal distribution (with the same remark as earlier).
* rq: This is a sampler for the proposal distribution.

This algorithm samples N times and accepts or rejects in each sample in a for loop. The result is stored in a data.frame. We keep all the samples to compare the results between rejection or not. The rst column is the samples, and the second column is a binary value indicating whether the sample has been accepted or not. The algorithm works as described in the theoretical part:

1. We rst create two vectors, accept and x, for storing the results.
2. We start a loop in which:
3. We sample z0 from the proposal distribution.
4. We sample u0 from a uniform on [0,1].
5. We accept or reject the value and store the result.

Let's do a few experiments to understand the behavior of this important algorithm. In the experiments, in order for the reader to be able to reproduce exactly the same results, we will use a xed random seed by doing:

set.seed(600)

Moreover, we will use a scaling factor k of 3.1 as:

k <- 3.1

So, the rst experiment is to run the algorithm with 100 samples:

x <- rejection(100, k, p, q, rq)

The results are stored in the data.frame x and the head of this data.frame is as follows:

head(x)  
 x accept  
1 -0.56007075 FALSE  
2 -0.18000011 FALSE  
3 -0.07572593 TRUE  
4 -0.72502107 FALSE  
5 -0.60916359 FALSE  
6 0.97963839 FALSE

We can see that not all the points have been accepted. In our example, only 47 points out of 100 have been accepted. Looking at the histogram of the accepted values, we are far from the target distribution. It means that we need to run the algorithm longer than we did:

t <- seq(-2, 2, 0.01)  
hist(x$x[x$accept], freq = F, breaks = 200, col = "grey")  
lines(t, p(t), col = 2, lwd = 2)

On this graph, we can see that the accepted samples regroup in the region of the high-probability mass. But running with such a small number of samples is not enough. The red curve is our target distribution. We now run the algorithm with 5,000 samples:

x <- rejection(5000, k, p, q, rq)  
hist(x$x[x$accept], freq = F, breaks = 200, col = "grey")  
lines(t, p(t), col = 2, lwd = 2)

And what we expect in this second run is to see a better concentration of the accepted samples into the regions of high probability of the target distribution. The following graph shows the result:

And indeed the graph looks better now. The histogram follows the true distribution but it is still not perfect. In fact, the number of accepted samples is not that high when we consider it:

sum(x$accept)  
1581

If we run the algorithm for longer, we will obtain a better sample set and approach the target distribution very closely. [ 139 ]

Approximate Inference So, now we run the algorithm with 50,000 samples. After running it, we nd that 16,158 of them have been accepted. And the result is far better of course:

The two modes of the distribution have been correctly captured and the empirical distribution follows precisely the target distribution. This is at the expense of running the algorithm for longer. If we draw the histogram of all the points sampled from the proposal distribution (accepted or not), they follow precisely the proposal distribution as expected:

hist(x$x, freq = F, breaks = 200, col = "grey")  
lines(t, q(t), col = 2, lwd = 2)

Finally, it is also interesting to look at the behavior of this algorithm from the point of view of the number of accepted samples. We run a simple function, as follows:

N <- sapply(seq(1000, 50000, 1000), function(n) {  
})  
x <- rejection(n, k, p, q, rq)  
sum(x$accept)

And we plot the result with:

plot(N, t = "o")

The result is not surprising. The more samples are drawn, the more are accepted, so it gives us an interesting clue: running the algorithm for longer will indeed improve the results with respect to the number of accepted samples. But the problem with rejection sampling is that we need to sample many points in order to have good results. Rejection sampling is still a good algorithm and can be used in many situations. In the next section we will explore an improvement on rejection sampling, called importance sampling, in which all the sample points are accepted.

## 重要性采样

Importance sampling is an improvement on rejection sampling. Again the assumptions are the same and we will use a proposal distribution q(x). We also assume that we can compute the value of the density of probability p(x). But we are unable to draw a sample from it because it is, again, too complex. [ 142 ]

Chapter 5 Importance sampling is based on the following reasoning, where we need to evaluate the expectation of a function f(x) with respect to the distribution p(x): E ( f ) = ∫ f ( x )p ( x ) dx At this stage, we simply introduce the distribution q(x) in the previous expression: E ( f ) = ∫ f ( x ) p ( x ) q ( x ) dx q(x) And, as before, we approximate it with a nite sum: 1∑L p(x(l)) ((l)) E(f) L l=1q(x(l))f x The ratio r = p (x(l ) ) is called importance weight and it is the bias introduced by l q(x(l)) sampling q(x) when in fact we wanted to sample from p(x). In this case, the algorithm is very simple because all the samples are used. Again, importance sampling is ef cient if the proposal distribution is close enough to the original distribution. If the function f(x) varies a lot, we might end up in a situation where f has high values in areas where the distribution p is small and the sum might be dominated by these areas of low probability. Therefore it becomes necessary to increase the number of samples in order to have better results. So the effective number of samples might in fact be lower than the real number of samples, even if there is no rejection. For a graphical model with discrete variables, it is possible to use importance sampling with the following approach:

• For each variable x in the graph: ° If the variable is in the evidence set (x is observed), then set it to its own observed value. ° Otherwise, it is sampled from p(x | pa(x)), in which the variables in pa(x) are set to their sampled (or observed) values. Therefore, sampling from p(x | pa(x)) becomes a simple problem. [ 143 ]

Approximate Inference The weighting associated with the sample produced by this algorithm is: r(x)=∏ p(x| pa(x))∏ p(x| pa(x))=∏ p(x| pa(x)) x∉E p(x|pa(x)) x∉E 1 x∉E The two algorithms we introduced are interesting in small dimensions and can be easily implemented. However, we saw that they suffer from severe limitations in high dimensions. Even importance sampling might need a long convergence time, despite the fact that all samples are accepted. The rest of this chapter is therefore dedicated to a very powerful and very framework based on Markov Chain and it is called Markov Chain Monte Carlo (MCMC).

### R语言实现

The difference between rejection sampling and importance sampling is that the latter is not so much an algorithm to sample from a distribution as a technique to approximate averages with respect to an intractable distribution. It is usual to use the following algorithm: ∫ f(x)p(x)q(x)dx X q(x) q(xi ) Here, xi ~ q, that is, x is drawn from the distribution q. This gives us a simple algorithm because all we have to sample from q and then apply the earlier formula. We can then estimate all sorts of f(x) function from this simple algorithm. In this case, the distribution function q doesn't have to be scaled as in the rejection algorithm. Moreover, all the samples are accepted because there is no such notion as rejection. However, the algorithm is restricted to the evaluation of integrals and can't generate samples from the target distribution. So we see that importance sampling has a different type of use case. [ 144 ] ∑N f(xi)p(xi) i=1 ∑N p(xi) q(xi) i=1

Chapter 5 The algorithm can be simply implemented in R and we will use the following code:

importance <- function(N, f, p, q, rq) {  
 x <- sapply(1:N, rq) # sample from the proposal distribution  
 A <- sum((f(x) \* p(x))/q(x)) # numerator  
 B <- sum(p(x)/q(x)) # denominator  
 return(A/B)  
}

The parameters are as follows:

* N: This is the number of samples
* f: This is the function we want to know the expectation of
* p: This is the target distribution function
* q: This is the proposal distribution function
* rq: This is a sampler from the proposal distributions

In the following examples, we will estimate the mean of several distributions and therefore the function f will be the identity function in R. The algorithm is very simple and follows the formula we saw before:

1. It draws N points from the proposal distribution.
2. It computes the numerator and denominator of the formula.
3. It returns the result.

In the next examples, we will compute the mean from the following examples:

* We will use the same mixture of Gaussian we used for rejection sampling and will approximate it with a Gaussian distribution
* We will approximate a Student's t distribution with a Gaussian
* We will approximate a Gamma distribution with an exponential distribution

And, as before, in order to be able to reproduce the results, we will set the seed beforehand by doing the following in R:

set.seed(600)

The rst example has two distributions, where the black curve is the target distribution and the red curve is the proposal distribution. We see the same mixture of Gaussian and Gaussian as in rejection sampling:

Then the next example uses a Student's t distribution and a Gaussian distribution as the proposal distribution. In this example, the Student's t distribution has 2 degrees of freedom and the Gaussian has a mean of 0 and a variance of 1.5. [ 146 ]

The following gure shows the two distributions:

Finally, the last example uses a Gamma distribution as the target and an exponential distribution as the proposal distribution. The shape parameter of the Gamma distribution is 2. As for the exponential distribution, the rate parameter is 0.5. These two distributions have a support from zero to in nity. Indeed, importance sampling requires that, if the proposal distribution gives a zero probability, then the target distribution must give a zero probability too for the same value. [ 147 ] Chapter 5

Approximate Inference The following gure shows the Gamma and the exponential distribution:

In R we de ne the function of our three examples as follows. For the mixture of Gaussian approximated by a Gaussian, we have:

p <- function(x) 0.6 \* dnorm(x, 0, 0.1) + 0.4 \* dnorm(x, 0.6, 0.2)  
q <- function(x) dnorm(x, 0, 0.5)  
rq <- function(x) rnorm(1, 0, 0.5)

For the Student's t distribution approximated by a Gaussian, we write:

p <- function(x) dt(x, 2)  
q <- function(x) dnorm(x, 0, 1.5)  
rq <- function(x) rnorm(1, 0, 1.5)

And, for the Gamma distribution approximated by an exponential function, we de ne:

p <- function(x) dgamma(x, 2)  
q <- function(x) dexp(x, 0.5)  
rq <- function(x) rexp(1, 0.5)

Then we run the rst experiments:

print(importance(1000, identity, p, q, rq))  
print(importance(10000, identity, p, q, rq))  
print(importance(50000, identity, p, q, rq))

The theoretical mean of the mixture of Gaussian is 0.24. Our code gives the following results:

[1] 0.2256604  
[1] 0.2364267  
[1] 0.2409898

We see that the more samples we have, the more accurate the estimation is. However, we also see that, with 10,000 samples, the result is already accurate enough. This is one of the advantages of importance sampling: we need fewer samples than rejection sampling to perform this kind of task. The second experiment with a Student's t and a Gaussian gives the following result:

[1] -0.00285064  
[1] 0.07353888  
[1] 0.06475101

The theoretical result is 0 because the Student's t is 0-centered. And the third experiment with a Gamma and an exponential distribution gives:

[1] 1.971177  
[1] 2.002985  
[1] 1.994183

Again, we see an improvement when we go from 1,000 samples to 10,000. After 10,000 samples, it seems the results are not improving a lot so we can stop the algorithm earlier. Also note that the previous examples can be run with exactly the same lines of code as shown earlier. The reader will have to take care to the rede ne the functions p, q, and rq each time. [ 149 ]

Approximate Inference The next experiment we are going to do is to run the algorithm with different sample sizes and look at how the estimated mean converges to the true value. We will rerun the following code three times, changing the functions p, q, and rq each time:

t <- seq(1000, 50000, 500)  
x <- sapply(t, function(i) importance(i, identity, p, q, rq))

This code iterates from 1,000 to 50,000 samples in steps of 500 on the importance sampling algorithm. The more samples we draw, the more accurate the estimated mean. The rst experiment with a mixture of Gaussian and a Gaussian gives the following result:

The second example with a Student's t distribution and a Gaussian distribution gives the following result:

We see in this example that, despite a good convergence, we can sometimes have surprising results. It seems indeed that our proposal distribution is not the best approximation. Indeed, importance sampling can be sensitive, as with rejection sampling, to the proposal distribution and gives results that are not completely stabilized. [ 151 ]

Approximate Inference The last experiment uses the Gamma distribution and an exponential distribution as the proposal distribution. The following screenshot shows the result:

Here we see a clear convergence of the results. At the beginning, the number of samples is too low to give an accurate mean; then, when the number of samples increases, the results are more and more accurate. In the next section, we will see a more advanced technique (to sample from arbitrary distributions) called Markov Chain Monte-Carlo sampling. It is a very powerful method and has many applications nowadays.

## 马尔科夫链蒙特卡洛

MCMC methods have their origin in physics with the work of Metropolis, Ulam, and Rosenbluth. It was in the 1980s that they began to have a signi cant impact on statistics. Many MCMC algorithms and methods have been proposed and they are among the most successful approaches to computing posterior distributions. [ 152 ]

Chapter 5 If we use the word framework and not algorithm, it is because there is no single MCMC algorithm; instead, there are many. Multiple strategies are possible to implement it based on the problem we need to solve. Monte-Carlo has been used for more than half a century to solve many complicated estimation problems. However, its main weakness was, as in rejection and importance sampling, its convergence in high-dimensional problems. So Markov Chains were used from the start to estimate the convergence and stability of those methods. But it wasn't until recently (the1980s and 1990s) that they started to be massively used in statistical estimation.

### 主要思想

Markov Chain Monte-Carlo methods are based on the same idea as previously, where we have a complex distribution p(x) and a proposal distribution q(x). However, in this case, the state of the variable x is kept along the way and the proposal distribution depends on this current state, that is we sample from q(x | xt-1). This sequence of x forms a Markov Chain. The main idea is that at each step of the algorithm we draw a sample from q(x | xt-1) and accept the sample upon certain criteria. If the sample is accepted then the parameters of q are updated according to the new sampled value and we start again, otherwise a new sample is drawn without changing q. So we want to choose a simple q distribution to make it ef cient. As before, the problem we want to solve is the expectation of a function with respect to a complex distribution: E(e)=∫f (x)p(x)dx We assume that we can evaluate p(x) or at least we can evaluate it up to a normalizing constant p ( x ) ∝ p% ( x ) . In order to solve the problem of sampling, the Metropolis-Hastings algorithm (proposed by Metropolis in 1953 and Hastings in 1970) gives a generic way to construct a Markov Chain that is ergodic and stationary with respect to the distribution p(x). In other words, it means that if xt ~ p(x) then xt+1 ~ p(x) and therefore the Markov Chain will converge to the distribution p(x). [ 153 ]

Approximate Inference The principle of MCMC algorithms is somehow contrary to the principle of rejection and importance sampling in the sense that, instead of aiming directly at the big picture with the proposal distribution, it tries to explore the space of p(x), with a simpler distribution. Let me give you an analogy, once explained by Professor Christian Robert from the University of Paris-Dauphine, France, and the University of Warwick, UK. Imagine you are a visitor in a museum and suddenly there is a blackout. The gallery is completely dark. Your only way to look at the paintings is to use a small torch. The beam of the torch is very narrow so at any time you will only see a small part of the painting. But you can move your torch along the painting until you discover all of it. Then you will have the big picture. Of course, you can argue that a painting is more than the sum of its parts, but that's another story.

### Metropolis-Hastings算法

This algorithm will build a sequence of sampled values xt, such that this sequence will converge to p(x). So the chain of values is a sample of p(x) and these values are approximately distributed according to p(x). However, at the beginning, and because each value is dependent on its previous value, the rst samples are also very dependent on the initial value x0. It is therefore recommended not to use the initial values and to give a period of warm-up to the algorithm. We recall a similar result in the previous algorithm. Based on the Markov Chain, it is possible to show that, even if it's hard to determine when the algorithm will reach stationarity, the average of the sampled values will converge almost surely to E(f), the empirical average de ned by: 1 ∑L ( (l) ) E(f)=L l=1fx This will converge almost surely to E(f). We recall that almost sure convergence is de ned as a sequence X1, X2, ..., Xn of random variables converging to a random variable X if: ({ }) p s∈S:limX (s)=X(s) =1 n→∞ n [ 154 ]

Chapter 5 Of course, we can't feasibly sample an in nite sequence of variables, but we have a guarantee it will converge. So in theory we know it will converge and therefore sampling from the Markov chain is equivalent to iid sampling from the distribution. In practice, we need to sample a lot to get good results. The Metropolis-Hastings algorithm works as follows:

1. Draw a value xt ~ q(xt | xt-1) where q(x) is our simple proposal distribution.
2. Compute the probability of acceptance with: ⎪ ()( )⎪ ρ ( x , x ) = m i n ⎧ 1 , p x t q x t − 1 | x t ⎫ t t−1 ⎨ p (x )q(x |x )⎬ ⎪t−1 tt−1⎪ ⎩⎭
3. Take xt with probability ρ(xt, xt-1) and xt-1 with probability 1– ρ(xt, xt-1).

Given the choice of q(x), this algorithm will preserve the stationarity of the distribution p(x) in the Markov chain. In theory, again, this algorithm is guaranteed to converge for an arbitrary distribution q(x). This is an impressive result, hence the popularity of this algorithm. However, in practice, things are a bit more complicated because the convergence might happen very late in the process, if for example the proposal distribution q(x) is too narrow. On the other hand, a too large distribution might end up with a very unstable algorithm. It will still converge but with a huge step and can miss the most important part of the original distribution p(x) by leaving the area with high probability mass too early. The sequence of samples xt represents a random walk and we can illustrate the previous problem of practical convergence by looking at what happens to such a trajectory in a simple example. We want to sample from a two-dimensional Gaussian distribution and we use a smaller two-dimensional Gaussian as the proposal distribution. In order to deal with multi-dimensional Gaussian, we need the MASS package:

library(MASS)  
bigauss <- mvrnorm(50000, mu = c(0, 0), Sigma = matrix(c(1, 0.1, 0.1, 1), 2))  
bigauss.estimate <- kde2d(bigauss[, 1], bigauss[, 2], n = 50)  
contour(bigauss.estimate, nlevels = 6, lty = 2)

This code snippet plots a simple two-dimensional Gaussian distribution as shown next. From there, we will use a smaller Gaussian distribution whose next mean value will be drawn from the current small Gaussian distribution. This is not yet an application of the Metropolis-Hastings algorithm, but just an example to visualize what happens with different covariance to the random walk and where it can go in just a few iterations. The starting point of the random walk is arbitrarily in the center of the big Gaussian distribution, that is, at coordinates (0,0) in this case:

The following gure shows the small proposal distribution in red contour lines:

The following code draws random samples from the small distribution in the center and updates its center with the previous value:

L <- 10  
smallcov <- matrix(c(0.1, 0.01, 0.01, 0.1), 2)  
x <- c(0, 0)  
for (i in 1:L) {  
 x2 <- mvrnorm(1, mu = x, Sigma = smallcov)  
 lines(c(x[1], x2[1]), c(x[2], x2[2]), t = "p", pch = 20)  
 x <- x2  
}

We show three examples, with 10, 100, and 1,000 points. Obviously, a pure random walk is completely off the initial big distribution after a few iterations. As for 1,000 iterations, they are totally off:

Now, let's complete the picture and implement the last bit of the Metropolis-Hastings algorithm in R. As we assumed that we can estimate p(x) on our target distribution, we will use here the dmvnorm function from the mvtnorm package. To illustrate the behavior of this algorithm, we will take a simple Gaussian distribution as p(x) and an even simpler uniform distribution as q(x). So the proposal distribution is simply: q(xt |xt−1)= 1 I(xt−1 −α,xt−1 +α)(x) 2α [ 158 ]

hist(sampler(l,1),main=paste(l,"iterations"),breaks=50,freq=F,xlim=c(- 4,4),ylim=c(0,1)) lines(x0,p(x0)) } The rst function is the evaluation of p(x). Then it is followed by the Metropolis-Hastings step using the proposal uniform distribution de ned before. [ 159 ] Chapter 5 And p(x) = N(0,1), a simple Gaussian distribution:

p = function(x) {  
 dnorm(x, 0, 1)  
}  
mh = function(x, alpha) {  
 xt <- runif(1, x - alpha, x + alpha)  
 if (runif(1) > p(xt)/p(x))   
 xt <- x  
 return(xt)  
}  
sampler = function(L, alpha) {  
 x <- numeric(L)  
 for (i in 2:L) x[i] <- mh(x[i - 1], alpha)  
 return(x)  
}  
par(mfrow = c(2, 2))  
for (l in c(10, 100, 1000, 10000)) {  
 hist(sampler(1000, a), main = paste("alpha=", a), breaks = 50, freq = F,   
 xlim = c(-4, 4), ylim = c(0, 1))  
 lines(x0, p(x0))  
}

Approximate Inference Then we implement the sampler, which takes two parameters: L is the number of iterations and alpha the width of the uniform distribution. The bigger alpha is, the larger the area this proposal distribution will cover. Too large a value will result in too many jumps and poor results. Too small a value will have a serious impact on the numerical convergence of the algorithm, even if we know that theoretically it will converge. Then the last part of the code draws some results for 10, 100, 1,000, and 10,000 iterations. Running this code, we obtain the following graph. Your results will be different because we draw numbers at random, of course, but the overall plot is similar:

It is clear that, with only 10 iterations, the results are poor. After a warm-up period of 100 iterations, it seems we are close to the mean value, but the variance of such a dataset will be completely off with respect to p(x). After 1,000 iterations, our dataset begins to be quite close to the target distribution. Finally, after 10,000 iterations, we have a superb histogram.

Next, we vary alpha, with 1,000 iterations. We use the following code:

par(mfrow = c(2, 2))  
for (a in c(0.1, 0.5, 1, 10)) {  
 hist(sampler(1000, a), main = paste("alpha=", a), breaks = 50, freq = F,   
 xlim = c(-4, 4), ylim = c(0, 1))  
 lines(x0, p(x0))  
}

Again, we see interesting results: the theory says it will converge. The practice needs a bit of tuning rst. It seems that alpha=0.5 or alpha=1 are large enough to cover the distribution. However, alpha=0.1 is too narrow and can't explore the space fast enough in only 1,000 iterations. On the other hand, alpha=10 gives a bi-modal distribution; the jumps are too big.

If we run the same experiment with many more iterations, such as 50,000, we see a stabilization of the algorithm and most of the proposal distribution seem to converge to the ideal solution. Again, alpha=0.1 and alpha=10 seem a bit weak, but the overall result is more than acceptable this time:

## 概率图模型MCMC算法R语言实现

In fact, this section could be the title of book. As a matter of fact, there are several books entirely devoted to this speci c topic. Research in this eld is extremely active, with many new algorithms coming every year. [ 162 ]

Chapter 5 There are numerous packages implementing MCMC algorithms for different types of algorithms. There are also more generic frameworks such as the famous BUGS (and its open source implementation OpenBUGS) and a new, even more powerful framework called Stan. Historically, BUGS was the rst framework to popularize MCMC inference in Bayesian statistics and literally led to a revolution in this eld, as everyone could bene t from Bayesian statistics right out of the box. Making an introduction to each of them and showing all the possibilities of MCMC for a few speci c graphical models would require another book as big as this one. In this section, we will therefore focus on a programming environment available in R. In fact, it works also with C++, Python, Matlab, Julia, Stata, and even on the command line! It allows the implementation of all sorts of Bayesian models, as we have seen so far. Stan mainly uses MCMC algorithms to perform inferences.

### 安装Stan和RStan

This procedure is detailed on the following web page: <https://github.com/stan-> dev/rstan/wiki/Rstan-Getting-Started. Thus we will just recall the basic steps in order to install Stan and RStan:

Sys.setenv(MAKEFLAGS = "-j4")  
install.packages("rstan", dependencies = TRUE)

Prepare yourself for a long installation, as Stan will need numerous packages. Finally, depending on your R installation, you might have to restart R, but in general it is not necessary. Finally, load RStan like any other package:

library(rstan)

You should see an introductory message such as this, telling you Stan is ready:

Loading required package: ggplot2  
rstan (Version 2.9.0, packaged: 2016-01-05 16:17:47 UTC, GitRev:  
05c3d0058b6a)  
For execution on a local, multicore CPU with excess RAM we recommend  
calling  
rstan\_options(auto\_write = TRUE)  
options(mc.cores = parallel::detectCores())

### RStan的简单例子

We show here some basic possibilities in RStan. The reader is encouraged to read more about it and try more examples. RStan is based on a probabilistic programming language used to describe the Bayesian models. For example, we can make a simple univariate Gaussian model:

parameters {  
real y; }  
model {  
}  
 y ~ normal(0,1);  
}

This code is Stan code, not R code. Then, in R, we can simulate this model, by doing:

fit = stan(file = "example.stan")

The model will be simulated using an MCMC algorithm and the results are displayed using:

print(fit)  
Inference for Stan model: example.  
4 chains, each with iter=2000; warmup=1000; thin=1;  
post-warmup draws per chain=1000, total post-warmup draws=4000.  
 mean se\_mean sd 2.5% 25% 50% 75% 97.5% n\_eff Rhat  
y 0.00 0.03 0.98 -1.93 -0.63 -0.01 0.64 1.98 1191 1  
lp\_\_ -0.48 0.02 0.69 -2.35 -0.63 -0.20 -0.05 0.00 1718 1  
Samples were drawn using NUTS(diag\_e)  
For each parameter, n\_eff is a crude measure of effective sample size,  
and Rhat is the potential scale reduction factor on split chains (at  
convergence, Rhat=1).

It's interesting to see that this package has indeed a warm-up procedure as we saw before and it tries to compute an estimate of the effective sample size for a simple Gaussian. The value of 1191 (the n\_eff column) is not too far from the examples we saw in the previous section. [ 164 ]

## 小结

In this chapter, we saw the second (and presumably most successful) approach to performing Bayesian inference, with algorithms such as rejection and importance sampling, which are based on the use of a proposal distribution simpler than the one we want to estimate. These two algorithms are usually ef cient with low-dimensional cases but suffer from long convergence problems, when they converge at all, in high dimensions. We then introduced the most important algorithm in Bayesian inference: the MCMC method using the Metropolis-Hastings algorithm. This algorithm is extremely versatile and has a nice property: it converges toward the distribution one wants to simulate. However, it needs careful tuning in order to converge, but its convergence is guaranteed in theory. In the next chapter, we will explore the most standard statistical model ever: linear regression. While it seems beyond the scope of this book, this model is so important that it needs to be introduced. However, we will not stop at the simple form of it but will explore its Bayesian interpretation, how it can be represented as a probabilistic graphical model, and what bene t we get from doing so.

# 第6章 贝叶斯建模——线性模型

A linear regression model aims at explaining the behavior of one variable with another one, or several others, and by so doing, the assumption is that the relationship between the variables is linear. In general, the expectation of the target variable, the one you need to explain, is an af ne transform of several other variables. Linear models are presumably the most used statistical models, mainly because of their simplicity and the fact they have been studied for decades, leading to all possible extensions and analysis one can imagine. Basically all statistical packages, languages, or software implement linear regression models. The idea of the model is really simple: a variable y is to be explained by several other variables xi by assuming a linear combination of x's—that is, a weighted sum of x's. This model appeared in the 18th century in the work of Roger Joseph Boscovich. Then again, his method has been used by Pierre-Simon de Laplace, Adrien-Marie Legendre, and Carl Friedrich Gauss. It seems that Francis Galton, a mathematical genius of the 19th century, coined the term "linear regression". The model is generally written as a linear combination of variables as follows: y = β0 + β1x1 + β2x2 + ... + βnxn + ε. Here, y is the variable to explain, the x's are the explaining variables, and ε is a random noise that can be explained by the x's. It is generally a Gaussian-distributed random variable of mean 0 and variance σ2. What does it mean in practice? The intuition behind this model is that each x after being rescaled will contribute a little bit to y. In other words, y is made of a sum of little pieces, each being an x. [ 167 ]

Bayesian Modeling – Linear Models There are many ways to estimate the value of the parameters from a dataset, and obviously in many situations the value of each parameter is of the utmost importance and needs to be carefully studied. The most used method is the least square method in which one tries to minimize the difference between the real y and its approximation by a sum of x's. Indeed, representing y as a sum of other variables is, as with many models, just an approximation of the reality. Many mathematical tools and algorithms have been developed for linear regression to answer the question of the quality of the model and its parameters.

The word difference is just an analogy. In this case, the correct term is mean squared error, of course.

In this chapter, we will quickly cover the basics of linear regression. A full-scale study would be beyond the scope of this book and we assume the reader has been exposed to such models before. The aim of this chapter is to go further and give a Bayesian avor to the linear regression. In fact, in the standard model, one only focuses on the expectation of y and the parameters. But as soon as each of these components is considered as a random variable, it is possible to explain the linear regression in a Bayesian way and open oneself to many new techniques and bene ts of dealing with full-probability distribution instead of just their expectations. We will review the following elements in this chapter:

* What is a linear regression and how to use it in R?
* What are the main hypotheses in a linear regression and what to do when they break?
* How to compute the parameters by hand and in R
* How to interpret a linear regression as a probabilistic graphical model
* How to estimate the parameters in the Bayesian way and what's the benefit of doing so?
* A review of R packages for Bayesian linear regression
* What is over-fitting, why is it so important to avoid it, and what is the Bayesian solution to it? [ 168 ]

## 线性回归

We start by looking at the most simple and most used model in statistics, which consists of tting a straight line to a dataset. We assume we have a data set of pairs (xi, yi) that are i.i.d and we want to t a model such that: y = βx +β0 + ε Here, ε is a Gaussian noise. If we assume that xi ε Rn then the expected value can also be written as: yˆ=β0+∑n xiβi i=1 Or, in matrix notation, we can also include the intercept β0 into the vector of parameters and add a column on 1 in X, such that X = (1, x1, ..., xn) to nally obtain: ŷ = XTβ The following gure shows an example (in one dimension) of a data set with its corresponding regression line:

In R, tting a linear model is an easy task, as we will see now. Here, we produce a small data set with an arti cial number, in order to reproduce the previous gure. In R, the function to t a linear model is lm() and it is the workhorse of this language in many situations. Of course, later in this chapter we will see more advanced algorithms:

N = 30  
x = runif(N, 0, 20)  
y = 1.2 \* x + 4 + rnorm(N, 0, 4)  
plot(x, y)  
m = lm(y ~ x)  
xx = seq(0, 20, 1)  
lines(xx, xx \* m$coefficients[2] + m$coefficients[1], col = 2, lw = 2)

In this example, we generate 30 random points between 0 and 20. Then we compute y, on a straight line of slope 1.2 and intercept 4, adding a random noise, with zero mean at a variance of 4. We compute the model in m with the function lm(). Finally we plot the result. Printing the variable m gives the following result:

Call:  
lm(formula = y ~ x)  
Coefficients:  
(Intercept) x  
 3.919 1.238

Here we see that the intercept is 3.919, close to the value of 4, and the slope is 1.238, also close to 1.2. As we added an important noise (of variance 4), it is not surprising to see a difference between the theoretical model and the tted model.

### 估计参数

In order to estimate the parameters, we need a discrepancy measure, or in other words, a function that measures some sort of difference between the model and the data set. Of course, the goal is to minimize this difference. Here the word difference has a very broad meaning and many functions could work. However, it has been found that one of the most useful and practical is the mean squared error de ned by: 21∑N(∑n 2 MSE= Y−XTβ 2 =N i=1 yi − j=1xjβj) [ 170 ]

Chapter 6 To estimate the parameters of the model, we resort to a method we saw earlier, called the maximum likelihood. The likelihood is the probability of observing a data set given some parameters, in this case. We assume as usual the data are identically and independently distributed. This allows us to write the maximum likelihood as follows: ˆ θ =argmaxθ p(D|θ) Here, θ is the set of all parameters θ = {β1, ...βp}. Basically we want to nd the parameters that maximize this probability. As we assume i.i.d data, we can write: ˆ ∏N instead of a big product: ˆ ∑N θ=argmaxθ i=1 p(yi |xiθ) Next, we need an analytical form for the probability and in the linear regression, we saw the target data is Gaussian distributed as follows: p(y | xθ) = N(y | XTθ, σ2) If in the log-likelihood we replace the probability with the density function of a Gaussian, we will obtain: θ=argmaxθ i=1 p(yi |xiθ) Then, to simplify the computations we take the log-likelihood and have a sum ⎡ 1 ⎛ T 2⎞⎤ N ⎢⎛ 1 ⎞2 ⎜−(yi−βXi)⎟⎥ logL(θ)=∑i=1log⎢⎜2πσ2 ⎟ exp⎜ 2σ2 ⎣⎝⎠⎝ ⎠⎦ ⎟⎥ ⎡ 1 ⎛ T 2⎞⎤ N⎢ ⎛1⎞2 ⎜−(yi−βXi)⎟⎥ =∑i=1⎢log⎜2πσ2 ⎟ +logexp⎜ 2σ2 ⎣⎝⎠⎝ ⎠⎦ ⎟⎥ 12 (y −β X ) N ⎛ 1 ⎞2 N i T i =∑i=1log⎜2πσ2 ⎟ −∑i=1 2σ2 ⎝⎠ N11∑N 2 = 2 log2πσ2 −2σ2 i=1(y −βT X ) ii [ 171 ]

Bayesian Modeling – Linear Models This long development leads to two terms in the last equation; one is the constant and the other one depends on the parameters. As our goal is to maximize this expression, we can get rid of the constant term without loss of generality. And because it easier (in most numerical packages) to minimize a function than to maximize it, we will take the negative of this: N L L ( θ ) = 12 ∑ N ( y i − β T X i ) 2 i=1 And we nd the famous results again: the maximum likelihood estimator for a linear regression model is nothing but minimizing the square error! In order to nd the solution, we need to do a bit more linear algebra again. First we write this expression in matrix form to make things simpler: 1T NLL(θ)=2(y−Xβ) (y−Xβ) Here, X is the design matrix—that is, the matrix of all the data set. Each row i of the matrix is a vector (x(i),..., x(i) ). This form is very convenient so that we deal with 1p the sum from within the scalar product. After expanding this expression we obtain: NLL(θ)=1βT XT Xβ+1(yT y−yT Xβ−βT XT y) 22 This again can be simpli ed. Indeed, we are only interested in the term dependent on the parameters. So the rest can be discarded. Moreover, we know that y ε RN and the same for βTXT so we can write yTXβ = βTXTy, which helps us to simplify the main expression again to nally obtain: NLL (θ ) = 1 β T X T X β − β T X T y 2 [ 172 ]

Chapter 6 The minimum of this convex function is reached when its rst derivative (in fact the Jacobian matrix) is equal to 0. So we derive it and obtain: ∂NLL(θ)=(XT X)β−XT y ∂β Solving this equation to zero, the solution is nally: ˆ −1 β=(XTX) XTy This is what the function lm() computes in R and most numerical languages or packages. However, we advise you not to implement it directly, especially when the data set is big. Indeed, inverting such a matrix like that could lead to a numerical instability that is hard to control. The main problem when doing linear regression arises when the parameters are not stable and a little change can have massive effects on parameters. This could be due to collinearity between parameters, parameters canceling each other, or many other reasons. One technique to solve this problem is known as shrinkage and its goal is to constrain the parameters to that they don't grow too far away. This usually gives better models with better predictive power. One simple approach to shrinkage is to put a Gaussian prior on the parameters. This technique is called ridge regression or L2 penalization. Practically speaking, we assume the following priors on the parameters: p(β)=∏jN(βj |0,τ2) Here, τ controls the amount of shrinkage one wants to apply to the model and the Gaussian distributions are zero-centered. The last point means we want the parameters not to go too far away from 0. So, the optimization problem to solve becomes the following: argmaxβ∑N logN(yi |β0 +βTxi,σ2)+∑D logN(βj |0,τ2) [ 173 ] i=1 j=1

Bayesian Modeling – Linear Models To be brief, we give the solution directly, because the calculus is similar to the previous development. The negative log-likelihood is therefore: 2 1T2 NLL(θ)= (y−XTβ)(y−XTβ)+λβ N σ2 The only difference, really, is that the last term: λ = τ 2 controls the amount of penalization. The higher this term, the more the parameters will be penalized when they grow too much. A good amount of penalization can lead some of the parameters to become close to zero. In that case, it can be a good idea to try to t the model again without those parameters. In a sense, it is a method for variable selection. A large τ will reduce λ, which follows the intuition that, if the prior on the parameters has a large variance, then, a broad range of values can be reached. If the variance is, on the contrary small then only a smaller range will have a high probability to be reached. The solution to this optimization problem is: −1 βridge =(λID +XT X) XT y Again, we advise you not to compute this directly in R but rather to rely on packages that have been speci cally implemented to be numerically stable. We recommend two packages:

* MASS with the function lm.ridge(), which is similar to lm()
* glmnet with its function, glmnet()

The second package implements several algorithms. You can use it for ridge regression, and for L1-penalization (Lasso). In L1-penalization, instead of using a Gaussian prior on the parameters, we have a Laplace prior. The Laplace distribution is very peaky in the center and this has a particular effect: collinear parameters can reach the value 0 exactly. In that case, they are simply eliminated from the model. It's a very powerful variable selection method. However, the problem doesn't have an analytical solution and needs a speci c optimization algorithm to nd the solution. [ 174 ]

Chapter 6 In R, we can t a model with lm() and glmnet() as follows, where we use the mtcars data set included in R directly, where we want to regress the variable mpg against the other variables:

m1 <- lm( mpg ~ . , mtcars)  
m1  
Call:  
lm(formula = mpg ~ ., data = mtcars)  
Coefficients:  
(Intercept) cyl disp hp drat  
wt  
 12.30337 -0.11144 0.01334 -0.02148 0.78711  
-3.71530  
 qsec vs am gear carb  
 0.82104 0.31776 2.52023 0.65541 -0.19942  
We can plot the model and see the theoretical line with the real data set:  
yhat <- ( as.matrix(mtcars[2:10] ) %\*% m1$coefficients[2:10] ) +  
m1$coefficients[1]

Also note that we use the scalar product %\*%:

plot(sort(mtcars$mpg))  
lines(sort(yhat), col = 2)

## 贝叶斯线性模型

In this section, we are going to extend the standard linear regression model using the Bayesian paradigm. One of the goals is to put prior knowledge on the parameters of the models to help to solve the over- tting problem.

### 模型过拟合

One immense bene t of going Bayesian when doing a linear model is to have better control of the parameters. Let's do an initial experiment to see what happens when the parameters are completely out of control. We are going to generate a simple model in R and look at the parameters when they are tted with the standard approach for linear models. Let's rst generate some data points at random to obtain 10 variables and plot them:

N <- 30  
x <- runif(N, -2, 2)  
X <- cbind(rep(1, N), x, x^2, x^3, x^4, x^5, x^6, x^7, x^8)  
matplot(X, t = "l")

Next we generate the dependent variable following the model: y = Xβ + ε

Here, ε is a Gaussian noise of variance σ2. We use the following code in R and plot the variable y. As we use randomly generated numbers, your plot might be different from the one in this book:

sigma <- 10  
eps <- rnorm(N, mean = 0, sd = sigma)  
y <- X %\*% true\_beta + eps  
plot(y, t = "l")

Then we estimate the coef cients of the model with the lm() function:

model <- lm(y ~ ., data = data.frame(X[, 2:ncol(X)]))  
beta\_hat <- model$coefficients

We plot the true coef cients and the estimated coef cients to see, visually, how close they are. And the result is ... not good!

If we have a closer look at the values, we can clearly see that the model tried to use all the variables when we gave them a zero coef cient in the true model. Moreover, it tried to compensate between all the variables, hence the positive and negative values all along the vector of parameters. Your results might be different from the values shown here, because we use random data, but the behavior will be similar:

> true\_beta  
[1] 10 -3 0 8 0 0 0 0 0  
> beta\_hat  
(Intercept) x V2 V3 V4 V5  
V6 V7 V8  
 10.012121 -30.091272  
 6.844616 4.410177  
 8.904295 62.005179 -12.913125 -28.102293  
-1.154756

In fact, most of the values are terribly wrong. This is a perfect example of over- tting. The model tried to t the data perfectly, ending in something completely off:

> true\_beta  
[1] 10 -3 0 8 0 0 0 0 0  
> beta\_hat  
 (Intercept) x V2 V3 V4 V5 V6 V7 V8  
10.012121 -30.091272 8.904295 62.005179 -12.913125 -28.102293  
6.844616 4.410177 -1.154756

In this case, we knew the true values beforehand but, in practice, we try to t a model on a dataset to nd out good values for the parameters. As we saw in the previous section, one good technique is called regularization and it is equivalent to placing prior distribution on the parameters. By doing so, we somehow constrain the parameters to stay within an acceptable range of values, with a higher probability.

### 线性模型的图模型

Before going further, we need to visualize the structure of a linear mode and better understand the relationships between variables. We can, of course, represent it as a probabilistic graphical model. The linear model captures the relationships between observable variables x and a target variable y. This relation is modeled by a set of parameters θ. But remember the distribution of y for each data point indexed by i: yi ~ N(Xiβ, σ2) Here, Xi is a row vector for which the rst element is always 1 to capture the intercept of the linear model. If you look at earlier pages in this chapter, you will realize that the linear model has been written with many different forms that are all equivalent. We leave it to the reader as an exercise to show they are all equivalent. For example, Xi could be a column vector, and so on. So our rst graphical model could be as follows:

The parameter θ is itself composed of the intercept, the coef cients β for each component of X, and the variance σ2 in the distribution of yi.

So, this decomposition leads us to a second version of the graphical model in which we explicitly separate the components of θ:

We introduce again plate notation in probabilistic graphical models. When a rectangle is drawn around a set of nodes with a number or a variable in a corner (N for example), it means the same graph is repeated many times. The likelihood function of a linear model is: L(θ)=∏N p(yi |Xi,β,σ) i=1 This form is representable as a graphical model and, based on the previous graph, we nally have:

In this graph, it is clearly stated that each yi is dependent on one xi. It is also clear that the parameters θ = {β,σ} are all shared as they are outside the plate. [ 180 ]

Chapter 6 For the sake of simplicity, we will keep β as a vector, but you could also decompose it into its univariate components and use the plate notation for those:

In the last two iterations of the graphical model, we see that the parameter β could have a prior probability on it instead of being xed. In fact, the parameter σ can also be considered as a random variable. For the time being, we will keep it xed.

### 后验分布

Many prior distributions could be used for β but they need to be tractable or simply have an appropriate meaning in the context of a linear regression. Here we need a distribution that has a zero mean, is symmetric, and has an in nite support. The reasons are:

* Zero-mean because we want our parameters to be driven toward zero if possible; this is the shrinkage effect. So we give the highest mass of probability to the center at zero.
* Symmetric because we want to give equal chances for positive and negative values. A priori, we don't know which direction the parameters will take.
* Infinite support because we don't want to block the parameters to have certain values. Obviously, with most of the probability mass in the center, the tail of the distribution, despite having an infinite support, will have a low probability. So we are trying to force the model not to have huge values like we saw in the previous example.
* The distribution needs to be simple enough that we can compute the posterior of the parameters and the predictive posterior of y.

Bayesian Modeling – Linear Models Given all these reasons, the Gaussian distribution seems to be a good candidate for our purpose. The conditional distribution for y is as usual: p(yi |Xiβσ2)=N(yi |Xβ,σ2) We remember we saw that the Maximum Likelihood Estimator (MLE) is: ˆ −1 β=(XTX) XTy And the estimator for the variance can also be estimated (left as an exercise) and is: 1∑N( ˆ 2 σˆ=N i=1 yi−βTXi) The good thing about having a Gaussian prior on the parameters is that it is conjugate to the likelihood function. It means that the posterior of the parameters is also a Gaussian distribution such that: Here: p(β | yXστ)∝ p(y|βXστ)p(β |τ)= N(β |m,S) m=σ−2SXT y −1 S = (τ I + σ −2 X T X ) Here we nd again the τ parameter, which controls how large the prior is on the parameters. It is the same parameter τ we used in the previous section on ridge regression. In fact, it can be shown that having Gaussian priors on the parameter β and the ridge regression are equivalent. The reader is encouraged to take the previous two formulas and calculate again the ridge regression to see the relation between the two. [ 182 ]

Chapter 6 Finally, the last thing we need to do is to compute the posterior predictive distribution. The posterior predictive distribution is the distribution of an unknown y when some Xs have been observed after computing the parameters. It is basically making a prediction with a model that has been learned with the previous method. The reason it is important is because we do a complete Bayesian treatment of the problem, instead of a pure statistical one. In the standard linear model, the scalar product of X and β is suf cient to compute the expected value of y when we wish to make a prediction. In other words, after nding the β parameters we simply do: yj =∑N xi.βi i=1 But because here we have a full Bayesian model, instead of just having the expectation of y, we have the full probability distribution. As we will see, the posterior distribution is also a Gaussian, and a Gaussian is de ned by its mean and variance. Therefore, by using a full Bayesian model, we also compute the posterior variance and have an estimation of the uncertainty of the prediction: Here: p ( y ′ | yτ σ 2 X ) = ∫ p ( y ′ | β σ 2 ) p ( β | yτ σ 2 ) d β = N(y′|mT X′,σ2 (X′)) σ2(X′)=σ2 +X′TSX′ X' and y' are respectively the new observed data upon which we want to do the prediction y'. [ 183 ]

Bayesian Modeling – Linear Models Finally, we can draw the graphical model for the full Bayesian interpretation of the linear model:

In fact, we only presented the case when the prior on β is Gaussian, but other priors such as a Laplace distribution can be used. This leads to an L1-penalization that doesn't have an analytical form. However, a very ef cient algorithm called the Lasso can be used to nd the parameters. A very ef cient implementation of it can be found in the glmnet package.

### R语言实现

Let's take an example from the beginning of this chapter again. When we tried to compute the parameters, we found strange values that were seriously off from their true value, meaning there was a problem in the estimation procedure. We saw that this problem is called over- tting. Then we looked at a solution by interpreting the linear model in the Bayesian framework and calculated the solutions of the problem. Implementing it is really easy when the priors on the parameters are Gaussian:

dimension <- length(true\_beta)  
lambda <- diag(0.1, dimension, dimension)  
posterior\_sigma <- sigma^2 \* solve(t(X) %\*% X + sigma^2 \* lambda)  
posterior\_beta <- sigma^(-2) \* as.vector(posterior\_sigma %\*% (t(X) %\*% y))

The posterior parameters are now:

posterior\_beta  
[1] 7.76069781 -0.06509725 1.18834799 2.72321814 0.16637478  
2.65759764  
[7] -0.10993147 -0.31961733 0.02273269

This is far better than what we had before. But it is still not perfect. Indeed, the true β parameters are:

true\_beta <- c(10, -3, 0, 8, 0, 0, 0, 0, 0)

And we can see that the second parameter is too small and many parameters are too high, with values going from 1 to 2 when they should be zero. Indeed, the penalization in the form of the lambda variable is too weak. It means the variance is too large. We can therefore give more penalization by doing:

lambda <- 0.5 \* diag(0.1, dimension, dimension)

We then recompute the results:

posterior\_beta  
[1] 9.6677088 -0.7393309 1.1248994 3.5526708 -0.1869873 2.8805658  
-0.3506464  
[8] -0.4582813 0.1190531

It is not perfect but the intercept is now closer to 10 (9.66) and the second parameter now has a better value. The other parameters are still off but they are going in the right direction. We can penalize even more and run the same code again:

lambda <- 0.1\*diag(0.1, dimension, dimension)  
posterior\_sigma <- sigma^2 \* solve(t(X) %\*% X + sigma^2 \* lambda)  
posterior\_beta <- sigma^(-2) \* as.vector(posterior\_sigma %\*% (t(X) %\*%  
y))  
posterior\_beta  
[1] 12.0750175 -3.8736938 0.6105363 8.0942494 -1.0959572 1.3938047  
-0.1099443  
[8] -0.3496412 0.1280143

Despite it not being perfect, we see that we are closer to the true solution. Needless to say, the example we used has been speci cally made to be hard to solve. Despite this, the Bayesian solution is able to converge to the true solution, where the simple linear regression was completely wrong. After running the previous model, we can plot the results with the following code:

t <- seq(-2, 2, 0.01)  
T <- cbind(rep(1, N), t, t^2, t^3, t^4, t^5, t^6, t^7, t^8)  
  
plot(x, y, xlim = c(-2, 2), ylim = range(y, T %\*% true\_beta))  
lines(t, T %\*% true\_beta, col = "black", lwd = 3)  
lines(t, T %\*% beta\_hat, col = "blue", lwd = 3)  
lines(t, T %\*% posterior\_beta, col = "red", lwd = 3)  
legend("topleft", c("True function", "OLS estimate", "Bayesian  
estimate"),   
 col = c("black", "blue", "red"), lwd = 3)

The rst two lines generate evenly spaced data points to draw the results. Then the rst plot draws the dataset (little black circles). Then we draw three curves on it:

* In black: This is the true model as defined in the R program
* In blue: This is the OLS estimate (the standard linear regression)
* In red: This is the Bayesian estimate with the penalization we found earlier

The blue curve (OLS estimate) tries to follow the data points as much as possible, tting more noise and being far away from the true function. This is a good example of over- tting. On the contrary, the red curve (Bayesian estimate) did a good job nding the true function that generated the data in the beginning:

Now we want to add to this graph the 95% posterior predictive interval. Thanks to the Bayesian approach, we can easily compute it from the posterior distribution. Therefore the code in R will be as follows:

pred\_sigma <- sqrt(sigma^2 + apply((T %\*% posterior\_sigma) \* T, MARGIN = 1,   
 FUN = sum))  
upper\_bound <- T %\*% posterior\_beta + qnorm(0.95) \* pred\_sigma  
lower\_bound <- T %\*% posterior\_beta - qnorm(0.95) \* pred\_sigma

The previous code computes the upper and lower bound along the dataset we used. And nally the following code draws the plot:

plot(c(0,0),xlim=c(-2,2), ylim=range(y,lower\_bound,upper\_  
bound),col='white')  
polygon( c(t,rev(t)), c(upper\_bound,rev(lower\_bound)), col='grey',  
border=NA)  
points(x,y)  
lines(t,T%\*%true\_beta, col='black', lwd=3)  
lines(t,T%\*%beta\_hat, col='blue', lwd=3)  
[ 187 ]  
   
Bayesian Modeling – Linear Models  
lines(t,T%\*%posterior\_beta, col='red', lwd=3)  
legend('topleft', c('True function', 'OLS estimate', 'Bayesian  
estimate'), col=c('black','blue','red'), lwd=3)

In this code we use the polygon function to draw the gray area representing the 95% predictive interval. We use the qnorm function to compute the values and you can play with theses values to change the interval.

### 一种稳定的实现

In the previous implementation, we used the solve() function from R, but it is not always a good idea to inverse a matrix directly as it can quickly lead to numerical instability. As a quick example, here is a little piece of code that generates random invertible matrices and computes the Froebenius distance between an identity matrix and the result of the random matrix multiplied by its inverse. If M is a matrix and M-1 its inverse, then M.M-1=I. We see in this little example that it is not numerically the case:

N <- 200  
result <- data.frame(i=numeric(N),fr=numeric(N))  
 for(i in 2:N)  
[ 188 ]  
   
{  
}  
x <- matrix(runif(i\*i,1,100),i,i)  
y <- t(x)%\*%x  
I <- y%\*%solve(y)  
I0 <- diag(i)  
fr <- sqrt(sum(diag((I-I0)%\*%t(I-I0))))  
result$i[i] <- i  
result$fr[i]<- fr  
}

The code generates square matrices of size going from 2 x 2 to 200 x 200 and computes the Froebenius distance between a perfect identity matrix and the identity matrix obtained by multiplying the random matrix with its inverse. Plotting the result shows that the distance is not zero all the time:

In fact, it increases with the size of the matrix, when more errors accumulate. If we plot the log of the distance, we clearly see the error getting bigger:

With a perfect inverse matrix, the distance would be zero all the time and therefore the log would be—in nite. So this simple example tells us that in Bayesian linear regression we can have problems when inverting the matrix X. We therefore need a better algorithm. Here we present a simple algorithm to solve the ridge regression problem with a numerically stable solution. The idea is to transform the problem of matrix inversion into a simpler problem to solve, where the matrix to inverse is triangular. If X is the matrix containing the data points, and y the vector containing the target, the idea is to rst extend the matrix and the vector y as follows: ⎛X⎞ X=ˆΛ=I ˆ ⎛yσ⎞11 ⎜Λ⎟andy=⎜0 ⎟where τ andΛ=τ2I ⎝⎠⎝D⎠ [ 190 ]

Chapter 6 The next step is to do a QR decomposition of Xˆ and the last step is to compute the inverse of R, which is easier because it's an upper triangular matrix. Finally, the coef cients of the linear regression are given by: ˆ β = R − 1 Q T yˆ The algorithm in R can be implemented as follows:

# the numerically stable function  
ridge <- function(X, y, lambda) {  
 tau <- sqrt(lambda)  
 Xhat <- rbind(X, (1/tau) \* diag(ncol(X)))  
 yhat <- c(y, rep(0, ncol(X)))  
 aqr <- qr(Xhat)  
 q <- qr.Q(aqr)  
 r <- qr.R(aqr)  
 beta <- solve(r) %\*% t(q) %\*% yhat  
 return(beta)  
}

This algorithm returns a vector of coef cients, where the rst value is the intercept. We assume here that the matrix X has a column of one on the left-hand side. We use the qr() function from R to do the QR decomposition. The following code runs an example:

set.seed(300)  
N <- 100  
# generate some data  
x <- runif(N, -2, 2)  
beta <- c(10, -3, 2, -3, 0, 2, 0, 0, 0)  
X <- cbind(rep(1, length(x)), x, x^2, x^3, x^4, x^5, x^6, x^7, x^8)  
y <- X %\*% beta + rnorm(N, 0, 4)

First of all, we generate random data. The reader will note that for the rst time we set the random seed manually so that the following results can be exactly reproduced. The reason for that is that we want to illustrate a nice behavior of the ridge regression. The code generates random values on the x axis, then we give the true beta coef cients, and nally we generate the data. We also add a random Gaussian noise to the target data y so as to test the capacity of the ridge regression with respect to the standard linear regression solved by an OLS. The X matrix has many columns but only four of them (plus the intercept) are used to generate y so we expect the linear regression to give very small coef cients to the unused columns (or even zero coef cients). Then we run the following code to generate results and plot them:

# plot the results  
t <- seq(-2, 2, 0.01)  
Xt <- cbind(rep(1, length(t)), t, t^2, t^3, t^4, t^5, t^6, t^7, t^8)  
yt <- Xt %\*% beta  
yridge <- Xt %\*% ridge(X, y, 0.9)  
plot(x, y)  
lines(t, yt, t = "l", lwd = 2, lty = 2)  
lines(t, yridge, col = 2, lwd = 2)  
olsbeta <- lm(y ~ X - 1)  
olsy <- Xt %\*% olsbeta$coefficients  
lines(t, olsy, col = 3, lwd = 2)

First of all, we generate a sequence of points on the x axis, then we compute the X matrix and then the real function called yt. This is our theoretical model without noise. [ 192 ]

Chapter 6 The next step is to compute the ridge regression coef cients with our new ridge function as de ned previously. Finally, we compute a standard OLS solution to the same problem. And we plot the results:

The gure shows the real data set as little circles. The dashed black line is the true function, the one we called our theoretical model before. It is the function without the noise. The red line is very close to the true function. This is the ridge regression. Because of the shrinkage effect of the ridge regression, it is less sensitive to the noise and gives a better solution. However, the green line is the standard OLS function. It has a wiggly behavior because it is more sensitive to the noise. This is an illustration of the over- tting problem. In this example, the OLS tries too hard to be close to the data and ends up with a solution that is more unstable than the ridge regression. [ 193 ]

Bayesian Modeling – Linear Models In order to illustrate the last point a bit more, we run a nal example in which we exaggerate the noise. Instead of having a standard deviation of 4, we use the value 16, making the data very noisy:

Obviously, this is an extreme example, but we see again that the ridge regression stays closer to the true function. The OLS solution, on the other hand, became completely unstable and is totally over- tting.

### 更多R语言程序包

Bayesian linear regression is a well-covered topic in R and many packages implement such models. Of course we mentioned glmnet before, which implements ridge regression and Lasso. It also implements a mixture of both, called the elastic net, in which two penalty functions are used at the same time. Another package is bayesm, which covers linear and multivariate linear regression, multinomial logit and probit, mixtures of Gaussians, and even Dirichlet process prior density estimation. [ 194 ]

Chapter 6 We can also mention the arm package, which provides Bayesian versions of glm() and polr() and implements hierarchical models. It is another very powerful package. Of course, we shouldn't stop here and ought to extend our study of Bayesian models to all sorts of algorithms and prior distributions. At some point, it becomes impossible to nd an analytical solution and one should use Monte-Carlo inference, as we saw in the previous chapter.

## 小结

In this chapter, we saw the standard linear model. This model is one of the most important models in statistics and provides a simple, additive way to represent relationships between observed variables and a target. Estimating good parameters for a linear model can be hard sometimes and one should be very careful not to trust the results immediately. However, a Bayesian approach to the problem helps to include prior knowledge into the model and drive it toward a more stable and usable solution. We saw ridge regression and Bayesian linear regression. We saw that, when the parameters have a Gaussian prior, then these two approaches are equivalent and very easy to compute. Using a simple example, we saw that a standard regression can lead to completely over- tted results and that the Bayesian approach solved the problem. In the next chapter, we will look at more advanced models for dealing with clusters of data, called mixture models. These models make the assumption that the data is generated by a different group and the goal will be to automatically discover the group and reveal the hidden process behind it. [ 195 ]

# 第7章 概率混合模型

We have seen an initial example of mixture models, namely the Gaussian mixture model, in which we had a nite number of Gaussians to represent a dataset. In this chapter, we will focus on more advanced examples of mixture models, going again from the Gaussian mixture model to the Latent Dirichlet Allocation. The reason for so many models is that we want to capture various aspects of the data that are not easily captured by a mixture of Gaussian. In many cases, we will use the EM algorithm to nd the parameters of the model from the data. Also, it appears that most of the mixture models can have intractable solutions and need solutions on approximate inferences. The rst type of model we will see is a mixture of simple distributions. The simple distribution can be a Gaussian, a Bernoulli, a Poisson, and so on. The principle is always the same but the applications are different. If Gaussian distributions are nice for capturing clouds of points, Bernoulli distributions can be ef cient to analyze black and white images, for example, in handwritten recognition. We will then relax one assumption of the mixture model and see a second type of model called mixture of experts, in which the chosen cluster is dependent on the data point. It can be seen as a rst approach to probabilistic decision trees. Finally, we will see a very powerful model called the Latent Dirichlet Allocation (LDA), in which we relax another assumption of the mixture models. In mixture models, a point is supposed to have been generated by one cluster. In the LDA, it can belong to several clusters at the same time. This model has been successfully used in text analysis, among other things. It belongs to a family of mixed memberships models. [ 197 ]

Probabilistic Mixture Models We will review the following elements in this chapter:

* Mixture models in general, with examples of several distributions
* Mixture of experts, when we assume clusters are dependent on the data points
* LDA when we assume a point belongs to several clusters

## 混合模型

The mixture model is a model of a larger distribution family called latent variable models, in which some of the variables are not observed at all. The reason is usually to simplify the model by grouping all the variables into subgroups with a different meaning. Another reason is also to introduce a hidden process into the model, the real reason for the data generation process. In other words, we assume that we have a set of models and something hidden will select one of these models, and then generate a data point from the selected model. When the data naturally exhibits clusters, it seems reasonable to say that each cluster is a small model. The whole problem is then to nd to what extent a submodel will participate in the data generation process and what the parameters for each sub model are. This is usually solved using the EM algorithm. There are many ways to combine small models in order to make a bigger or more generic model. The approach generally used in mixture modeling is to give a proportion to each sub model, such that the sum of proportions is one. In other words, we build an additive model as follows: p(xi |θ)=∑K πk pk (xi |θ) i=1 In this, πk is the proportion of each sub model. And each sub model is captured by the probability distribution pk. Of course, in this form, the sum of πk is 1. Also, the proportions can be considered as random variables and the model can be extended in a Bayesian way. The model is therefore called a mixture model and the probability distribution pk is called the base distribution. [ 198 ]

Chapter 7 There are, theoretically, no constraints on the form of the base distribution and, depending of the function, several types of model arise. In Machine Learning: A Probabilistic Perspective, the following taxonomy helps us to understand many popular models:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Base distribution | Latent var. distribution | Notes |
| Mixture of Gaussian | Gaussian | Discrete | A Gaussian is chosen among K |
| Probabilistic PCA | Gaussian | Gaussian | . |
| Probabilistic ICA | Gaussian | Laplace | Used for sparse coding |
| Latent Dirichlet Allocation | Discrete | Dirichlet | Used for text analysis |

These are just a few examples to show that many models are possible based on the same principle. However, it does not mean they are all easy to solve and, in many cases, advanced algorithms will be necessary. For example, the mixture of Gaussian model is de ned as follows: we consider that each sub model is a Gaussian distribution (base distribution) and the latent variable distribution is discrete. For each distribution, we have a mean and a variance. Sampling from such a model could give the following data set, for example:

Probabilistic Mixture Models The base density is: p(x |θ,z =k)=N(x |μ ,σ2) iiikk And the latent variable distribution is a categorical distribution Π = (π1,...,πK ). The model is therefore: p(xi |θ)=∑K N(x |μ ,σ2) i=1 ikk In the case of a multidimensional Gaussian, the variance σ2k will be replaced by the covariance matrix Σk.

## 混合模型的期望最大化

The standard way for tting mixture models is the EM algorithm or Expectation Maximization. This algorithm was the focus of Chapter 3, Learning Parameters. So here, we just recall the basic principles of this algorithm again, to later show a Bernoulli mixture model. A good package to use in R is mixtools to learn mixture models. A thorough presentation of this package is given in the Journal of Statistical Software, Oct 2009, Vol 32, Issue 6, mixtools: An R Package for Analyzing Finite Mixture Models. The EM algorithm is a good choice for learning a mixture model. Indeed, in Chapter 3, Learning Parameters, we saw that when data is missing or even when variables are hidden (that is, all their respective data is missing), the EM algorithm will proceeds in two steps: rst compute the expected value of the missing variables, so that to do as if the data is fully observed, and then maximize an objective function, usually the likelihood. Then, given the new set of parameters, the process is iterated again until some convergence criterion is reached. [ 200 ]

In the mixture model represented by the preceding graphical model, it is clear that the variable z is the latent variable and the xi is observed. We usually adopt plate notation to give a comprehensive view of the data generation process as a graphical model and use the following graph:

As in many probabilistic models, tting the parameters can be solved by nding the set of parameters that maximizes the probability to generate the data. In other words, we aim at maximizing the log-likelihood of the model: L(θ)=∏N p(xi |θ)=∑N log(∑K p(xi,zi |θ)) i=1 i=1 zi And that's where the problem is because the sum inside the log is hard to reduce and in many cases it is intractable. You will also note that this likelihood is written in terms of the observed data. So, if we follow the previous graphical model, we can write the complete log-likelihood by introducing the latent variables as follows: Lc(θ)=∑N logp(xi,zi|θ) i=1 The EM algorithm will solve the problem of computing this likelihood, in which z is completely hidden, by performing the following two steps. First of all, we de ne the expected complete data log likelihood by: Q(θt,θt−1)=E(Lc (θ)|Dataθt−1) This expectation is the expected complete data log likelihood computed from the parameters found in the previous step. Of course, at the beginning of the algorithm, the parameters are initialized to some arbitrary value. We saw in Chapter 3, Learning Parameters that this value can be anything, but choosing values at random can lead to a very long convergence time. Nevertheless, it has been proved that the EM algorithm converges for any value. The E-step in the EM algorithm will compute this expected value—that is, the expected parameters given the data and the previous parameters. Then the M-step will maximize this expectation given the newly found set of parameters θ that will solve the problem: θt =argmaxθQ(θt,θt−1) [ 202 ]

Chapter 7 In the mixtools package, it is possible to t a mixture of Gaussian using the function normalmixEM. Here is how to do so. First, we generate a simple data set of two univariate Gaussian:

x1 ← rnorm(1000,-3,2)  
x2 ← rnorm(850,3,1)

Then, we plot the result to see how they are empirically distributed using the hist function:

hist(c(x1, x2), breaks = 100, col = "red")

And we obtain the following gure, where we can easily identify the two clusters of data and see they are approximately distributed. Given that we use random generators, your result might be slightly different from what is shown in this book:

model ← normalmixEM( c(x1,x2) , lambda=.5, k=2)

[ 203 ]

Probabilistic Mixture Models This model should take between 30 to 40 iterations to run. We give an initial proportion of 50% to each class and set the number of clusters to 2. In our results, we obtain the following parameters:

* The mixing proportions are 54.9% and 45.1%. This precisely corresponds to the proportion we gave to our data sets x1 and x2.
* The μ parameters are -2.85 and 3.01, which are extremely close to the initial values we gave too.

We can plot this histogram again with the Gaussian distributions superimposed on it:

hist(c(x1, x2), breaks = 100, col = "red", freq = F, ylim = c(0, 0.4))  
lines(x, dnorm(x, model$mu[1], model$sigma[1]), col = "blue")  
lines(x, dnorm(x, model$mu[2], model$sigma[2]), col = "green")

The result is obviously far off what we expected in terms of proportion. If we redraw it by adding the proportions now, we obtain the expected mixture distribution:

hist(c(x1,x2),breaks=100,col='red',freq=F)  
lines(x,  
 model$lambda[1]\*dnorm(x,model$mu[1],model$sigma[1]) + model$lambda[2]\*d  
norm(x,model$mu[2],model$sigma[2]) ,  
lwd=3)

The number of clusters is very important and can change the results dramatically when not chosen properly. For example, if we try the following values, we end up with different results:

model <- normalmixEM(c(x1,x2), lambda=.5, k=3)  
number of iterations= 774  
model <- normalmixEM(c(x1,x2), lambda=.5, k=4)  
WARNING! NOT CONVERGENT!  
 number of iterations= 1000

With three clusters, the EM algorithm still converges. With four clusters, we need to increase the number of iterations for the algorithm to converge. In fact, even with three clusters, the result is interesting if not surprising. Plotting the density function, we obtain:

In this plot, we see that the second cluster is somehow attached to the rst one. If you carefully look at the thick black line in the middle, you will see that the left-hand side distribution is not totally Gaussian. Inspecting the model parameters, we see that the means of the Gaussian are -3.74, -1.08, and 2.9. The middle one is indeed closer to the rst cluster. The proportion is interesting: 38%, 15.4%, and 46.5%. So it seems the EM algorithm split the biggest cluster (1,000 points against 850) into 2 Gaussian to respect our choice of three clusters. A slow convergence can sometimes be an indication that our hyperparameters are not totally adequate and more values should be explored. [ 206 ]

## 伯努利混合

The mixture of Bernoulli is another interesting problem. As we saw earlier, the graphical model to represent such a model is always the same. Only the probability distribution functions associated to each node change compared to the previous example. The mixture of Bernoulli nds applications in particular in analyzing black and white images, where an image is made of one Bernoulli variable for each pixel. Then the goal is to classify an image, that is, to say which value of the latent variable produced it, given some observed pixels. For example, the following (very stylized) gure represents the letter A: In a real application, we would use more pixels. But the idea of the application is to associate the value of pixels to each value of the latent variable, each one representing a different letter. This is a simple model for character recognition. The distribution of a Bernoulli variable is: 1−x p(x|θ)=θx (1−θ) Now, let's say we have D Bernoulli variables. Each Bernoulli variable is parameterized by one parameter θi. So the likelihood of such a model can be written as: i=1 Here, X =(x1, ..., xD) and θ = (θ1, ..., θD). ii ∏D (1−x ) p(X|θ)= θxi (1−θ) i [ 207 ] Chapter 7

Probabilistic Mixture Models If we introduce the mixture of all the variables then the joint distribution can be written as: p(X|θπ)=∑K πKp(x|θk) i=1 Here, π = (π1, ...πD) are the mixing parameters and the distribution p is inside as: In order to t this model, we need now to nd the log-likelihood and, as expected, its expression will not be suitable for a direct optimization. The reason is, as usual, that we introduce latent variables for which we have no observations and we therefore need to use an EM algorithm. The log-likelihood is taken from the main joint distribution: logp(x|θπ)=∑N log(∑Kπkp(xi|θi)) n=1 i=1 This is a pretty standard way of computing the log-likelihood. And, as is usually the case for mixture models, the sum inside the log cannot be pushed out. So we end up with a quite complex expression to minimize. Now we introduce the latent variable z into the model. It has a categorical distribution with K parameters, such that: p(z|π)=∏K πkzk k=1 And the joint distribution with x is as follows: p(x|zθ)=∏K p(x|θk )zk [ 208 ] ∏D This is in fact the same distribution as before but for a case k only. p(x|θk )= θxi i=1 k,i k,i k=1 ( )1−x 1−θ i

Chapter 7 As before with the Gaussian mixture, we will write the complete data log-likelihood. This likelihood is what we would optimize if the data set were complete—that is, without latent variables: logp(X,Z|θπ)=∑N ∑K n=1 k=1 zn,k (logπk + ⎡xlogθ+1−xlog1−θ⎤ ∑D ( )( )) n,i k,i n,i k,i i=1⎣ ⎦ In this (very long) expression, we consider X and Z to be matrices, so in fact we use the design matrix notation, where each row vector of X (resp. Z) is one set of observations of each variable xi (resp. z). Using the Bayes formula, we can calculate the expectation of the complete-data log-likelihood with respect to the distribution of the latent variable. This step is necessary for the E-step of the EM algorithm, where we want to complete the data set: Ez(logp(X,Z|θπ))=∑N ∑K E(zn,k)(logπk n=1 k=1 And the expected value of the latent variable is: + ⎡x logθ +1−x log1−θ ⎤ ∑D ( )( )) n,i k,i n,i k,i i=1⎣ ⎦ π p(x |θ ) E(z)=∑k i k n,k K π p(x |θ ) j=1 In a sense, this is not surprising because in the end we need to compute the ratio of zi for each subset of the dataset in which it appears after the posterior computations. Finally, in the M-step, we can derive the complete-data log-likelihood with respect to the parameters θk and π and set it equal to zero. We obtain the following estimators: θk=1∑N E(zn,k)xn Nk n=1 πk =Nk jnj [ 209 ] N

Probabilistic Mixture Models Here, Nk =∑N E(zn,k ). n=1 The EM algorithm will alternate between computing the expectation of z and the new values for the parameters θ and π until convergence.

More details about the derivation can be found in Pattern Recognition and Machine Learning, Christopher M. Bishop, Springer, 2007

This model can be extended and the same principles applied to other types of distribution—for example, a mixture of Poisson or Gamma. On the other hand, the mixture of Bernoulli can be extended to the multinomial case with the same type of derivation. In all those models, however, we consider that we have one model for all the data point space. In other words, we somehow use the same model for the whole support of the distribution. One extension is to consider that each subspace has its own model and therefore the choice of the sub model made by the latent variable is dependent on the data points. In Adaptive mixtures of local experts, Jacobs, R.A., Jordan, M.I, Nowlan, S.J., and Hinton, G.E. (1991) in Neural Computation, 3, 79-87, such a model is presented. We give a brief review of this interesting model in its simple form.

## 专家混合

The idea behind mixture of experts is to use a set of linear regressions for each sub space of the original data space and combine them with weighting functions that will successively give weight to each linear regression. Consider the following example dataset, which we generate with the following toy code:

x1=runif(40,0,10)  
x2=runif(40,10,20)  
e1 = rnorm(20,0,2)  
[ 210 ]  
   
e2 = rnorm(20,0,3)  
y1 = 1+2.5\*x1 + e1  
y2 = 35+-1.5\*x2 + e2  
xx=c(x1,x2)  
yy=c(y1,y2)

Plotting the result, and doing a simple linear regression on it, gives the following:

Obviously, the linear regression does not capture the behavior of the data at all. It barely captures a general trend in the data that more or less averages the data set. Chapter 7 [ 211 ]

Probabilistic Mixture Models The idea of mixture of experts is to have several sub models within a bigger model— for example, having several regression lines, as the following graph:

In this graph, the red and green lines seems to better represent the data set. However, the model needs to choose when to choose each one. Again, a mixture model could be a solution, except that, in this case, we want the mixture to be dependent on the data points. So the graphical model will be a bit different: p(y |x,z,θ)=N(y |wTx,σ2) iii ikik This is the linear model as we know it. Next we introduce the dependence of the latent variable to the data points with: p(zi |xiθ)=Mult zi |S VTxi ( ( )) [ 212 ]

Chapter 7 Here, S(.) is, for example, a sigmoid function. The function p(zi | xiθ) is usually called the gating function. The graphical model associated with such a model is quite different now because it introduces a dependency between the latent variable and the observations:

In general, mixture of experts models uses a softmax gating function such that: f (x)= exp(βT x) ∑k exp(βT x) n=1 [ 213 ]

Probabilistic Mixture Models The EM algorithm is usually a good algorithm to t such a model. For example, the mixtools package includes a function hmeME to t mixture of experts models. At the time of writing, this function is limited to two clusters. The combination of all the gating functions requires us to sum to one at each point; for example, in our example we could use two sigmoids with the following effect:

And such a combination could give a nal model that better interprets the initial data set, such as this graph:

We recommend the reader develop his or her own EM algorithm to t such models and try different types of gating functions. Techniques such as shrinkage or using a Bayesian approach on the parameters could be useful to avoid over- tting too, which can be problematic when the number of sub models grows quickly.

## 隐狄利克雷分布

The last model we want to present in this book is called the Latent Dirichlet Allocation. It is a generative model that can be represented as a graphical model. It's based on the same idea as the mixture model, with one notable exception. In this model, we assume that the data points might be generated by a combination of clusters and not just one cluster at a time, as was the case before. [ 215 ]

Probabilistic Mixture Models The LDA model is primarily used in text analysis and classi cation. Let's consider that a text document is composed of words making sentences and paragraphs. To simplify the problem, we can say that each sentence or paragraph is about one speci c topic, such as science, animals, sports, and so on. Topics can also be more speci c, such as cats or European soccer. Therefore, there are words that are more likely to come from speci c topics. For example, the word cat is likely to come from the topic cats. The word stadium is likely to come from the topic European soccer. However, the word ball should come with a higher probability from the topic European soccer, but it is not unlikely to come from the topic cats, because cats like to play with balls too. So it seems the word ball might belong to two topics at the same time with a different degree of certainty. Other words such as table will certainly belong equally to both topics and presumably to others. They are very generic. Unless, of course, we introduce another topic such as furniture. A document is a collection of words, so a document can have complex relationships to a set of topics. But, in the end, it is more likely we will see words coming from the same topic or the same topics within a paragraph, and to some extent in the document. In general, we model a document with a bag-of-words model that is, we consider a document to be a randomly generated set of words, using a speci c distribution over the words. If this distribution is uniform over all the words, then the document will be purely random without a speci c meaning. However, if this distribution has a speci c form, with more probability mass to related words, then the collection of words generated by this model will have a meaning. Of course, generating documents is not really the application we have in mind for such a model. What we are interested in is the analysis of documents, their classi cation, and automatic understanding.

### LDA模型

Let's say zi is a categorical variable (in other words, a histogram) representing the probability of appearance of all words from a dictionary. Usually, in this kind of model, we restrict ourselves to long words and remove the small words such as and, to, but, the, a, and so on. These words are usually called stop words. Let wj be the j-th word in a document. A document generating model could be represented with the following graphical model:

Let θ be a distribution over topics, then we can extend this model by choosing which kind of topic will be selected at any time and then generate a word out of it. Therefore, the variable zi now becomes the variable zij, which is the topic i selected for the word j. The graphical model is extended as follows:

[ 217 ]

Probabilistic Mixture Models We can go even further and decide we want to model a collection of documents, which seems natural if we consider we have a big dataset. Assuming that documents are i.i.d, we can draw the following graphical model again, in which we capture M documents (on the right in the earlier gure). And because the distribution on θ is categorical, we want to be Bayesian about it, mainly because it will help to model (not to over- t) and because we consider the selection of topics for a document to be a random process in itself. Moreover, we want to apply the same treatment to the word variable by having a Dirichlet prior. This prior is used to avoid non-observed words having a zero probability. It smooths the distribution of words per topic. A uniform Dirichlet prior will induce a uniform prior distribution on all the words. The nal graphical model is given by the following gure:

[ 218 ]

Chapter 7 This is quite a complex graphical model but techniques have been developed to t the parameters and use this model. So, if we follow this graphical model carefully, we have a process that generates documents based on a certain set of topics:

* α chooses the set of topics for a document
* From θ we generate a topic zij
* From this topic, we generate a word wj

In this model, only the words are observable. All the other variables will have to be determined without observation, exactly like in the other mixture models. So documents are represented as random mixtures over latent topics, in which each topic is represented as a distribution over words. The distribution of a topic mixture based on this graphical model can be written as: p(θ,z,w|αβ)=p(θ|α)∏N p(zi |θ)p(wi |zi,β) i=1 You see in this formula that for each word we select a topic, hence the product from 1 to N. Integrating over θ, and summing over z, the marginal distribution of a document is as follows: p(w|αβ)=∫p(θ|α)(∏N ∑ p(zi |θ)p(wi |ziβ))dθ i=1 zi The nal distribution can be obtained by taking the product of marginal distributions of single documents, so as to get the distribution over a collection of documents (assuming documents are independently and identically distributed). Here, D is the collection of documents: p(D|αβ)=∏M ∫p(θ |α)(∏Nd ∑ p(z |θ )p(w |z β))dθ d and z given a document. By applying the Bayes formula we know that: p(θ,z|wαβ)= p(θ,z,w|α,β) p(w|αβ) [ 219 ] d,i d d,i d,i The main problem to solve now is how to compute the posterior distribution over θ d=1 i=1 z d,i d

Probabilistic Mixture Models Unfortunately, this is intractable because of the normalization factor at the denominator. The original paper on LDA therefore refers to a technique called variational inference, which aims at transforming a complex Bayesian inference problem into a simpler approximation which can be solved as a (convex) optimization problem. This technique is the third approach to Bayesian inference and has been used on many other problems. In the next section, we brie y review the principles of variational inference and, nally, we will show an example in R to conclude this section.

### 变分推断

The main idea in variational inference is to consider a family of lower bounds indexed by variational parameters and optimize on those parameters to nd the tightest lower bound. Practically speaking, the idea is to approximate a complicated distribution we wish to evaluate by a simpler distribution such that the distance (or any suitable metric between the distributions) can be minimized with a convex optimization procedure. The reason we want things to be convex is essentially because convex problems have a global minimum. In general, a good approximation for a graphical model consists in simplifying the graph of the model by decoupling variables. In practice, we remove edges. In the LDA model, the proposed variational problem is done by decoupling the variables θ and β. The resulting graphical model after decoupling no longer shows the connection between θ and zi but includes new free variational parameters. The nal distribution is given by: q(θz|γφ)=q(θ|γ)∏N q(zi |φi) i=1 Here, γ is a Dirichlet variable and Φ a multinomial. The optimization problem requires a way to calculate some distance or discrepancy between the simpli ed distribution and the real distribution. This is usually done by using the Kullback-Leibler divergence between the two distributions. The optimization problem is now to nd (γ, φ) such that: (γ∗,φ∗)=argmin D(q(θ,z|γ,φ) p(θ,z|w,α,β)) [ 220 ] (γ,φ)

The tting of the parameters of the model can be done again using the EM algorithm. However, as in inference, the E-step is intractable but can be solved with the variational approximation of this problem. The E-step consists in nding the values for the variational parameters for each document. Then the M-step consists in maximizing the lower bound of the log-likelihood with respect to the parameters α and β. The steps are repeated until convergence of the lower bound on the log-likelihood.

### 示例

We will use the RtextTools and topicmodels packages. The second one contains an implementation of the LDA model as described before. First we load some data:

data(NYTimes)  
data ← NYTimes[ samples(1:3100, size=1000,replace=F) ]

The resulting data.frame contains titles and subject and an associated topic.code. This dataset contains headlines from the New York Times. Then we create a matrix suitable for the LDA() function in the topicmodels package:

matrix ← create\_matrix(cbind(as.vector(data$Title),as.  
vector(data$Subject)), language="english," removeNumbers=TRUE,  
stemWords=TRUE)

Next, we set up the number of topics. This is computed by looking at the number of unique topic.code in the original data set. This data set has been specially compiled for this task:

k <- length(unique(data$Topic.Code))

Finally, we run the learning algorithm using the variational EM algorithm. This function also provide a Gibbs sampling method to solve the same problem:

lda <- LDA(matrix, k)

The result is a topic model with 27 topics, as expected. Let's see this in detail. The returned object is an S4 object (so you will notice we use the @ notation in R). [ 221 ] Chapter 7 Many optimization algorithms are able to solve this problem.

Probabilistic Mixture Models Let's take the rst document and look at its posterior distribution over the topics:

print(lda@gamma[1,])  
 [1] 0.649978052 0.004191364 0.004191364 0.004191364 0.004191364  
0.004191364 0.004191364 0.004191364 0.004191364 0.004191364 0.004191364  
0.004191364 0.004191364 0.004191364 0.004191364 0.004191364  
[17] 0.004191364 0.004191364 0.115045483 0.004191364 0.004191364  
0.004191364 0.134383733 0.004191364 0.004191364 0.004191364 0.004191364

We see that the rst topic has a higher probability. We can plot this to view it better:

And we will also look at an average graph over all the documents by doing:

plot(colSums(lda@gamma)/nrow(lda@gamma), t = "h")

From there, we can see the distribution over the topics is clearly not uniform, which would have been really surprising. So we have a simple way to extract the most probable topic from each document. Note that, in the case of the rst document, one topic was highly probable and two others appeared too. The rest were insigni cant. We can, for example, search for the number of documents that have two or more topics with a probability higher than 10%:

sum(sapply(1:nrow(lda@gamma), function(i) sum(lda@gamma[i, ] > 0.1) > 1))

We nd 649 over 1,000 documents. However, if we look at intervals of 10% from 0% to 100%, we see that this number drops quickly. So it seems that our data set has a lot of documents that identify themselves to only one topic at a time. The following graph shows this progression:

For example, at 30% only a couple of hundred documents still share their topics among at least two topics. Then the number drops. All sorts of analysis can be done on such a collection, such as nding the words belonging to a topic.

## 小结

In this last chapter, we saw more advanced probabilistic graphical models, whose solution is not easy to compute with standard tools such as the junction tree algorithm. This chapter set out to show that the graphical model framework can still be used even if one has to develop a special algorithm for each model. Indeed, in the LDA model, the solution to the variational problem appeared by looking at the graph of the original LDA and by transforming this graph, thus leading to a better approximation of the initial problem. So, even if the nal algorithm does not use the graph directly like a junction tree algorithm would do, the solution came from the graph anyway. [ 224 ]

Chapter 7 This chapter proved how powerful probabilistic graphical models can be, and all the possibilities and new models that can be created from simpler models. Indeed, each of these models can again be extended either by combining them—for example, in the mixture of experts model. In this model, each expert function could be replaced by another mixture of experts model, creating a hierarchical mixture of experts. This is a probabilistic version of the decision tree but with smooth transitions and an increased ability to deal with uncertainty. We have nally reached the end of our journey into the world of probabilistic graphical models. But, as with all journeys, it is just a start and we encourage the reader to look for all the R packages dedicated to graphical models and to write his or her own algorithms. Following the graphs and the generic recipes found in this book, it is possible to go way beyond the standard models and solutions we presented here; the only limit is your imagination. [ 225 ]

# 附录

## 参考文献

The following references were used while writing this book. We encourage those of you who want to go further into the eld of probabilistic graphical models and Bayesian modeling to read at least some of them. Many of our examples and presentations of algorithms took inspiration from these books and papers.

### 有关贝叶斯历史的书籍

* Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B, Vehtari, A., and Rubin, D.B.. Bayesian Data Analysis, 3rd Edition. CRC Press. 2013. This is a reference book on Bayesian modeling covering topics from the most fundamental aspects to the most advanced, with the focus on modeling and also on computations.
* Robert, C.P.. The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation. Springer. 2007. This is a beautiful presentation of the Bayesian paradigm with many examples. The book is more theoretical but has a rigorous presentation of many aspects of the Bayesian paradigm.
* McGrayne, Sharon Bertsch. The Theory That Would Not Die. Yale University Press. 2011. This talks about how Bayes' rule cracked the Enigma code, hunted down Russian submarines, and emerged triumphant from two centuries of controversy. It is a brilliantly written history of Bayes' rule going from the seminal paper of Thomas Bayes to the latest advances in the 21st century.

### 有关机器学习的书籍

* Murphy, K.P.. Machine Learning: A Probabilistic Perspective. The MIT Press. 2012. This is a book on machine learning in general with a lot of algorithms. It covers more than just graphical models and Bayesian models. It is one of the best references.
* Bishop, C.M. Pattern Recognition and Machine Learning. Springer. 2007. This is one of the best books on machine learning, covering many aspects and going through many details of the implementation of each algorithm.
* Barber, D.. Bayesian Reasoning and Machine Learning. Cambridge University Press. 2012. This is another excellent reference book covering many aspects of machine learning with a specific focus on Bayesian models.
* Robert, C.P.. Monte Carlo Methods in Statistics. 2009. (<http://arxiv.org/> pdf/0909.0389.pdf) This is an excellent paper on the Monte Carlo methods, and it is very pedagogical.
* Koller, D. and Friedman, N.. Probabilistic Graphical Models: Principles and Techniques. The MIT Press. 2009. This is the most complete and advanced book on probabilistic graphical models. It covers all aspects of the domain. This book is very dense, with thorough details on many algorithms related to PGM and useful demonstrations. Probably the best book on PGM.
* Casella, G. and Berger, R.L.. Statistical Inference, 2nd Edition. Duxbury. 2002: This is a reference book on standard statistics with many detailed demonstrations. It's a book that anyone doing statistics should read.
* Hastie, T., Tibshirani, R., and Friedman, J.. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer. 2013: This is a book best-seller and covers the most important aspects of machine learning from a statistical point of view.

### 文章

* Jacobs, R.A., Jordan, M.I, Nowlan, S.J., and Hinton, G.E. Adaptive mixtures of local experts. 1991 in Neural Computation, 3, 79-87: This is the reference paper on mixture of experts as seen in Chapter 7, Probabilistic Mixture Models.
* Blei, David M., Ng, Andrew, Y, Jordan, Michael, I. Latent Dirichlet Allocation. January 2003, Journal of Machine Learning Research 3 (4–5), p993–1022: This is a reference paper on the LDA model as seen in Chapter 7, Probabilistic Mixture Models.

### 索引

A ancestral sampling 128 arm package 195 B basic sampling algorithms about 129 standard distributions 129-133 Bayesian Linear models about 176 graphical model 179-181 implementation, in R 184-188 over- tting 176-179 packages 194 packages, in R 195 posterior distribution 181-184 stable implementation 188-194 Bayesian Naive Bayes 104-106 Bayesian theory references 227 bayesm package 194 Bayes' rule about 11 conditional probability 12 example 13-19 formula, interpreting 13 Bernoulli distribution 108 Beta-Binomial Index cluster nodes 54 conjugacy property 112, 113 continuous random variable 36 D Dirichlet distribution 105 discrete random variable 36 discrete sepal width (dsw) 74 d-separation 37 E empirical distribution relating, to model distribution 79-81 Expectation Maximization (EM) algorithm about 70 applying, to graphical models 93, 94 derivation 91, 92 for mixture models 200-206 principles 90 with hidden variables 88 G gating function 213 Gaussian mixture model about 115 de ning 116-122 example 115 generative models 70 [ 229 ] about 106-111 posterior distribution, with conjugacy property 112, 113 prior distribution 111, 112 values, selecting for Beta parameters 113 Binomial distribution 110 bnlearn R package about 65 URL 65 C

glmnet package 184, 194 graphical models building 35 Expectation Maximization (EM) algorithm, applying to 93, 94 graphs, building 37 of Bayesian Linear models 179-181 random variable, types 36 graphs building 37 probabilistic expert system 37-40 probabilistic graphical models, basic structures 40-43 H Hidden Markov Model 42 hidden variables about 88 Expectation Maximization (EM) algorithm, using 88 latent variables 89, 90 I importance sampling about 132-144 implementation, in R 144-152 importance weight 143 independently and identically distributed (iid) 70 J joint probability distribution, uncertainty about 10 marginalization 11 junction tree about 54 building 52-55 junction tree algorithm about 51, 52 implementing 55-61 K Kullback-Leibler divergence 80 L L1-penalization (Lasso) 174, 184 L2 penalization 173 Latent Dirichlet Allocation (LDA) about 197, 215, 216 examples 221-224 graphical model 216-220 variational inference 220, 221 latent variable models 88 latent variables about 89, 90 using 115 learning by inference about 75 probability, estimating 75-78 likelihood 13 linear regression about 169 example 170 parameters, estimating 170-174 M machine learning about 2, 4 references 228 MAP queries 33 Markov Chain Monte-Carlo (MCMC) about 144, 152 for probabilistic graphical models 163 methods 153, 154 Metropolis-Hastings algorithm 154-162 RStan, installing 163 Stan, installing 163 Markov Model 42 maximum a posteriori 70 maximum likelihood application 86-88 empirical distribution, relating to model distribution 79-81 estimation 79 ML algorithm, implementing 82-86 Maximum Likelihood Estimator (MLE) 182 medical expert system 63 Metropolis-Hastings algorithm 154-162 mixing proportions 116 mixture components 116 [ 230 ]

mixture models about 198 example 199, 200 Expectation Maximization (EM) algorithm, using 200-206 mixture of Bernoulli 207-210 mixture of experts 210-215 ML algorithm implementing 82-86 model calibration 69 model distribution relating, to empirical distribution 79-81 Monte-Carlo sampling 125 N Naive Bayes model about 98 Bayesian Naive Bayes 104-06 example 98, 99 implementing 101-103 representation 100, 101 Noisy-OR model 64 O over- tting about 104 solving, with Bayesian Linear models 176-179 P packages, R arm package 195 bayesm package 194 glmnet package 194 parameter tting 69 parameter learning about 69 Iris dataset, estimating 72-74 Iris dataset, loading 71 plate notation 75 posterior distribution probabilistic expert system about 40 example 37-40 reference link 37 probabilistic graphical models (PGM), examples medical expert system 63 models, with more than two layers 64, 65 sprinkler example 62 tree structure 66, 67 probabilistic graphical models (PGM) about 3, 20 applications 26-29 basic structures 40-43 directed models 24 distribution, factorizing 23 example 26-29 graphs and conditional independence 21-23 Markov Chain Monte-Carlo (MCMC) 163 probabilistic models 20, 21 undirected models 25 probabilistic queries 33 probability calculus, uncertainty and random variables 8, 9 event 8 probability 8 realization 8 sample space 8 pseudo-random numbers 129 Q queries MAP queries 33 probabilistic queries 33 R R random variable about 36 continuous random variable 36 about 13 used, with Bayesian Linear models 181-184 with conjugacy property 112, 113 prior distribution 13, 107-112 Bayesian Linear models, implementing 184-188 importance sampling, implementation 144-152 Markov Chain Monte-Carlo (MCMC) 163 rejection sampling, implementation 135-142 [ 231 ]

discrete random variable 36 random variables, uncertainty about 8 probability distribution 9 rejection sampling about 132-135 implementation, in R 135-142 ridge regression 173 RStan example 164 installing 163 URL 163 S sampling from distribution 126-129 sampling algorithms 125 seed 129 separator nodes 54 sprinkler example 62 Stan installing 163 T trees about 34 tree structure 66, 67 U uncertainty as probabilities 6 Bayesian interpretation 6 Bayes' rule 11, 12 conditional probability 7 frequentist interpretation 6 joint probability distributions 10, 11 probability calculus 7 random variables 8 representing, with probabilities 5 V variable elimination 44, 46 variational inference 220, 221 standard distributions 129-133 stop words 216 structural learning 24 sum-product variable elimination algorithm about 47, 48 implementing 49, 50 [ 232 ]

1. 原书此处为厘米，似乎有问题。 [↑](#footnote-ref-53)
2. 原书中“as the bad light bulbs arrive, the probability that the machine will fail quickly falls”，应有误。 [↑](#footnote-ref-60)
3. 英文版有误，为2n+1。 [↑](#footnote-ref-76)