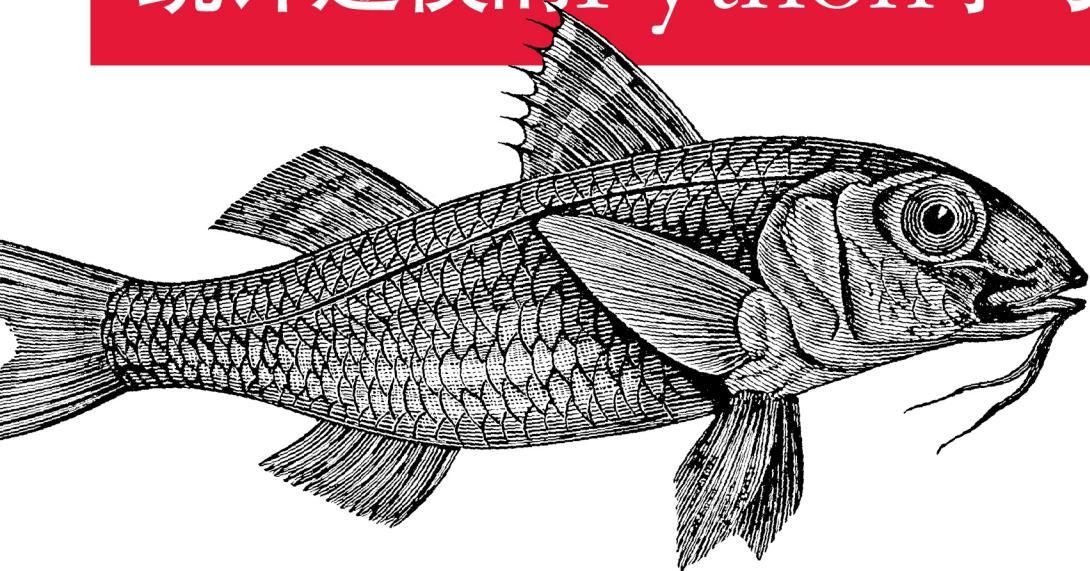


Think Bayes

贝叶斯思维： 统计建模的Python学习法



[美] *Allen B. Downey* 著
许杨毅 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

贝叶斯思维：统计建模的 Python 学习法

[美] *Allen B. Downey* 著

许杨毅 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北京

版权声明

Copyright ©2013 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2014. Authorized translation of the English edition, 2013 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书中文简体版由 O'Reilly Media, Inc. 授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不得以任何方式复制或传播。

版权所有，侵权必究。

◆ 著 [美] Allen B. Downey
译 许杨毅
责任编辑 王峰松
责任印制 张佳莹

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京 印刷有限公司印刷

◆ 开本: 787×1000 1/16
印张: 11.75
字数: 232 千字 2015 年 4 月第 1 版
印数: 1 - 0 000 册 2015 年 4 月北京第 1 次印刷
著作权合同登记号 图字: 01-2013-8996 号

定价: 00.00 元

读者服务热线: (010) 81055410 印装质量热线: (010) 81055316

反盗版热线: (010) 81055315

内容提要

这本书旨在帮助那些希望用数学工具解决实际问题的人们，仅有的要求可能就是懂一点概率知识和程序设计。贝叶斯方法是一种常见的利用概率学知识去解决不确定性问题的数学方法，对于一个计算机专业人士，应当熟悉其在诸如机器翻译、语音识别、垃圾邮件检测等常见的计算机领域的应用。

本书实际上会扩大你的视野，即使不是一个计算机专业人士，你也可以看到在战争环境下（第二次世界大战德军坦克问题），法律问题上（肾肿瘤的假设验证），体育博彩领域中（棕熊队和加人队 NHL 比赛问题）贝叶斯方法的威力。怎么从有限的信息判断德军装甲部队的规模？你所支持的球队有多大可能赢得冠军？在《龙与地下城》勇士中，你应当对游戏角色属性的最大值有怎样的预期？甚至在普通的彩弹射击游戏中，拥有一些贝叶斯思维也能帮助你提高游戏水平。

除此以外，本书在共计 15 章的篇幅中讨论了怎样解决十几个现实生活中的实际问题。在这些问题的解决过程中，作者还潜移默化地帮助读者形成了建模决策的方法论，建模误差和数值误差怎么取舍，怎样为具体问题建立数学模型，如何抓住问题中的主要矛盾（模型中的关键参数），再一步一步地优化或者验证模型的有效性或者局限性。在这个意义上，这本书又是一本关于数学建模的成功样本。

推荐序

很多人把世界理解为基于简单的、确定的，非一即零、非黑即白的。但是真实的世界却是非常复杂的，不是一两个公式可以完美总结概括的。就像我们的高考成绩和我们的学习水平，确实有很大的联系，但是最后又会受到很多因素的影响（比如身体状况，是否休息好了，心情，天气等），进而使得我们的最终成绩在真实水平上下有很大的波动。这就像我们分析很多事情时，经常得到的结论，“既有必然性，又有偶然性”。

这个时候，基于概率和统计的方法给了我们很多的帮助。很多时候，我们不能给出每一个人、每一件事的确定结果。但是当我们观察大量的相同事件后，我们就会发现从一个集体的意义上的规律是存在的。而单个事件每次可能得到不同的结果，这些结果以最有可能的结果为中心，服从一定的概率分布。了解这些分布数据，使我们更加容易理解和预期真实世界的多边形。

回顾在进行计算机自然语言处理过程中走过的路，我们就会发现从研究规则到研究统计的转变。最初，研究人员都认为，语言是基于语法规则。这个也很容易理解，因为我们学习语言的时候，总是背单词，学语法，然后掌握语言。基于这种思维，自然语言处理经历了多年的发展后，遇到了巨大的挑战。那就是即便语法规则已经非常复杂，仍然不能处理大多数的语言情况。从结果上而言，自然语言处理的准确度远低于人类，不具有真正的使用价值。而后，有一批学者开始另辟蹊径，基于统计的思路进行探索。如果语言是根据人类沟通需求自然发生，然后才有总结出来的语法呢？基于这种思想，研究人员放弃语法规则，开始建立基于统计的模型。他们使用了大量的真实文本数据，分析每个词和它前后的词出现的统计关系，用贝叶斯方法以及马尔科夫过程，建立了新的自然语言处理模型。这一次，语言处理准确率有了巨大的提升，进而达到可以实用的要求。今天，当我们使用谷歌翻译、苹果的 Siri 语音服务的时候，后面都有基于统计的模型的功劳。

还有很多的真实世界的事情都是这样的，比如路上的交通是否阻塞、银行排队的时间、球赛的比赛结果，都是以一种概率的形式出现的。了解贝叶斯方法，也是了解真实世

界运行的一种有效途径。本书中也列举了很多的真实实例来告诉我们，贝叶斯方法和真实世界的联系。

另外，在我们正在经历的大数据时代，作为数据分析方法的一个巨大分支，基于贝叶斯的机器学习算法也在被广泛地使用，并产生很多实际意义。比如简单贝叶斯算法、贝叶斯信念网络等，被广泛地应用于分类和预测。对于海量数据的文本分类问题，例如，垃圾邮件的甄选和过滤，基于贝叶斯方法的算法取得了非常好的效果，并在很多公司中正在使用，帮助我们远离垃圾邮件的骚扰。

更加难能可贵的是，本书作者用相对简单的 Python 语言，对所涉及的实例进行了编程。对于有一定计算机基础的人来说，通过程序，可以进一步理解贝叶斯方法的应用，真正掌握并且可以利用这些程序达到举一反三的效果。

本书用简洁的语言，大量的实例和故事，辅之以简单的 Python 语言，把原本枯燥的概率理论讲得生动且容易理解。在学习到理论的同时，还了解了它的真实意义以及可以使用的地方。对于有追求的工程师和感兴趣的读者而言，这是一本提升自我的很好的图书。

酷我音乐 雷鸣^①

^① 雷鸣，现任酷我音乐董事长、CEO，国家千人计划特聘专家，百度创始七剑客之一，百度搜索引擎的早期设计者和技术负责人之一。获北京大学计算机科学硕士学位和斯坦福大学商学院 MBA 学位，曾任北京大学计算机系学生会主席和斯坦福大学中国学生学者联合会副主席。

前言

学习之道

这本书以及 Think 系列其他书籍的一个前提是：只要懂得编程，你就能用这个技能去学习其他的内容。

绝大多数贝叶斯统计的书使用数学符号并以数学概念的形式表示数学思想，比如微积分。但本书使用了 Python 代码而不是数学，离散近似而不是连续数学。结果就是原本需要积分的地方变成了求和，概率分布的大多数操作变成了简单的循环。

我认为这样的表述是易于理解的，至少对于有编程经验的人们来说是这样的。当作建模选择时也非常实用，因为我们可以选取最合适的模型而不用担心偏离常规分析太多。

另外，这也提供了一个从简化模型到真实问题的平滑发展路线，第 3 章就是一个好示例。它由一个关于骰子的简单例子开始，那是基本概率的一个主题；紧接着谈到了一个我从 Mosteller《50 个挑战的统计学难题》(Fifty Challenging Problems in Probability)一书中借用的火车头问题；最后是德军坦克问题，这个第二次世界大战中成功的贝叶斯方法应用案例。

建模和近似

本书中多数章节的灵感都是由真实世界里的问题所激发的，所以涉及了一些建模知识，在应用贝叶斯方法（或者其他的方法）前，我们必须决定真实世界中的哪些部分可以被包括进模型，而哪些细节可以被抽象掉。

例如，第 7 章中那个预测冰球比赛获胜队伍的例子，我将进球得分建模为一个泊松过程，这预示着在比赛的任何时段进球机会都是相等的，这并不完全符合实际情况，但就大多数目的来说可能就够了。

第 12 章中，问题是对 SAT 得分进行解释（SAT 是用于全美大学的入学标准测试）。我以一个假设所有 SAT 试题难度相同的简化模型开始，但其实 SAT 的试题设计中既包括了相对容易，也包括了相对较难的试题。随后提出了第二个反映这一设计目的的模型，结果显出两个模型在最终效果上没有大的差别。

我认为在解决问题的过程中，明确建模过程作为其中一部分是重要的，因为这会提醒我们考虑建模误差（也就是建模当中简化和假设带来的误差）。

本书中的很多方法都基于离散分布，这让一些人担心数值误差，但对于真实世界的问题，数值误差几乎从来都小于建模误差。

再者，离散方法总能允许较好的建模选择，我宁愿要一个近似的良好的模型也不要一个精确但却糟糕的模型。

从另一个角度看，连续方法常在性能上有优势，比如能以常数时间复杂度的解法替换掉线性或者平方时间复杂度的解法。

总的来说，我推荐这些步骤的一个通用流程如下。

1. 当研究问题时，以一个简化模型开始，并以清晰、好理解、实证无误的代码实现它。注意力集中在好的建模决策而不是优化上。
2. 一旦简化模型有效，再找到最大的错误来源。这可能需要增加离散近似过程当中值的数量，或者增加蒙特卡洛方法中的迭代次数，或者增加模型细节。
3. 如果对你的应用而言性能就已经足够了，则没必要再优化。但如果要做，有两个方向可以考虑：评估你的代码以寻找优化空间，例如，如果你缓存了前面的计算结果，你也许能避免重复冗余的计算；或者可以去发现找到计算捷径的分析方法。

这一流程的好处是第一、第二步较快，所以你能在投入大量精力前研究多个可替代的模型。

另一个好处是在第三步，你可以从一个大体正确的可参考实现开始进行回归测试。也就是说，检查优化后的代码是否得到了同样的结果，至少是近似的结果。

代码指南

本书中的很多例子使用了在 `thinkbayes.py` 当中定义的类和函数，可以从 <http://thinkbayes.com/thinkbayes.py> 下载这个模块。

本书大多数章节包括了可以从 <http://thinkbayes.com> 下载的代码，其中有一些依赖代码也需要下载，我建议你将这些文件全部放入同一个目录，这样代码间就可以彼此引用而无需变更 Python 的库文件搜索路径。

你可以在需要时再下载这些代码，或者一次性从 http://thinkbayes.com/thinkbayes_code.zip 下

载，这个文件也包括了某些程序使用的数据文件，当解压时，将创建名为 `thinkbayes_code` 的包括本书中所有代码的目录。

另外，如果是 Git 用户，你可以通过 `fork` 和 `clone` 来一次性获得这个仓库：<https://github.com/AllenDowney/ThinkBayes>。

我用到的模块之一是 `thinkplot.py`，它对 `pyplot` 中一些函数进行了封装，要使用它需要安装好 `matplotlib`，如果还没有，检查你的软件包管理器看看它是否存在，否则你可以从 <http://matplotlib.org> 得到下载指南。

最后，本书中一些程序使用了 NumPy 和 SciPy，可以从 <http://numpy.org> 和 <http://scipy.org> 获得。

编码风格

有经验的 Python 程序员会注意到本书中的代码没有符合 PEP 8 这一最通用的 Python 编码指南（<http://www.python.org/dev/peps/pep-0008/>）。

确切地说，PEP 8 使用带有词间下划线的小写函数名 `like_this`，而在本书中和实现的代码里，函数和方法名以大写开头并使用间隔式的大小写，`LikeThis`。

没有遵循 PEP 8 规范的原因是在我为书中内容准备代码时正在谷歌做访问学者，所以就遵循了谷歌的编码规范，它只在少数地方沿袭了 PEP 8，一用上了谷歌风格我就喜欢上了，现在要改太麻烦。

同样，在主题风格上，如在“Bayes’s theorem”中，`s` 放在单引号后，在某些风格指南中倾向这样使用而在其他指南当中不是。我没有特别的偏好，但不得不选择其一，所以就是你们现在看到的这个。

最后一个排版上的注脚是：贯穿全书，我使用 PMF 和 CDF 表示概率密度函数或累积分布函数这些数学概念，而 `Pmf` 和 `Cdf` 是指我所表述的 Python 对象。

预备条件

还有几个出色的能在 Python 中进行贝叶斯统计的模块，包括 `pymc` 和 `OpenBUGS`，由于读者需要有相当多的背景知识才能开始使用这些模块，因此本书中我没有使用它们，而且我想使阅读本书的预备条件最小。如果你了解 Python 和一点点概率知识，就可以开始阅读本书。

第 1 章关于概率论和贝叶斯定理，没有程序代码。第 2 章介绍了 `Pmf`，一望而知是用来表示概率密度函数（PMF）的 Python 字典对象。然后第 3 章我介绍了 `Suite`，一个 `Pmf` 对象，也是一个能进行贝叶斯更新的框架，因而万事具备了。

好了，随后的章节中，我使用了高斯（正态）分布，二次和泊松分布，beta 分布等各种分析型的概率分布，在第 15 章，我介绍了不太常见的狄利克雷分布，不过接着也进行了解释。如果你不熟悉这类分布，可以从维基百科了解它们。也可以阅读本书的一本指南《统计思维》(Think Stats)，或其他入门级的统计学书籍（不过，恐怕大多数类似书籍都会采取对实战没有太大帮助的数学方法来阐述）。

书中使用的惯例写法

本书中使用了下面的印刷惯例。

斜体 (*Italic*)

表示新术语，URL，邮件地址，文件名和文件扩展名。

等宽 (Constant width)

用于程序代码，也包括那些表示程序代码元素的段落，例如，变量和函数名，数据库，数据类型，环境变量，声明和关键字。

等宽粗体 (**Constant width bold**)

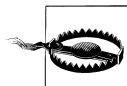
命令或者其他由用户输入的文字。

等宽斜体 (*Constant width italic*)

应该由用户输入值替换或者由上下文决定的文本。



这个图标表示这是一个提示、建议或者一般性的注记。



这个图标表示这是提醒或者警示。

我们的联系方式

如果你想就本书发表评论或有任何疑问，敬请联系出版社。

美国：

O'Reilly Media Inc.
1005 Gravenstein Highway North

Sebastopol, CA 95472

中国:

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室 (100035)
奥莱利技术咨询 (北京) 有限公司

我们还为本书建立了一个网页，其中包含了勘误表、示例和其他额外的信息。你可以通过地址访问该网页：<http://oreil.ly/think-bayes>。

关于本书的技术性问题或建议，请发邮件到：bookquestions@oreilly.com。

欢迎登录我们的网站 (<http://www.oreilly.com>)，查看更多我们的书籍、课程、会议和最新动态等信息。

我们的其他联系方式如下。

Facebook: <http://facebook.com/oreilly>

Twitter: <http://twitter.com/oreillymedia>

YouTube: <http://www.youtube.com/oreillymedia>

贡献者列表

如果你发现本书有需要更正的地方或者其他建议，请发送电子邮件至 downey@allendowney.com。一旦根据你的反馈进行了修正，我会将你加入贡献者列表（除了要求不署名的情况）。

提供包含错误之处的段落部分，会让我更容易找到它们。只提供页和节数也可以，但还是不太容易找到错误之处。这里先致谢！

- 首先，我要感谢大卫·麦凯 (David MacKay) 的优秀作品《信息理论、推理和学习算法》(Information Theory, Inference, and Learning Algorithms)，我从这本书里第一次理解了贝叶斯方法。他允许我使用他书中的几个问题来作为例子。
- 这本书也得益于我和圣乔恩·马哈的相互配合，2012 的秋天我在欧林学院审核了他的贝叶斯推理课程。
- 在参加波士顿 Python 用户组项目时，我在夜班时间完成了本书的部分内容，所以我也要感谢他们以及他们所提供的比萨。
- 乔纳森·爱德华兹提交了第一个拼写错误。
- 乔治·珀金斯发现了一个标记错误。
- 奥利维尔提出了几个有益的建议。
- 尤里·帕西奇尼克发现了几个错误。

- 克里斯托弗 · 欧霍特提交了一个更正和建议的清单。
- 罗伯特 · 马库斯发现了一个错误放置的小写 i。
- 麦克斯 · 黑尔珀林建议在第 1 章提供一个澄清章节。
- 马库斯 · 杜布勒指出 “从碗中有放回的取出饼干” 并不是一个真实的场景。
- 汤姆 · 波拉德和保罗 A. 吉安纳罗斯指出，在火车头案例中的某些数量有版本问题。
- 兰姆 · 林布发现了一个拼写错误，还建议了澄清章节。
- 2013 春天在我的《贝叶斯统计计算》课程上，学生们提出了许多有益的修正和建议，他们是：凯 · 奥斯汀，克莱尔 · 巴尼斯，卡里 · 本德尔，瑞秋 · 铂伊，凯特 · 门多萨，阿琼 · 伊耶，本 · 克罗普，内森 · 林之，凯尔 · 麦克康诺亥，亚历克 · 雷德福，布伦丹 · 里特，埃文 · 辛普森。
- 格雷戈 · 马拉和马特 · 艾提帮我澄清了“正确的价格”这个问题的一些讨论。
- 马库斯 · 奥格伦指出火车头问题的原有声明是有些含糊的。
- O'Reilly Media 的贾斯敏和丹在校对书的过程中也发现了许多可改进的地方。

目录

第 1 章 贝叶斯定理	1
1.1 条件概率	1
1.2 联合概率	2
1.3 曲奇饼问题	2
1.4 贝叶斯定理	3
1.5 历时诠释	4
1.6 M&M 豆问题	5
1.7 Monty Hall 难题	6
1.8 讨论	8
第 2 章 统计计算	9
2.1 分布	9
2.2 曲奇饼问题	10
2.3 贝叶斯框架	11
2.4 Monty Hall 难题	12
2.5 封装框架	13
2.6 M&M 豆问题	14
2.7 讨论	15
2.8 练习	16
第 3 章 估计	17
3.1 骰子问题	17
3.2 火车头问题	18
3.3 怎样看待先验概率？	20
3.4 其他先验概率	21
3.5 置信区间	23
3.6 累积分布函数	23

3.7	德军坦克问题.....	24
3.8	讨论	24
3.9	练习	25
第 4 章	估计进阶	27
4.1	欧元问题.....	27
4.2	后验概率的概述.....	28
4.3	先验概率的湮没.....	29
4.4	优化	31
4.5	Beta 分布	32
4.6	讨论	34
4.7	练习	34
第 5 章	胜率和加数	37
5.1	胜率	37
5.2	贝叶斯定理的胜率形式.....	38
5.3	奥利弗的血迹.....	39
5.4	加数	40
5.5	最大化	42
5.6	混合分布.....	45
5.7	讨论	47
第 6 章	决策分析	49
6.1	“正确的价格”问题.....	49
6.2	先验概率.....	50
6.3	概率密度函数.....	50
6.4	PDF 的表示	51
6.5	选手建模.....	53
6.6	似然度	55
6.7	更新	55
6.8	最优出价.....	57
6.9	讨论	59
第 7 章	预测.....	61
7.1	波士顿棕熊队问题.....	61
7.2	泊松过程.....	62
7.3	后验	63
7.4	进球分布.....	64
7.5	获胜的概率.....	66
7.6	突然死亡法则.....	66
7.7	讨论	68

7.8	练习	69
第 8 章	观察者的偏差	71
8.1	红线问题	71
8.2	模型	71
8.3	等待时间	73
8.4	预测等待时间	75
8.5	估计到达率	78
8.6	消除不确定性	80
8.7	决策分析	81
8.8	讨论	83
8.9	练习	84
第 9 章	二维问题	85
9.1	彩弹	85
9.2	Suite 对象	85
9.3	三角学	87
9.4	似然度	88
9.5	联合分布	89
9.6	条件分布	90
9.7	置信区间	91
9.8	讨论	93
9.9	练习	94
第 10 章	贝叶斯近似计算	95
10.1	变异性假说	95
10.2	均值和标准差	96
10.3	更新	98
10.4	CV 的后验分布	98
10.5	数据下溢	99
10.6	对数似然	100
10.7	一个小的优化	101
10.8	ABC (近似贝叶斯计算)	102
10.9	估计的可靠性	104
10.10	谁的变异性更大?	105
10.11	讨论	107
10.12	练习	108
第 11 章	假设检验	109
11.1	回到欧元问题	109
11.2	来一个公平的对比	110

11.3	三角前验	111
11.4	讨论	112
11.5	练习	113
第 12 章	证据	115
12.1	解读 SAT 成绩	115
12.2	比例得分 SAT	115
12.3	先验	116
12.4	后验	117
12.5	一个更好的模型	119
12.6	校准	121
12.7	效率的后验分布	122
12.8	预测分布	123
12.9	讨论	124
第 13 章	模拟	127
13.1	肾肿瘤的问题	127
13.2	一个简化模型	128
13.3	更普遍的模型	130
13.4	实现	131
13.5	缓存联合分布	132
13.6	条件分布	133
13.7	序列相关性	135
13.8	讨论	138
第 14 章	层次化模型	139
14.1	盖革计数器问题	139
14.2	从简单的开始	140
14.3	分层模型	141
14.4	一个小优化	142
14.5	抽取后验	142
14.6	讨论	144
14.7	练习	144
第 15 章	处理多维问题	145
15.1	脐部细菌	145
15.2	狮子，老虎和熊	145
15.3	分层版本	148
15.4	随机抽样	149
15.5	优化	150
15.6	堆叠的层次结构	151

15.7	另一个问题	153
15.8	还有工作要做	154
15.9	肚脐数据	156
15.10	预测分布	158
15.11	联合后验	161
15.12	覆盖	162
15.13	讨论	164

第1章

贝叶斯定理

1.1 条件概率

所有贝叶斯统计的方法都基于贝叶斯定理，如果有条件概率的学习基础，意识到这一点很自然。因此我们会从概率、条件概率开始，然后到贝叶斯定理，最后讨论贝叶斯统计的内容。

概率表示为 0 和 1 之间的数字（包括 0 和 1），含义是某一事件或者预测行为的可信程度，1 值表示“事件为真”的情形肯定发生，或表述为预测成真；而 0 值则表示“事件为真”这一情形为假。

其他中间值表示确定性的程度。例如，0.5 通常也会写成 50%，意味着一个预测结果发生和不发生有同等可能性。例如，在一个掷硬币事件中，人像面（正面）朝上的概率就非常接近 50%。

条件概率是带有某些（前提条件）背景约束下的概率问题。例如，我想了解一下明年自己心脏病发作的可能性。根据疾病控制中心的数据，每年大约有 78.5 万名美国人罹患心脏病 (<http://www.cdc.gov//heartdisease/fact.html>)。

美国约有 3.11 亿人，假设随机挑选一个美国人，那么其在明年心脏病发作的概率大约是 0.3%。

但就具体个例而言，“我”可不是那个被随意选中的美国人。流行病学家们已经明确了多种影响心脏病发作的风险因素，根据这些因素我的风险则有可能高于或低于平均值。

本人男，45 岁，有临界高胆固醇，这些因素增加了我发病的可能性；然而，血压低、不抽烟这些因素则降低了可能性。

把上面这些条件输入在线计算器 <http://hp2010.nhlbihin.net/atpii/calculator.asp>，我发现自己明年心脏病发作的风险约为 0.2%，低于全国平均水平。这个值就是一个条件概率，因为它是基于一系列前提因素的，这些因素构成了我患心脏病的“条件”。

通常条件概率的记号是 $p(A|B)$ ，表示在给定 B 条件下 A 事件发生的概率。在这个例子中， A 表示我明年罹患心脏病带的概率，而 B 表示了上面所罗列的条件。

1.2 联合概率

联合概率：是指两个事件同时发生的概率。 $p(A \text{ 和 } B)$ 是 A 和 B 事件的发生都为真的概率。

如果你已经理解了投骰例子和它的背景，我们开始学习下面的公式：

$$p(A \text{ 和 } B) = p(A)p(B) \quad \text{提醒：表达式并非总是成立。}$$

例如，如果我投掷两个硬币， A 表示第一枚硬币正面朝上， B 表示第二枚硬币正面朝上，那么 $p(A) = p(B) = 0.5$ ，同样的 $p(A \text{ 和 } B) = p(A)p(B) = 0.25$ 。

但是上面公式仅在 A 和 B 都是独立事件的情况下才成立。即：已知 A 事件的结果并不影响或改变 B 事件发生的概率。或更正式表示为， $p(B|A) = p(B)$ 。

再考虑另一个事件之间并不独立的例子。假设 A 表示今天下雨的事件， B 表示明天会下雨的事件。如果我已经知道今天下雨，则明天还有可能下雨（译注：与仅仅单独考虑某一天会下雨的概率相比较），所以 $p(B|A) > p(B)$ 。

通常意义下，联合概率表述为

$$p(A \text{ and } B) = p(A) p(B|A)$$

对于任何 A 、 B 事件，如果任意一天下雨的机会是 0.5，连续两天就不会是 0.25，而是可能更高一点。

1.3 曲奇饼问题

我们即将开始讨论到贝叶斯定理，但我还想通过一个被称为“曲奇饼问题”的例子来介绍它。假设有两碗曲奇饼，碗 1 包含 30 个香草曲奇饼和 10 个巧克力曲奇饼，碗 2 有上述两种饼干各 20 个。

现在设想你在不看的情况下随机地挑一个碗拿一块饼，得到了一块香草曲奇饼。我们的问题是：从碗 1 取到香草曲奇饼的概率是多少？

这就是一个条件概率问题；我们希望得到概率 $p(\text{碗 1}|\text{香草})$ ，但怎样进行计算并非显而易见。问题如果换成在碗 1 中香草曲奇饼的概率则简单得多。

$$p(\text{香草}|\text{碗 } 1) = 3/4$$

不巧的是， $p(A|B)$ 并不和 $p(B|A)$ 相同，但有方法从一个计算出另一个：贝叶斯定理。

1.4 贝叶斯定理

现在，我们准备好进行贝叶斯定理推导需要的所有条件了。首先，我们注意到，联合概率是乘积可交换（乘法交换律）的，即：

$$p(A \text{ and } B) = p(B \text{ and } A)$$

对于任何 A, B 表示的事件都成立。

然后，我们写出一个联合概率的表达式：

$$p(A \text{ and } B) = p(A) p(B|A)$$

由于我们并没有明确定义 A 和 B 的含义，因而可以对 A, B 进行互换操作。

交换它们的位置：

$$p(B \text{ and } A) = p(B) p(A|B)$$

把这些表达式连接起来，我们得到下面的表达式：

$$p(B) p(A|B) = p(A) p(B|A)$$

这意味着我们有两种方式计算联合概率，已知 $p(A)$ ，乘以 $p(B|A)$ ；或者从另一方向，已知 $p(B)$ ，乘以 $p(A|B)$ 。两种方法是相同的。

最后，将上式除以 $p(B)$ ，得到：

$$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

这正是贝叶斯定理！看起来不起眼，不过它会显示出令人吃惊的强大之处。

例如，我们可以用它来解决曲奇饼问题。

假设 B_1 表示曲奇饼属于碗 1 的概率， V 表示曲奇饼是香草曲奇饼的概率。

带入贝叶斯定理我们得到：

$$p(B_1|V) = \frac{p(B_1)p(V|B_1)}{p(V)}$$

等式左边就是我们希望得到的，一块香草曲奇饼来自碗 1 的概率。

等式的右边表示：

- $p(B_1)$: 这是我们忽略得到曲奇饼这个条件时（零条件下）选中碗 1 的概率。因为选择碗的过程是随机的，我们可以假设 $p(B_1)=1/2$ 。
- $p(V|B_1)$: 这是从碗 1 得到一个香草曲奇饼的概率 = 3/4。
- $p(V)$: 从任意碗里得到一个香草曲奇饼的概率。因为考虑到选择碗的机会均等，而且每个碗的曲奇饼数量都是 40，得到曲奇饼的机会是相同的。两个碗中香草和巧克力曲奇饼总数各是 50 和 30，因此 $p(V)=5/8$ 。

把它们放在一起，我们得到：

$$p(B_1|V) = \frac{(1/2)(3/4)}{5/8}$$

结果是 3/5。所以，“得到一块香草曲奇饼”是支持于假设“来自碗 1”的证据，因为香草曲奇饼来自碗 1 的可能性更大。

这个例子演示了一个应用贝叶斯定理的案例：它提供了一个从 $p(B|A)$ 得到 $p(A|B)$ 的策略。这种策略在解决类似“曲奇饼问题”的情况下是有用的，即从贝叶斯等式的右边计算要比左边容易的情况下。

1.5 历时诠释

还有另外一种理解贝叶斯定理的思路：它给我们提供的是一种根据数据集 D 的内容变化更新假设概率 H 的方法。

这种对贝叶斯定理的理解被称为“历时诠释”。

“历时”意味着某些事情随着时间而发生；在本例，即是假设的概率随着看到的新数据而变化。

在考虑 H 和 D 的情况下，贝叶斯定理的表达式可以改写成：

$$p(H|D) = \frac{p(H)p(D|H)}{p(D)}$$

在这种解释里，每项意义如下：

- $p(H)$ 称为先验概率，即在得到新数据前某一假设的概率。
- $p(H|D)$ 称为后验概率，即在看到新数据后，我们要计算的该假设的概率。
- $p(D|H)$ 是该假设下得到这一数据的概率，称为似然度。
- $p(D)$ 是在任何假设下得到这一数据的概率，称为标准化常量。

有些情况，我们可以基于现有背景信息进行计算。比如在曲奇饼问题中，我们就将随

机选中碗 1 或碗 2 的概率假设为均等。

在其他情况下，先验概率是偏主观性的；对某一先验概率，理性派的人可能会有不同意见，或许由于他们使用不同的背景信息做出判断，或者因为他们针对相同的前提条件做出了不同的解读。

似然度是贝叶斯计算中最简单的部分，在曲奇饼问题中曲奇饼来自来自哪个碗，则我们就计算那个碗中香草曲奇饼的概率。

标准化常量则有些棘手，它被定义为在所有的假设条件下这一数据出现的概率，但因为考虑的正是最一般的情况，所以不容易确定这个常量在具体应用场合的现实意义。

最常见的，我们可以指定一组如下的假设集来简化。

互斥的：集合中，至多一个假设为真。

完备的：集合中，至少一个假设必为真，且集合包含了所有的假设。

我使用 suite 这个词来表示具备上述属性的假设集。

在曲奇饼问题中，仅有两个假设：饼干来自碗 1 或者碗 2，它们就是互斥的和完备的。

在本例中，我们可以用全概率公式计算 $p(D)$ ，即如果发生某一事件有互不容的两个可能性，可以像下面这样累加概率：

$$p(D) = p(B_1) p(D|B_1) + p(B_2) p(D|B_2)$$

代入饼干问题中的实际值，得到：

$$p(D) = (1/2)(3/4) + (1/2)(1/2) = 5/8$$

我们早前心算得到的结果也是一样的。

1.6 M&M 豆问题

M&M 豆是有各种颜色的糖果巧克力豆。制造 M&M 豆的 Mars 公司会不时变更不同颜色巧克力豆之间的混合比例。

1995 年，他们推出了蓝色的 M&M 豆。在此前一袋普通的 M&M 豆中，颜色的搭配为：30%褐色，20%黄色，20%红色，10%绿色，10%橙色，10%黄褐色。这之后变成了：24%蓝色，20%绿色，16%橙色，14%黄色，13%红色，13%褐色。

假设我的一个朋友有两袋 M&M 豆，他告诉我一袋是 1994 年，一袋是 1996 年。

但他没告诉我具体哪个袋子是哪一年的，他从每个袋子里各取了一个 M&M 豆给我。一个是黄色，一个是绿色的。那么黄色豆来自 1994 年的袋子的概率是多少？

这个问题类似于曲奇饼问题，只是变化了我抽取样品的方式（碗还是袋）。这个问题也给了我一个机会演示纸面方法：也就在仅仅在纸上画画就可以解决类似这样的问题（译注：作者为后续章节的计算型方法铺垫）。在下一章中，我们将以计算方法解这些问题。

第一步是枚举所有假设。取出黄色 M&M 豆的袋子称为袋 1，另一个称为袋 2，所以假设是：

- A：袋 1 是 1994 年的，袋 2 是 1996 年的。
- B：袋 1 是 1996 年的，袋 2 是 1994 年的。

接着我们设计一个表格，每行表示每个假设，每列表示贝叶斯定理中的每一项：

先验概率 $p(H)$		似然度 $p(D H)$	$p(H) p(D H)$	后验概率 $p(H D)$
假设 A	1/2	(20) (20)	200	20/27
假设 B	1/2	(10) (14)	70	7/27

第一列表示先验。基于问题的声明，选择 $p(A)=p(B)=1/2$ 是合理的。

第二列表示似然度，表明了问题的背景信息。举例来说，如果 A 为真，黄色 M&M 是来自 1994 年的袋概率 20%，而绿色来自 1996 包的概率为 20%。因为选择是独立的，我们将其相乘以得到联合概率。

第三列由前两列得到。此列的总和 270 是归一化常数（译注：参考全概率公式）。为了得到最后一列的后验概率，我们将第三列的值归一化后得到第四列的值。

就是这样。简单吧？

还有，你可能会被一个细节所困扰。我将 $p(D|H)$ 写成了百分数的形式而不是概率形式，这意味着它没有除以因子 10000。但是当我们将其除以归一化常数时就抵消了，因此这不影响结果。

当设定的假设是互斥和穷举的，你可以将似然度乘以任何因子，如果方便，将同一个因子应用到整列上。

1.7 Monty Hall 难题

蒙蒂大厅（Monty Hall problem）难题可能是历史上最有争议的概率问题。问题看似简单，但正确答案如此有悖常理以致很多人不能接受，很多聪明人都难堪于自己搞错了反而据理力争，而且是公开的。

蒙蒂大厅是游戏节目“来做个交易”（Let's Make a Deal）的主场。蒙蒂大厅难题也是这一节目的常规游戏之一。如果你参加节目，规则是这样的：

- 蒙蒂向你示意三个关闭的大门，然后告诉你每个门后都有一个奖品：一个奖品是一辆车，另外两个是像花生酱和假指甲这样不值钱的奖品。奖品随机配置。
- 游戏的目的是要猜哪个门后有车。如果你猜对了就可以拿走汽车。
- 你先挑选一扇门，我们姑且称之为门 A，其他两个称为门 B 和门 C。
- 在打开你选中的门前，为了增加悬念，蒙蒂会先打开 B 或 C 中一个没有车的门来增加悬念（如果汽车实际上就是在 A 门背后，那么蒙蒂打开门 B 或门 C 都是安全的，所以他可以随意选择一个）。
- 然后蒙蒂给你一个选择。坚持最初的选择还是换到剩下的未打开的门上。

问题是，你应该“坚持”还是“换”？有没有区别？

大多数人都有强烈的直觉，认为这没有区别。剩下两个门没有打开，车在门 A 背后的几率是 50%。

但是，这是错的。事实上，如果你坚持选择门 A，中奖概率只有 $1/3$ ；而如果换到另外一个门，你的机会将是 $2/3$ 。

运用贝叶斯定理，我们可以将这个问题分解成几个简单部分，也许这样可以说服自身，“正确”的答案实际上的的确确是对的。

首先，我们应该对数据进行仔细描述。在本例中为 D 包括两个部分：蒙蒂打开了门 B，而且没有车在后面。

接下来，我们定义了三个假设：A，B 和 C，表示假设车在门 A，门 B，或门 C 后面。同样，采用表格法：

先验概率 $p(H)$		似然度 $p(D H)$	$p(H) p(D H)$	后验概率 $p(H D)$
假设 A	$1/3$	$1/2$	$1/6$	$1/3$
假设 B	$1/3$	0	0	0
假设 C	$1/3$	1	$1/3$	$2/3$

填写先验很容易，因为我们被告知奖品是随机配置的，这表明该车可能在任何门后面。

定义似然度需要一些思考，在充分合理的考虑后，我们确信正确的似然度如下：

- 考虑假设 A：如果汽车实际上是在门 A 后，蒙蒂可以安全地打开门 B 或门 C。所以他选择门 B 的概率为 $1/2$ 。因为车实际上是在门 A 后，也就是说车不在门 B 后的概率是 1。
- 考虑假设 B：如果汽车实际上是在门 B 后，蒙蒂不得不打开门 C，这样他打开门 B 的概率就是 0（译注：也就是这个假设的似然度为 0，不可能发生）。
- 最后考虑假设 C：如果车是在门 C 后，蒙蒂打开门 B 的概率为 1，发现车不在那

儿的概率为 1 (译注: 因为在选手已经选了 A 门这个情况下, 可供蒙蒂增加悬念开门的选择只有 B 和 C, 而假设 C 有车, 蒙蒂肯定不会选, 因此蒙蒂会打开 B 门的概率为 1, 也就是在这个假设下, 数据 D 的似然度为 1)。

现在我们已经完成有难度的部分了, 剩下无非就是算术。第三列的总和为 $1/2$, 除以后得到 $p(A|D) = 1/3$, $p(C|D) = 2/3$, 所以你最好是换个选择。

该问题有许多变形。贝叶斯方法的优势之一就是可以推广到这些变形问题的处理上。

例如, 设想蒙蒂总是尽可能选择门 B, 且只有在迫不得已的时候才选门 C (比如车在门 B 后)。在这种情况下, 修正后的表如下:

先验概率 $p(H)$		似然度 $p(D H)$	$p(H)p(D H)$	后验概率 $p(H D)$
假设 A	1/3	1	1/3	1/2
假设 B	1/3	0	0	0
假设 C	1/3	1	1/3	1/2

唯一的变化是 $p(D|A)$ 。如果车在门 A 后, 蒙蒂可以选择打开 B 或 C。但在这个变形问题里面, 他总是选择 B, 因此 $p(D|A) = 1$ 。

因此, 对 A 和 C, 似然度是相同的, 后验也是相同的: $p(A|D) = p(C|D) = 1/2$, 在这种情况下, 蒙特选择 B 门显示不了车位置的任何信息, 所以无论选手选择坚持不变还是改变都无关紧要。

反过来的情况下, 如果蒙蒂打开门 C, 我们就知道 $p(B|D) = 1$ (译注: 因为蒙蒂总是优先选择门 B, 而门 D 是他打开了门 C, 因此在假设车在门 B 后的前提下, 他必然会打开门 C, 概率为 1, 即 $p(B|D)=1$)。

本章中我介绍了蒙蒂问题, 因为我觉得这里有它的趣味性, 也因为贝叶斯定理使问题的复杂性更易控制。但这并不算是一个典型的贝叶斯定理应用, 所以如果你觉得它令人困惑, 没什么好担心的!

1.8 讨论

对于涉及条件概率的很多问题, 贝叶斯定理提供了一个分而治之的策略。如果 $p(A|B)$ 难以计算, 或难以用实验衡量, 可以检查计算贝叶斯定理中的其他项是否更容易, 如 $p(B|A)$, $p(A)$ 和 $p(B)$ 。

如果蒙蒂大厅问题让你觉得有趣, 我在一篇文章 “All your Bayes are belong to us” 中收集了很多类似问题, 你可以去单击链接进行阅读 <http://allendowney.blogspot.com/2011/10/all-your-bayes-are-belong-to-us.html>。

统计计算

2.1 分布

在统计上，分布是一组值及其对应的概率。

例如，如果滚动一个六面骰子，可能的值是数字 1 至 6，与每个值关联的概率是 $1/6$ 。

再举一个例子，你应该有兴趣了解在日常的英语使用中每个单词出现的次数。你可以建立一个包含每个字及它出现的次数的分布。

为了表示 Python 中的分布，可以使用一个字典映射某个值和它的概率。我编写了一个名为 `Pmf` 的类，利用 Python 字典实现了上述功能，而且提供了一些有用的方法。为了对应概率质量函数这种分布的数学表示法，我将其命名为 `Pmf`。

`Pmf` 的定义在一个我为本书完成的 Python 模块 `thinkbayes.py` 中。可以从 <http://thinkbayes.com/thinkbayes.py> 下载。欲了解更多信息参见前言的“代码指南”。

要使用 `Pmf`，可如下导入：

```
from thinkbayes.py import Pmf
```

下面的代码建立一个 `Pmf` 来表示六面骰子的结果分布：

```
pmf = Pmf()
for x in [1,2,3,4,5,6]:
    pmf.Set(x, 1/6.0)
```

`Pmf` 创建一个空的没有赋值的 `pmf`。`Set` 方法设置每个值的概率为 $1/6$ 。

这里是另一个例子，计算每个单词在一个词序列中出现的次数：

```
pmf = Pmf()
for word in word_list:
    pmf.Incr(word, 1)
```

Incr 为每个单词的相应“概率”加 1。如果一个词还没有出现在 Pmf 中，那么就将这个词添加进去。

我把“概率”加上引号是因为在这个例子中概率还没有归一化，也就是说它们的累加和不是 1，因此不是真正的概率。但在本例中单词计数与概率成正比。所以当完成了所有的计数，就可以通过除以计数的总值来计算得到概率。

Pmf 提供了一种 Normalize 方法来实现上述功能：

```
pmf.Normalize()
```

一旦有一个 Pmf 对象，你可以像下面这样得到任何一个值相关联的概率：

```
print pmf.Prob('the')
```

这会打印输出单词“the”在词序列中出现的频率。

Pmf 使用 Python 字典来存储值及其概率，所以 Pmf 中的值可以是任意可被哈希的类型。概率可以是任意数值类型，但通常是浮点数（float 类型）。

2.2 曲奇饼问题

在贝叶斯定理的语境下，可以很自然地使用一个 Pmf 映射每个假设和对应的概率。

在曲奇饼问题里面，该假设是 B_1 和 B_2 。在 Python 中可以使用字符串来表示它们：

```
pmf = Pmf()  
pmf.Set('Bowl1', 0.5)  
pmf.Set('Bowl2', 0.5)
```

这一分布包含了对每个假设的先验概率，称为先验分布。

要更新基于新数据（拿到一块香草曲奇饼）后的分布，我们将先验分别乘以对应的似然度。

从碗 1 拿到香草曲奇饼的可能性是 $3/4$ ，碗 2 的可能性是 $1/2$ 。

```
pmf.Mult('Bowl1', 0.75)  
pmf.Mult('Bowl2', 0.5)
```

如你所愿，Mult 将给定假设的概率乘以已知的似然度。

更新后的分布还没有归一化，但由于这些假设互斥且构成了完全集合（意味着完全包含了所有可能假设），我们可以进行重新归一化如下：

```
pmf.Normalize()
```

其结果是一个包含每个假设的后验概率分布，这就是所说的后验分布。

最后，我们可以得到假设碗 1 的后验概率如下：

```
print pmf.Prob('Bowl 1')
```

答案是 0.6。你可以从 <http://thinkbayes.com/cookie.py> 下载这个例子。欲了解更多信息，请参见前言的“代码指南”。

2.3 贝叶斯框架

在继续讨论其他的问题前，我想在上一节的基础上重写代码以使其更通用。首先我将定义一个类来封装与此相关的代码：

```
class Cookie(Pmf):  
  
    def __init__(self, hypos):  
        Pmf.__init__(self)  
        for hypo in hypos:  
            self.Set(hypo, 1)  
        self.Normalize()
```

Cookie 对象是一个映射假设到概率的 Pmf 对象。`__init__`方法给每个假设赋予相同的先验概率。如上一节中就有两种假设：

```
hypos = ['Bowl1', 'Bowl2']  
pmf = Cookie(hypos)
```

Cookie 类提供了 `Update` 方法，它以 `data` 为参数并修正相应的概率：

```
def Update(self, data):  
    for hypo in self.Values():  
        like = self.Likelihood(data, hypo)  
        self.Mult(hypo, like)  
    self.Normalize()
```

`Update` 遍历 `suite` 中的每个假设，并将其概率乘以数据在某一假设下的似然度，似然度由 `Likelihood` 计算：

```
mixes = {  
    'Bowl 1': dict(vanilla=0.75, chocolate=0.25),  
    'Bowl 2': dict(vanilla=0.5, chocolate=0.5),  
}  
  
def Likelihood(self, data, hypo):  
    mix = self.mixes[hypo]  
    like = mix[data]  
    return like
```

`Likelihood` 使用 `mixes`，它使用 Python 的字典结构来映射碗名和在碗中曲奇饼的混合比例。

如下面这样进行更新：

```
pmf.Update('vanilla')
```

然后我们就可以打印输出每个假设的后验概率：

```
for hypo, prob in pmf.Items():
    print hypo, prob
```

其结果是

```
Bowl 1 0.6
Bowl 2 0.4
```

结果和我们之前得到的结论一样。比起我们在前面章节看到的，这段代码更复杂。

一个优点是，它可以推广到从同一个碗取不只一个曲奇饼（带替换）的情形：

```
dataset= ['vanilla', 'chocolate', 'vanilla']
for data in dataset:
    pmf.Update(data)
```

另一优点是，它提供了解决许多类似问题的框架。在下一节中，我们将解决蒙蒂大厅问题的计算，然后看看框架的哪些部分相同。

本节中的代码可以从 <http://thinkbayes.com/cookie2.py> 获得。欲了解更多信息，请参见前言的“代码指南”。

2.4 Monty Hall 难题

为了解决蒙蒂大厅（Monty Hall）问题，我将定义一个新的类：

```
class monty(Pmf):
    def __init__(self, hypos):
        Pmf.__init__(self)
        for hypo in hypos:
            self.Set(hypo, 1)
        self.Normalize()
```

到目前为止，蒙蒂大厅和曲奇饼是完全一样的。创建 Pmf 的代码也一样，除了假设的名称：

```
hypos='ABC'
pmf =Monty(hypos)
```

对 Update 的调用几乎是相同的：

```
data='B'
pmf.Update(data)
```

Update 的实现也是完全一样的：

```
def Update (self, data):
```

```

for hypo in self.Values():
    like = self.Likelihood(data, hypo)
    self.Mult(hypo, like)
    self.Normalize()

```

唯一需要些额外工作的是 Likelihood:

```

def Likelihood (self,data,hypo):
    if hypo==data:
        return 0
    elif hypo=='A':
        return 0.5
    else:
        return 1

```

最后，打印输出的结果是一样的：

```

for hypo, prob in pmf.Items():
    print hypo,prob

```

答案是

A 0.333333333333
B 0.0
C 0.666666666667

在本例中，Likelihood 的编写有一点点复杂，但该贝叶斯框架的 Update 很简单。本节中的代码可以从 <http://thinkbayes.com/monty.py> 获得。欲了解更多信息，请参见前言的“代码指南”。

2.5 封装框架

现在，我们看看框架的哪些元素是相同的，这样我们就可以把它们封装进一个 Suite 对象，即一个提供 __init__、Update 和 Print 方法的 pmf 对象：

```

class Suite(Pmf):
    """代表一套假设及其概率。"""

    def __init__(self, hypo=tuple()):
        """初始化分配。"""

    def Update(self, data):
        """更新基于该数据的每个假设。"""

    def Print (self):
        """打印出假设和它们的概率。"""

```

Suite 的实现 in thinkbayes.py 中。要使用 Suite 对象，你应当编写一个继承自 Suite 的类，并自行提供 Likelihood 方法的实现。例如，这是一个以蒙蒂大厅问题改写的使

用 Suite 的方案：

```
from thinkbayes import Suite

class Monty(Suite):

    def Likelihood(self, data, hypo):
        if hypo == data:
            return 0
        elif hypo == 'A':
            return 0.5
        else:
            return 1
```

而下面是一个使用这个类的代码：

```
suite=Monty('ABC')
suite.Update('B')
suite.Print()
```

你可以从 <http://thinkbayes.com/monty2.py> 下载这个例子。更多信息见前言的“代码指南”。

2.6 M&M 豆问题

我们可以使用 Suite 框架来解决 M&M 豆的问题。除了编写 Likelihood 有点棘手，其他一切都很简单。

首先，需要对 1995 年之前和之后的颜色混合情况进行封装：

```
mix94=dict(brown= 30,
            yellow= 20,
            red= 20,
            green= 10,
            orange= 10,
            tan= 10)

mix96=dict(blue= 24,
            green= 20,
            orange= 16,
            yellow= 14,
            red= 13,
            brown= 13)
```

然后，封装假设：

```
hypoA =dict(bag1 = mix94, bag2 = mix96)
hypoB =dict(bag1 = mix96, bag2 = mix94)
```

hypoA 表示假设袋 1 是 1994 年，袋 2 是 1996 年。hypoB 是相反的组合。

接下来，从假设的名称来映射其含义：

```
hypotheses=dict(A=hypoA, B=hypoB)
```

最后，开始编写 Likelihood。在这种情况下，假设 hypo 是一个 A 或 B 的字符串，数据是一个指定了袋子年份和颜色的元组。

```
def Likelihood(self, data, hypo):
    bag, color = data
    mix = self.hypotheses[hypo][bag]
    like = mix[color]
    return like
```

下面是创建该 suite 对象并进行更新的代码：

```
suite = M_and_M('AB')

suite.Update(('bag1', 'yellow'))
suite.Update(('bag2', 'green'))

suite.Print()
```

结果如下：

```
A 0.740740740741
B 0.259259259259
```

A 的后验概率大约是 20/27，正是我们之前得到的。

本节中的代码可以从 http://thinkbayes.com/m_and_m.py 获得。欲了解更多信息，请参见前言的“代码指南”。

2.7 讨论

本章介绍了 Suite 类，它封装了贝叶斯 update 框架。

Suite 是一个抽象类 (abstract type)，这意味着它定义了 Suite 应该有的接口，但并不提供完整的实现。Suite 接口包括 Update 和 Likelihood 方法，但只提供了 Update 的实现，而没有 Likelihood 的实现。

具体类 (concrete type) 是继承自抽象父类的类，而且提供了缺失方法的实现。例如，Monty 扩展自 Suite，所以它继承了 Update 并且实现了 Likelihood。

如果你熟悉设计模式，你可能会意识到这其实是设计模式中模板方法的一个例子。你可以从 http://en.wikipedia.org/wiki/Template_method_pattern 了解这个模式。

大多数在以下章节中的例子遵循相同的模式，对于每个问题我们定义一个扩展的 Suite 对象，继承了 Update 方法，并提供了 Likelihood。在少数情况下，会重写 Update 方法，

通常是为了提高性能的缘故。

2.8 练习

练习 2-1。

在第 11 页的“贝叶斯框架”里面，我提到了曲奇饼问题的解法是简化的，是曲奇饼有补充的取多个饼的情况（有放回的情况）。

但更可能的情况是，我们吃掉了取出的曲奇饼，那么似然度就依赖于之前的取曲奇饼行为（曲奇饼少了）。

修改本章中的解法以处理没有曲奇饼补充的情况。提示：添加 `Cookie` 的实例变量来表示碗的假设状态，并据此修改似然度。你可能要定义一个 `Bowl` 对象。

第3章

估计

3.1 骰子问题

假设我有一盒骰子，里面有 4 面的骰子、6 面的骰子、8 面的骰子、12 面的骰子和 20 面的骰子各 1 个。如果曾经玩过游戏《龙与地下城》，你当然会明白我的所指。

假如我随机从盒子中选一个骰子，转动它得到了 6。那么每一个骰子被选中的概率是多少？

我通过一个三步策略来解决这个问题。

1. 选择假设的表示方法。
2. 选择数据的表示方法。
3. 编写似然度函数。

在前面的例子中我用字符串来表示假设和数据，但骰子问题中我将使用数字。

确切地说我将使用整数 4、6、8、12 和 20 来表示假设：

```
suite=Dice([ 4, 6, 8, 12, 20 ])
```

另外，从 1 到 20 的整数作为数据。有了这些表达式，编写似然函数就很简单了：

```
class Dice(Suite):  
    def Likelihood(self, data, hypo):  
        if hypo < data:  
            return 0  
        else:  
            return 1.0/hypo
```

这里 Likelihood 的原理是：如果 $\text{hypo} < \text{data}$ ，意味着投骰子值大于骰子的面数，显然这是一个不可能的情形，所以似然度是 0。

另外的情形下，问题变成“考虑到所有假设的点数，得到某个点数结果的机会是多少？”

答案是 $1/\text{hypo}$ 无论数据是什么。

下面是使用 `update` (如果转动得到 6) 的语句声明：

```
suite.Update (6)
```

后验分布的结果如下：

```
4 0.0
6 0.392156862745
8 0.294117647059
12 0.196078431373
20 0.117647058824
```

当得到 6 后，骰子是 4 面的概率是 0。最可能的备选是 6 面骰，但也有约 12% 的可能是 20 面骰。

如果我们多摇几次，得到 6, 8, 7, 7, 5, 4 这样一组数据的情况下呢？

```
for roll in [ 6, 8, 7, 7, 5, 4 ] :
    suite.Update (roll)
```

结合该组数据，可以去掉 6 面骰的可能性了（因为有大于 6 的值），8 面骰看起来可能性很大。

结果如下：

```
4 0.0
6 0.0
8 0.943248453672
12 0.0552061280613
20 0.0015454182665
```

现在有 94% 的可能我们转动了一个 8 面骰，同时还有 1% 可能是一个 20 面骰。

骰子问题参考了我在 Sanjoy Mahajan 的贝叶斯推论课上看到的一个例子。

你可以从 <http://thinkbayes.com/dice.py> 下载代码。欲了解更多信息，请参见前言的“代码指南”。

3.2 火车头问题

我是在弗雷德里克·莫斯泰勒的《五十个概率难题的解法》(多佛出版社, 1987)一书中发现火车头问题的：

铁路上以 1 到 N 命名火车头。有一天你看到一个标号 60 的火车头，请估计铁路上有多少火车头？

基于这一观察结果，我们知道铁路上有 60 个或更多的火车头。但这个数字到底是多少？要应用贝叶斯进行推理，我们可以将这个问题分成两步骤进行：

1. 在得到数据之前，我们对 N 的认识是什么？
2. 已知一个 N 的任意值后，得到数据（“标志为 60 号的火车头”）的似然度？

第一个问题的答案就是问题的前置概率。第二个问题是似然度。

在选择前置概率上，我们还没有太多的基本信息，但我们可以从一些简单情况开始，再考虑进一步的方法。假设 N 可以是从 1 到 1000 等概率的任何值。

```
hypos= xrange (1, 1001)
```

接着我们需要的是一个似然函数。先假设存在一个有 N 个火车头的车队，我们看到 60 号火车头的概率是多少？假设只有一个列车运营公司（或者只有一个我们关注的公司），看到任意一个火车头有同等可能，那么看到的任何特定火车头的机会为 $1/N$ 。

似然度函数如下：

```
class Train(Suite) :  
    def Likelihood(self, data, hypo) :  
        if hypo<data:  
            return 0  
        else:  
            return 1.0/hypo
```

看起来很熟悉，似然函数在火车头问题和骰子问题上是相同的。

Update 如下：

```
suite=Train (hypos)  
suite.Update (60)
```

因为有太多的假设（1000）要打印输出，所以我绘制了如图 3-1 所示的结果。意料之中的是， N 中 60 以下的所有值都被去掉了。

如果非要猜测的话，最可能的值是 60。这似乎算不上很好的结果，毕竟，想想你恰好看到最高标志号火车头的机会是多少呢（应该不高吧）？不过，如果想使猜到的答案完全正确的可能性最大化，你应该猜 60。

不过，这还不是我们的目标。另一个可选的方法是计算后验概率的平均值分布：

```
def Mean(suite):  
    total= 0  
    for hypo, prob in suite.Items ():  
        total += hypo*prob  
    return total  
print Mean(suite)
```

或者你可以用由 Pmf 提供的非常类似的方法：

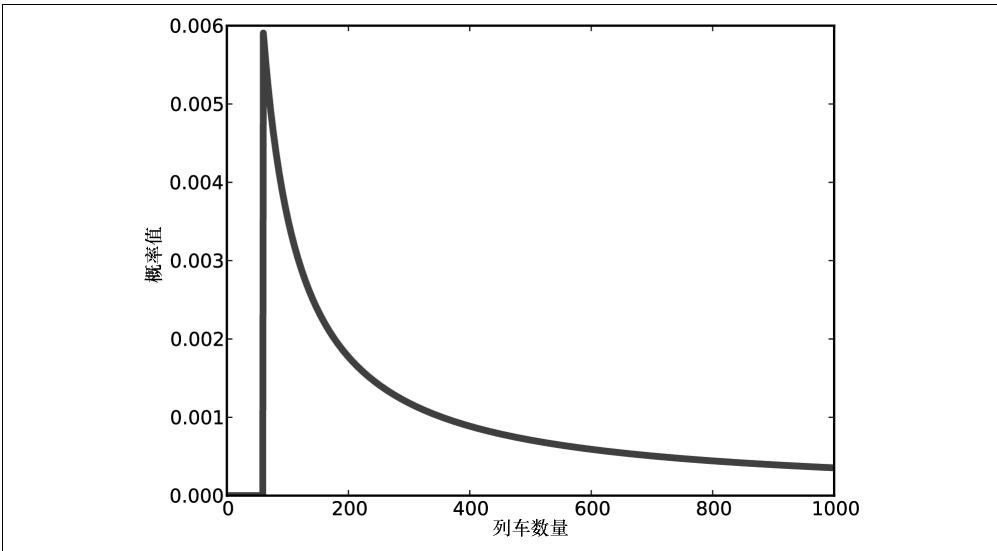


图 3-1 火车头问题的后验分布，基于均匀分布的先验

```
print suite.Mean ()
```

后验的平均值是 333，所以要是你想最大限度地减少错误，这也许是一个很好的猜测结果。

如果你一遍又一遍地玩这个猜谜游戏，使用后验概率的平均值来作为估计值会减少从长远来看的均方差（参考 http://en.wikipedia.org/wiki/Minimum_mean_square_error）。

你可以从 <http://thinkbayes.com/train.py> 下载这个例子。如需更多信息参见前言的“代码指南”。

3.3 怎样看待先验概率？

为了进一步解决火车头问题，我们必须进行一些假设。

其中一些是相当武断的。尤其当我们选择了在 1-1000 间的均匀随机分布的先验概率。其实并没有特别的理由选择 1000 作为上界或均匀随机来作为分布特征。

相信一个运营着 1000 个火车头的铁路公司算不上疯狂，但一个理性点的人可能会对这一问题做出更多或更少的猜测。

因此，我们可能想知道是否后验概率分布对这些假设敏感。仅仅依赖一次观察的小量数据，情况可能真是如此的（敏感）。

回忆一下，从 1 到 1000 的均匀分布的先验概率，后验概率的平均值是 333。同样上界

为 500，我们得到的后验平均值为 207，一个 2000 的上界后验平均值为 552。

所以结果很糟（猜测结果对上界敏感）。有两种方法继续进行分析：

- 获取更多的数据。
- 更多的背景信息。

有了更多的数据后，基于不同的先验概率，后验分布趋于收敛。

例如，假设除了列车 60 我们也看到列车 30 和 90。我们可以这样更新分布：

```
for data in[60, 30, 90]:  
    suite.Update (data)
```

采用这些数据时，后验概率的均值是

上限	后验均值
500	152
1000	164
2000	171

这样差异就较小了。

3.4 其他先验概率

如果没有更多的数据，另一个方法是通过收集背景资料优化先验。

大型和小型公司有同等可能性的假设也许相当不合理，大型公司可能有 1000 台火车头，小型公司仅有 1 台火车头。

通过一些努力，我们很有可能发现在观察区域内火车运营公司的清单，或者可以采访铁路运输专家来收集这些公司一般规模的信息。

但是，即使没有深入了解铁路产业的一些具体情况，我们也可以做一些猜测。在大多数领域，有大量小型公司，一些中型公司，一个到两个非常大型的公司。

事实上，公司规模的分布往往遵循幂律，参考罗伯特·阿克斯特尔在《科学》杂志上的报道 (<http://www.sciencemag.org/content/293/5536/1818.full.pdf>)。

这规律表明，如果少于 10 个火车头的公司有 1000 家，100 个火车头的公司可能有 100 家，1000 个火车头的公司有 10 家，10000 个火车头的公司可能仅有 1 家。

在数学上，幂律表示公司的数量与公司规模成反比，或

$$\text{PMF}(x) \propto \left(\frac{1}{x}\right)^{\alpha}$$

其中 $\text{PMF}(x)$ 是 x 的概率质量函数， α 是一个通常接近于 1 的参数。

我们可以构造一个服从幂律分布的先验如下：

```
class Train (Dice):  
  
    def __init__ (self, hypos, alpha = 1.0):  
        Pmf.__init__ (self)  
        for hypo in hypos:  
            self.Set (hypo, hypo** (-alpha))  
        self.Normalize ()
```

而下面是生成前置概率的代码：

```
hypos=range(1,1001)  
suite=Train(hypos)
```

再次说明，上限是任意的，但对于幂律分布的先验概率，后验概率对这一选择的敏感性较小。

图 3-2 表示了基于幂律后与之前基于均匀随机后验概率的比较。

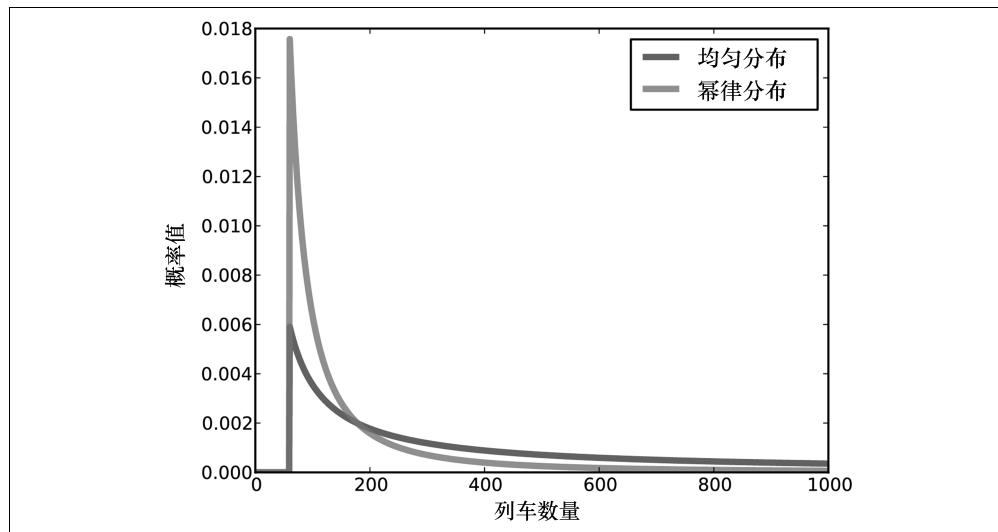


图 3-2 前验为幂律的后验分布，对比于均匀前验分布

考虑了幂律分布作为背景资料后。我们可以消除大于 700 的 N 值了。

如果基于这种先验概率，在观察到列车 30、60 和 90 时，后验概率分别是：

上限	后验均值
500	131
1000	133
2000	134

现在的差异小得多了（考虑不同上界的假设）。事实上，考虑一个任意大的上界，平均值都收敛于 134。

所以基于幂律分布的先验概率是比较现实的，因为它基于公司规模的一般情况，并且在实际中表现得更好。

你可以从 <http://thinkbayes.com/train3.py> 下载本节程序的例子。欲了解更多信息，请参见前言的“代码指南”。

3.5 置信区间

一旦计算出的后验分布，通过单点估计或区间对后验分布进行总结通常是有用的。

对于点估计，通常使用平均数、中位数或最大似然值。

对于区间，我们通常给出两个计算值，使得未知量有 90% 的可能落入这两个值之间（或者任何其他的概率值）。这些值定义了一个置信区间。

计算置信区间的一个简单方法是在后验概率分布中累加其中的概率，并记录对应于概率 5% 和 95% 的值。也就是说，第 5 和第 95 百分位。

thinkbayes 提供了一个函数计算百分位数：

```
def Percentile(pmf, percentage):
    p = percentage / 100.0
    total = 0
    for val, prob in pmf.Items():
        total += prob
        if total >= p:
            return val
```

下面是一个应用代码：

```
interval = Percentile(suite, 5), Percentile(suite, 95)
print interval
```

在前面的示例（看到了三个火车，且呈幂律分布的先验概率的火车头问题）中 90% 置信区间为 (91,243)。如此大的范围其实确切的表明，（尽管平均值收敛了）我们仍然相当不确定究竟有多少火车头存在。

3.6 累积分布函数

在上一节中，我们通过数值迭代计算出百分比和 Pmf 的概率。如果我们需要计算多个百分位数，更有效方法是使用累积分布函数，或 Cdf。

由于包含有某个分布的相同的信息，在这个意义上 Cdf 和 Pmf 是等价的，并可以随时

从一个转换到另一个。Cdf 的优点是可以更有效地计算百分位数。

thinkbayes 有一个表示累积分布函数的 Cdf 类。Pmf 提供了一种方法生成相应的 Cdf:

```
cdf = suite.MakeCdf ()
```

Cdf 提供了一个名为 Percentile 的函数

```
interval = cdf.Percentile(5), cdf.Percentile(95)
```

将 Pmf 转换为 Cdf 需要正比于值数量 $\text{len}(\text{pmf})$ 的运算时间。Cdf 将值和概率存储在有序列表里 (list)，所以查询某个概率得到相应的值需要“对数时间 (log time)”: 即，时间和值的数量的对数成正比。查询一个值获得对应的概率也是对数时间，所以 Cdf 对于很多计算来说都是有效的。

本节中的例子位于 <http://thinkbayes.com/train3.py>。欲了解更多信息参见前言的“代码指南”。

3.7 德军坦克问题

第二次世界大战期间，在伦敦的美国大使馆战争经济部门使用统计分析来估计德国生产的坦克和其他装备^①。

西方盟军获得了一份记录簿，记录了存货和修理记录，其中包括坦克的底盘和发动机的序列号。

这些记录的分析表明，制造商为坦克类型分配了以 100 为一个区间的序列号，每个区间内的号码都是成序列的，但并不是每个区间内的数字都用到了。如此一来，在每个 100 的区间内，估算德军坦克问题的范围就可以按照之前的火车头问题来缩小了。

基于这种认识，美国和英国的分析师们完成了远远小于源自情报部门其他形式情报的估算结果。而在战后，记录显示这些结果实质上更准确。

他们也对轮胎、卡车、火箭等设备进行类似的分析，产生准确实用的经济情报。

德军坦克问题是个历史上有趣的问题，也是一个很好的在现实世界应用统计估计的例子。到目前为止，本书的许多例子都是游戏性质的问题，但我们马上会开始解决实际问题。我认为它是贝叶斯分析的优点，特别是在我们正在使用的计算方法上，它提供了一条从基础介绍到研究前沿的捷径。

3.8 讨论

贝叶斯当中，有两种途径选择先验分布。一些人建议选择最能代表问题相关背景资料

^① 拉格尔斯，布罗迪.实证方法经济情报大战[J].《美国统计协会》杂志，1947，卷 42，第 237 号。

的先验概率，在这种情况下，先验被认为是“信息”。问题是，人们可能会使用不同的背景信息（或者进行不同的诠释）。所以基于信息的先验往往显得主观。

另一种方法是所谓的“无信息参考的先验”，其目的是为了让数据来说话，越没有约束越好。在某些情况下，你可以选择包含一些期望属性的特殊先验，例如，就估计量设置一个最小先验。

“无信息先验”观点存在是因为它们似乎更为客观。但通常，我倾向使用先验信息。为什么呢？首先，贝叶斯分析总是基于模型决策的。选择先验就是决策之一，但它不是唯一的一部分，甚至可能不是最主观的。因此，即使无信息先验较为客观，整个分析本身仍然是主观的。

另外，对于大多数实际中的问题，你很可能是在两个（对立面）之间：也许有大量的数据，也许没有。如果你有大量的数据，先验的选择不是特别关键；信息先验和无信息先验会得到几乎相同的结果。我们会在下一章看到类似的例子。

不过，如果像火车头问题，如果你没有太多的参考数据，那么采用相关的背景信息（如幂律分布）就有很大区别了。

而比如德军坦克问题，如果必须基于你的结论做出生死存亡的决策，你就应该利用所有的信息，而不是在“要保持客观”的幻觉中假装不了解具体情况。

3.9 练习

练习 3-1。

为火车头问题写一个似然函数，我们必须要回答这个问题：“如果铁路上有 N 个火车头，我们看到 60 号的概率是多少？”

答案取决于当我们观察到火车头时，所采用的取样过程。

在本章中，我通过指定只有一个列车运营公司（或只有一个我们关心）解决了这个模棱两可的问题。

但是假设有很多家使用不同火车头号码编排的公司，再假设看到任意公司经营的任意火车的可能性相同。在这种情况下，似然函数是不同的，因为你更有可能看到一列由大型公司运营的火车。

作为练习，实现火车头似然函数的这种变化，并比较结果。