Probabilistic Programming and Bayesian Methods for Hackers

概率编程和贝叶斯方法实践

2014-7-15

作者:钟超

目录

第 1 章 贝叶斯方法原则及概率编程初步

1.1 贝叶斯推断的哲学意义

即使你是一个非常牛 B 的程序猿，你开发的程序也会有 bug。一天，你开发了一个巨牛 B 的算法，然后开始尝试你的算法，第一步算法通过了一个简单的测试，然后你增加了测试的难度，你的算法又通过了，最终你的算法一次又一次的通过了越来越难的测试。这时你认为你的算法中不会存在 bug 了。

如果你按照上面的描述中思考问题，恭喜你，你已经是个贝叶斯者了！贝叶斯推断过程就是通过观察新发生的现象，来更新某一事件能否发生的可信度（例如开发出的算法无 bug 的可信度有多高？由于算法一次次的通过了测试，所以算法无 bug 的可信度就逐渐很高）。贝叶斯方法很少对结果给出确定性答案，但是他给的答案却很值得相信。就像在上面的例子中，我们无法确定算法 100%无 bug，除非把所有可能的情况都测试一遍，在实际当中这是无法实现的。然而，我们可以在相对大量的可能情况下进行测试，如果测试结果都无误，即使无法确认算法一定准确无误，我们也会对自己的算法很有信心，随着新的测试结果出现，算法无误的可信度也在逐渐改变。总之，由于无法穷举所有可能性，贝叶斯推断基本上不能给出肯定的结果。

1.2 贝叶斯思维方式

贝叶斯方法对推断结果的不确定性，使其与传统的统计推断有很大的不同。这让人感觉贝叶斯方法不那么牛 B 了。难道统计学不就是从随机中获取确定性的信息吗？为了更好的理解确定性和随机性，我们需要像贝叶斯方法一样思考。

在贝叶斯世界里，概率是对一个事件可信度的度量，也就是说，我们有多大信心相信一个事件会发生。事实上，稍后我们会发现，这正是对概率最原生的解释。

为了解释清楚概率，我们以非正统的观点解释什么是概率：在经典的统计学中，频率轮者认为，大量事件发生的频度就是概率。例如，在频率论者的思想体系中，飞机发生事故的概率，就是很长一段时间内飞机发生事故的频度。频率学观点可以解释很多事件发生的概率，但是当事件发生的频度变小时，即不是长期统计的结果，频率学派的概率估计的结果误差会很大。Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of occurrences defines the probability.

而贝叶斯方法，采用更符合直觉的方式来解释这一问题。贝叶斯方法将概率看成是对事件发生的可信度（或者对事件发生的信心）的度量。简单的说，概率是对某一判定的概述。如果一个人认为某一事件发生的可信度为 0，说明这个人不相信这个事件会发生；反之，如果其认为事件发生的可信度为 1，说明其相信这个事件一定会发生。由于还存在其他的可能结果，所以可信度的取值区间为[0,1]。贝叶斯这种解释概率的方式，依然可以很好的解释飞机出现故障的概率问题，不考虑其它的任何外界因素，飞机出现故障的可信度就等于出现故障的频率。所以，在这种概率定义方式下，概率就等于可信度，所以此时谈论总统大选结果的概率问题时才有意义：你对候选总统 A 能够赢得竞选的信心有多大？

以上段落介绍的可信度（概率），都是对个人而言的

1.3 贝叶斯推断在实际中的应用

.....

1.4 频率论者的方法错了吗

频率论者的方法也没有错。例如，基于最小二乘的线性回归，LASSO 回归，EM 算法，都是很高效的方法。贝叶斯方法只是对频率学派进行补充，处理一些频率学派无法处理的问题。

注意：在对大数据进行预测，分析时使用的算法往往相对简单[2][4]。可以断言，处理大数据的困难不在于算法如何复杂，而在与大数据计算、存储等方面的困难（One should also consider Gelman's quote from above and ask "Do I really have big data?"）。

与大数据相比，对中等规模数据的分析会增加不少难度，对小规模数据的分析确实是个不小的挑战。就如 Gelman's 所说，如果能够很便利的处理大数据，我们更应该关心中等规模或者规模很小的数据问题。

1.5 我们的贝叶斯框架

我们关心可信度，因为其和贝叶斯理论中的可信性有关。由于有以前的信息存在，我们对事件 A 有一个先验的可信度。例如在上面讲到的代码 bug 测试的例子中，在测试之前就存在先验的可信度。

继续以代码 bug 测试为例，如果代码通过了 X 次测试，我们想通过现在的这个结果更新我们对代码 bug 存在与否的可信度。将新的可信度称为后验概率。通过下式可以对可信度进行更新，即贝叶斯理论

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)} \propto \mathrm{P}(X|A)P(A)$$

以上的表示并不是贝叶斯推断所独有的，其它地方也存在这种数学表示。只是贝叶斯推使用它将先验概率$P(A)$和后验概率$P(A|X)$联系起来而已。

1.6 示例

1.6.1 抛硬币实验

任何统计的文章中都会有抛硬币的例子，这里将展示抛硬币实验与通常文章中介绍的将不尽相同。假设你不太确定头朝上的概率是多少（剧透一下，头朝上的概率为 50%）。你坚信真理就藏在每次抛硬币的结果中，头朝上的概率可能为p吗，但是又不知道p的先验概率是多少，怎么办？

我们开始抛硬币，记录观察结果，结果为H或者T。有了这些观测数据。我们可能会问一个有趣的问题，随着观察的数据越来越多，推断如何变化？更具体的说，当数据很少时后验概率是什么样的，如果数据变大后验概率又该如何变化？

随着数据增加（抛硬币的次数），下面画出一系列的后验概率更新图。Python 脚本如下。

```
"""
The book uses a custom matplotlibrc file, which provides the unique styles for
matplotlib plots. If executing this book, and you wish to use the book's
styling, provided are two options:
   1. Overwrite your own matplotlibrc file with the rc-file provided in the
      book's styles/ dir. See http://matplotlib.org/users/customizing.html
   2. Also in the styles is  bmh_matplotlibrc.json file. This can be used to
      update the styles in only this notebook. Try running the following code:
       import json, matplotlib
       s = json.load( open("../styles/bmh_matplotlibrc.json") )
       matplotlib.rcParams.update(s)
"""
# The code below can be passed over, as it is currently not important, plus it
# uses advanced topics we have not covered yet. LOOK AT PICTURE, MICHAEL!
%matplotlib inline
from IPython.core.pylabtools import figsize
import numpy as np
from matplotlib import pyplot as plt
figsize(11, 9)
import scipy.stats as stats
dist = stats.beta
n_trials = [0, 1, 2, 3, 4, 5, 8, 15, 50, 500]
data = stats.bernoulli.rvs(0.5, size=n_trials[-1])
x = np.linspace(0, 1, 100)
# For the already prepared, I'm using Binomial's conj. prior.
for k, N in enumerate(n_trials):
   sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
   plt.xlabel("$p$, probability of heads") \
      if k in [0, len(n_trials) - 1] else None
   plt.setp(sx.get_yticklabels(), visible=False)
   heads = data[:N].sum()
   y = dist.pdf(x, 1 + heads, 1 + N - heads)
   plt.plot(x, y, label="observe %d tosses,\n %d heads" % (N, heads))
   plt.fill_between(x, 0, y, color="#348ABD", alpha=0.4)
   plt.vlines(0.5, 0, 4, color="k", linestyles="--", lw=1)
   leg = plt.legend()
   leg.get_frame().set_alpha(0.4)
   plt.autoscale(tight=True)
plt.suptitle("Bayesian updating of posterior probabilities",
             y=1.02,
             fontsize=14)
plt.tight_layout()
```

注意：

Python 脚本中的曲线实际上是由下面这条语句画出的 y = dist.pdf(x, 1 + heads, 1 + N - heads)这条语句对应的是 Beta 分布，公式如下

$$\text{Beta}(\mu|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1}$$

μ相当于 x，a,b 分别对应于参数 1+heads 和 1+N-heads

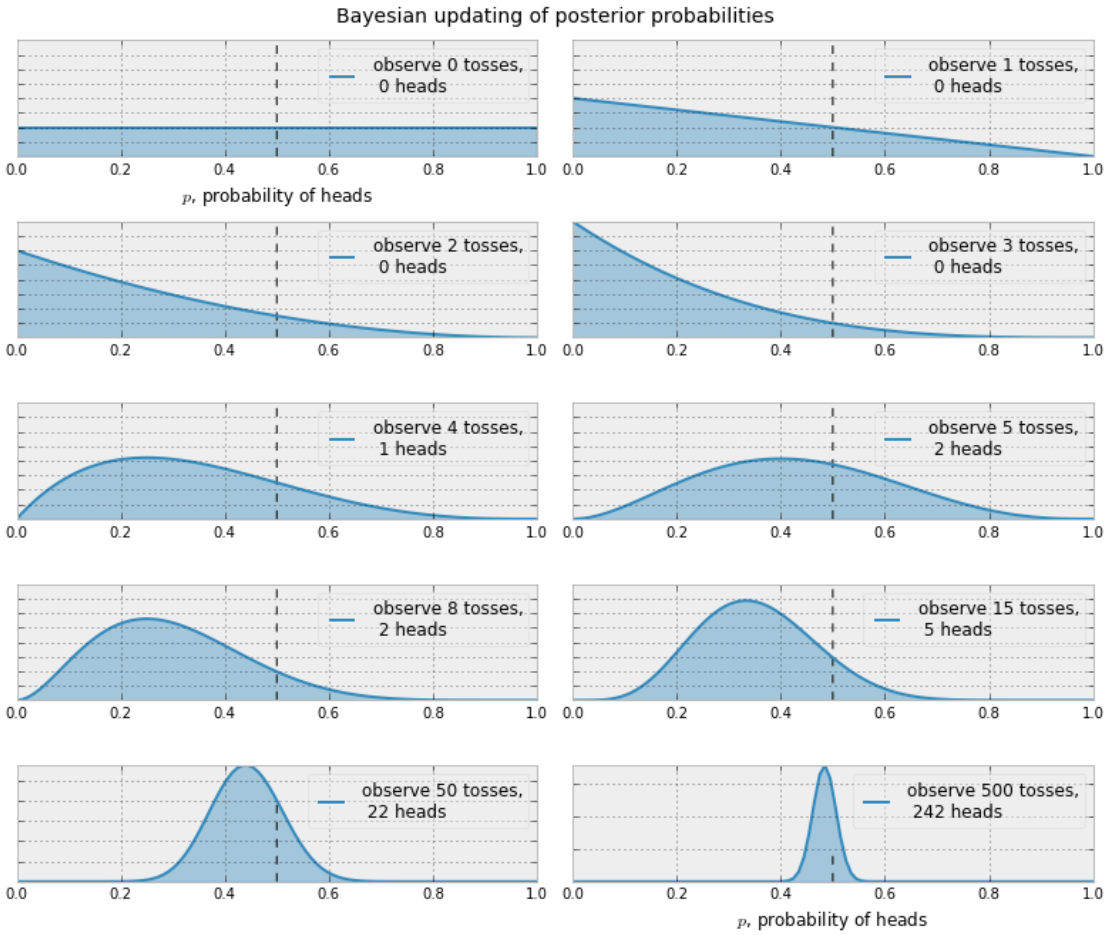


图** 后验概率密度曲线变化图

上图是后验概率曲线，不确定性与曲线的宽度成比例。从上图中可以看出，当观察数据变化时曲线也在移动变化。最终随着观察数据变的越来越多，概率密度越来越靠近真实值$p = 0.5$（图中的虚线标定的位置）。

注意，图中的最终曲线并不在峰值 0.5 处。回忆一下前面的假设，我们并不知道先验概率p是多少。事实上我们观测的是很极端的数据，抛了 8 次硬币，其中只有 1 次是头朝上，所以分布曲线的峰值离 0.5 很远（因为在没有先验知识的情况下，显然无法通过 8 次试验中的一次头朝上的结果，表明头朝上的概率为 0.5）。随着数据累加，概率会越来越向 0.5 移动，虽然不会完全等于 0.5。

1.6.2 Bug, or just sweet, unintended feature?

用 A 表示代码无 Bug 事件。用 X 表示代码通过了所有测试这一事件。现在就可以将无 Bug 的先验概率表示为$P(A) = p$。

我们感兴趣的量是$P(A|X)$，即在代码通过测试 X 的条件下，代码无 Bug 的概率是多少。使用上面介绍的贝叶斯推理框架，我们需要计算一些先验概率。

$P(X|A)$代表在代码无 Bug 条件下，代码测试通过的概率是多少。如果在没有 Bug 的条件下全部通过了代码测试则$P(X|A) = 1$。

$P(X)$的处理就有点复杂，事件X可以被分成两部分，事件 X 中确实有 Bug 存在的部分用$\sim A$表示，没有 Bug 存在的部分依然用A表示，此时$P(X)$可以表示成如下形式

$$P(X) = P(X \text{ and A}) + P(X \text{ and } \sim\text{A}) = P(X|A)P(\text{A}) + P(X|\sim A)P(\sim\text{A})$$
$$= P(X|A)p + P(X|\sim A)(1-p)$$

上面已经计算了$P(X|A)$。另一方面从直觉来看$P(X|\sim A)$表示的意义是：虽然随着代码通过测试，代码存在的 Bug 可能性在减少，但是代码仍然可能有 Bug 存在。注意，$P(X|\sim A)$和实验的次数，以及实验过程中的复杂程度有关。给$P(X|\sim A)$设一个保守值为$P(X|\sim A) = 0.5$，此刻有

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)} = \frac{2p}{1+p}$$

这就是后验概率，$p\epsilon[0,1]$。

下面的脚本画出了后验概率$P(A|X)$

```
import numpy as np
import matplotlib.pyplot as plt
p = np.linspace(0, 1, 50)
plt.plot(p, 2 * p / (1 + p), color="#348ABD", lw=3)
# plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A60628"])
plt.scatter(0.2, 2 * (0.2) / 1.2, s=140, c="#348ABD")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel("Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title("Are there bugs in my code?")
plt.show()
```

图如下

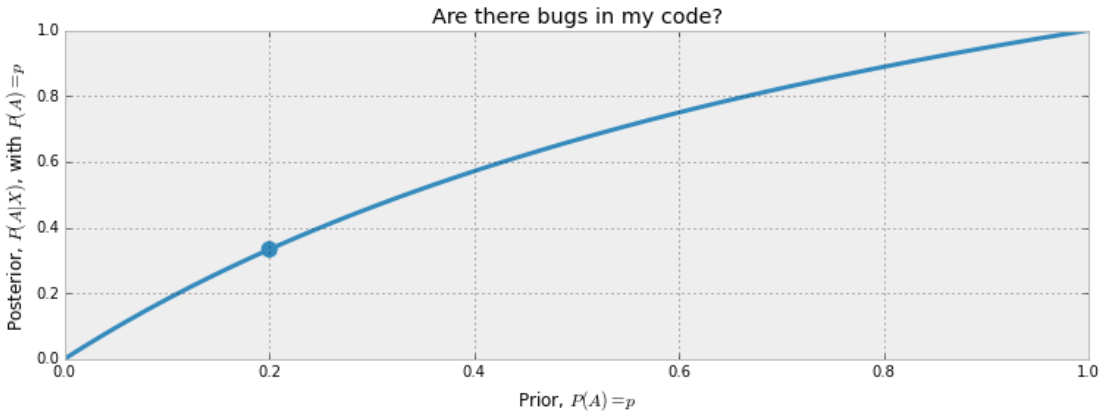


图** 后验概率图

从图中可以看出当先验概率p较小时，如果观测到 X 测试都通过了，那么可以等到较大的后验概率。让我们为先验概率设置一个具体的值。我自认为自己是个很牛 B 的程序猿，所以设置一个比较真实的先验概率值为 0.2，也就是说我写的代码有 20%的可能性是没有 Bug 的。为了实际情况更吻合，这个先验概率是代码复杂度和代码规模的函数，我们在图中标注出 0.2 对应的位置。此时我的代码无 Bug 的后验概率是 0.33。

由于此处的先验信息代表的是没有 Bug 的概率p，那么有 Bug 的先验概率就是$1-p$。同理，$P(A|X)$表示所有测试都通过的前提下没有 Bug 的概率是多少，$1-P(A|X)$就表示在所有测试都通过的条件下，再次测试存在 Bug 的概率是多少。我们的后验概率看起来到底应该是什么样的？下面给出先验概率和后验概率的柱状图。

```
import numpy as np
import matplotlib.pyplot as plt
colours = ["#348ABD", "#A60628"]
prior = [0.20, 0.80]
posterior = [1. / 3, 2. / 3]
plt.bar([0, .7], prior, alpha=0.70, width=0.25,
      color=colours[0], label="prior distribution",
      lw="3", edgecolor=colours[0])
plt.bar([0 + 0.25, .7 + 0.25], posterior, alpha=0.7,
      width=0.25, color=colours[1],
      label="posterior distribution",
      lw="3", edgecolor=colours[1])
plt.xticks([0.20, .95], ["Bugs Absent", "Bugs Present"])
plt.title("Prior and Posterior probability of bugs present")
plt.ylabel("Probability")
plt.legend(loc="upper left");
plt.show()
```

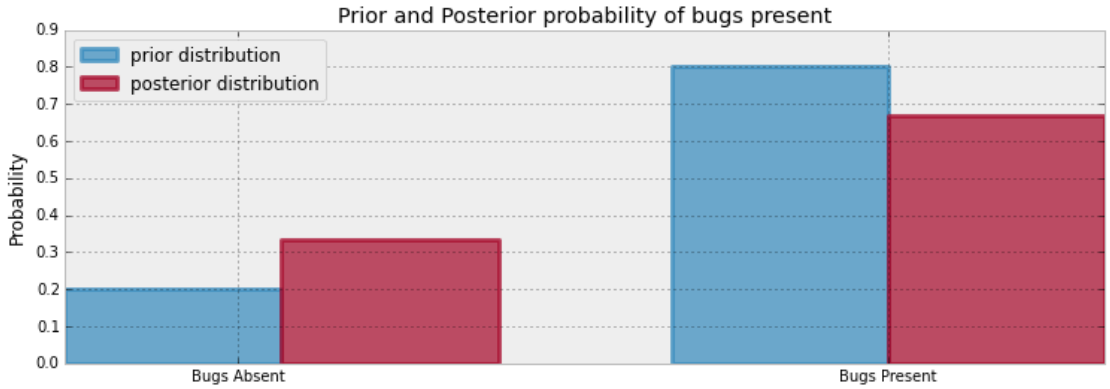


图** 先验和后验柱状图

注意，随着观测到的X发生，代码没有 Bug 的概率在减少。通过的测试越来越多，我们离概率 1 越来越近。确信没有 Bug 存在。

这是一个非常简单的贝叶斯推断和贝叶斯准则的例子。不幸的是，除了上面这特特殊构造的示例外，贝叶斯推断都需要复杂的数学计算，这就导致贝叶斯推断变的非常难。稍后会看到，实际上这种类型的数学分析是不必要的。首先我们要扩展我们的模型工具。下一个例子是概率分布，如果你已经很熟悉了，可以跳过下个示例。

1.6.3 概率分布

首先迅速回忆下什么是概率分布：假设Z为随机变量，则有个函数描述不同的 Z 发生的可能性。如果用图形来表示，概率分布就是一个曲线，曲线值越大表示概率越大。

随机变量可以分为三类

- **Z 是离散的**：离散的随机变量只能取特殊的序列值。比如流行度，电影评分，选票数都可以作为离散随机变量的取值，当我们将离散随机变量与。。。对比时我们对它的认识将变得非常清晰。
- **Z 是连续的**：连续随机变量可以取任何值。比如，温度、速度、时间、颜色等
- **Z 是混合的**：离散和连续的随机变量都可以有概率存在，它是连续和离散的组合。

离散随机变量

如果变量Z是离散的，则它的分布称为概率分布函数，表示Z取k的可能性，用$P(Z = k)$表示。用概率分布函数完全能够描述随机变量 Z，也就是说如果知道了概率分布函数，也就知道了变量 Z 的行为。有些经常出现且非常流行的概率分布函数，在需要的时候我们会逐步介绍，不过首先介绍个非常有用的概率分布函数，即泊松分布

$$\mathrm{P}(Z = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad \mathrm{k} = 0,1,2,\cdots$$

λ是分布参数，它控制着分布形状。对于泊松分布来说，参数λ可以取任何正数。增加λ，峰值向右移动（横轴上大的值取得概率较大），减小λ峰值向左移动（横轴上小的值取得概率较大）。可以将λ看成泊松分布的强度控制参数。

k必须为非负整数。这是非常重要的，因为如果你要用泊松分布模拟流行度，你不可能用 4.25 或者 5.612 表示商品的销售数量。

如果变量 Z 符合泊松概率分布，可以表示成如下形式

$$Z \sim \mathrm{Poi}(\lambda)$$

泊松有个非常重要的性质，它的期望等于它的参数，即

$$E\{Z|\lambda\} = \lambda$$

会经常使用这个性质。下面画出不同λ参数的概率分布函数。首先应该看到随着λ增加，大的 Z 获得了更大的概率，其次虽然分布图只显示到 15 这个位置，都是实际的分布并没有结束。每个非负整数都会有一个概率存在。

```
import scipy.stats as stats
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(16)
poi = stats.poisson
lambda_ = [1.5, 4.25]
colours = ["#348ABD", "#A60628"]
plt.bar(a, poi.pmf(a, lambda_[0]), color=colours[0],
      label="$\lambda = %.1f$" % lambda_[0], alpha=0.60,
      edgecolor=colours[0], lw="3")
plt.bar(a, poi.pmf(a, lambda_[1]), color=colours[1],
      label="$\lambda = %.1f$" % lambda_[1], alpha=0.60,
      edgecolor=colours[1], lw="3")
plt.xticks(a + 0.4, a)
plt.legend()
plt.ylabel("probability of $k$")
plt.xlabel("$k$")
plt.title("Probability mass function of a Poisson random variable; differing \
$\lambda$ values")
plt.show()
```

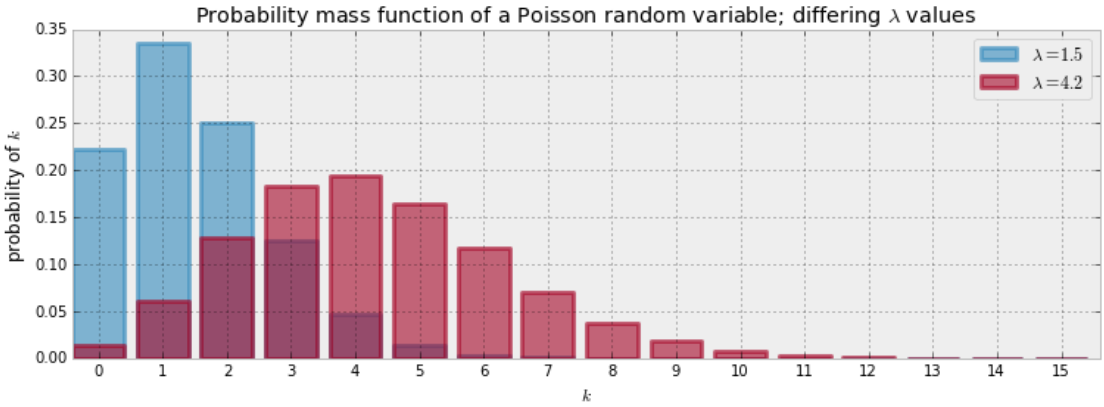


图** 不同参数的泊松分布图

连续随机变量

连续的随机变量有概率密度函数。看起来不必要区分密度函数和分布函数，实际上这两者是不同的概念。连续随机变量具有指数衰减密度。一个指数随机变量的密度函数有如下形式

$$f_Z(z|\lambda) = \lambda e^{-\lambda z}, z \geq 0$$

和泊松随机变量类似，指数随机变量只取非负值。但是和泊松随机变量不同的是，指数随机变量可以去任何非负值，包括非负的浮点数，如 4.25 或 5.612401。这一特点是其基本上不能用在离散数据上，因为离散数据必须是整数。但是当数据是时间，温度，或者其它任何精度的正数时，就需要使用连续的密度函数。下面的图形显示了不同λ的密度函数。

当随机变量 Z 是指数分布时，就说 Z 是指数的，表示如下

$$Z \sim \text{Exp}(\lambda)$$

给定具体的参数λ，指数随机变量的期望等于参数的倒数，即

$$\text{E}\{Z|\lambda\} = \frac{1}{\lambda}$$

Python 脚本如下

```
import scipy.stats as stats
import matplotlib.pyplot as plt
import numpy as np
plt.figure(2)
a = np.linspace(0, 4, 100)
expo = stats.expon
lambda_ = [0.5, 1]
for l, c in zip(lambda_, colours):
    plt.plot(a, expo.pdf(a, scale=1. / l), lw=3,
             color=c, label="$\lambda = %.1f$" % l)
    plt.fill_between(a, expo.pdf(a, scale=1. / l), color=c, alpha=.33)
plt.legend()
plt.ylabel("PDF at $z$")
plt.xlabel("$z$")
plt.ylim(0, 1.2)
plt.title("Probability density function of an Exponential random variable;\
 differing $\lambda$");
plt.show()
```

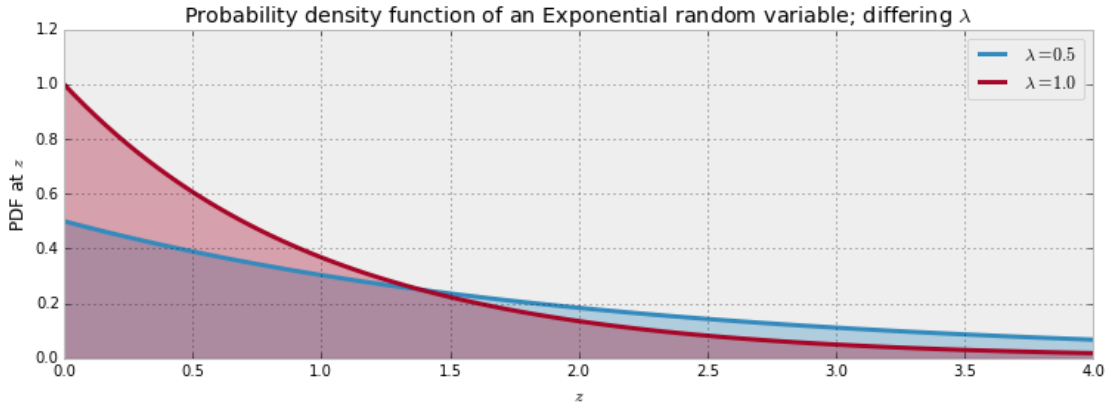


图** 不同参数的指数分布图

λ是什么

有哪些因素在控制着统计特征？对于我们来说，在现实世界中参数**λ**隐藏的。我们能看到的仅仅是Z，必须通过观测到的Z才能确定**λ**。由于没有从Z到λ的一一对应关系，所以求解参数λ比较困难。有很多方法都可以用来估计参数λ，但是由于参数λ没办法确切观测到，没有人敢说某一个方法是最好的。

贝叶斯推断所关心的是与参数**λ**相关的可信度是什么样的。贝叶斯推断并不会猜测确切的λ是什么样的，而其只是考虑如果给参数λ附上一个概率分布，那么λ的后验概率将会变成什么样子。

这看起来似乎有点奇怪。毕竟，λ是确定的参数，它不是也不必要成为随机的！我们如何给一个非随机变量附上一个概率？嗯，在传统的频率论者面前我们已经失败了。回忆一下，如果频率解释为可信度，那么我们就可以给一个非随机变量附上一个概率值。完全可以用可信度描述参数**λ，所以也就可以为参数λ**赋上一个概率，当然可以将**λ**当成随机变量了。

1.6.4 从短信中推断行为

这是一个更有趣的例子，我们关心短信用户发送和接受短信的概率是多少：

给你一些短信信息。横轴为时间，数据显示如下图。我们想知道用户的发短信行为是否会随着时间逐渐变化，还是会有剧烈的变化，如何来模拟这一过程。

```
import numpy as np
import matplotlib.pyplot as plt
count_data = np.loadtxt("data/txtdata.csv")
n_count_data = len(count_data)
plt.bar(np.arange(n_count_data), count_data, color="#348ABD")
plt.xlabel("Time (days)")
plt.ylabel("count of text-msgs received")
plt.title("Did the user's texting habits change over time?")
plt.xlim(0, n_count_data)
plt.show()
```

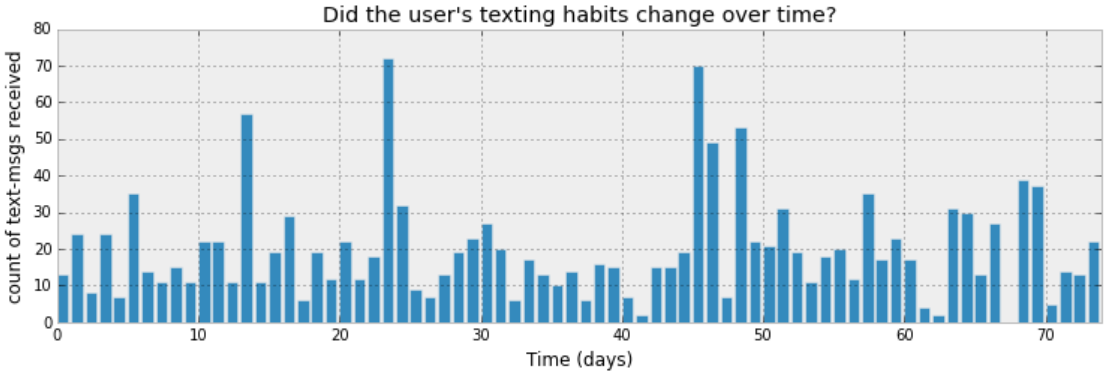


图** 每天收到的短信数量图

开始模拟之前，从上图可以看出什么规律。从中可以看出这段时间内短信行为上有改变吗？

如何模拟这一过程？从数据中可以看出，泊松分布非常适合模拟这一数据。用C_i表示第i天的短信数量，C_i服从泊松分布，记为

$$C_i \sim Poisson(\lambda)$$

λ为未知参数。从泊松分布的图中可以看到随着观测到的样本越多，概率会变得越大，这也就等价于在某一个点上的概率随着参数λ增加而增加（回忆一下在泊松分布示例中，λ参数越大，概率密度函数的峰值越向右移动，也就意味着如果λ变大，那么在某一天中发送较多消息的概率就比较大）。

如何这一过程用数学进行描述？假设在某一个时间段内（用τ表示），参数λ突然变大。所以λ可以取两个值，在每个时间段τ内有个参数λ与之对应。这种突然发生变化的情况称之为阶跃函数

$$\lambda = \begin{cases} \lambda_1, & if \ t < \tau \\ \lambda_2, & if \ t \geq \tau \end{cases}$$

如果实际中λ没有变化，则$\lambda_1 = \lambda_2$，后验概率λ_s也与之类似。

我们最关心的是如何推断出后验概率λ_s。使用贝叶斯推断时，需要给λ提供先验概率。什么样的先验概率对于λ_1, λ_2来说才是最有的哪。λ可以为任意的正数。在前面我们已经看到，对于正数来说，指数分布具有连续的概率密度函数，这种特性正好可以用来模拟参数λ_i。但是指数分布自己还有一个参数，所以在我们将要建立的模型中也要包括指数分布的参数。指数分布的参数为α，λ_1, λ_2服从的分布分别如下

$$\lambda_1 \sim Exp(\alpha)$$
$$\lambda_2 \sim Exp(\alpha)$$

α称之为超参数。该参数能够影响其它参数。超参数α的初始值不会对模型有太大的影响，所以在选择参数α的值时可以随意一些。不过有个经验法则可以选取该参数，一般会把该参数设置为每天短信数量的平均值倒数。因为我们希望使用指数分布表示参数λ的先验概率密度，可以将期望值表示为

$$\frac{1}{N}\sum_{i=0}^{N} C_i \approx E\{\lambda|\alpha\} = \frac{1}{\alpha}$$

可以尝试选择两个先验概率，每个λ_i对应一个。选择两个不同的指数分布作为先验分布，说明在不同时间段观测到事件发生的可信度是不同的。

如何处理时间区间τ。因为数据中有噪声存在，所以很难选择参数τ。为了解决这一问题，可以让为每天赋上一个均匀的先验可信度。这也就意味着τ服从如下分布

$$\tau \sim \text{DiscreteUniform}(1,70),$$

$$\Rightarrow P(\tau = k) = \frac{1}{70}$$

完成了以上的准备工作，此时要问未知参数的后验概率此时变成什么样子了？我们要明白这样一个道理，只有数学家才会喜欢那些看起来杂乱无章的，丑陋的公式。如果我们的模型变得很复杂，那么有些事情也就不那么美了。无论如何我们所关心的只是后验概率。

下一节将会介绍 PyMC，这是一个用于分析贝叶斯推断的 Python 库

1.6.5 第一个分析工具：PyMC

PyMC 是一个用于贝叶斯分析的 Python 程序库。其运行速度快，且便于维护。唯一的缺点就是在某些领域它的文档不是很完善，这使得初学者使用比来难度较大。本书的主要作用就是为了解决这一问题，同时还会展示为什么 PyMC 是如此的酷。

下面将使用 PyMC 解决短信推断问题。这种类型的程序称为概率编程（*probabilistic programming*），概率规划的给人感觉就好像程序代码是随机产生的，它使用户感到困惑，且有点望而生畏，结果就是不敢涉足这一领域。实际上，概率编程并不是说代码是随机产生的，而是使用程序中的变量作为模型参数，以概率的观点进行程序开发。在 PyMC 框架中模型参数是第一类的参数实体。

B. Cronin [5]对概率编程有一段激动人心的描述

> Another way of thinking about this: unlike a traditional program, which only runs in the forward directions, a probabilistic program is run in both the forward and backward direction. It runs forward to compute the consequences of the assumptions it contains about the world (i.e., the model space it represents), but it also runs backward from the data to constrain the possible explanations. In practice, many probabilistic programming systems will cleverly interleave these forward and backward operations to efficiently home in on the best explanations.

鉴于概率编程容易让人迷惑，所以以后尽量避免使用概率编程这一说法。而取之以编程这一说话，因为概率编程要表达的却是是编程的意思而无其它。

PyMC 的代码非常容易理解。唯一有点新奇的东西就是它的语法，下面将会对代码进行拆解，并解释相关细节。只要简单的记住我们需要将模型中的各个要素$(\tau, \lambda_1, \lambda_2)$表示成变量，Python 代码如下

```
import pymc as pm
alpha = 1.0 / count_data.mean()  # Recall count_data is the variable that holds our txt counts
lambda_1 = pm.Exponential("lambda_1", alpha)
lambda_2 = pm.Exponential("lambda_2", alpha)
tau = pm.DiscreteUniform("tau", lower=0, upper=n_count_data)
```

在上面代码中创建了 PyMC 变量λ_1, λ_2，并将它们设置成 PyMC 的随机变量，称之为随机变量是因为后端的程序会把它们当成随机数生成器。通过调用内建的函数 random() 生成随机变量的取值

```
print "Random output:", tau.random(), tau.random(), tau.random()
```

随机数的输出是：39，10，32

下面的脚步创建了个新的函数 lambda_，但是我们依然可以把这个函数想象成为一个随机变量。因为 lambda_1，lambda_2 和 tau 都是随机的，lambda_也就是随机的。

@pm.determinist 是一个标志词，该标志词告知 PyMC 这是一个确定性函数。也就是说如果函数参数是确定性的（实际上参数可不是确定性的），输出也是确定性的，在第二章中将会介绍确定性函数。

```
@pm.deterministic
def lambda_(tau=tau, lambda_1=lambda_1, lambda_2=lambda_2):
    out = np.zeros(n_count_data)
    out[:tau] = lambda_1  # lambda before tau is lambda1
    out[tau:] = lambda_2  # lambda after (and including) tau is lambda2
    return out
```

变量 observation 将数据 count_data，以及 lambda_关联在一起。将 observed 设置为 True，就是要告诉 PyMC 在分析过程中观测数据保持不变。最终，PyMC 收集我们感兴趣的变量，并创建了一个模型实例。有了这个模型，我们就会非常轻松的获取相关结果。

```
observation = pm.Poisson("obs", lambda_, value=count_data, observed=True)
model = pm.Model([observation, lambda_1, lambda_2, tau])
```

下面这段代码将会在第 3 章详细介绍，在这里介绍这段代码只是想告诉读者这里看到的结果是如何而来的。姑且把这段代码看成学习过程中的一个步骤而已。蒙特卡洛方法也将在第 3 章介绍。这种技术能够计算出$\lambda_1, \lambda_2, \tau$的后验概率。可以通过画出直方图看出后验概率的模样。下图是样本的直方图（called traces in the MCMC literatur）。

```
# Mysterious code to be explained in Chapter 3.
mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000, 1)
```

```
[****************100%******************]  40000 of 40000 complete
```

```
lambda_1_samples = mcmc.trace('lambda_1')[:]
lambda_2_samples = mcmc.trace('lambda_2')[:]
tau_samples = mcmc.trace('tau')[:]
```

```
# histogram of the samples:
ax = plt.subplot(311)
ax.set_autoscaley_on(False)
plt.hist(lambda_1_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label="posterior of $\lambda_1$", color="#A60628", normed=True)
plt.legend(loc="upper left")
plt.title(r"""Posterior distributions of the variables
    $\lambda_1,\;\lambda_2,\;\tau$""")
plt.xlim([15, 30])
plt.xlabel("$\lambda_1$ value")
ax = plt.subplot(312)
ax.set_autoscaley_on(False)
plt.hist(lambda_2_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label="posterior of $\lambda_2$", color="#7A68A6", normed=True)
plt.legend(loc="upper left")
plt.xlim([15, 30])
plt.xlabel("$\lambda_2$ value")
plt.subplot(313)
w = 1.0 / tau_samples.shape[0] * np.ones_like(tau_samples)
plt.hist(tau_samples, bins=n_count_data, alpha=1,
         label=r"posterior of $\tau$",
         color="#467821", weights=w, rwidth=2.)
plt.xticks(np.arange(n_count_data))
plt.legend(loc="upper left")
plt.ylim([0, .75])
plt.xlim([35, len(count_data) - 20])
plt.xlabel(r"$\tau$ (in days)")
plt.ylabel("probability");
```

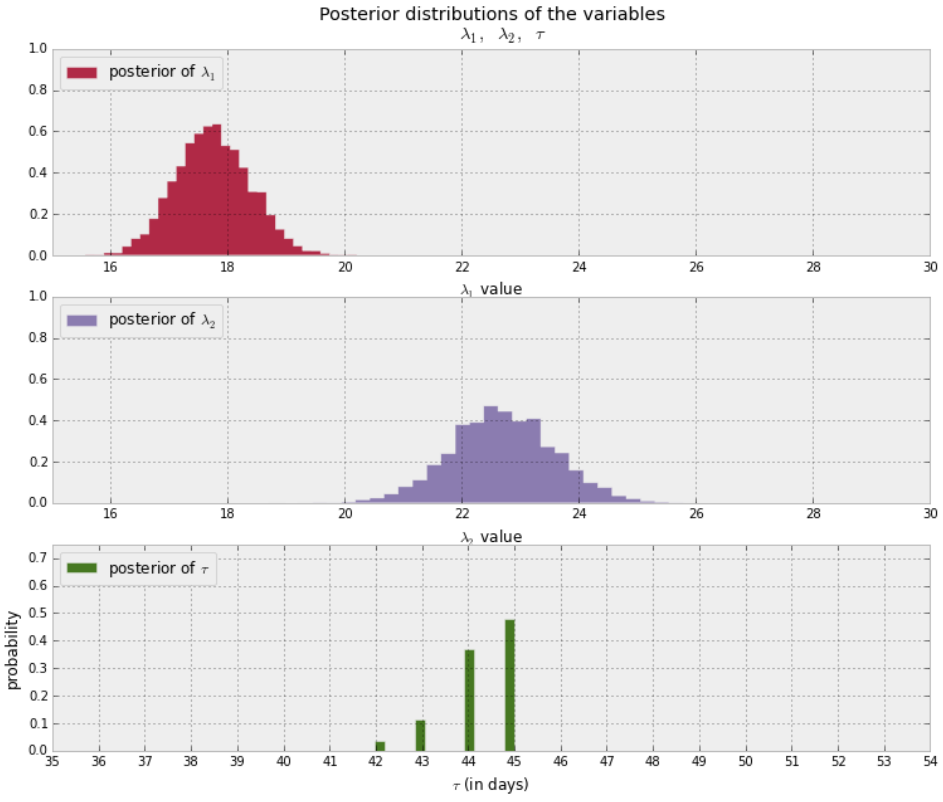


图** $\lambda_1, \lambda_2, \tau$后验分布直方图

1.6.6 相关解释

贝叶斯方法的最终结果是一个分布。我们用分布来描述未知参数λ_s和τ。从分布图中可以看到不确定性是什么样的：分布的曲线越平坦，后验可信度的不确定性越大。从上面的分布图中可以看出λ_1, λ_2比较合理的取值是 18 和 23。λ_s的两个后验分布明显不同，表面用户在使用短信时的行为确实不同。

还可以从分布图中观察到其它特性吗？如何结合最原始的数据，这些观察到的结果是合理的吗？

λ_s的后验分布看起来并不像指数分布，虽然它们的先验分布是指数形式的。事实上后验分布并不是先验分布的形式有多大关系。从计算学观点看这是有益处的，But that's OK! This is one of the benefits of taking a computational point of view. If we had instead done this analysis using mathematical approaches, we would have been stuck with an analytically intractable (and messy) distribution. Our use of a computational approach makes us indifferent to mathematical tractability.

τ也有后验分布，它的后验分布和λ_1, λ_2看起来不一样，因为τ是离散的随机变量，所以在区间上并没有概率值。从分布图中可以看出，在 45 天的时候有 50%的概率用户的行为会改变，在τ的后验概率分布延伸较大的地方说明用户在短信上的行为没有变化或者变化缓慢，这些天可以作为τ的候选值。通过对比发现只有三到四天是潜在的行为跳跃点。

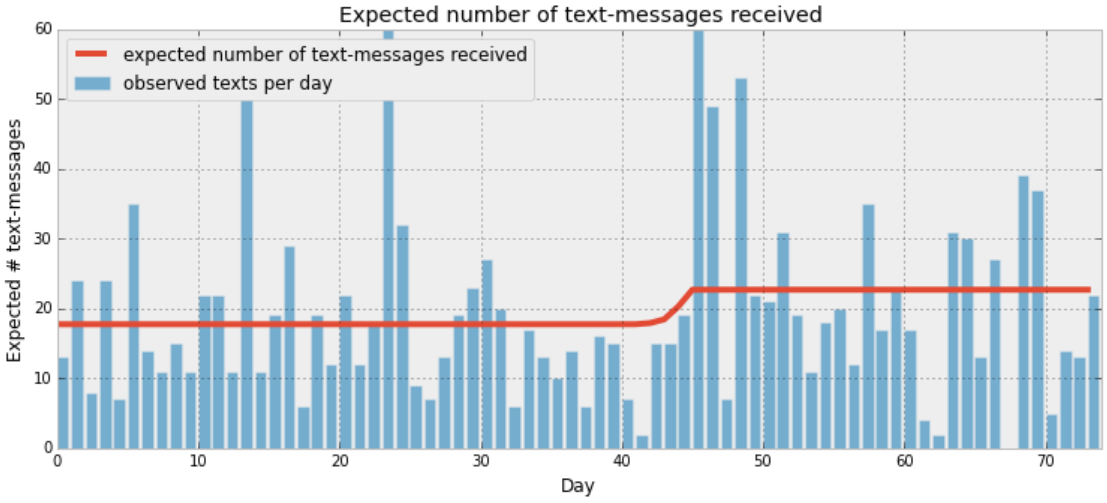
1.6.7 为什么要对后验概率进行采样

本书的余下部分会详细解决这个问题，有许多令人惊异的结果会展现出来。以下面的例子结束本章内容。

用后验采样回答下面的问题：在第t天会有多少个短信，$0 \le t \le 70$。因为泊松变量的期望等于参数λ，因此问题就变成，在时刻 t 与λ等价的期望是什么。

下面代码中的 i 是从后验分布中采样数据的索引。给定t，对所有的λ_i取平均，当$t < \tau_i$时$\lambda_i = \lambda_1$，否则$\lambda_i = \lambda_{2,i}$。

```
figsize(12.5, 5)
# tau_samples, lambda_1_samples, lambda_2_samples contain
# N samples from the corresponding posterior distribution
N = tau_samples.shape[0]
expected_texts_per_day = np.zeros(n_count_data)
for day in range(0, n_count_data):
    # ix is a bool index of all tau samples corresponding to
    # the switchpoint occurring prior to value of 'day'
    ix = day < tau_samples
    # Each posterior sample corresponds to a value for tau.
    # for each day, that value of tau indicates whether we're "before"
    # (in the lambda1 "regime") or
    #  "after" (in the lambda2 "regime") the switchpoint.
    # by taking the posterior sample of lambda1/2 accordingly, we can average
    # over all samples to get an expected value for lambda on that day.
    # As explained, the "message count" random variable is Poisson distributed,
    # and therefore lambda (the poisson parameter) is the expected value of
    # "message count".
    expected_texts_per_day[day] = (lambda_1_samples[ix].sum()
                                   + lambda_2_samples[~ix].sum()) / N
plt.plot(range(n_count_data), expected_texts_per_day, lw=4, color="#E24A33",
         label="expected number of text-messages received")
plt.xlim(0, n_count_data)
plt.xlabel("Day")
plt.ylabel("Expected # text-messages")
plt.title("Expected number of text-messages received")
plt.ylim(0, 60)
plt.bar(np.arange(len(count_data)), count_data, color="#348ABD", alpha=0.65,
        label="observed texts per day")
plt.legend(loc="upper left");
```



我们的分析有力的证明了用户的行为确实会改变，而且变化是突然发生的，而不是逐渐变化的。我们会猜测是什么引起这样的变化的：是短信变便宜了，是最近订阅了天气预报短信，还是其它什么新的关系导致的。

练习

1. Using lambda_1_samples and lambda_2_samples, what is the mean of the posterior distributions of λ_1 and λ_2?

```
# type your code here.
```

2. What is the expected percentage increase in text-message rates? hint: compute the mean of lambda_1_samples/lambda_2_samples. Note that this quantity is very different from lambda_1_samples.mean()/lambda_2_samples.mean().

```
# type your code here.
```

3. What is the mean of λ_1 given that we know τ is less than 45. That is, suppose we have been given new information that the change in behaviour occurred prior to day 45. What is the expected value of λ_1 now? (You do not need to redo the PyMC part. Just consider all instances where tau_samples < 45.)

```
# type your code here.
```

References

[1] Gelman, Andrew. N.p.. Web. 22 Jan 2013. N is never large enough.

[2] Norvig, Peter. 2009. The Unreasonable Effectiveness of Data.

[3] Patil, A., D. Huard and C.J. Fonnesbeck. 2010. PyMC: Bayesian Stochastic Modelling in Python. Journal of Statistical Software, 35(4), pp. 1-81.

[4] Jimmy Lin and Alek Kolcz. Large-Scale Machine Learning at Twitter. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD 2012), pages 793-804, May 2012, Scottsdale, Arizona.

[5] Cronin, Beau. "Why Probabilistic Programming Matters." 24 Mar 2013. Google, Online Posting to Google . Web. 24 Mar. 2013. https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1.

第 2 章 PyMC 实战

本章将介绍 PyMC 的语法和设计模式，以及如何用贝叶斯方法去模拟一个系统。还会介绍一些小技巧和可视化技术，这些方法能否更好的拟合贝叶斯模型。

2.1 PyMC 语法介绍

2.1.1 父子关系

为了便于描述贝叶斯关系，并和 PyMC 的文档保持一致，下面介绍名词术语父变量和子变量。

- 父变量是指能够影响其它变量的变量
- 子变量是只被其它变量影响的变量，它率属于父变量

一个变量既可以是父变量又可以是子变量。例如下面的 PyMC 代码

```
import pymc as pm
parameter = pm.Exponential("poisson_param", 1)
data_generator = pm.Poisson("data_generator", parameter)
data_plus_one = data_generator + 1
```

参数 parameter 控制参数 data_generator，因此 parameter 影响 data_generator 的值。前者是后者的父变量。通过对称性，data_generator 是 parameter 子变量。

同理 data_generator 是变量 data_plus_one 的父变量（因此 data_genetator 既是父变量又是子变量）。data_plus_one 可以看成是 PyMC 变量，因为它是另外一个 PyMC 变量的函数，因此 data_plus_one 是 data_generator 的子变量。

这些术语有助于理解 PyMC 模型中的关系。可以通过变量上的属性 children 和 parents 来识别代码中的子变量和父变量。

```
print "Children of `parameter`: "
print parameter.children
print "\nParents of `data_generator`: "
print data_generator.parents
print "\nChildren of `data_generator`: "
print data_generator.children
```

```
Children of `parameter`:
set([<pymc.distributions.Poisson 'data_generator' at 0x1087e14d0>])

Parents of `data_generator`:
{'mu': <pymc.distributions.Exponential 'poisson_param' at 0x104efac50>}

Children of `data_generator`:
set([<pymc.PyMCObjects.Deterministic '(data_generator_add_1)' at 0x109119790>])
```

一个子变量可以有多个父变量，一个父变量也可以有多个子变量。

2.1.2 PyMC 变量

所有的 PyMC 变量都会有一个属性 value。通过这个属性可以拿到当前变量的内部值。如果一个变量是子变量，它的值就会收到父变量值影响。

```
print "parameter.value =", parameter.value
print "data_generator.value =", data_generator.value
print "data_plus_one.value =", data_plus_one.value
```

parameter.value = 0.0281609254306
data_generator.value = 0
data_plus_one.value = 1

PyMC 关心两类变量：随机变量和确定性变量。

- 随机变量指的是那些不确定性变量，即使知道所有父变量的值，子变量依然可能是随机的。例如泊松分布，离散的均匀分布和指数分布
- 非随机变量称之为确定性变量，例如父变量已知时子变量是确定性变量。刚开始时这可以有些令人费解，有个快速判断方法，如果变量 foo 所有的父变量都是已知的，foo 的值就是确定的。

初始化随机变量

用参数 name 可以初始化随机变量，不同的类还会有不同的附加参数需要设置。例如

some_variable = pm.DiscreteUniform("discrete_uni_var", 0, 4)

0，4 是离散变量的上下边界值。PyMC 文档（http://pymc-devs.github.io/pymc/distributions.html）介绍了各个随机变量的具体参数。

可以用属性 name 获取后验概率的分布，所以在给变量命名时最好能够使用一个语义明确的名称。通常使用 Python 变量的名字为 name。

对于多变量问题，不可以只创建一个随机变量 Python 数组，而是在函数调用时给 Stochastic 变量传递 size 属性创建多个独立的随机变量数组。随机变量数组和 Numpy 数组类似，通过属性 value 可以返回 Numpy 数组。

可以使用参数 size，模拟多个变量$\beta_i i = 1, \cdots, N$。不需要为每个变量都设置一次，例如

beta_1 = pm.Uniform("beta_1", 0, 1)
beta_2 = pm.Uniform("beta_2", 0, 1)
…

可以将他们表示成一个变量数组，如下

betas = pm.Uniform("betas", 0, 1, size=N)

调用 random()

可以调用随机变量的方法 random()，该方法可以产生新的随机值。下面以前一章中的短信例子展示这种用法

```
lambda_1 = pm.Exponential("lambda_1", 1)  # prior on first behaviour
lambda_2 = pm.Exponential("lambda_2", 1)  # prior on second behaviour
tau = pm.DiscreteUniform("tau", lower=0, upper=10)  # prior on behaviour change
print "lambda_1.value = %.3f" % lambda_1.value
print "lambda_2.value = %.3f" % lambda_2.value
print "tau.value = %.3f" % tau.value
print
lambda_1.random(), lambda_2.random(), tau.random()
print "After calling random() on the variables..."
print "lambda_1.value = %.3f" % lambda_1.value
print "lambda_2.value = %.3f" % lambda_2.value
print "tau.value = %.3f" % tau.value
```

lambda_1.value = 0.539
lambda_2.value = 1.858
tau.value = 1.000

After calling random() on the variables...
lambda_1.value = 0.343
lambda_2.value = 0.316
tau.value = 6.000

在调用 random 方法时，程序会将一个新的值存储到 value 属性中。实际上为了效率，这个新的值是预先存储在计算机缓存中的。

Warning: Don't update stochastic variables' values in-place.

Straight from the PyMC docs, we quote [4]:

Stochastic objects' values should not be updated in-place. This confuses PyMC's caching scheme... The only way a stochastic variable's value should be updated is using statements of the following form:

A.value = new_value

The following are in-place updates and should never be used:

A.value += 3
A.value[2,1] = 5
A.value.attribute = new_attribute_value

确定性变量

因为将要模拟的大部分变量都是随机的，为了区别随机变量，用一个关键字 pymc.deteministic 表示（如果不太了解 Python 中的语法，也没关系，只要在声明的变量前面增加 pymc.deterministic）。用如下形式将 Python 函数声明成确定性变量

```
@pm.deterministic
def some_deterministic_var(v1=v1,):
    #jelly goes here.
```

可以将对象 some_deterministic_var 看成是一个变量而不是一个 Python 函数。

用修饰符声明确定性变量，可以完成基本操作，例如加法运算，指数运算等。但这不是唯一方法。下面的代码也可以返回确定性变量

```
type(lambda_1 + lambda_2)
```

pymc.PyMCObjects.Deterministic

在第一章的例子中已经使用了 deterministic 声明了确定性变量。泊松分布的参数λ如下

$$\lambda = \begin{cases} \lambda_1, & if \ \ t < \tau \\ \lambda_2, & if \ t \geq \tau \end{cases}$$

PyMC 的代码如下

```
import numpy as np
n_data_points = 5  # in CH1 we had ~70 data points
@pm.deterministic
def lambda_(tau=tau, lambda_1=lambda_1, lambda_2=lambda_2):
    out = np.zeros(n_data_points)
    out[:tau] = lambda_1  # lambda before tau is lambda1
    out[tau:] = lambda_2  # lambda after tau is lambda2
    return out
```

当$\tau, \lambda_1, \lambda_2$确定时，λ也就完全确定，所以λ是确定性变量。

在对确定性变量的描述中，随机变量以标量或者 Numpy 数组（多维随机变量）的形式传递，而不是以随机变量的形式。例如运行如下代码将返回 AttributeError 错误，因

为 stoch 没有 value 属性。应该将 stoch.value**2 改为 stoch**2。在运行过程中值是通过 value，而不是通过变量本身传递的。

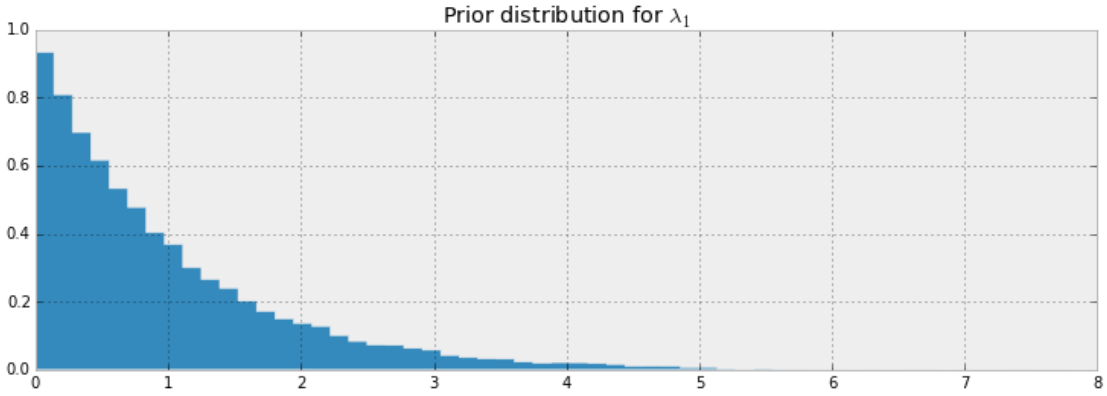
```
@pm.deterministic
def some_deterministic(stoch=some_stochastic_var):
  return stoch.value**2
```

在创建确定性变量时，一定要给函数中的每个变量都设置缺省值。

2.1.3 设置观测量

At this point, it may not look like it, but we have fully specified our priors。例如，可以回答以下问题“λ_1的先验概率看起来像什么样？”。

```
%matplotlib inline
from IPython.core.pylabtools import figsize
from matplotlib import pyplot as plt
figsize(12.5, 4)
samples = [lambda_1.random() for i in range(20000)]
plt.hist(samples, bins=70, normed=True, histtype="stepfilled")
plt.title("Prior distribution for $\lambda_1$")
plt.xlim(0, 8);
```



To frame this in the notation of the first chapter, though this is a slight abuse of notation, we have specified$P(A)$。下面的目标是要将观测变量X包含到模型中。

PyMC 的随机变量有一个关键字 observed，该变量接收一个 boolean 类型值（缺省值是 False）。observed 关键字的作用非常简单，就是为了确定变量的当前值，使 value 不可改变。必须在创建变量时初始化 value，这等价于设置模型中的观测变量。例如

```
data = np.array([10, 5])
fixed_variable = pm.Poisson("fxd", 1, value=data, observed=True)
print "value: ", fixed_variable.value
print "calling .random()"
fixed_variable.random()
print "value: ", fixed_variable.value
```

```
value:  [10  5]
calling .random()
value:  [10  5]
```

这说明可以通过给随机变量设定一个初始值，达到给模型设置观测数据的目的。

为了完成短信例子，给 PyMC 变量 obserbations 设置观测到的数据集

```
# We're using some fake data here
data = np.array([10, 25, 15, 20, 35])
obs = pm.Poisson("obs", lambda_, value=data, observed=True)
print obs.value
```

```
[10 25 15 20 35]
```

最终，将所有变量都统一到 pm.Model 类上。有了 Model 类，我们就可以对变量进行分析了。这种做法也不是必须的，因为拟合算法可以发送变量数组而不是 Model 类。所以在以后的例子中不一定会使用这种方式。

```
model = pm.Model([obs, lambda_, lambda_1, lambda_2, tau])
```

2.1.4 建模方法

学习贝叶斯建模的最好方式就是用贝叶斯方法生成数据。假设你是一个非常牛逼 B 的人，你会如何创建你的数据集。

在上一章，研究了短信数据，此时，此时我们会想我们的观测数据时如何产生的：

1 首先我们会想，什么样的随机变量才能最好的描述每天短信数量。泊松分布是不错的选择因为它表示的是计数变量。所以可以用泊松分布描述 sms 接受到的数据。

2 接着我们会想，假设 sms 是泊松分布，确定泊松分布的其它量是什么。泊松分布的参数是λ。

3 我们知道λ的值是多少吗？事实上不知道，我们猜测λ有两个值，一个用于描述早期行为，另一个用于描述后期行为，我们不知道行为变化的时间点在哪里，但是可以将这个跳变点定义为τ。

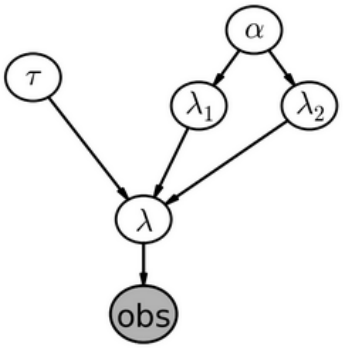
4 对于两个λ_s，哪个值对于的分布较好？指数分布是不错的选择，因为在指数分布中正实数都存在一个概率，指数分布也有参数，称之为α。

5 我们知道参数α是多少吗？此时不知道，可以给α设定一个分布，but it's better to stop once we reach a set level of ignorance: whereas we have a prior belief about λ, ("it probably changes over time", "it's likely between 10 and 30", etc.), we don't really have any strong beliefs about α. So it's best to stop here.

α的最优值是什么？我们认为λ_s的取值范围为 10-30，如果将α设置的很小（对应概率密度函数的峰值偏右），将无法很好的反映出先验概率，如果α值过大又会使先验的可信度降低。所以α的设定要能够反映先验概率的可信度，α的取值标准一般为：在给定α下，λ的均值等于观测到的均值。下一章将会看到这种用法。

6 对于τ我们没有先验的专家经验。所以在整个时间区间上假设τ是离散的均匀分布。

下图描述了上面介绍的整个逻辑关系，图中箭头表示父子关系(provided by the Daft Python library)



PyMC 和其它概率编程语言的核心是描述数据是如何生成的。更一般化的描述如下

> Probabilistic programming will unlock narrative explanations of data, one of the holy grails of business analytics and the unsung hero of scientific persuasion. People think in terms of stories - thus the unreasonable power of the anecdote to drive decision-making, well-founded or not. But existing analytics largely fails to provide this kind of story; instead, numbers seemingly appear out of thin air, with little of the causal context that humans prefer when weighing their options

2.1.5 同样的故事，不同的结果

有趣的是当我们重述以上的故事时，却有不同的结果，如果将 2.1.4 中介绍的六个步骤倒过来，又可模拟出另一种可能的数据集。

1 用户行为改变符合（0,80）的离散均匀分布

```
tau = pm.rdiscrete_uniform(0, 80)
print tau
```

37

2 让λ_1,λ_2符合$\text{Exp}(\alpha)$分布

```
alpha = 1. / 20.
lambda_1, lambda_2 = pm.rexponential(alpha, 2)
print lambda_1, lambda_2
```

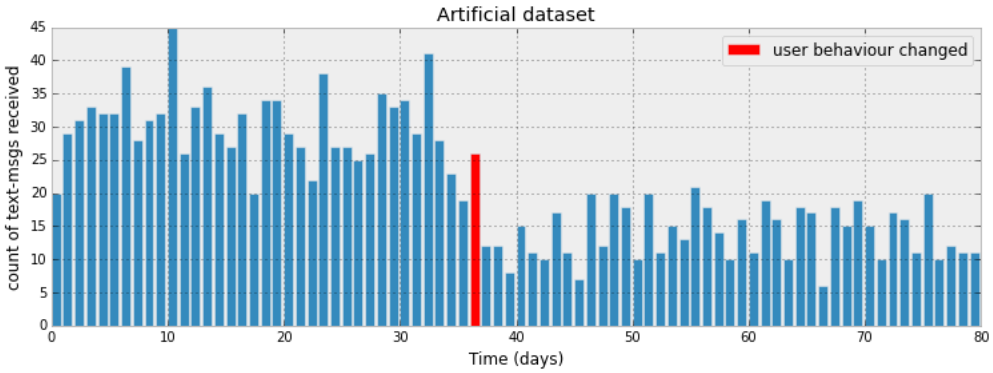
29.0394614963 13.5921161471

3 在τ之前的时间内，用$\text{Poi}(\lambda_1)$表示用户接受到的 SMS 数量分布，在 τ 之后的时间内用$\text{Poi}(\lambda_2)$表示接受到的 SMS 数量分布

```
data = np.r_[pm.rpoisson(lambda_1, tau), pm.rpoisson(lambda_2, 80 - tau)]
```

4 画出构造的数据集

```
plt.bar(np.arange(80), data, color="#348ABD")
plt.bar(tau - 1, data[tau - 1], color="r", label="user behaviour changed")
plt.xlabel("Time (days)")
plt.ylabel("count of text-msgs received")
plt.title("Artificial dataset")
plt.xlim(0, 80)
plt.legend();
```

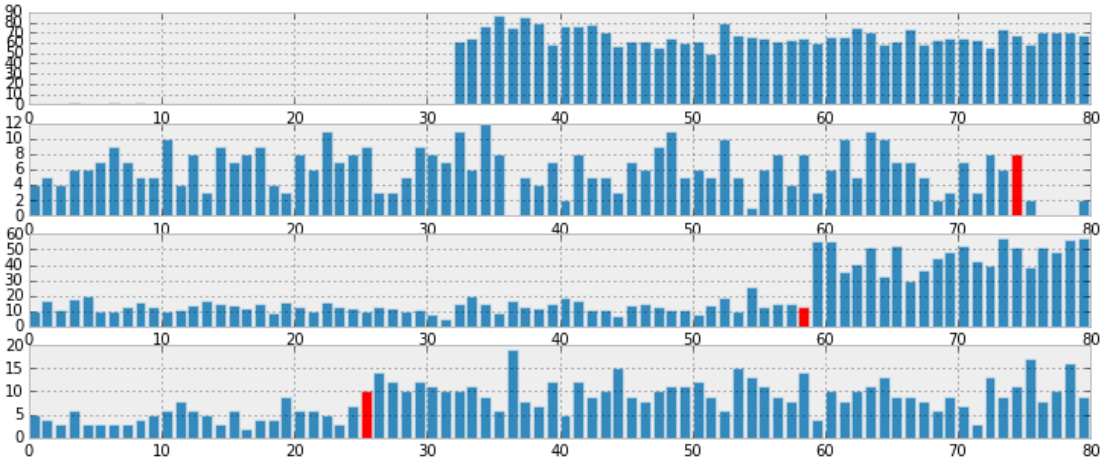


我们构造的数据集和我们观察到的并不一致，因为概率非常小。PyMC 的引擎通过使概率最大，找到最优的参数λ_i和 τ 。

能够制造人工数据集是我们模型非常有趣的一个方面，我们将会看到这对于贝叶斯推断来说非常重要。下面将产生新的数据集

```
def plot_artificial_sms_dataset():
    tau = pm.rdiscrete_uniform(0, 80)
    alpha = 1. / 20.
    lambda_1, lambda_2 = pm.rexponential(alpha, 2)
    data = np.r_[pm.rpoisson(lambda_1, tau), pm.rpoisson(lambda_2, 80 - tau)]
    plt.bar(np.arange(80), data, color="#348ABD")
    plt.bar(tau - 1, data[tau - 1], color="r", label="user behaviour changed")
    plt.xlim(0, 80)

figsize(12.5, 5)
plt.title("More example of artificial datasets")
for i in range(4):
    plt.subplot(4, 1, i)
    plot_artificial_sms_dataset()
```



图**

稍后将会看到如何使用这种方法对模型进行预测和近似。

2.2 示例

2.2.1 贝叶斯 A/B 测试

A/B 测试是一种统计设模式，它能够测试两种不同设计方案之间的差异。例如，一个制药公司对药物 A 和 B 之间的效果差异就非常关注。公司会在一小部分实验上做 A 药物的相关测试，在 B 药物上做另一部分实验的相关测试（两种实验样本的比例一般为 0.5，但是也可能放松这种假设，不需要一定是 0.5）。在足够的实验完成之后，对实验数据进行统计，就可以确定哪种药物效果更好。

网站的前端开发人员关心他们设计的网站是否能够带来更多的销售额，或者其他一些对网站有益的特性。网站会人为的将网站的流量切换到 A 前端，将剩下流量的切换到 B 前端，并记录每个浏览者是否购买的实时信息。通过分析这些详细，就可以知道设计的前端是 A 好，还是 B 好。

通常采用假设检验对实验的数据进行分析，例如 difference of means test or difference of proportions test。假设检验中的 Z（Z-score）得分和 p 值（p-values）难以理解。如果你学习过概率统计的课程，你可能已经学习过这种技术。但是如果你和我一样对这种技术的理解始终不那么深刻，那就太好了，贝叶斯方法以一种非常自然的方式解释了假设检验中的相关理论。

一个简单的例子

因为这是一本黑客用书，所以继续以网络开发为例。此刻我们将集中精力分析 A 网站。假设用户浏览网站 A 并最终从该站购买东西的概率为$0 < p_A < 1$。这是一个真正有效的网站。目前我们还不知道网站的质量到底如何。

假设网站A一共向N个人展示过，其中n个人从站上买了商品。有人就坦率的认为$p_A = \frac{n}{N}$。不幸的是观测到的频率$\frac{n}{N}$并不等于p_A，这说明在观测到的频率和真实的频率之间有一定的差异。真实的频率可以解释为事件发生的概率。例如，掷色子时 1 朝上的概率为$\frac{1}{6}$。在网站这个例子中，我们要先知道如下现象才能知道事件的真实频率

- 发生购买行为的用户比例
- 社会属性的频率
- 网民养猫的比例等信息

不幸的是这些现象隐藏了真实频率，必须从观察数据中推断出它们真实频率。顾名思义真实频率就是我们看到的频率，例如掷 100 次色子其中有 20 次正面朝上。观察频率就是 0.2，这和真实频率$\frac{1}{6}$是不同的。如果有近似先验和观测到的数据，可以利用贝叶斯统计推断可能的真实概率。

创建一个贝叶斯模型，首先需要为未知的量设置一个先验概率。A 的先验知识表示我们所认为的p_A应该是什么样的。例如我们不确定p_A到底应该设置为多少，所以可以设置p_A服从$[0,1]$的均匀分布。

```
import pymc as pm
# The parameters are the bounds of the Uniform.
p = pm.Uniform('p', lower=0, upper=1)
```

如果我们很了解某个领域的先验知识，可以将这种信念表示在先验分布中。

在这个例子中，假设$p_A = 0.05$，网站 A 向$N = 1500$人展示过，根据这些信息我们将要模拟用户在网站上是否会购买东西。为了模拟N次试验，将采用 Bernoulli 分布：如果$X \sim Ber(p)$，则X是 1 的概率为p，是 0 的概率为$1 - p$，当然，实际上并不知道p_A的取值为多少，但是我们将用它来模拟数据。

```
# set constants
p_true = 0.05  # remember, this is unknown.
N = 1500
# sample N Bernoulli random variables from Ber(0.05).
# each random variable has a 0.05 chance of being a 1.
# this is the data-generation step
occurrences = pm.rbernoulli(p_true, N)
print occurrences  # Remember: Python treats True == 1, and False == 0
print occurrences.sum()
```

```
[False False False …, False False  True]
87
```

观察到的频率是

```
# Occurrences.mean is equal to n/N.
print "What is the observed frequency in Group A? %.4f" % occurrences.mean()
print "Does this equal the true frequency? %s" % (occurrences.mean() == p_true)
```

```
What is the observed frequency in Group A? 0.0580
Does this equal the true frequency? False
```

将 PyMC 的 observed 变量设置为观测变量，并运行推断程序

```
# include the observations, which are Bernoulli
obs = pm.Bernoulli("obs", p, value=occurrences, observed=True)
# To be explained in chapter 3
mcmc = pm.MCMC([p, obs])
mcmc.sample(18000, 1000)
```

```
[****************100%******************]  18000 of 18000 complete
```

未知参数p_A的后验概率分布图如下

```
figsize(12.5, 4)
plt.title("Posterior distribution of $p_A$, the true effectiveness of site A")
plt.vlines(p_true, 0, 90, linestyle="--", label="true $p_A$ (unknown)")
plt.hist(mcmc.trace("p")[:], bins=25, histtype="stepfilled", normed=True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x109862250>
```

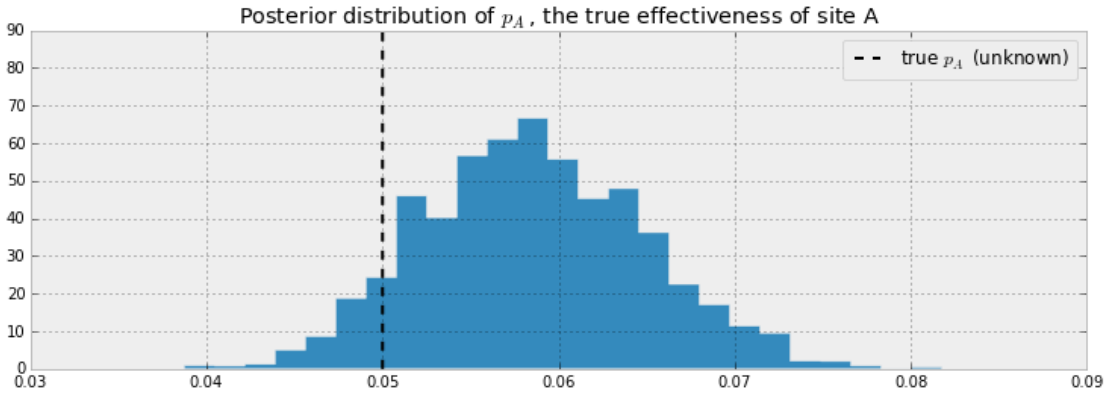


图** p_A的后验分布图

从图中可以看出，在p_A附近的值有更高的权重，在左右的长尾部分也有一些权重分布。在给定观测数据的条件下，后验分布可以度量不确定性的程序。改变N的大小，观测后验概率的变化。

将 A 和 B 放在一起

可以用相似的方法，通过分析网站 B 的响应数据得到p_B。但是真正让人感兴趣的是p_A和p_B间的差异。用 PyMC 的确定性变量计算p_A，p_B，$\text{delta} = p_A - p_B$（在这个例子中假设$p_B = 0.04$，所以$\text{delta} = 0.01$，$N_B = 750$，明显比N_A小，用模拟网站 A 的类似方法模拟网站 B）。

```
import pymc as pm
figsize(12, 4)
# these two quantities are unknown to us.
true_p_A = 0.05
true_p_B = 0.04
# notice the unequal sample sizes -- no problem in Bayesian analysis.
N_A = 1500
N_B = 750
# generate some observations
observations_A = pm.rbernoulli(true_p_A, N_A)
observations_B = pm.rbernoulli(true_p_B, N_B)
print "Obs from Site A: ", observations_A[:30].astype(int), "..."
print "Obs from Site B: ", observations_B[:30].astype(int), "..."
```

Obs from Site A: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] ...
Obs from Site B: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0] ...

```
print observations_A.mean()
print observations_B.mean()
```

0.056
0.0306666666667

```
# Set up the pymc model. Again assume Uniform priors for p_A and p_B.
p_A = pm.Uniform("p_A", 0, 1)
p_B = pm.Uniform("p_B", 0, 1)
# Define the deterministic delta function. This is our unknown of interest.
@pm.deterministic
def delta(p_A=p_A, p_B=p_B):
    return p_A - p_B
# Set of observations, in this case we have two observation datasets.
obs_A = pm.Bernoulli("obs_A", p_A, value=observations_A, observed=True)
obs_B = pm.Bernoulli("obs_B", p_B, value=observations_B, observed=True)
# To be explained in chapter 3.
mcmc = pm.MCMC([p_A, p_B, delta, obs_A, obs_B])
mcmc.sample(20000, 1000)
```

[****************100%******************] 20000 of 20000 complete

下面画出三个未知变量的后验分布图

```
p_A_samples = mcmc.trace("p_A")[:]
p_B_samples = mcmc.trace("p_B")[:]
delta_samples = mcmc.trace("delta")[:]
```

```
figsize(12.5, 10)
# histogram of posteriors
ax = plt.subplot(311)
plt.xlim(0, .1)
plt.hist(p_A_samples, histtype='stepfilled', bins=25, alpha=0.85,
        label="posterior of $p_A$", color="#A60628", normed=True)
plt.vlines(true_p_A, 0, 80, linestyle="--", label="true $p_A$ (unknown)")
plt.legend(loc="upper right")
plt.title("Posterior distributions of $p_A$, $p_B$, and delta unknowns")
ax = plt.subplot(312)
plt.xlim(0, .1)
plt.hist(p_B_samples, histtype='stepfilled', bins=25, alpha=0.85,
        label="posterior of $p_B$", color="#467821", normed=True)
plt.vlines(true_p_B, 0, 80, linestyle="--", label="true $p_B$ (unknown)")
plt.legend(loc="upper right")
ax = plt.subplot(313)
plt.hist(delta_samples, histtype='stepfilled', bins=30, alpha=0.85,
        label="posterior of delta", color="#7A68A6", normed=True)
plt.vlines(true_p_A - true_p_B, 0, 60, linestyle="--",
         label="true delta (unknown)")
plt.vlines(0, 0, 60, color="black", alpha=0.2)
plt.legend(loc="upper right");
```

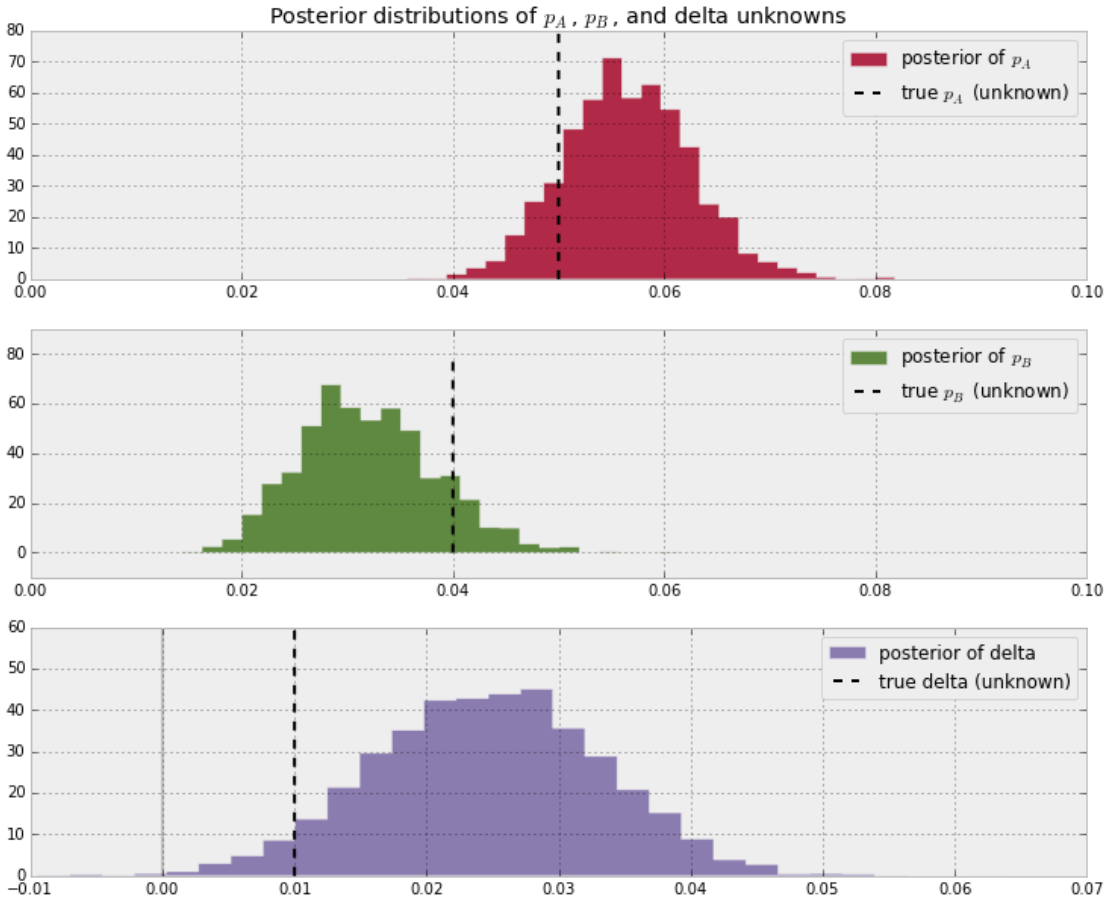


图** $p_A, p_B,$ delta的后验分布图

由于$N_B < N_A$，所以从 B 站得到的数据要比 A 站上的少，p_B的后验分布就比较平坦，这表明p_B接近真实值得不确定性比p_A要大，即p_A接近真实概率值得可信度要更高。

从 delta 的后验分布图中可以看到分布大多数集中在大于 0 的部分，这表明网站 A 的响应很可能优于网站 B 的响应。这种推理不正确的概率很容易计算

```
# Count the number of samples less than 0, i.e. the area under the curve
# before 0, represent the probability that site A is worse than site B.
print "Probability site A is WORSE than site B: %.3f" % \
   (delta_samples < 0).mean()
print "Probability site A is BETTER than site B: %.3f" % \
   (delta_samples > 0).mean()
```

```
Probability site A is WORSE than site B: 0.003
Probability site A is BETTER than site B: 0.997
```

如果觉得网站 A 优于网站 B 的概率 0.997 太高，可以在网站 B 上做更多的实验（因为网站 B 的样本较少，所以对于网站 B 来说，任何一个额外的数据都有更多的推断信息）。

调整参数 true_p_A，true_p_B，N_A 和 N_B，观察 delta 的后验分布的变化。我们一直没有关注过网站 A 和网站 B 之间样本的差异，这种情况下很适合贝叶斯分析。

我希望与传统的假设检验相比，这种类型的 A/B 检验更容易理解。在余下部分将会看到 A/B 测试的两个扩展部分：第一个扩展用于动态的调整设计不佳的网站，第二个扩展是通过减少简单方程的分析以改善计算速度。

测谎算法

社会学的统计数据有趣的地方在于人们并不总是诚实的回答问题，这就让推断变得复杂。例如，当有人问你“在考试时你作过弊吗？”，这种问法就包括了别人对你的不信任在里面。What you can say for certain is that the true rate is less than your observed rate (assuming individuals lie only about not cheating; I cannot imagine one who would admit "Yes" to cheating when in fact they hadn't cheated).

一种优雅的解决方案将用来解决这种不诚信问题，同时展示贝叶斯模型。首先引入二项分布。

二项分布

二项分布是一种很流行的分布，因为其简单且非常有用。不像本书中的其它分布，二项分布只有两个参数：N和p，N是整数，表示N次试验，p表示事件发生一次的概率。和泊松分布类似，二项分布也是离散分布，但是和泊松分布又有不同，它的概率密度分布函数的取值范围只在$[0, N]$之间。分布函数如下

$$P(X = k) = \binom{N}{k} p^k (1-p)^{N-k}$$

X是二项随机变量，参数为p和N，$X \sim \text{Bin}(N, p)$，X表示在 N 次试验中事件发生的次数$0 \leq X \leq N$，p是单个事件的概率。p越大表面有越多的事件发生。二项分布的期望值等于Np。下面画出不同参数所对应的概率分布函数。

```
figsize(12.5, 4)
import scipy.stats as stats
binomial = stats.binom
parameters = [(10, .4), (10, .9)]
colors = ["#348ABD", "#A60628"]
for i in range(2):
    N, p = parameters[i]
    _x = np.arange(N + 1)
    plt.bar(_x - 0.5, binomial.pmf(_x, N, p), color=colors[i],
            edgecolor=colors[i],
            alpha=0.6,
            label="$N$: %d, $p$: %.1f" % (N, p),
            linewidth=3)
plt.legend(loc="upper left")
plt.xlim(0, 10.5)
plt.xlabel("$k$")
plt.ylabel("$P(X = k)$")
plt.title("Probability mass distributions of binomial random variables");
```

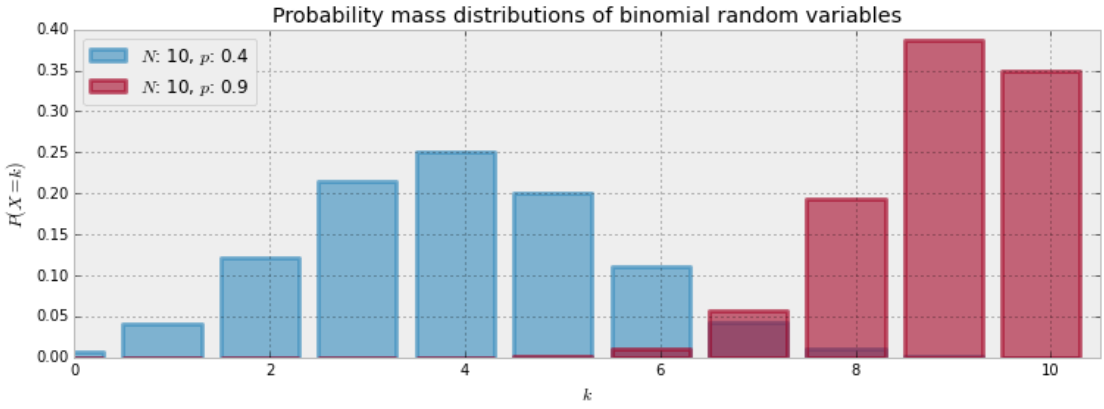


图** 不同参数的二项分布

当$N=1$时，对应于伯努利分布。如果$X_1, X_2, \cdots, X_N$是伯努利随机变量，这些随机变量具有相同的参数p，则$Z=X_1+X_2+\cdots+X_N \sim \text{Binomial}(N,p)$。当$N=1$，对应于伯努利随机变量，伯努利随机变量的期望是$p$。所以二项分布的期望为$Np$。

2.2.2 学生作弊

下面将用二项分布估计学生在考试中作弊的频率。设N表示参加考试的学生数目，假设参加考试的每个学生在考试结束后都会接受提问是否作弊。其中X表示承认作弊的学生个数。我们的目标是，在给定观测数据X，p的先验概率，学生个数 N 的条件下计算p的后验概率。

这是一个荒谬的模型。即使不会被处罚，也没有学生会承认自己作弊。我们需要一个更好的算法来询问学生，他们是否存在欺骗行为。理想情况下算法应该在保护隐私的情况下鼓励学生要诚实守信。下面将介绍一个设计新颖且有效的算法。

> 在对学生提问的过程中，让学生抛硬币，硬币的结果提问者看不到。如果硬币的正面朝上，那么学生就要诚实的回答问题。如果硬币的背面朝上，那么学生要再抛一次硬币，如果这次是正面朝上，学生就要回答他作弊了，如果还是背面朝上那么学生就回答没作弊。这就导致提问者并不知道学生的回答是由于内疚才承认作弊还是由于抛两次硬币都是正面朝上才承认作弊。这种做法既能保护隐私，又能得到学生的诚实回答。

称这种算法为隐私算法。One could of course argue that the interviewers are still receiving false data since some Yes's are not confessions but instead randomness, but an alternative perspective is that the researchers are discarding approximately half of their original dataset since half of the responses will be noise. But they have gained a systematic data generation process that can be modeled. Furthermore, they do not have to incorporate (perhaps somewhat naively) the possibility of deceitful answers. We can use PyMC to dig through this noisy model, and find a posterior distribution for the true frequency of liars.

假设调查了 100 个学生，我们希望用这些数据计算出不诚实学生的概率p。在 PyMC 中有一些方法可以模拟这一过程。下面将介绍一种最清晰的方法，然后介绍一种简化版本。两种版本都能达到相同的推断目的。在我们的生成模型中，从一个先验分布中采样出p，p的先验分布符合均匀分布$\text{Uniform}(0,1)$。

```
import pymc as pm
N = 100
p = pm.Uniform("freq_cheating", 0, 1)
```

将伯努利随机变量分配给 100 个学生：1 代表作弊了，0 代表没作弊。

```
true_answers = pm.Bernoulli("truths", p, size=N)
```

接下来算法就要模拟每个学生第一次抛硬币的过程了。可以从参数$p=\frac{1}{2}$的伯努利分布中采样出 100 个随机变量：1 代表正面，0 代表反面。

```
first_coin_flips = pm.Bernoulli("first_flips", 0.5, size=N)
print first_coin_flips.value
```

```
[False  True  True  True  True False False  True False False False False
 False  True  True False  True  True False False  True  True False False
 False False False  True False False  True False  True  True  True  True
  True False  True False  True  True  True  True  True False False False
```

```
 True  True  True  True  True  True False  True False False False  True
 True  True False False False False  True False  True False False  True
 True False False False False  True  True False False  True False  True
False False  True False False False  True  True  True  True  True False
False False False False]
```

虽然不是每个人都抛两次硬币，我们仍然可以模拟可能第二次抛硬币的过程

```
second_coin_flips = pm.Bernoulli("second_flips", 0.5, size=N)
```

有了这些变量就可以计算出承认作弊的学生概率

```
@pm.deterministic
def observed_proportion(t_a=true_answers,
                        fc=first_coin_flips,
                        sc=second_coin_flips):

    observed = fc * t_a + (1 - fc) * sc
    return observed.sum() / float(N)
```

fc*t_a + (1-fc)*sc 是隐私算法的核心。当且仅当满足以下条件时数组中 observed 的元素是 1，否则为 0：i）第一次抛硬币是正面朝上同时学生作弊，或者 ii）第一次抛硬币是反面朝上，第二次抛是正面朝上，最后一行对 observed 求和，并除以 float(N)。结果为

```
observed_proportion.value
```

```
0.42999999999999999
```

接下来需要一个数据集。在进行抛硬币访问后，共收集到 35 个承认作弊的数据。我们需要以一种相互比较的视角来看待这些数据，如果确实没有作弊的学生，我们希望看到收集的数据中承认作弊的学生概率为 1/4（有一半的概率硬币反面朝上，在这一半中又有一半正面朝上），所以只有 25%的作弊概率。如果所有学生都作弊了，我们希望在收集到的数据中有 3/4 的学生承认作弊。

当$N = 100$，p = observed_proportion，value = 35，观察伯努利随机变量

```
X = 35
observations = pm.Binomial("obs", N, observed_proportion, observed=True,
                           value=X)
```

下面将所有相关的变量加入到 Model 容器中，同时运行黑箱算法

```
model = pm.Model([p, true_answers, first_coin_flips,
                  second_coin_flips, observed_proportion, observations])
# To be explained in Chapter 3!
mcmc = pm.MCMC(model)
mcmc.sample(40000, 15000)
```

```
[****************100%******************]  40000 of 40000 complete
```

```
figsize(12.5, 3)
p_trace = mcmc.trace("freq_cheating")[:]
plt.hist(p_trace, histtype="stepfilled", normed=True, alpha=0.85, bins=30,
         label="posterior distribution", color="#348ABD")
plt.vlines([.05, .35], [0, 0], [5, 5], alpha=0.3)
plt.xlim(0, 1)
plt.legend();
```

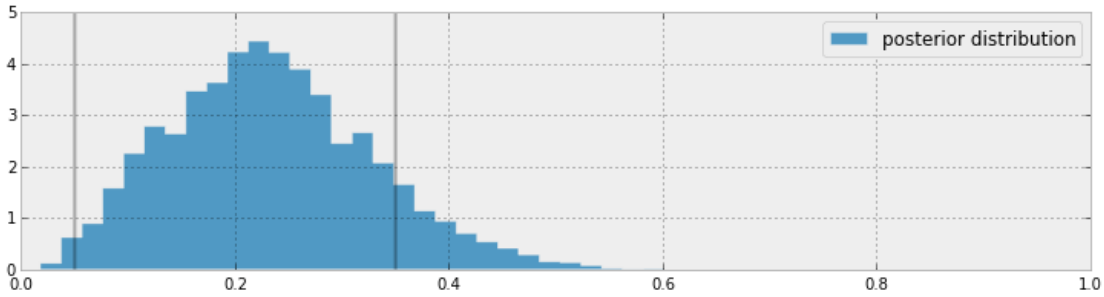


图** p的后验概率分布

从以上图中仍然不能看出作弊的真实频率是多少，但是可以把区间限制在 0.05 到 0.35 之间，图中的实线之间。这样的结果相当不错，因为不知道先验分布是什么样的，所以将先验分布取为均匀分布。但是从另一种角度看，这个结果又非常不好，因为真值得区间长度只有 0.3。用这区间内的信息就真的可以获得所有我们想要的东西吗，或者说我们依然对真实的频率还怀有很大的怀疑吗？

可以说已经发现了一些信息。这些信息又有些不合情理，从后验概率分布图中可以看出没有作弊者，因为在$p = 0$时概率很小。因为我们选择了均匀分布的先验概率，这就导致所有p的值都差不多是一样的，but the data ruled out p=0 as a possibility, we can be confident that there were cheaters.

这种算法可以用来收集用户的隐私信息，虽然数据中含有一定的噪声，但是数据还是有一定的可信度的。

另一种 PyMC 模型

给定一个概率p，我们可以找到学生承认作弊的概率

$$P("Yes") = P(Heads\ on\ first\ coin)P(cheater) + P(Tail\ on\ first\ coin)P(Heads\ on\ second\ coin) = \frac{1}{2}p + \frac{1}{2}\frac{1}{2} = \frac{1}{2}p + \frac{1}{4}$$

有了上式，当知道p时就知道学生承认作弊（“Yes”）的概率了。在 PyMC 中创建一个确定性函数用来计算$P("Yes")$

```
p = pm.Uniform("freq_cheating", 0, 1)
@pm.deterministic
def p_skewed(p=p):
  return 0.5 * p + 0.25
```

如果知道了“Yes”的概率，用 p_skewed 表示，$N = 100$，“Yes”的个数就是二项随机变量，此二项分布的参数为 N 和 p_skewed。

下面的代码中的二项分布中，value=35，表示“Yes”的个数为 35，设置 observed=True。

```
yes_responses = pm.Binomial("number_cheaters", 100, p_skewed,
                    value=35, observed=True)
```

下面的代码将所有相关变量加入到模型容器中，并在模型上执行黑箱算法

```
model = pm.Model([yes_responses, p_skewed, p])
# To Be Explained in Chapter 3!
mcmc = pm.MCMC(model)
mcmc.sample(25000, 2500)
```

```
[****************100%******************]  25000 of 25000 complete
```

```
figsize(12.5, 3)
p_trace = mcmc.trace("freq_cheating")[:]
plt.hist(p_trace, histtype="stepfilled", normed=True, alpha=0.85, bins=30,
      label="posterior distribution", color="#348ABD")
plt.vlines([.05, .35], [0, 0], [5, 5], alpha=0.2)
plt.xlim(0, 1)
plt.legend();
```

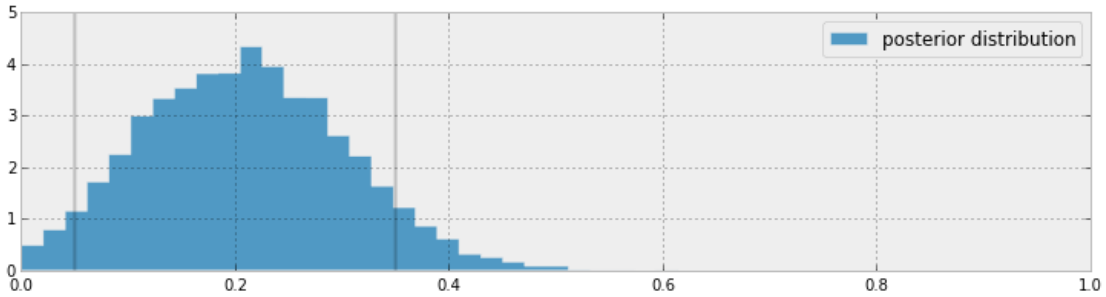


图** PyMC 计算的后验分布

更多的 PyMC 技巧

Protip：Lighter deterministic variables with Lambda class

有时候用@pm.deterministic 声明确定性变量让人感觉很繁琐，尤其对于一个小的函数来说。已经说过初等函数的运算可以隐式的定义确定性变量，但是数组的索引操作和切片操作也可以隐性的推断出确定性变量吗。内建的 Lambda 函数可以简单而优美的解决索引操作和切片操作的问题。例如

```
beta = pm.Normal("coefficients", 0, size=(N, 1))
x = np.random.randn((N, 1))
linear_combination = pm.Lambda(lambda x=x, beta=beta: np.dot(x.T, beta))
```

Protip: Arrays of PyMC variables

没有理由不存储多个异构的 PyMC 变量在 Numpy 数组中。只要在初始化数组时为相应的 object 添加上 dtype 属性即可，例如

```
N = 10
x = np.empty(N, dtype=object)
for i in range(0, N):
   x[i] = pm.Exponential('x_%i' % i, (i + 1) ** 2)
```

本章的余下内容将会介绍 PyMC 的例子和模型。

2.2.3 挑战者号航天飞机灾难

1986 年 1 月 28 日，美国航天飞机计划以灾难告终。在起飞后不就，航天飞机的其中一个助推器发生爆炸，七名宇航员全部遇难。总统的调查委员会得出的结论是，这场灾难是由于火箭推进器上的一个密封圈的失效引起的，导致失效的原因是密封圈对外界许多因素非常敏感，包括温度。从之前的 24 次发射中，可以得到此次失效的 O 形密封圈的 23 批次的数据（其中一个密封圈丢失在海上了）。在挑战者发射的前一天晚上，对这 23 组数据进行了专门的讨论，不幸的是得出只有七组数据反映出密封圈会造成损坏事故，这说明没有明显的趋势表明密封圈会出问题。数据展示如下

```
Temp (F), O-Ring failure?
[[ 66.  0.]
```

```
[ 70.  1.]
[ 69.  0.]
[ 68.  0.]
[ 67.  0.]
[ 72.  0.]
[ 73.  0.]
[ 70.  0.]
[ 57.  1.]
[ 63.  1.]
[ 70.  1.]
[ 78.  0.]
[ 67.  0.]
[ 53.  1.]
[ 67.  0.]
[ 75.  0.]
[ 70.  0.]
[ 81.  0.]
[ 76.  0.]
[ 79.  0.]
[ 75.  1.]
[ 76.  0.]
[ 58.  1.]]
<matplotlib.text.Text at 0x109766650>
```

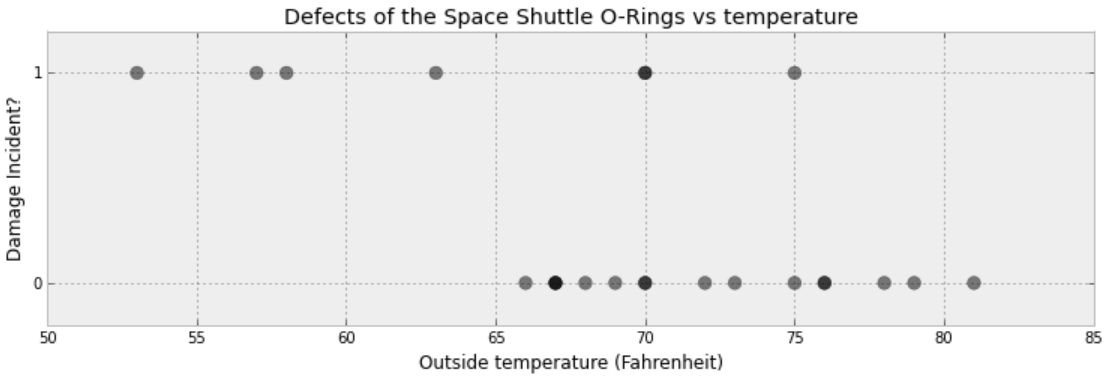


图** 外部温度与事故的关系

从图中可以看出随着外部温度降低，事故发生的概率在增加。我们的目标在于模拟事故发生的概率，在温度和事故之间看起来没有明显的截止点。最终的问题是在温度t时，事故发生的概率是多少。这个例子的目标就是要回答这个问题。

需要一个温度的函数，称之为$p(t)$，该函数的值域为[0,1]，随着温度的专家函数值从 1 增加到 0。有很多这样的函数，最常用的是 logistic 函数

$$p(t) = \frac{1}{1 + e^{\beta t}}$$

β是这个模型中的参数，也是需要求解的。下面画出$\beta = 1,3,-5$时的$p(t)$的曲线

```
figsize(12, 3)
def logistic(x, beta):
    return 1.0 / (1.0 + np.exp(beta * x))
x = np.linspace(-4, 4, 100)
plt.plot(x, logistic(x, 1), label=r"$\beta = 1$")
plt.plot(x, logistic(x, 3), label=r"$\beta = 3$")
plt.plot(x, logistic(x, -5), label=r"$\beta = -5$")
plt.legend();
```

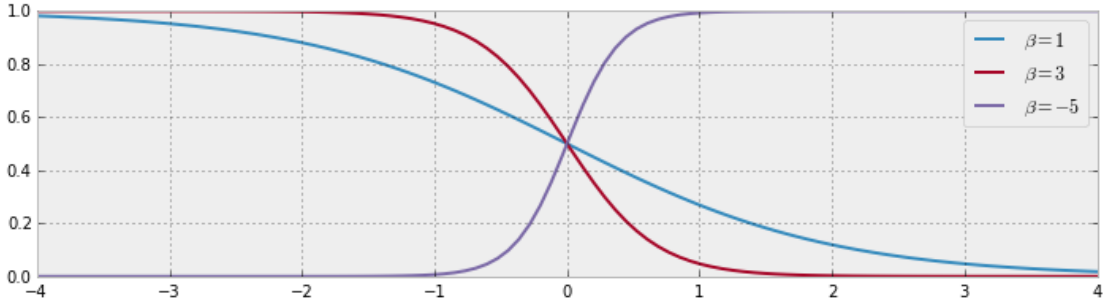


图** 不同β参数的 logistic 函数曲线

在上图中，在$t = 0$附件概率会发生改变，但是在实际的数据中在 65 到 70 这个范围内概率发生改变。所以还需要在 logistic 函数中添加一个偏置项，函数变为

$$p(t) = \frac{1}{1 + e^{(\beta t + \alpha)}}$$

不同的α对应的 logistic 函数如下

```
def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))
x = np.linspace(-4, 4, 100)
plt.plot(x, logistic(x, 1), label=r"$\beta = 1$", ls="--", lw=1)
plt.plot(x, logistic(x, 3), label=r"$\beta = 3$", ls="--", lw=1)
plt.plot(x, logistic(x, -5), label=r"$\beta = -5$", ls="--", lw=1)
plt.plot(x, logistic(x, 1, 1), label=r"$\beta = 1, \alpha = 1$",
         color="#348ABD")
plt.plot(x, logistic(x, 3, -2), label=r"$\beta = 3, \alpha = -2$",
         color="#A60628")
plt.plot(x, logistic(x, -5, 7), label=r"$\beta = -5, \alpha = 7$",
         color="#7A68A6")
plt.legend(loc="lower left");
```

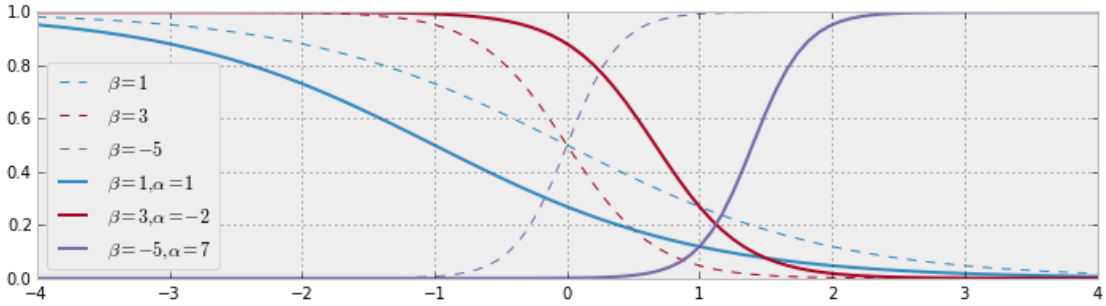


图** 不同β, α参数的 logistic 函数曲线

α参数控制着曲线在横轴上左右移动，这也就是把α称为偏置的原因。

下面将用 PyMC 模拟 logistic 函数。参数β, α取值为实数域，所以可以用正态分布的随机变量模拟参数β, α，下面将介绍正态分布。

正态分布

正态分布随机变量表示为$X\sim N(\mu|\,1/\tau)$，正态分布有两个参数均值μ，方差τ。也可以用σ^2代替$1/\tau$，它们二者互为倒数。The change was motivated by simpler mathematical analysis and is an artifact of older Bayesian methods。τ越小，正态分布的曲线越平坦（不确定性越大），反之亦然（不确定性越小）。τ的取值范围为正数。

概率密度函数为$N(\mu|\,1/\tau)$的随机变量为

$$\mathrm{f}(x|\mu,\tau)=\sqrt{\frac{\tau}{2\pi}}exp\left(-\frac{\tau}{2}(x-\mu)^2\right)$$

画出的图像如下

```
import scipy.stats as stats
nor = stats.norm
x = np.linspace(-8, 7, 150)
mu = (-2, 0, 3)
tau = (.7, 1, 2.8)
colors = ["#348ABD", "#A60628", "#7A68A6"]
parameters = zip(mu, tau, colors)
for _mu, _tau, _color in parameters:
    plt.plot(x, nor.pdf(x, _mu, scale=1. / _tau),
             label="$\mu = %d,\;\\tau = %.1f$" % (_mu, _tau), color=_color)
    plt.fill_between(x, nor.pdf(x, _mu, scale=1. / _tau), color=_color,
                     alpha=.33)
plt.legend(loc="upper right")
plt.xlabel("$x$")
plt.ylabel("density function at $x$")
plt.title("Probability distribution of three different Normal random \
variables");
```

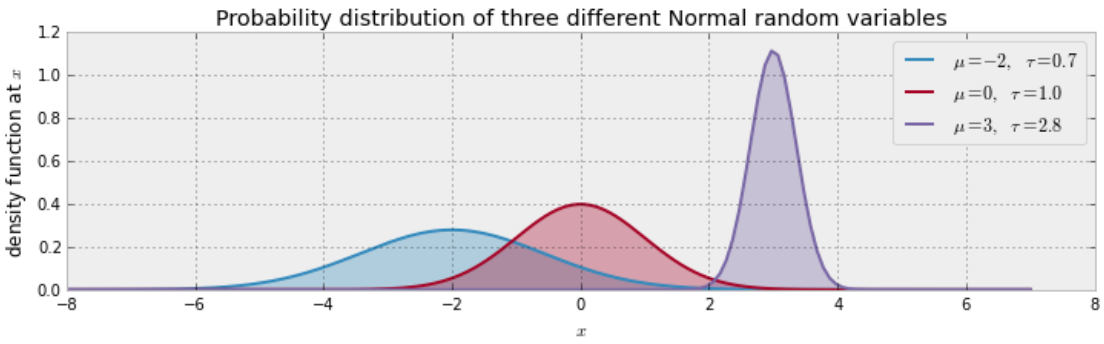


图** 不同参数的正态分布

正态分布的随机变量取值范围可以扩展到整个实数轴，但是随机变量大部分值集中在μ附件。正态分布的随机变量的期望等于μ

$$\mathrm{E}[X|\mu,\tau]=\mu$$

方差等于τ

$$\mathrm{Var}[X|\mu,\tau]=\frac{1}{\tau}$$

继续模拟挑战者号航天飞机

```
import pymc as pm
temperature = challenger_data[:, 0]
D = challenger_data[:, 1]  # defect or not?
# notice the`value` here. We explain why below.
beta = pm.Normal("beta", 0, 0.001, value=0)
alpha = pm.Normal("alpha", 0, 0.001, value=0)
@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta * t + alpha))
```

已经有了参数β, α的先验概率，如何将它们和观测数据融合到一起哪。伯努利随机变量的参数为p，分布为$\text{Ber}(p)$，伯努利随机变量取 1 时的概率为p，其它情况取 0，这样，我们的模型如下所示

$$\text{Defiect Incident}，\ D_i \sim Ber\big(p(t_i)\big), i = 1 \cdots N$$

$p(t)$logistic 回归函数，t_i代表温度。注意，在上面的代码中将 beta 和 alpha 设置为 0。这是因为如果 beta 和 alpha 的值非常大，p就可能为 1 或者 0。pm.Bernoulli 并不会返回概率 0 和 1，虽然 pm. Bernoulli 符合它数学上定义的概率。所以将系数值设置为 0，对应的 p 值作为初值，是合理的。这对结果没有什么影响，或者说在先验信息中包括了任何情况，在 PyMC 中计算非常简单

```
p.value
```

```
array([ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5])
```

```
# connect the probabilities in `p` with our observations through a
# Bernoulli random variable.
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)
model = pm.Model([observed, beta, alpha])
# Mysterious code to be explained in Chapter 3
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)
```

```
[****************100%******************]  120000 of 120000 complete
```

在观测数据上训练模型，并从后验概率上采样，β, α的后验概率如下

```
alpha_samples = mcmc.trace('alpha')[:, None]  # best to make them 1d
beta_samples = mcmc.trace('beta')[:, None]
figsize(12.5, 6)
# histogram of the samples:
plt.subplot(211)
plt.title(r"Posterior distributions of the variables $\alpha, \beta$")
plt.hist(beta_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\beta$", color="#7A68A6", normed=True)
plt.legend()
plt.subplot(212)
plt.hist(alpha_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\alpha$", color="#A60628", normed=True)
plt.legend();
```

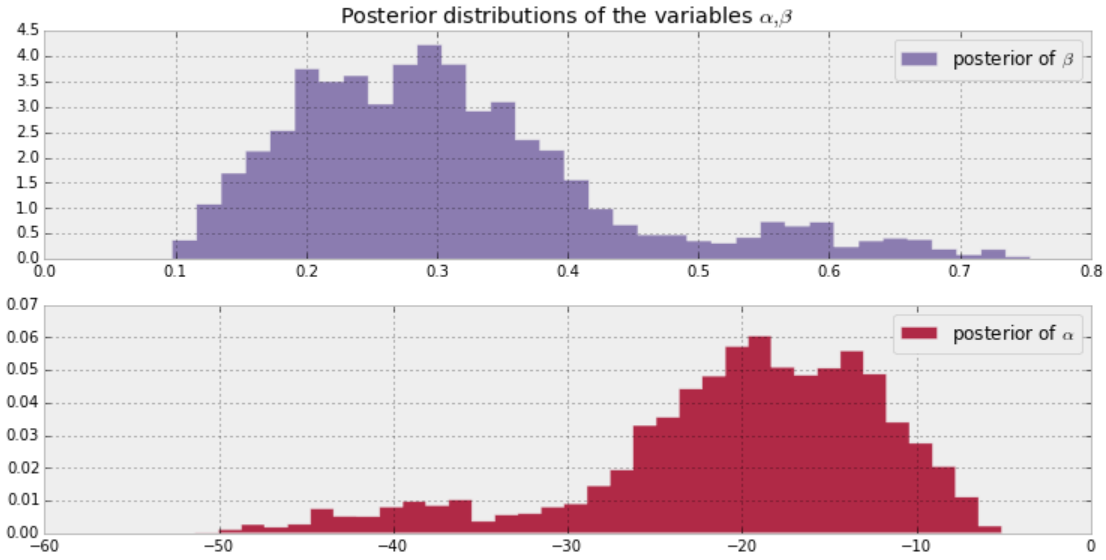


图** β, α的后验概率图

所有的β样本值都大于 0。如果后验概率集中在 0 附件，假设$\beta = 0$，说明温度对失败的概率没有影响。

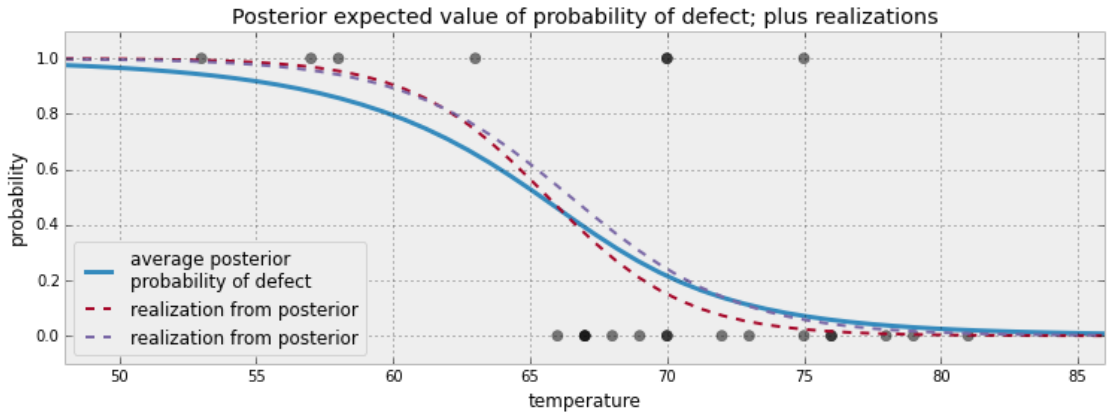
所有的α后验值都是负的且远离 0，可以确信α一定是小于 0 的。

鉴于数据的分布情况，我们不确定真实的参数β, α到底是什么（though considering the low sample size and the large overlap of defects-to-nondefects this behaviour is perhaps expected）。

下面看看对于特定的温度下，期望概率是多少。也就是对所有后验概率的样本进行平均得到一个可能的$p(t_i)$值

```
t = np.linspace(temperature.min() - 5, temperature.max() + 5, 50)[:, None]
p_t = logistic(t.T, beta_samples, alpha_samples)
mean_prob_t = p_t.mean(axis=0)
```

```
figsize(12.5, 4)
plt.plot(t, mean_prob_t, lw=3, label="average posterior \nprobability \
of defect")
plt.plot(t, p_t[0, :], ls="--", label="realization from posterior")
plt.plot(t, p_t[-2, :], ls="--", label="realization from posterior")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.title("Posterior expected value of probability of defect; \
plus realizations")
plt.legend(loc="lower left")
plt.ylim(-0.1, 1.1)
plt.xlim(t.min(), t.max())
plt.ylabel("probability")
plt.xlabel("temperature");
```



图**

上图中画出了两条实际系统的可能曲线。Both are equally likely as any other draw。蓝色的曲线是对 20000 个虚线样本点进行平均后的结果。

一个有趣的问题是，什么样的温度导致事故发生的概率最大？下面画出期望值曲线，每个温度有一个概率区间。

```
from scipy.stats.mstats import mquantiles

# vectorized bottom and top 2.5% quantiles for "confidence interval"
qs = mquantiles(p_t, [0.025, 0.975], axis=0)
plt.fill_between(t[:, 0], *qs, alpha=0.7,
                 color="#7A68A6")
plt.plot(t[:, 0], qs[0], label="95% CI", color="#7A68A6", alpha=0.7)
plt.plot(t, mean_prob_t, lw=1, ls="--", color="k",
         label="average posterior \nprobability of defect")
plt.xlim(t.min(), t.max())
plt.ylim(-0.02, 1.02)
plt.legend(loc="lower left")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.xlabel("temp, $t$")
plt.ylabel("probability estimate")
plt.title("Posterior probability estimates given temp. $t$");
```

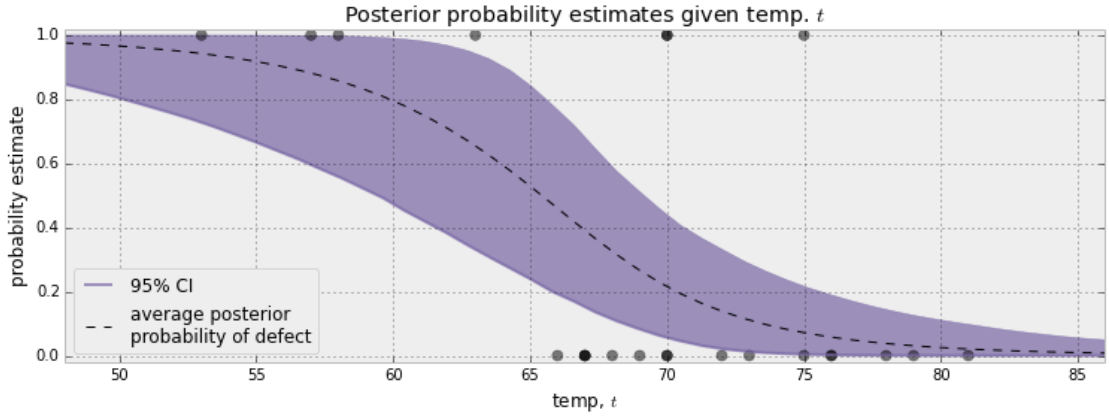


图** 在t时刻的后验概率估计

紫色区域表示 95%置信区间或者 95% CI，对应着每个温度下发生事故的概率区间。例如在 65 度时，有 95%的确信度相信事故发生的概率范围为 0.25 到 0.75 之间。

更一般情况，当温度到达 60 度时，CI's 快速变窄，当温度超过 70 度时 CI's 也有类似的情况。这将给我提供接下来如何分析的思路：我们可能在温度区间 60-65 上做更多的 O 形密封圈的测试，这样就能得到更好的估计概率。所以，当你向科学家介绍你的估计结果时，要非常小心而不是简单的告诉他们概率期望的结果是多少，因为我们看到其没有真正的反映出后验分布的宽度有多大。

挑战者发生灾难的那天天气如何？

挑战者号发生灾难的那天，外部的温度是 31 华氏度。在此时的温度下，发生灾难的后验分布如何？分布图如下。看起来挑战者号必定会发生由于 O 形密封环导致的灾难。

```
figsize(12.5, 2.5)
prob_31 = logistic(31, beta_samples, alpha_samples)
plt.xlim(0.995, 1)
plt.hist(prob_31, bins=1000, normed=True, histtype='stepfilled')
plt.title("Posterior distribution of probability of defect, given $t = 31$")
plt.xlabel("probability of defect occurring in O-ring");
```

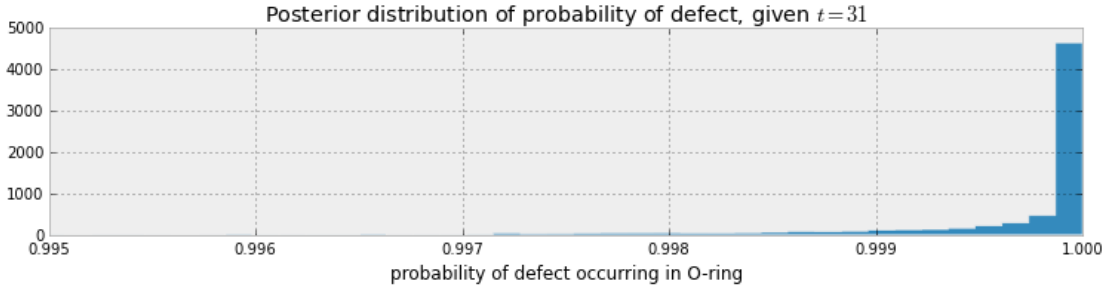


图** 某一温度下的事故发生的后验概率

我们的模型合理吗？

有的读者可能会问。本文选择了 logistic 函数和先验概率是合理的吗。如果选择了其它函数结果也许就不一样了。如何能够知道我们选择的模型是好的。这种怀疑完全正确。考虑一种极端情况，如果让$p(t) = 1, \forall t$，这保证事故一定会发生，我们应该可以再次预测在 1 月 28 号会发生灾难。显然这是一个很差的模型。另一方面，如果选择了 logistic 函数，并让先验概率都在 0 附近，可能会得到不同的后验概率。如何能够知道模型能够对数据进行了很好的表达？这就需要检验模型的吻合度问题。

我们可能会想：如何验证我们的模型的吻合度是否不好？一种想法是将观测数据和人造的数据集进行对比。这种做法的理论根据是，如果模拟出的数据和观测数据在统计学上没有相似的地方，那么我们的模型很可能没有准确的表示观测数据。

在本章的前面部分，我们为 SMS 仿真出了数据集。SMS 的例子是从先验分布中采样出数据。我们看到仿真出的数据和观测到的数据有很大不同。在当前这个例子中，从后验概率中采样出数据，模拟出尽量合理的数据。幸运的是，贝叶斯框架让采样的事情非常简单。只需要创建一个 Stochastic 变量，这个随机变量模拟观测到的随机变，但是不包括观测到的随机变量值。If you recall, our Stochastic variable that stored our observed data was:

```
observed = pm.Bernoulli( "bernoulli_obs", p, value=D, observed=True)
```

下面得到仿真数据

```
simulated_data = pm.Bernoulli("simulation_data", p)
```

仿真 10000 次

```
simulated = pm.Bernoulli("bernoulli_sim", p)
N = 10000
mcmc = pm.MCMC([simulated, alpha, beta, observed])
mcmc.sample(N)
```

```
[****************100%******************]  10000 of 10000 complete
```

```
figsize(12.5, 5)
simulations = mcmc.trace("bernoulli_sim")[:]
print simulations.shape
plt.title("Simulated dataset using posterior parameters")
figsize(12.5, 6)
for i in range(4):
    ax = plt.subplot(4, 1, i + 1)
    plt.scatter(temperature, simulations[1000 * i, :], color="k",
                s=50, alpha=0.6)
```

```
(10000, 23)
```

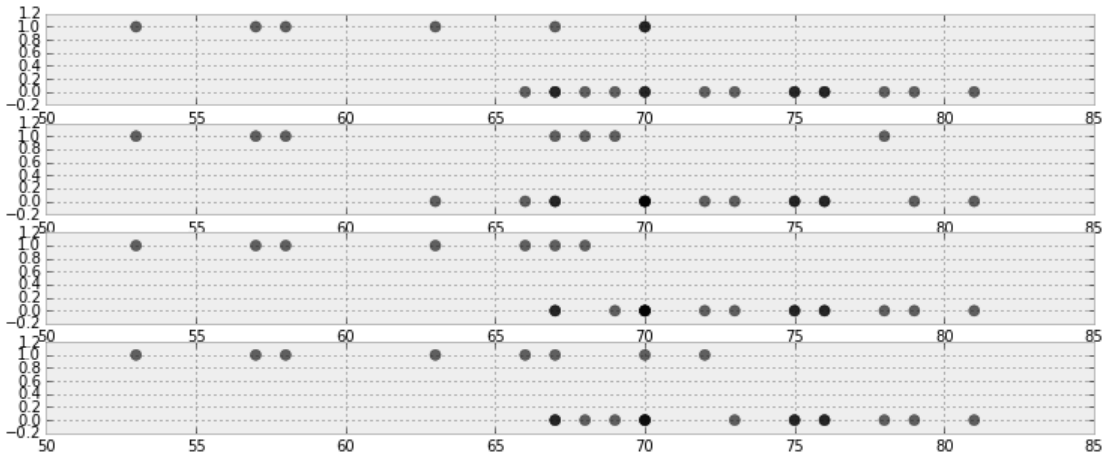


图** 仿真数据

注意，上面画出的图是有差异的。

我们希望知道模型是如何的好。“好”是个带有主观意思的词汇，所以结果必须和其它结果进行比较。

下面将使用图形进行比较，这看起来跟不客观。另一种方法是使用贝叶斯的 p 值。这仍然有些主观性在里面，因为好坏之间的分界线依然是任意的。Gelman 强调图形方法比 p 值检验更直观且具有启发性。

对于逻辑回归来说，下面的图形检验方法是一种的新的 data-viz 方法，画出的图被称之为分离图。如果有很多个模型，每个模型都要画出一个分离图，更多分离图的细节见《The Separation Plot: A New Visual Method for Evaluating the Fit of Binary Models》，下面将简要说明其用法。

对于每个模型，计算特定温度下 1 的后验仿真次数的比例，例如计算 $P(\text{Defect} = 1|t, \alpha, \beta)$的平均值。这就可以计算出在数据集中任意数据点的事故概率。例如对于以上的模型有

```
posterior_probability = simulations.mean(axis=0)
print "posterior prob of defect | realized defect "
for i in range(len(D)):
    print "%.2f                    |   %d" % (posterior_probability[i], D[i])
```

posterior prob of defect | realized defect
0.47 | 0
0.20 | 1
0.25 | 0
0.32 | 0
0.39 | 0
0.11 | 0
0.08 | 0
0.19 | 0
0.94 | 1
0.71 | 1
0.20 | 1
0.02 | 0
0.39 | 0
0.98 | 1
0.40 | 0
0.05 | 0
0.20 | 0
0.01 | 0
0.03 | 0
0.01 | 0
0.05 | 1
0.04 | 0
0.92 | 1

按照后验概率排序

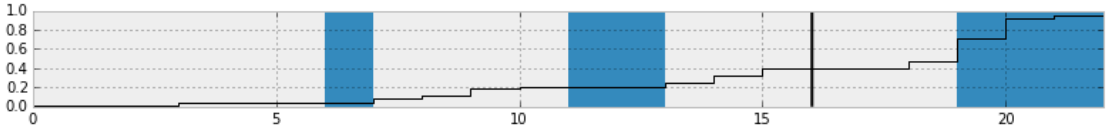
```
ix = np.argsort(posterior_probability)
print "probb | defect "
for i in range(len(D)):
    print "%.2f  |   %d" % (posterior_probability[ix[i]], D[ix[i]])
```

probb | defect
0.01 | 0
0.01 | 0
0.02 | 0
0.03 | 0
0.04 | 0
0.05 | 0
0.05 | 1
0.08 | 0
0.11 | 0
0.19 | 0
0.20 | 0
0.20 | 1
0.20 | 1
0.25 | 0
0.32 | 0
0.39 | 0
0.39 | 0

0.40 | 0
0.47 | 0
0.71 | 1
0.92 | 1
0.94 | 1
0.98 | 1

将以上的数据画成分离图

```
from separation_plot import separation_plot
figsize(11., 1.5)
separation_plot(posterior_probability, D)
```



图**

图中的蛇形线就是排序后的概率，蓝色的柱子表示事故，没有柱子的区域表示没有事故。随着概率增加，可以看到事故发生的概率越来越大，在图中的右边，说明后验概率越来越大（因为蛇形线接近 1），表示更多的事故发生。理想情况所有的蓝色柱子都要靠近右边，这里面会存在一些偏差，偏差存在的地方预测也就缺失了。

垂直的线表示在这个模型下，所能观测到的事故次数。这根垂直的线可以让我们比较模型预测出的事故次数和真实事故次数各是多少。

将这个模型的分离图和其它模型的分离图进行对比，会发现更多的有用信息。下面将上面的模型与其它三个进行对比：

1 如果模型很完美，当事故确实发生时，它预测的后验概率是 1

2 一个随机模型预测的结果完全是随机的，将和温度没有任何关系

3 对于一个常数模型，即$\mathrm{P}(D = 1|t) = c, \forall t$。$c$的最优值是观测到的事故发生的概率，即$c = 7/23$。

```
figsize(11., 1.25)
# Our temperature-dependent model
separation_plot(posterior_probability, D)
plt.title("Temperature-dependent model")
# Perfect model
# i.e. the probability of defect is equal to if a defect occurred or not.
p = D
separation_plot(p, D)
plt.title("Perfect model")
# random predictions
p = np.random.rand(23)
separation_plot(p, D)
plt.title("Random model")
# constant model
constant_prob = 7. / 23 * np.ones(23)
separation_plot(constant_prob, D)
plt.title("Constant-prediction model")
```

<matplotlib.text.Text at 0x10c392b90>

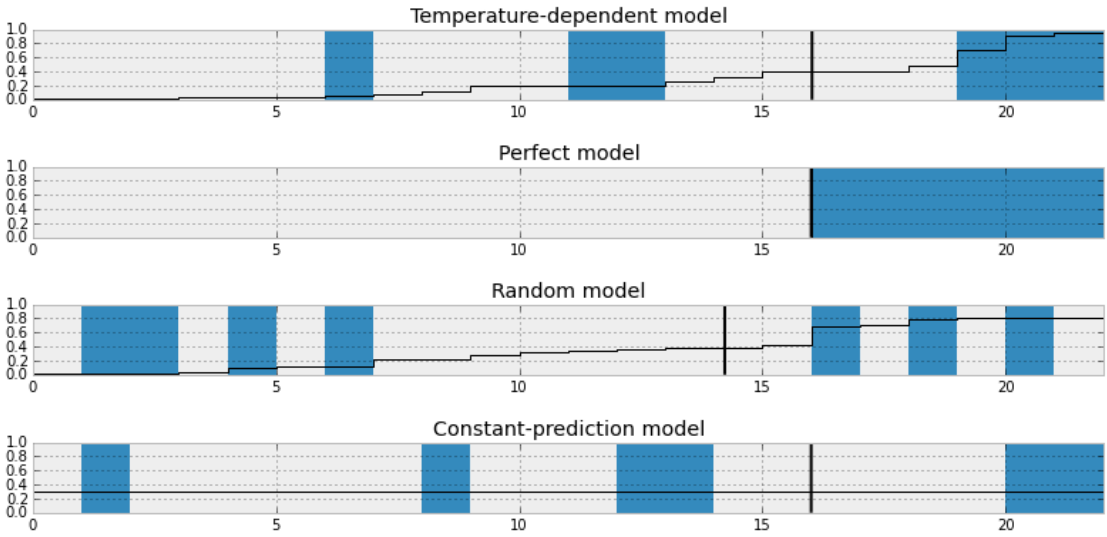


图** 各种模型对比图

在随机模型中，可以看到，随着概率的增加，事故并没有向右侧聚集。常数模型也是这个情况。

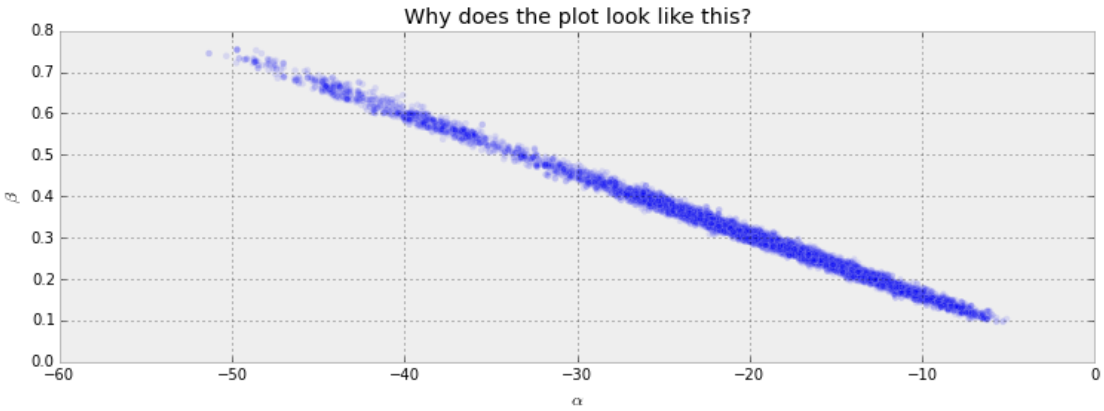
完美模型中没有显示概率曲线。因为它一直位于柱子的顶端。此处的完美模型只是用于演示，并不能从中推断出什么科学依据。

练习

1. Try putting in extreme values for our observations in the cheating example. What happens if we observe 25 affirmative responses? 10? 50?

2. Try plotting α samples versus β samples. Why might the resulting plot look like this?

```
# type your code here.
figsize(12.5, 4)
plt.scatter(alpha_samples, beta_samples, alpha=0.1)
plt.title("Why does the plot look like this?")
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$\beta$")
```



2.3 References

[1] Dalal, Fowlkes and Hoadley (1989),JASA, 84, 945-957.

[2] German Rodriguez. Datasets. In WWS509. Retrieved 30/01/2013, from http://data.princeton.edu/wws509/datasets/#smoking.

[3] McLeish, Don, and Cyntha Struthers. STATISTICS 450/850 Estimation and Hypothesis Testing. Winter 2012. Waterloo, Ontario: 2012. Print.

[4] Fonnesbeck, Christopher. "Building Models." PyMC-Devs. N.p., n.d. Web. 26 Feb 2013. http://pymc-devs.github.com/pymc/modelbuilding.html.

[5] Cronin, Beau. "Why Probabilistic Programming Matters." 24 Mar 2013. Google, Online Posting to Google . Web. 24 Mar. 2013. https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1.

[6] S.P. Brooks, E.A. Catchpole, and B.J.T. Morgan. Bayesian animal survival estimation. Statistical Science, 15: 357–376, 2000

[7] Gelman, Andrew. "Philosophy and the practice of Bayesian statistics." British Journal of Mathematical and Statistical Psychology. (2012): n. page. Web. 2 Apr. 2013.

[8] Greenhill, Brian, Michael D. Ward, and Audrey Sacks. "The Separation Plot: A New Visual Method for Evaluating the Fit of Binary Models." American Journal of Political Science. 55.No.4 (2011): n. page. Web. 2 Apr. 2013

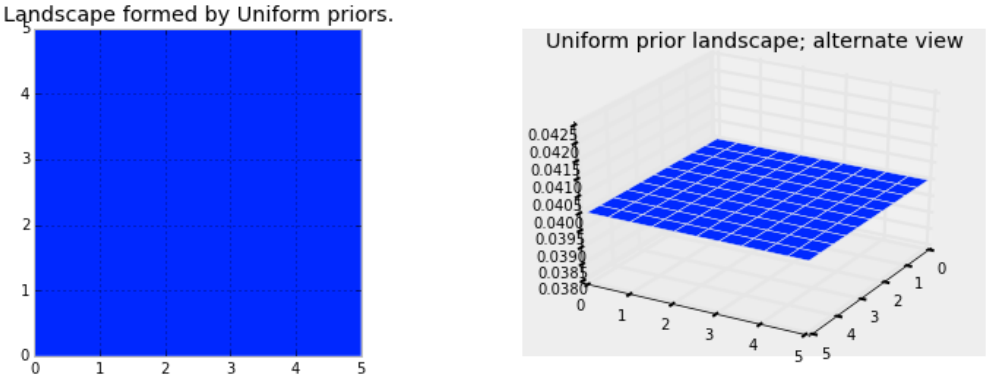
第 3 章 马尔科夫蒙特卡洛方法及其使用

前面两章没有介绍 PyMC 的细节，亦没有介绍更多的蒙特卡洛方法（MCMC）。本章有三个重点。第一，将会讨论任何贝叶斯推断的书都会讨论的 MCMC，所以本文也无法跳过此部分。第二，介绍 MCMC 的实现过程，向读者介绍 MCMC 算法如何收敛，收敛到何处。第三，将会介绍为什么从后验分布中能返回成千上万的样本，刚开始时这种采样过程看起来有点奇怪。

3.1 贝叶斯方法

当构造一个有N个未知变量的贝叶斯推断问题时，首先要隐式的创建 N 维空间（可以理解为 N 个随机变量）的先验分布。这 N 个随机变量的函数，是一个曲面或者曲线，曲线或者曲面反映了一个点的概率，即概率密度函数。例如有两个未知的随机变量p_1和p_2，这两个随机变量的先验分布为 Uniform(0,5)，p_1和p_2构成了一个5×5的平面空间，代表概率密度的曲面位于这个5×5的平面空间上（表示任何点的概率都是相同的）。

```
%matplotlib inline
import scipy.stats as stats
from IPython.core.pylabtools import figsize
import numpy as np
figsize(12.5, 4)
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
jet = plt.cm.jet
fig = plt.figure()
x = y = np.linspace(0, 5, 100)
X, Y = np.meshgrid(x, y)
plt.subplot(121)
uni_x = stats.uniform.pdf(x, loc=0, scale=5)
uni_y = stats.uniform.pdf(y, loc=0, scale=5)
M = np.dot(uni_x[:, None], uni_y[None, :])
im = plt.imshow(M, interpolation='none', origin='lower',
                cmap=jet, vmax=1, vmin=-.15, extent=(0, 5, 0, 5))
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title("Landscape formed by Uniform priors.")
ax = fig.add_subplot(122, projection='3d')
ax.plot_surface(X, Y, M, cmap=plt.cm.jet, vmax=1, vmin=-.15)
ax.view_init(azim=390)
plt.title("Uniform prior landscape; alternate view");
```



图**

如果两个先验分布是 Exp(3)和 Exp(10)，则构成的空间是在二维平面上的正数，同时表示先验概率的曲面就像从(0,0)点开始的瀑布。

下图中画出了此时的先验分布情况。颜色越红的地方，说明概率密度越大。反之，蓝色越深的地方说明概率密度越小。

```
figsize(12.5, 5)
fig = plt.figure()
plt.subplot(121)
exp_x = stats.expon.pdf(x, scale=3)
exp_y = stats.expon.pdf(x, scale=10)
M = np.dot(exp_x[:, None], exp_y[None, :])
CS = plt.contour(X, Y, M)
im = plt.imshow(M, interpolation='none', origin='lower',
                cmap=jet, extent=(0, 5, 0, 5))
#plt.xlabel("prior on $p_1$")
#plt.ylabel("prior on $p_2$")
plt.title("$Exp(3), Exp(10)$ prior landscape")
ax = fig.add_subplot(122, projection='3d')
ax.plot_surface(X, Y, M, cmap=jet)
ax.view_init(azim=390)
plt.title("$Exp(3), Exp(10)$ prior landscape; \nalternate view")
```

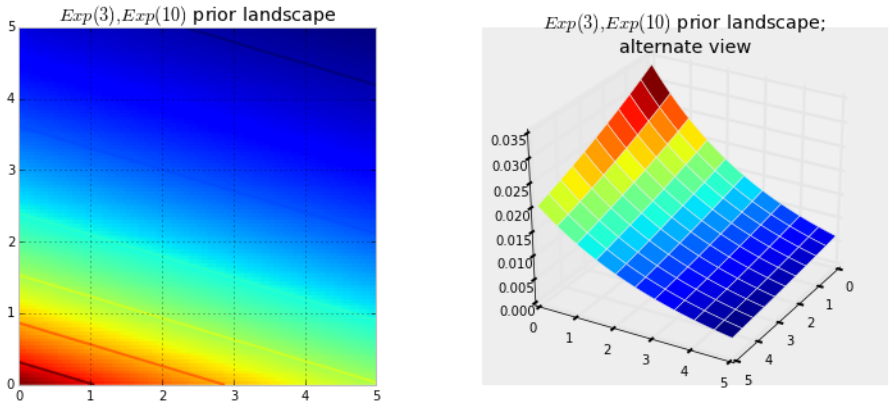


图** 二维空间上的先验概率

这是二维空间上的简单例子，人的大脑可以很好的理解二维空间。事实上平时碰到的实际问题大多是高维问题。

如果这些曲面描述的是未知变量的先验分布，那么当加入观测数据X后，后验分布会变成什么样哪？观测数据X并不会改变变量的空间，它只是通过拉伸和压缩先验分布的曲面，让曲面更符合实际的参数分布。数据X越多拉伸和压缩就越厉害，这样后验分布就变化的越厉害，可能完全看不出和先验分布的曲面有什么关系，或者说随着数据的增加先验分布对于后验分布的影响越来越小。如果数据X较小，那么后验分布的形状会更多的反映出先验分布的形状。

再次说明，在高维空间上是看不到这种直观的分布曲面的。在二维空间上，数据通过压缩二维空间上的曲面，让曲面变成高高的山包形状。观测数据在特定的区域推高先验概率，后验概率变化程度受先验概率影响，如果先验概率大的地方后验概率也会变得较大，先验概率小的地方后验概率也会变化较小，所以可以将先验概率小的地方看成阻力较大。所以对于双指数先验分布函数，在(0,0)位置的峰值比(5,5)位置的峰值要大的

多，因为在(5,5)位置有更大的阻力（先验概率比较小）。后验概率的峰值所反映的是位置参数真值所在的位置，如果一个位置的先验概率为 0，则该位置的后验概率也为 0。

如果以上所说的先验分布用两个不同参数λ参数的泊松分布表示。在加入一些后验分布的数据后，将会看到一些新的景像（后验分布）。

```
# create the observed data
# sample size of data we observe, trying varying this (keep it less than 100 ;)
N = 1
# the true parameters, but of course we do not see these values...
lambda_1_true = 1
lambda_2_true = 3
#...we see the data generated, dependent on the above two values.
data = np.concatenate([
    stats.poisson.rvs(lambda_1_true, size=(N, 1)),
    stats.poisson.rvs(lambda_2_true, size=(N, 1))
], axis=1)
print "observed (2-dimensional,sample size = %d):" % N, data
# plotting details.
x = y = np.linspace(.01, 5, 100)
likelihood_x = np.array([stats.poisson.pmf(data[:, 0], _x)
                         for _x in x]).prod(axis=1)
likelihood_y = np.array([stats.poisson.pmf(data[:, 1], _y)
                         for _y in y]).prod(axis=1)
L = np.dot(likelihood_x[:, None], likelihood_y[None, :])
```

```
observed (2-dimensional,sample size = 1): [[2 2]]
```

```
figsize(12.5, 12)
# matplotlib heavy lifting below, beware!
plt.subplot(221)
uni_x = stats.uniform.pdf(x, loc=0, scale=5)
uni_y = stats.uniform.pdf(x, loc=0, scale=5)
M = np.dot(uni_x[:, None], uni_y[None, :])
im = plt.imshow(M, interpolation='none', origin='lower', cmap=jet, vmax=1, vmin=-.15,
extent=(0, 5, 0, 5))
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title("Landscape formed by Uniform priors on $p_1, p_2$.")
plt.subplot(223)
plt.contour(x, y, M * L)
im = plt.imshow(M * L, interpolation='none', origin='lower', cmap=jet, extent=(0, 5, 0, 5))
plt.title("Landscape warped by %d data observation;\n Uniform priors on $p_1, p_2$." % N)
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.subplot(222)
exp_x = stats.expon.pdf(x, loc=0, scale=3)
exp_y = stats.expon.pdf(x, loc=0, scale=10)
M = np.dot(exp_x[:, None], exp_y[None, :])
plt.contour(x, y, M)
im = plt.imshow(M, interpolation='none', origin='lower', cmap=jet, extent=(0, 5, 0, 5))
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title("Landscape formed by Exponential priors on $p_1, p_2$.")
plt.subplot(224)
# This is the likelihood times prior, that results in the posterior.
plt.contour(x, y, M * L)
im = plt.imshow(M * L, interpolation='none', origin='lower',cmap=jet, extent=(0, 5, 0, 5))
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.title("Landscape warped by %d data observation;\n Exponential priors on \
$p_1, p_2$." % N)
plt.xlim(0, 5)
plt.ylim(0, 5)
```

`

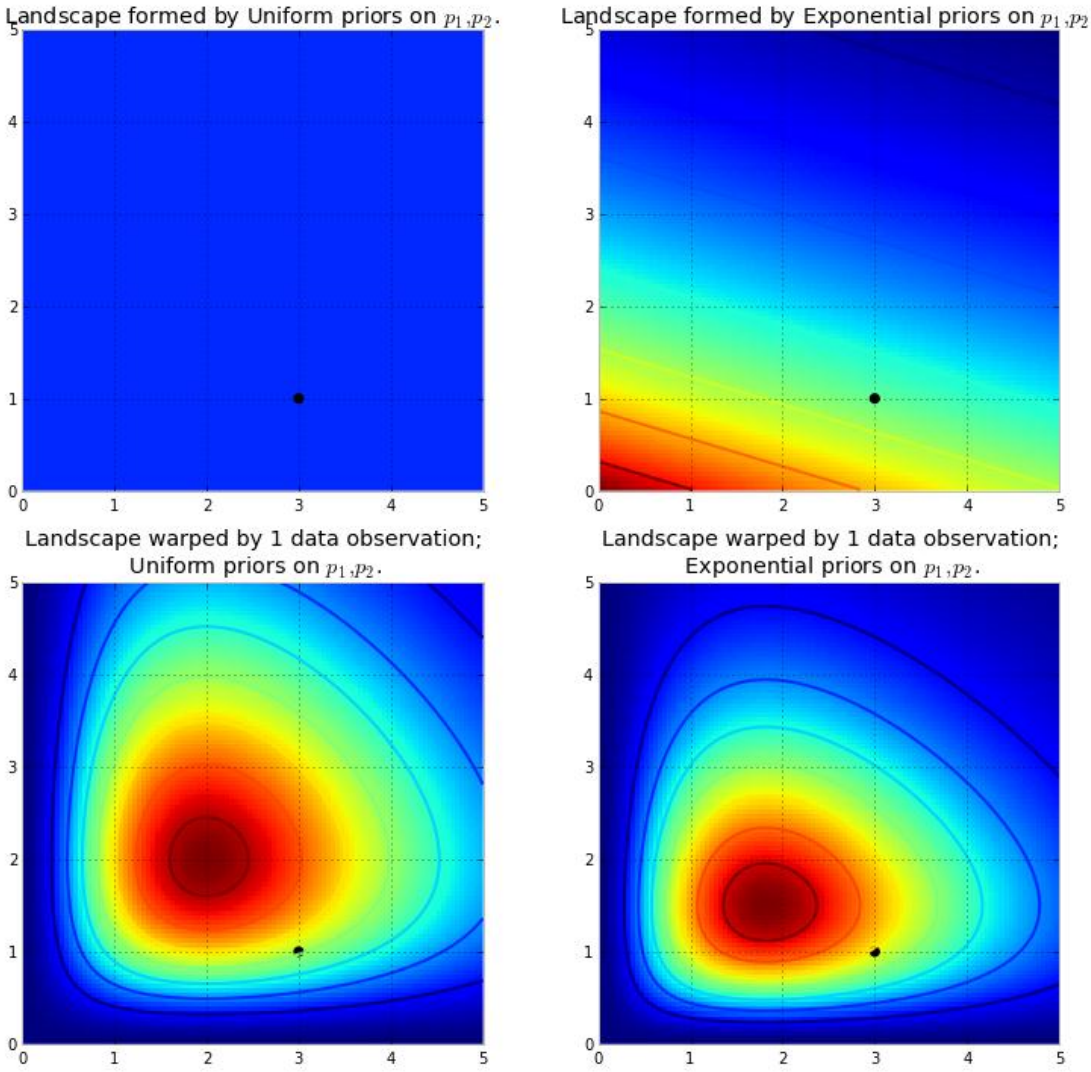


图** 先验分布和后验分布的关系

左边的图是先验分布为 Uniform(0,5)，加上观测数据后的后验概率变化图；右边的图是先验概率为指数分布，加上观测数据后的后验概率变化图。从图中可以看出，虽然加入的观测数据一致，后验分布的两个图的形状看起来有很大不同。原因如下，注意到右下方的图，它是指数先验分布的后验分布，该图右上角的后验概率权重较小，因为在先验概率中该处的权重亦很小。左下方的图中是先验分布为均匀分布的后验分布图，该图中右上角的权重较大，因为先验分布在那个位置的权重也较大。

在右下方的图中，图中颜色最深的地方，即等高线值最大的地方向(0,0)偏移，那是因为指数先验分布的权重在(0,0)处比其它地方要大。

图中的黑点表示参数的真值所在的位置，即使观测数据只有一个点，后验概率也在向真值靠近。当然了用一个样本进行推断是不合适的，这里只是为了演示而已。

可以改变观测样本的数量，观测后验概率的峰值变化情况。

3.2 MCMC 奇幻之旅

研究由先验分布和观测数据构造出的后验分布空间，目的是要找出后验分布的峰值。然而我们不能鲁莽的搜索整个空间，因为任何计算机科学家都会告诉你 traversing N-dimensional space is exponentially difficult in N，随着 N 的增加空间的大小会呈爆炸式的增加

（见维数灾难 http://en.wikipedia.org/wiki/Curse_of_dimensionality）。我们希望如何找到这些隐藏的峰值哪？MCMC 背后的思想就是用一种智能的方法搜索空间。“搜索”一词表面是在寻找一个特定的点，也许这种寻找方法并不准确，因为寻找的峰值有一定的分布范围。

我们知道 MCMC 是从后验分布而不是自身分布中采样样本。MCMCT 通过改变后验分布使其峰值越来越大，MCMC 所做的工作类似于反复询问这样一个问题“我们所找到的石头与我们期待找到的那座山的相似度到底有多大？”，为了使找到的石头和我们期待的那座上一致，MCMC 方法类似于返回一些列的相关石头，用这些石头重构我们需要的山。用 MCMC 和 PyMC 术语来说，MCMC 返回的“石头”是样本，它们累积起来称为迹。

MCMC 是智能搜索方法，MCMC 很有希望收敛到后验概率高的区域。MCMC 通过探索附件的位置，并向高概率位置移动。“收敛”很可能无法描述 MCMC 的计算过程。收敛通常是说移动到空间中的某一点，但是 MCMC 移动到空间中的一块广阔的区域，并在空间中随机游走，从所位于的局部空间中抽取样本。

为什么需要成千上万的样本？

首先，返回成千上万的样本让人觉得这是一种低效的描述后验概率的方式。实际上这是一种非常高效的方法。下面是其它可能用于计算后验概率的方法

1 用解析表达式描述“山峰区域”（后验分布），这需要描述带有山峰和山谷的 N 维曲面。

2 Returning the "peak" of the landscape, while mathematically possible and a sensible thing to do as the highest point corresponds to most probable estimate of the unknowns, ignores the shape of the landscape, which we have previously argued is very important in determining posterior confidence in unknowns.

除了计算原因，另一个主要原因是，利用返回的样本可以利用大数定理解决非常棘手问题。下一章将会讨论这个问题。有了成千上万的样本，就可以利用直方图技术，重构后验分布曲面。

3.3 MCMC 技术

有很多算法实现 MCMC。大部分的算法可以表述如下（详细的数据表示见附录）

1 选定初始位置。

2 移动到一个新的位置（探索附件的点）。

3 根据新数据位置和先验分布的紧密程度拒绝或接受新的数据点（石头是否来自于期望的那座山上）。

4 A，如果接受，移动到新的点，返回到步骤 1。

B，否则不移动到新的点返回到步骤 1。

5 在经过大量的迭代之后，返回所有接受的点。

新数据移动的方向指向先验概率存在的区域，在移动的过程中保守的收集样本。一旦抵达后验分布区域，就可以很容易的收集到样本，因为样本属于后验分布的可能性很大。

如果 MCMC 算法当前所位于的区域概率值很小，这可能是算法刚开始的区域，the algorithm will move in positions that are likely not from the posterior but better than everything else nearby。所以 MCMC 算法中第一次移动获得的样本不会反映出后验分布的特性。

在以上的算法伪代码中，只提及到了当前点（新的点要通过当前点得到）。可以将这种特性称之为无记忆特质的，算法不关心它是如何到达当前点的。

3.4 其它近似的后验概率求解方法

除了 MCMC 外，还有其它方法可以用于求解后验分布。拉普拉斯近似可以用简单函数近似后验分布。还有更高级的方法是变分贝叶斯方法（http://en.wikipedia.org/wiki/Variational_Bayesian_methods）。这三种方法拉普拉斯近似，变分贝叶斯，经典的 MCMC 方法，各有自己的优缺点。本书中将只关注 MCMC 方法。That being said, my friend Imri Sofar likes to classify MCMC algorithms as either "they suck", or "they really suck". He classifies the particular flavour of MCMC used by PyMC as just sucks ;)

3.4 示例：无监督的混合模型聚类

假设有如下数据集

```
figsize(12.5, 4)
data = np.loadtxt("data/mixture_data.csv", delimiter=",")
plt.hist(data, bins=20, color="k", histtype="stepfilled", alpha=0.8)
plt.title("Histogram of the dataset")
plt.ylim([0, None])
print data[:10], "..."
```

```
[ 115.85679142  152.26153716  178.87449059  162.93500815  107.02820697
  105.19141146  118.38288501  125.3769803   102.88054011  206.71326136] ...
```

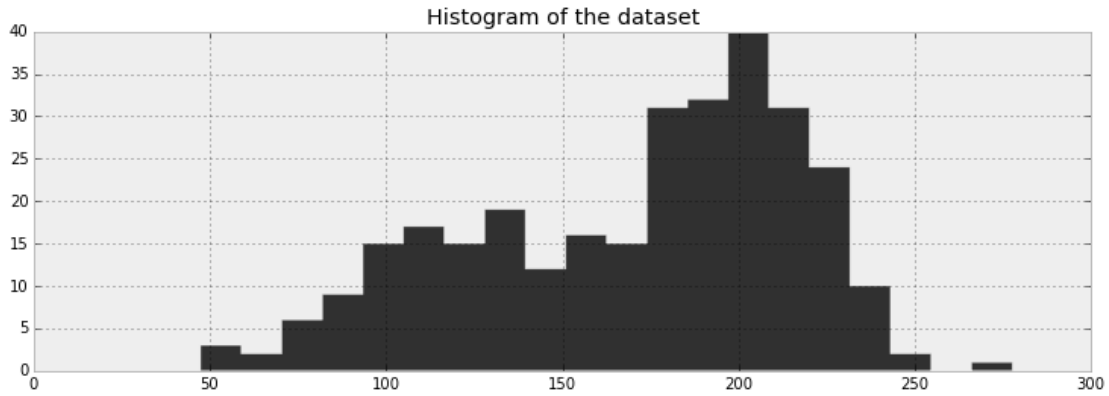


图** 用于聚类的数据集

从数据中能看出什么规律？数据看起来有两个模型，即有两个峰值，其中一个位于横坐标的 120 附件，另一个位于横坐标的 200 附件。所以，可以将数据聚成两个族。

这个例子能否很好的演示，上一章中介绍的数据生成模型生成数据的过程。下面介绍数据是如何产生的。生成的数据方法如下

1 对于每个数据点，让它以概率p属于类别 1，以概率$1-p$属于类别 2。

2 画出均值μ_i和方差σ_i的正态分布的随机变量，i是 1 中的类别 id，可以取 1 或者 2。

3 重复 1，2 步骤。

这个算法可以产生与观测数据相似的效果。所以选择这个算法作为模型。但是还不知道参数p和状态分布的参数。所以要学习或者推断出这些未知变量。

用Nor_0，Nor_1分别表示正态分布。两个正态分布的参数都是未知的，参数分别表示为μ_i, σ_i，$i = 0,1$。对于某一个具体的数据点来说，它可能来自Nor_0也可能来自Nor_1，假设数据点来自Nor_0的概率为p。

有一种近似的方法，可以使用 PyMC 的类别（Categorical）随机变量将数据点分配给某一类别。PyMC 类别随机变量有一个k维概率数组变量，必须对k维概率数组变量进行求和使其和变成 1，PyMC 类别随机变量的 value 属性是一个 0 到$k-1$的值，该值如何选择由概率数组中的元素决定（在本例中$k=2$）。目前还不知道将数据分配给类别 1 的先验概率是多少，所以选择 0 到 1 的均匀随机变量作为先验分布。此时输入类别变量的概率数组为$[p, 1-p]$。

```
import pymc as pm
p = pm.Uniform("p", 0, 1)
assignment = pm.Categorical("assignment", [p, 1 - p], size=data.shape[0])
print "prior assignment, with p = %.2f:" % p.value
print assignment.value[:10], "..."
```

```
prior assignment, with p = 0.31:
[1 1 0 0 0 0 1 1 1 1] ...
```

观察上面的数据集，可以猜测到两个正态分布的标准方差是不相同的。由于不知道标准方差，所以将初始方差初始化为 0 到 100 均匀分布的数值。回忆之前说道过的参数τ，它表示正态分布的精度。PyMC 需要将标准方差转化为精度，公式如下

$$\tau = \frac{1}{\sigma^2}$$

在 PyMC 中，通过如下代码就可以实现

```
taus = 1.0/pm.Uniform( "stds", 0, 100, size= 2)**2
```

参数 size=2，表示有两个 PyMC 参数τs。这种表示方法并不表示两个参数τs之间有什么关系，只是为了简洁方便才这样表示。

还需要定义类别中心值的先验分布。类别中心是正态分布参数μ。它的先验分布可以用正态分布模拟。根据目测，从数据图像中可以看出两个聚类中心大约位于 120 和 190 附件。因此设置$\mu_0 = 120$，$\mu_1 = 120$，$\sigma_{0,1} = 10$（在 PyMC 变量中$\tau = \frac{1}{\sigma^2}0.01$）。

```
taus = 1.0 / pm.Uniform("stds", 0, 100, size=2) ** 2
centers = pm.Normal("centers", [120, 190], [0.01, 0.01], size=2)
"""
The below deterministic functions map an assignment, in this case 0 or 1,
to a set of parameters, located in the (1,2) arrays `taus` and `centers`.
"""
@pm.deterministic
def center_i(assignment=assignment, centers=centers):
    return centers[assignment]
@pm.deterministic
def tau_i(assignment=assignment, taus=taus):
    return taus[assignment]
print "Random assignments: ", assignment.value[:4], "..."
print "Assigned center: ", center_i.value[:4], "..."
print "Assigned precision: ", tau_i.value[:4], "..."
```

```
Random assignments:  [1 1 0 0] ...
Assigned center:  [ 174.74278858  174.74278858  126.25164007  126.25164007] ...
Assigned precision:  [ 0.00034557  0.00034557  0.00012237  0.00012237] ...
```

```
# and to combine it with the observations:
observations = pm.Normal("obs", center_i, tau_i, value=data, observed=True)
# below we create a model class
model = pm.Model([p, assignment, taus, centers])
```

PyMC 有个 MCMC 类，MCMC 位于 PyMC 名称空间中，其实现了 MCMC 算法。用 Model 对象实例化 MCMC 类，如下

mcmc = pm.MCMC(model)

用 MCMC 的 sample(iterations)方法可以用于探究随机变量的空间，iteration 是算法执行的步数。下面将算法设置为执行 50000 步数

```
mcmc = pm.MCMC(model)
mcmc.sample(50000)
```

[****************100%******************] 50000 of 50000 complete

下面画出变量的路径（“迹”），并采用到目前为止讨论过的未知参数（中心，精度和p）。用 MCMC 对象中的方法 trace 计算路径，该方法的输入参数为 PyMC 变量 name。例如，mcmc.trace("centers")将获得 Trance 对象（可以使用[:]或者.gettrance()得到所有的路径，更方便的用法为[1000:]）。

```
figsize(12.5, 9)
plt.subplot(311)
lw = 1
center_trace = mcmc.trace("centers")[:]
# for pretty colors later in the book.
colors = ["#348ABD", "#A60628"]
if center_trace[-1, 0] > center_trace[-1, 1] \
    else ["#A60628", "#348ABD"]
plt.plot(center_trace[:, 0], label="trace of center 0", c=colors[0], lw=lw)
plt.plot(center_trace[:, 1], label="trace of center 1", c=colors[1], lw=lw)
plt.title("Traces of unknown parameters")
leg = plt.legend(loc="upper right")
leg.get_frame().set_alpha(0.7)
plt.subplot(312)
std_trace = mcmc.trace("stds")[:]
plt.plot(std_trace[:, 0], label="trace of standard deviation of cluster 0",
     c=colors[0], lw=lw)
plt.plot(std_trace[:, 1], label="trace of standard deviation of cluster 1",
     c=colors[1], lw=lw)
plt.legend(loc="upper left")
plt.subplot(313)
p_trace = mcmc.trace("p")[:]
plt.plot(p_trace, label="$p$: frequency of assignment to cluster 0",
     color="#467821", lw=lw)
plt.xlabel("Steps")
plt.ylim(0, 1)
plt.legend()
```

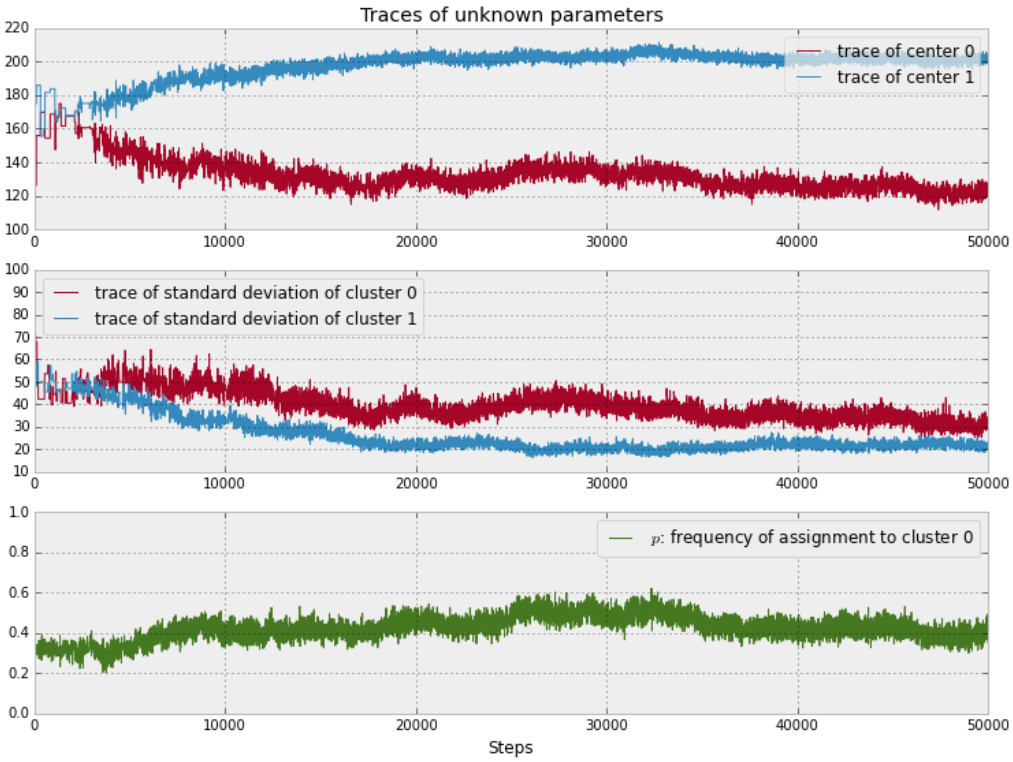


图** 未知参数的路径

注意以下特性：

1 迹并不会收敛于一个特定的点，而是会收敛到一个可能点的分布。这是 MCMC 算法中的收敛的意义，和以往的收敛意义不同。

2 使用刚开始的采样点进行推断不是一种明智的做法。因为它们与最终我们关心的分布不相关。因此在使用采样得到的样本进行推断之前要把刚开始的样本去掉。称收敛之前的那段时间为老化期。

3 迹是空间中的随机游走，它能够展示当前位置和之前位置的相关性。这既有一定的好处又有缺点。因为当前位置总是和之前位置有一定的相关性，如果相关性过多，将意味着算法没有很好的对空间中的特性进行探究。

为了达到更完美的收敛，MCMC 需要计算更多的步数。在 MCMC 算法收敛后，在收敛的基础上再次重新启动 MCMC 算法进行计算。在上面介绍的伪代码中，与下一个位置相关的只是当前位置（只在当前位置附近寻找下一个位置点），当前位置信息存储在 PyMC 变量的 value 属性中。这样就可以方便的挂起一个只在进行计算的算法并检查它的进度，稍后继续计算。value 属性值不会被覆盖。

下面将用 MCMC 算法进行成百上千次的采样，并可视化整个采样过程。

```
mcmc.sample(100000)
```

```
[****************100%******************]  100000 of 100000 complete
```

```
figsize(12.5, 4)
center_trace = mcmc.trace("centers", chain=1)[:]
prev_center_trace = mcmc.trace("centers", chain=0)[:]
x = np.arange(50000)
plt.plot(x, prev_center_trace[:, 0], label="previous trace of center 0",
    lw=lw, alpha=0.4, c=colors[1])
plt.plot(x, prev_center_trace[:, 1], label="previous trace of center 1",
    lw=lw, alpha=0.4, c=colors[0])
x = np.arange(50000, 150000)
plt.plot(x, center_trace[:, 0], label="new trace of center 0", lw=lw, c="#348ABD")
plt.plot(x, center_trace[:, 1], label="new trace of center 1", lw=lw, c="#A60628")
plt.title("Traces of unknown center parameters")
leg = plt.legend(loc="upper right")
leg.get_frame().set_alpha(0.8)
plt.xlabel("Steps")
```

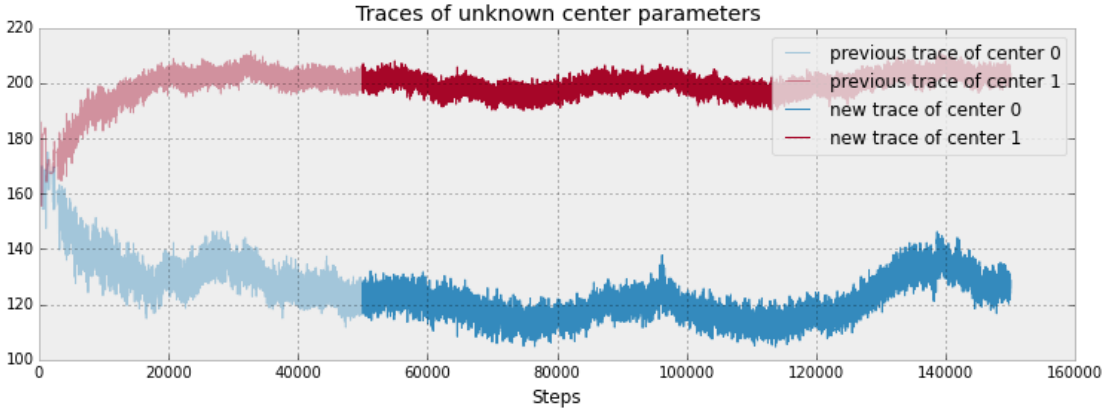


图** 中心参数的迹

MCMC 对象中的 trace 方法有个关键参数 chain，这个参数可以控制 sample 方法，让其返回你所期望的采样值（通常需要多次调用 sample，能够找回以前样本的程序将非常有用）。chain 的缺省值是-1，它将从最新的调用中返回样本值并传递给 sample。

3.4.1 对同一类别数据进行研究

最具挑战性的工作就是要找到同一类别中的数据。我们已经确定了未知参数的后验分布。下面画出聚类中心和标准方差的后验分布：

```
figsize(11.0, 4)
std_trace = mcmc.trace("stds")[:]
_i = [1, 2, 3, 0]
for i in range(2):
    plt.subplot(2, 2, _i[2 * i])
    plt.title("Posterior of center of cluster %d" % i)
    plt.hist(center_trace[:, i], color=colors[i], bins=30, histtype="stepfilled")
    plt.subplot(2, 2, _i[2 * i + 1])
    plt.title("Posterior of standard deviation of cluster %d" % i)
    plt.hist(std_trace[:, i], color=colors[i], bins=30, histtype="stepfilled")
    # plt.autoscale(tight=True)
plt.tight_layout()
```

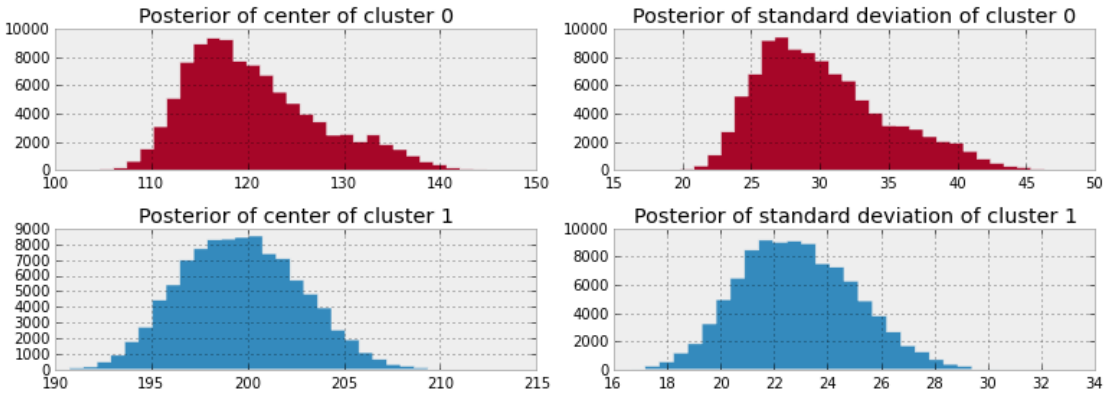


图** 聚类中心和标准方差的后验分布

MCMC 算法已经给出聚类中心最可能的地方分别在 120 和 200 处。同理，标准方差也可以有类似的推断过程。

同时也给出同一类别下的数据的后验分布，这些数据可以通过 mcmc.trace("assignment")方法得到。下图是对同一类别下数据的后验分布的可视化图。The y-axis represents a subsample of the posterior labels for each data point.x 轴表示先验数据的排序值。红色的地方表示类别 1，蓝色地方代表类别 0。

```
import matplotlib as mpl
figsize(12.5, 4.5)
plt.cmap = mpl.colors.ListedColormap(colors)
plt.imshow(mcmc.trace("assignment")[::400, np.argsort(data)],
    cmap=plt.cmap, aspect=.4, alpha=.9)
plt.xticks(np.arange(0, data.shape[0], 40),
    ["%.2f" % s for s in np.sort(data)[::40]])
plt.ylabel("posterior sample")
plt.xlabel("value of $i$th data point")
plt.title("Posterior labels of data points")
```

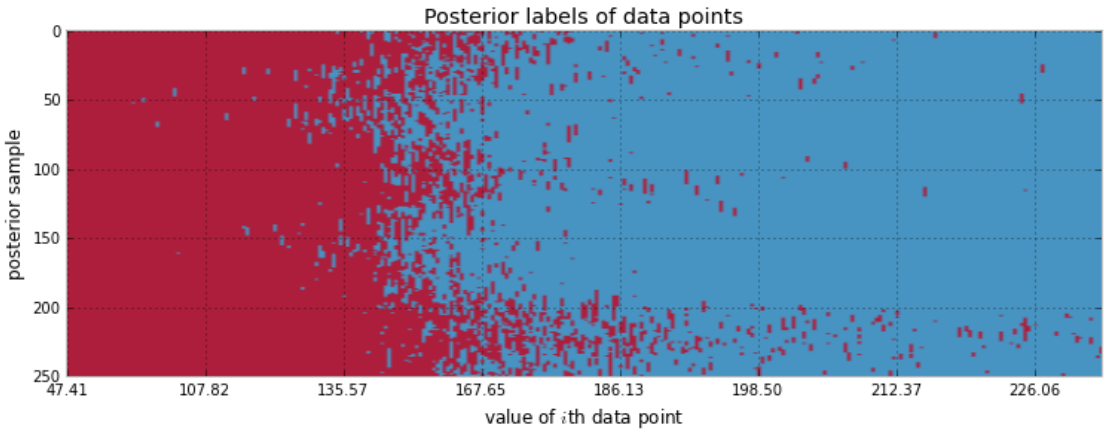


图** 数据点的后验标签

从上图中看出大多数不确定的点位于 150 到 170 之间。上图中的表示有点小问题，因为 x 轴不是真实的数据度量（它显示的是排序后第 i 个数据点的值）。更清晰的图象如下，在下图中估计出了每个数据点属于 0 和属于 1 的频率。

```
cmap = mpl.colors.LinearSegmentedColormap.from_list("BMH", colors)
assign_trace = mcmc.trace("assignment")[:]
plt.scatter(data, 1 - assign_trace.mean(axis=0), cmap=cmap,
      c=assign_trace.mean(axis=0), s=50)
plt.ylim(-0.05, 1.05)
plt.xlim(35, 300)
plt.title("Probability of data point belonging to cluster 0")
plt.ylabel("probability")
plt.xlabel("value of data point")
```

```
<matplotlib.text.Text at 0x116274490>
```

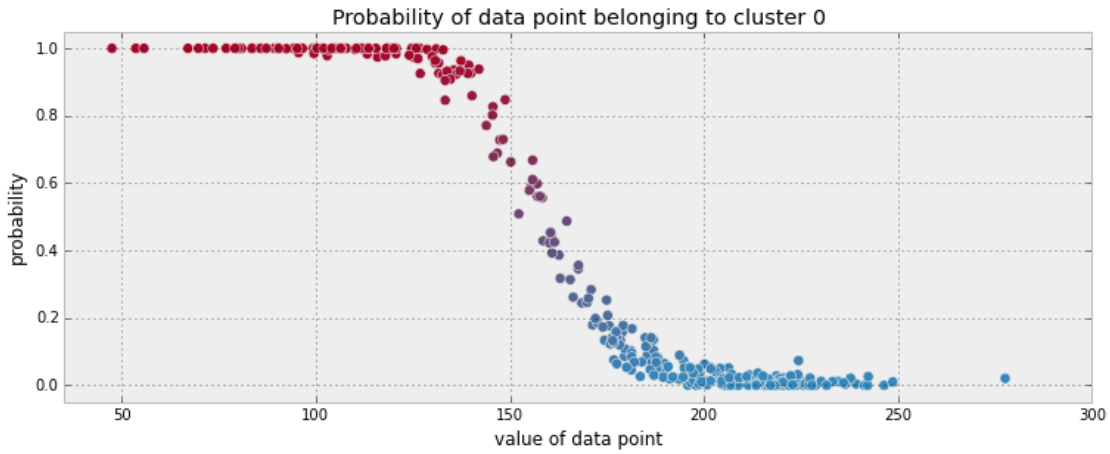
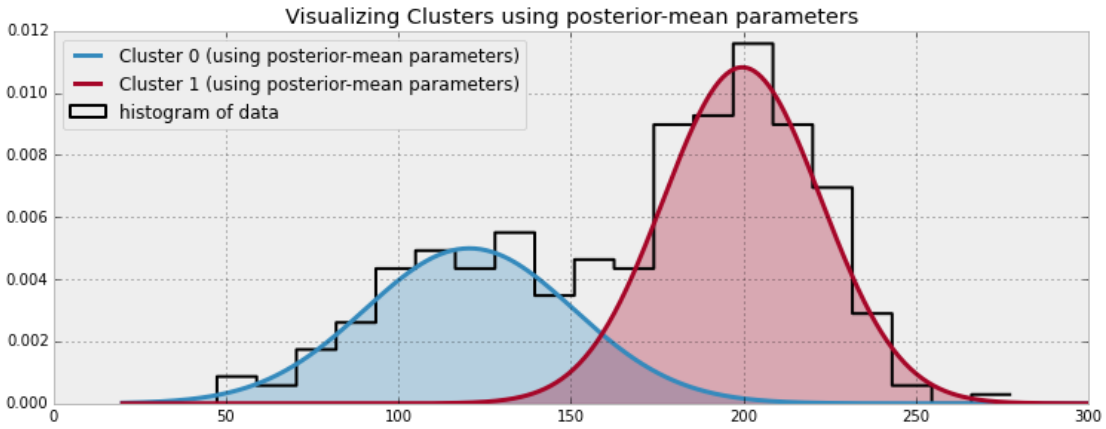


图** 每个数据点属于 0 和 1 的概率

虽然我们使用正态分布模拟每个类别中的数据，我们仍然没有得到一个能够更好的拟合数据的正态分布。如何选择两队均值和方差的值才能得到最佳的高斯拟合结果？

最佳方式是利用均值的后验结果。下面画出了用均值的后验分布加上观测数据后每个类别数据的分布情况，同时画出标准正态分布作为对比。

```
norm = stats.norm
x = np.linspace(20, 300, 500)
posterior_center_means = center_trace.mean(axis=0)
posterior_std_means = std_trace.mean(axis=0)
posterior_p_mean = mcmc.trace("p")[:].mean()
plt.hist(data, bins=20, histtype="step", normed=True, color="k",lw=2, label="histogram of
data")
y = posterior_p_mean * norm.pdf(x, loc=posterior_center_means[0],
                        scale=posterior_std_means[0])
plt.plot(x, y, label="Cluster 0 (using posterior-mean parameters)", lw=3)
plt.fill_between(x, y, color=colors[1], alpha=0.3)
y = (1 - posterior_p_mean) * norm.pdf(x, loc=posterior_center_means[1],
                          scale=posterior_std_means[1])
plt.plot(x, y, label="Cluster 1 (using posterior-mean parameters)", lw=3)
plt.fill_between(x, y, color=colors[0], alpha=0.3)
plt.legend(loc="upper left")
plt.title("Visualizing Clusters using posterior-mean parameters")
```



图**

3.4.2 不要混合后验样本

在上面的例子中类别 0 的标准方差可能很大，类别 0 的方差可能很小。即使和原始的推断有些差异，这仍然和事实相符。两种分布都具有很小的方差是不可能的，因为数据根本就不支持这种假设。因此两个标准方差相互影响：如果其中一个变小另一个会变大。事实上所有未知变量都有相似行为。例如，如果方差很大则均值分布的区域就比较大。反之，如果方差很小则均值分布在一个很小的区域中。

在 MCMC 中，返回的向量表示未知变量的后验分布样本。不同向量中的元素不能在一起使用，因为这会破坏以上的逻辑：perhaps a sample has returned that cluster 1 has a small standard deviation, hence all the other variables in that sample would incorporate that and be adjusted accordingly. It is easy to avoid this problem though, just make sure you are indexing traces correctly.

一个小例子可以说明这一点。假设两个变量x和y，它们的关系为$x + y = 10$。用均值为 4 的正态随机变量模拟x，产生 500 个样本点。

```
import pymc as pm
x = pm.Normal("x", 4, 10)
y = pm.Lambda("y", lambda x=x: 10 - x, trace=True)
ex_mcmc = pm.MCMC(pm.Model([x, y]))
ex_mcmc.sample(500)
plt.plot(ex_mcmc.trace("x")[:])
plt.plot(ex_mcmc.trace("y")[:])
plt.title("Displaying (extreme) case of dependence between unknowns")
```

```
[****************100%******************]  500 of 500 complete
<matplotlib.text.Text at 0x11620b490>
```

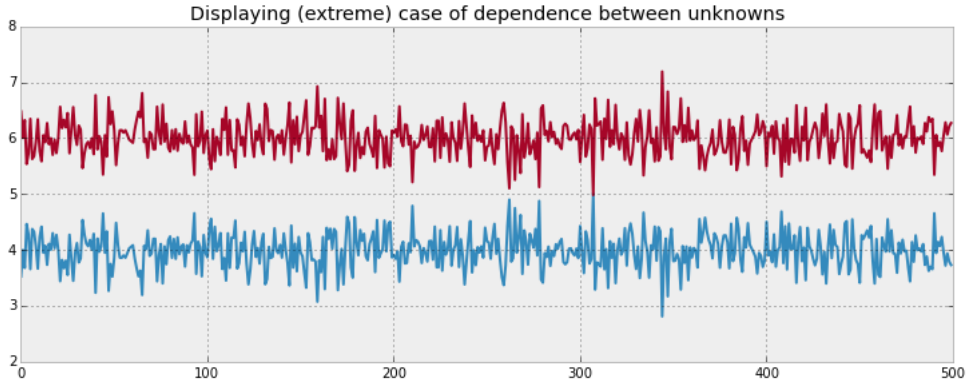


图** 演示两个随机变量之间的依赖关系

就如图所见，两个变量不相关，如果将x的第i个样本添加到y的第j个样本中将会引起错误，除非$i = j$。

返回预测的类别

以上的聚类可以生成k个类别。$k = 2$时能够更好的观测 MCMC，并可以看到一些有趣的现象。

如何预测？假设有个新的点$x = 175$，我们想知道这个点属于哪个类别。如果将这个点直接归入到最近的一个类别中，这是一种愚蠢的做法，因为这忽略了某一类别数据的标准方差。从上面的例子中可以看出，对方差的考量是非常重要的。更一般的情况是：我们更感兴趣$x = 175$这个点属于类别 1 的概率是多大。将x属于某一类别表示成L_x，我们感兴趣的是$P(L_x|x = 175)$。

有一种朴素的方法可以计算$P(L_x|x = 175)$，只要用额外的点重新计算 MCMC 即可。这个方法的缺点是，对于每个新的数据点该方法的推断速度较慢。不过可以牺牲某些精度，达到加速的目的。

下面将使用贝叶斯定理解决后验推断问题。贝叶斯定理如下

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}$$

在此时的例子中A用$L_x = 1$表示，X是观测到的数据，即$x = 175$。对于后验分布的参数$(\mu_0, \sigma_0, \mu_1, \sigma_1, p)$，我们感兴趣的是“x 属于 1 的概率是不是比 x 属于 0 的概率要大？”，概率的大小和选择的参数有关

$$P(L_x = 1|x = 175) > P(L_x = 0|x = 175)$$

$$\frac{P(x = 175|L_x = 1)P(L_x = 1)}{P(x = 175)} > \frac{P(x = 175|L_x = 0)P(L_x = 0)}{P(x = 175)}$$

因为分母都是一个样的所以可以将其忽略掉，实际上$P(x = 175)$的计算也是比较困难的。

```
norm_pdf = stats.norm.pdf
p_trace = mcmc.trace("p")[:]
x = 175
v = p_trace * norm_pdf(x, loc=center_trace[:, 0], scale=std_trace[:, 0]) > \
    (1 - p_trace) * norm_pdf(x, loc=center_trace[:, 1], scale=std_trace[:, 1])
print "Probability of belonging to cluster 1:", v.mean()
```

```
Probability of belonging to cluster 1: 0.025
```

与传统的非概率方法

L = 1 if prob > 0.5 else 0

相比，用概率表示样本属于某个类别的可能程度是很有用的。

可以使用损失函数优化猜测到的结果，第 5 章将重点介绍损失函数的求解。

3.4.3 用 MAP 改善收敛

如果运行以上的程序，将会发现每次得出的结果都不一样：也许类别的分散程度较大，也可能较小。这种结果的原因在于计算出的迹是 MCMC 算法中初值的函数。

从数学上来说可以让 MCMC 运行较长时间，计算更多步骤，这样算法就可能忘记初始数据的位置。这也就是说 MCMC 收敛意味着要计算较长的时间（在实际中虽然我们从来没有达到整体的收敛）。因此，如果观察不同后验分布的分析结果，很可能因为 MCMC 没有完全收敛，因此不能使用这些没有收敛的样本。

实际上如果初始点选择的不好，会导致无法收敛，或者是收敛速度变慢。理想情况下，我们非常希望初值在峰值附近能够连续出现，因为峰值是后验分布真正存在的地方。因此如果初值可以取在峰值附件可以避免较长的老化期（burn-in period）存在，以及避免不正确的推断出现。一般来说称“峰值”为最大后验概率或者为 MAP。

当然了，我们并不知道 MAP 的位置在哪里。如果找不到 MAP 的确切位置，PyMC 有一个对象可以近似的求得 MAP。在 MAP 对象位于 PyMC 的主命名空间中，它接受一个 PyMC 的 Model 实例作为参数。调用 MAP 实例的 fit()方法，将模型中的参数设置成 MAP 的值。

```
map_ = pm.MAP( model )
map_.fit()
```

MAP.fit()方法可以让用户方便的选择算法的优化方法，不同的优化算法产生的计算结果是不相同的。默认的优化算法是 scipy 中 fmin 方法（该方法会最小化目标函数的负值）。其它的优化算法包括 Powell 方法，该方法是 PyMC 博客的博主 Abraham Flaxman（http://healthyalgorithms.com/）的最爱，调用 Powell 方法为 fit(method='fmin_powell')。就经验来说，首先受用默认的优化算法，如果收敛效果不好或者收敛的较慢，可以尝试 Powell 方法。

MAP 还可以用于推断问题，因为它是未知参数的最可能取值。但是就像本章前一部分提及到的，推断出的结果不具有随机性，也就意味着推断出的结果不会是一个分布函数。

常常使用如下函数调用方法 MAP(model).fit()。这种间接调用 fit 方法会节省计算资源，并减少计算时间，因为老化期（burn-in period）比较短。

谈谈老化期（burn-in period）问题

It is still a good idea to provide a burn-in period, even if we are using MAP prior to calling MCMC.sample, just to be safe。PyMC 可以自动去掉前 n 个样本，在调用 sample 方法时，通过参数 burn 即可达到此目的。由于不知道什么时候数据链会收敛，通常会把样本的前半部分的数据去掉，有时候 90%的数据样本运行时间都会很长。为了继续以上的聚类问题，新代码如下

```
model = pm.Model( [p, assignment, taus, centers ] )
map_ = pm.MAP( model )
map_.fit() #stores the fitted variables' values in foo.value
mcmc = pm.MCMC( model )
mcmc.sample( 100000, 50000 )
```

3.4.4 对收敛性进行讨论

自相关性

自相关性衡量一些列的样本与其自身的相似程度。1 表示和自身完全正相关，0 表示没有自相关性，-1 表示完全负相关。如果你熟悉标准方法，自相关就是序列x_τ在时刻t和$t-k$的相关程度。

$$R(k) = Corr(x_t, x_{t-k})$$

例如有如下两个序列

$$x_t \sim \text{Normal}(0,1), x_0 = 1$$
$$y_t \sim \text{Normal}(y_{t-1}, 1), y_0 = 1$$

这两个序列的图像如下

```
figsize(12.5, 4)
import pymc as pm
x_t = pm.rnormal(0, 1, 200)
x_t[0] = 0
y_t = np.zeros(200)
for i in range(1, 200):
    y_t[i] = pm.rnormal(y_t[i - 1], 1)
plt.plot(y_t, label="$y_t$", lw=3)
plt.plot(x_t, label="$x_t$", lw=3)
plt.xlabel("time, $t$")
plt.legend()
```

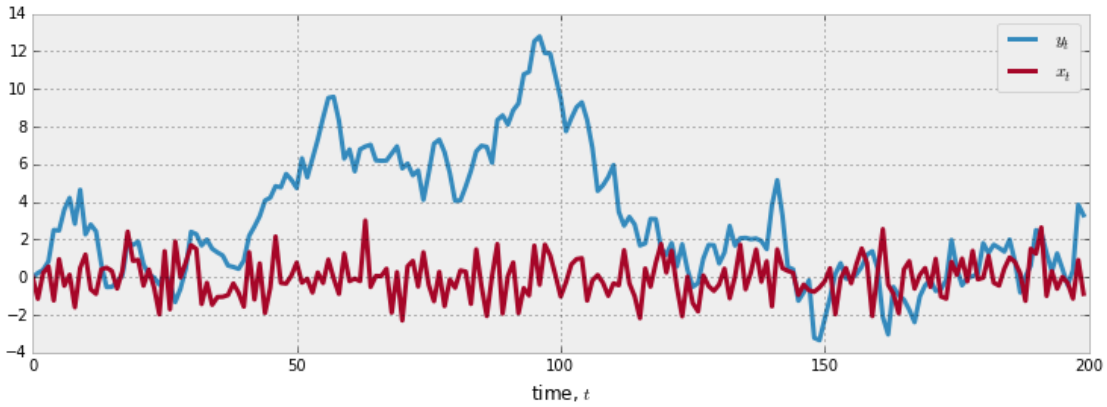


图** 序列图

可以这样来理解自相关，“如果知道在时刻s序列的位置，它能否帮助我们了解序列在时刻t时的位置在哪里”，在序列x_t中，是无法通过s时刻的序列位置判断t时刻的序列位置的。如果知道$x_2 = 0.5$，能否有利于猜测x_3的位置在哪里，答案是不能。

另一方面y_t是自相关的。如果知道$y_2 = 10$，可以很自信的说y_3离 10 的位置不会太远。对y_4可以进行类似的猜测（猜测的结果的可信度应该比y_3要小些），它不可能离 0 或者 20 比较近，离 5 近的可能性较大，对y_5的猜测的可信度又比y_4小。所以可以得出如下结论，随着k的减小，即数据点之间的间隔变小，相关性会增加。这个结论如下图

```
def autocorr(x):
    # from http://tinyurl.com/afz57c4
    result = np.correlate(x, x, mode='full')
    result = result / np.max(result)
    return result[result.size / 2:]
colors = ["#348ABD", "#A60628", "#7A68A6"]
x = np.arange(1, 200)
plt.bar(x, autocorr(y_t)[1:], width=1, label="$y_t$",
        edgecolor=colors[0], color=colors[0])
plt.bar(x, autocorr(x_t)[1:], width=1, label="$x_t$",
        color=colors[1], edgecolor=colors[1])
plt.legend(title="Autocorrelation")
plt.ylabel("measured correlation \nbetween $y_t$ and $y_{t-k}$.")
plt.xlabel("k (lag)")
plt.title("Autocorrelation plot of $y_t$ and $x_t$ for differing $k$ lags.")
```

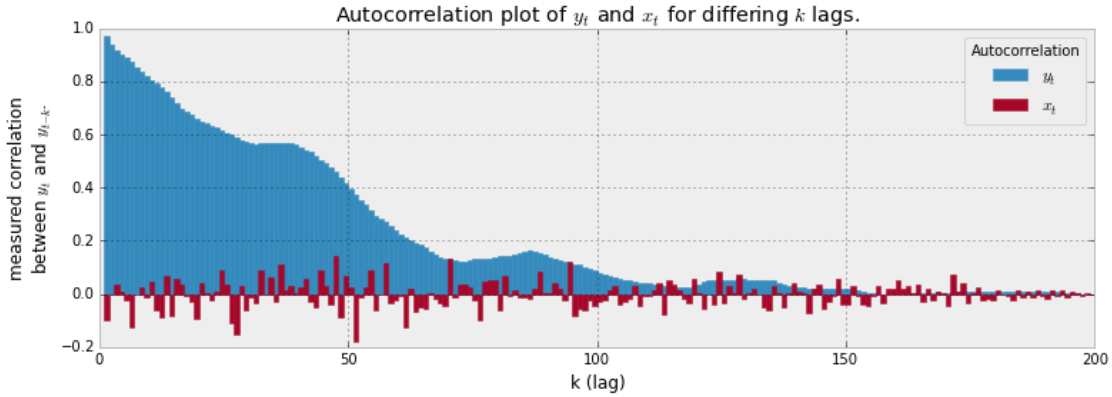


图** 不同 k 对应的x_t和y_t的自相关性

从图中可以看出，随着k的增加，y_t的自相关性从最高点快随减小。x_t的自相关性看起来就像噪声一样（实际上它就是噪声，白噪声），因此可以得出x_t没有自相关性。

自相关性和 MCMC 的收敛性有什么关系

对于 MCMC 算法来说，它总是返回具有自相关性的样本，这是因为样本序列是用一个位置移动到另一个位置产生的。

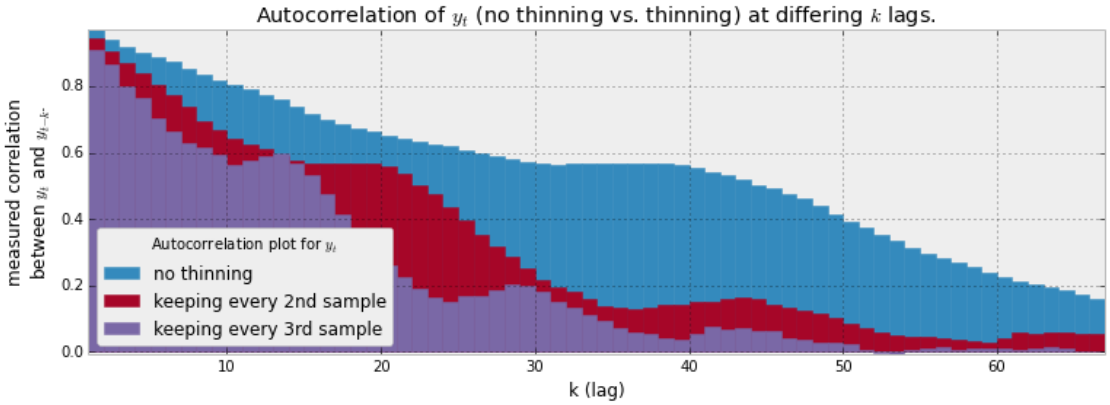
一个能够很好描述未知变量空间的样本序列应该具有很高的自相关性。从图形曲线中可以看出，如果数据样本的迹像一条永不停息流淌的河流，此时的数据序列就具有很高的自相关性。

这并不是说收敛的 MCMC 序列自相关性就比较低。因此低自相关性不是收敛的必要条件，只是充分条件。

3.4.5 对数据样本进行稀释

如果后验样本之间的自相关较高可能会导致一些问题。一些后处理算法要求样本间要相互独立。为了满足独立性要求，可以以 n 为周期从序列中抽取出样本，这会去掉一些自相关性。下图对y_t进行不同程度的稀释。

```
max_x = 200 / 3 + 1
x = np.arange(1, max_x)
plt.bar(x, autocorr(y_t)[1:max_x], edgecolor=colors[0],
      label="no thinning", color=colors[0], width=1)
plt.bar(x, autocorr(y_t[::2])[1:max_x], edgecolor=colors[1],
      label="keeping every 2nd sample", color=colors[1], width=1)
plt.bar(x, autocorr(y_t[::3])[1:max_x], width=1, edgecolor=colors[2],
      label="keeping every 3rd sample", color=colors[2])
plt.autoscale(tight=True)
plt.legend(title="Autocorrelation plot for $y_t$", loc="lower left")
plt.ylabel("measured correlation \nbetween $y_t$ and $y_{t-k}$.")
plt.xlabel("k (lag)")
plt.title("Autocorrelation of $y_t$ (no thinning vs. thinning) \
at differing $k$ lags.")
```



图**

随着稀释程度的增加，自相关性迅速减小。这里有一种折中考虑：如果稀释的很厉害，MCMC 就需要跟多的迭代次数才能得到需要的样本数。例如，10000 个样本是 100000 个样本稀释 10 倍后得到的。

稀释到什么程度是最好的？无论再做怎样的稀释，退回的样本仍然具有一些相关性。这时的稀释程度是最好的。只要自相关性趋近于零，这时可能是最好的。通常来说，不需要稀释 10 倍，数据就可以达到要求。

在调用 sample 时 PyMC 有一个稀释参数，例如

sample(10000, burn = 5000, thinning = 5)

pymc.Matplot.plot()

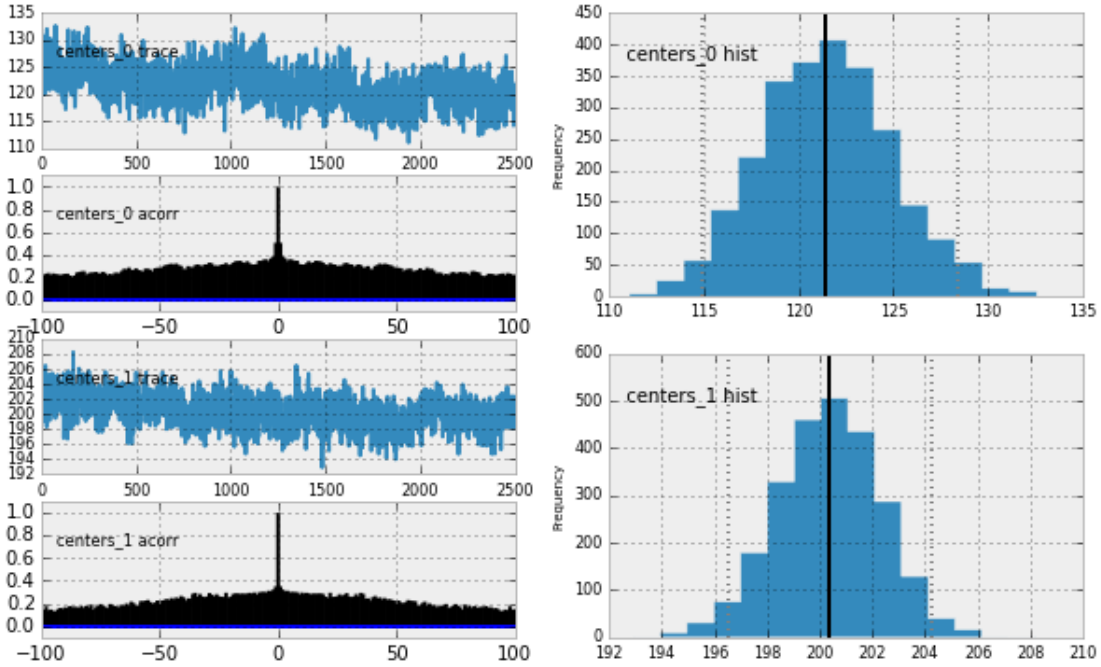
在进行 MCMC 计算时，手动的画直方图，自相关图，迹图，看起来不是聪明的办法。PyMC 中已经包括了自动可视化的工具。

pymc.Matplot 模块包括了一个 plot 函数，在导入的时候将其命名为 mcplot，这样就不会和其它命名空间冲突了。它可以接受一个 MCMC 对象，并返回每个变量的后验分布，迹，自相关系数等共十个变量。

下面用该工具画出每个聚类结果，该数据进行了 25000 次采样，稀释了 10 倍。

```
from pymc.Matplot import plot as mcplot
mcmc.sample(25000, 0, 10)
mcplot(mcmc.trace("centers", 2), common_scale=False)
```

[****************100%******************] 25000 of 25000 completePlotting centers_0
Plotting centers_1



图**

这里有两幅图，没一幅代表了每个未知的聚类中心变量。在每个图中，左上部分的图是变量的迹。图中的迹的蜿蜒变化时未收敛的结果。

右侧的图是样本的直方图，同时图上还标出了其它一些特征。黑色的竖线表示后验的均值，它是描述后验分布的关键变量。在后验分布图中两条垂直虚线之间的区间是95%的 credible interval，而不是 95% confidence interval。95%的 credible interval 表示的是参数有 95%的可能性落在这个区间。当用你的结果和其它人的结果进行比较时，这个区间的作用非常重要。学习贝叶斯方法，其中一个重要原因就是要清晰的理解未知变量的不确定性。Combined with the posterior mean, the 95% credible interval provides a reliable interval to communicate the likely location of the unknown (provided by the mean) and the uncertainty (represented by the width of the interval).

有 center_0_acorr 和 center_1_acorr 标签的图是自相关图。它和之前看到的自相关图不一样，but the only difference is that 0-lag is centered in the middle of the figure, whereas I have 0 centered to the left.

3.4.6 智能选择初值

将初始选择在后验概率附近，这样花很少的时间就可以计算出正确结果。通过 Stochastic 变量的 value 参数，告知算法我们所希望的后验分布所在的位置，可以让算法更好的计算出结果。在许多情况下我们可以为参数猜测一个合理的结果。例如，如果有数据符合正态分布，希望估计参数μ，最优的初值就是数据的均值

mu = pm.Uniform("mu", 0, 100, value = data.mean())

对于大多数的模型参数，都要对其进行一些频率估计。这些预估计的参数作为 MCMC 的初值。当然了，有些参数是无法估计出频率值的，不过还是要尽量的近似的初始值，这样有利于计算。即使你的猜测是错误的，MCMC 也会收敛到合适的分布，所以即使估计错误也没有什么损失。

这就是为什么使用 MAP 的原因，将初值赋给 MCMC 算法。所以为什么不喜欢给算法提供自定义的参数哪？给 MAP 赋以好的参数值，有助于得到最大的后验值。

另外重要的一点是，不合适的初值是 PyMC 出 bug 的主要原因，同时也不利于收敛。

先验分布

如果先验分布选择的不好，MCMC 算法可能不会收敛，至少收敛比较困难。考虑下，如果先验分布没有包含真的参数：类别 0 的先验概率将是未知的，因此属于类别 0 的后验概率也将未知。这可能会导致病态的结果。

所以要小心选择先验分布。一般来说，如果收敛性不好或者样本拥挤的分布在某一个边界，说明先验分布选择的有问题。

Covariance matrices and eliminating parameters

The Folk Theorem of Statistical Computing

If you are having computational problems, probably your model is wrong.

3.5 结论

PyMC 提供了一整套方便的工具用于贝叶斯推断，对于用户来说，它内部具有抽象的 MCMC 机制。Despite this, some care must be applied to ensure your inference is not being biased by the iterative nature of MCMC.

3.6 References

Flaxman, Abraham. "Powell's Methods for Maximization in PyMC." Healthy Algorithms. N.p., 9 02 2012. Web. 28 Feb 2013. http://healthyalgorithms.com/2012/02/09/powells-method-for-maximization-in-pymc/.

```
from IPython.core.display import HTML
def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

第 4 章 大数定理

本章所介绍的定理几乎在每本统计学的书上都会介绍，却很少将其用到实际的统计应用中。实际上到目前为止，在每个例子中都使用了这个简单的思想。

4.1 大数定理

设Z_i为服从特定分布的N个独立的样本，按照大数定理理论，只要Z的期望$E[Z]$是有限的，则就有如下关系

$$\frac{1}{N}\sum_{i=1}^{N} Z_i \to E[Z], N \to \infty \tag{4-1}$$

式（4-1）所要表达的意思是：来自于同一分布的随机变量序列的平均值收敛到该分布的期望值。

大数定理看似一点用处都没有，实际上它是最有用的工具。

直觉认识

如果觉得大数定理有点不可思议，可以通过一个例子看清它的面目。

设随机变量Z，只有两个取值c_1和c_2。假设有大量Z的样本值，每个样本用Z_i表示。大数定理表明可以用Z样本值的平均值近似Z的期望值。平均值的表达式如下

$$\frac{1}{N}\sum_{i=1}^{N} Z_i \tag{4-2}$$

由于Z_i，只能取c_1和c_2两个值，因此可以将求和公式拆分成如下两部分

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N} Z_i &= \frac{1}{N}\left(\sum_{Z_i=c_1} c_1 + \sum_{Z_i=c_2} c_2\right) = c_1 \sum_{Z_i=c_1}\frac{1}{N} + c_2 \sum_{Z_i=c_2}\frac{1}{N} \\ &= c_1 \times c_1\text{的近似频率} + c_2 \times c_2\text{的近似频率} \\ &\approx c_1 \times P(Z=c_1) + c_2 \times P(Z=c_2) \\ &= E[Z]\end{aligned} \tag{4-2}$$

以上等式只有在极限条件下成立，但是随着参加平均计算的样本越来越多，均值将越来越接近Z的期望值$E[Z]$。除了一些极少数的特殊情况外，大数定理对于大多数分布都是正确的。

4.1.1 大数定理计算均值

下面是三个不同的泊松随机变量在大数定理上的实例，用三个不同的曲线表示。

泊松分布的参数$\lambda = 4.5$，样本个数 sample_size=100000 个（泊松分布的均值等于其参数λ）。计算$n = 1$到 sample_size 时样本的均值，脚本如下

```
%matplotlib inline
import numpy as np
from IPython.core.pylabtools import figsize
import matplotlib.pyplot as plt
figsize(12.5, 5)
import pymc as pm
sample_size = 100000
expected_value = lambda_ = 4.5
poi = pm.rpoisson
N_samples = range(1, sample_size, 100)
for k in range(3):
    samples = poi(lambda_, size=sample_size)
    partial_average = [samples[:i].mean() for i in N_samples]
    plt.plot(N_samples, partial_average, lw=1.5, label="average \
of  $n$ samples; seq. %d" % k)
plt.plot(N_samples, expected_value * np.ones_like(partial_average),
         ls="--", label="true expected value", c="k")
plt.ylim(4.35, 4.65)
plt.title("Convergence of the average of \n random variables to its \
expected value")
plt.ylabel("average of $n$ samples")
plt.xlabel("# of samples, $n$")
plt.legend();
```

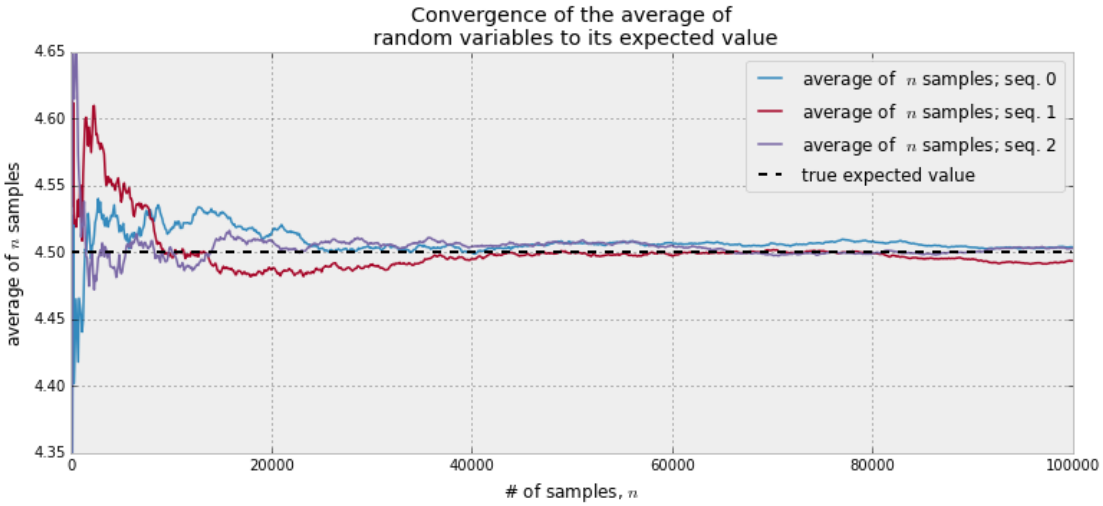


图 4.1 三个泊松分布的随机变量大数定理曲线

从图 4.1 看出，当样本个数较少时，样本平均值和实际的期望之间有很大误差（在样本数少时，平均值曲线呈锯齿状，且剧烈波动，随着样本数量增加，均值曲线变的平发滑多了）。从图 4.1 看出三条曲线最终都到达了值 4.5 附近，随着 N 变大曲线在 4.5 附近抖动的幅度越来越小。数学家或者统计学家将这种抖动逐步减小的现象称为收敛（convergence）。

另一个问题是关于收敛速度的，样本的均值最快能以什么样的速度收敛到随机变量的期望？下面介绍如何度量收敛速度。假设有N_Y组样本，每组样本个数为N，计算每组样本均值和实际期望，即 4.5，之间的距离，然后对所有组样本产生的距离再次取平均。这里又用到了大数定理。按照以上介绍，对于一个具体的N，样本平均值和真实期望之间距离的期望可以表示成如下形式

$$D(N) = \sqrt{E\left[\left(\frac{1}{N}\sum_{i=1}^{N} Z_i - 4.5\right)^2\right]} \tag{4-3}$$

上式是求取期望的标准定义，它可以用大数定理进行近似求解。首先看如下等式

$$Y_k = \left(\frac{1}{N}\sum_{i=1}^{N} Z_i - 4.5\right)^2 \tag{4-4}$$

式（4-4）表示第k组N个样本的均值与真实期望的距离，因为共有N_Y组样本所以要计算N_Y次，那就得到N_Y个Y_k值，因为Y_k表示的是随机变量，对Y_k进行平均，如下

$$\frac{1}{N_Y}\sum_{k=1}^{N_Y} Y_k \rightarrow E[Y_k] = E\left[\left(\frac{1}{N}\sum_{i=1}^{N} Z_i - 4.5\right)^2\right] \tag{4-5}$$

均方根为

$$\sqrt{\frac{1}{N_Y}\sum_{k=1}^{N_Y} Y_k} \approx D(N) \tag{4-6}$$

```
figsize(12.5, 4)
N_Y = 250  # use this many to approximate D(N)
N_array = np.arange(1000, 50000, 2500)  # use this many samples in the approx. to the variance.
D_N_results = np.zeros(len(N_array))
lambda_ = 4.5
expected_value = lambda_  # for X ~ Poi(lambda) , E[ X ] = lambda
def D_N(n):
    """
    This function approx. D_n, the average variance of using n samples.
    """
    Z = poi(lambda_, size=(n, N_Y))
    average_Z = Z.mean(axis=0)
    return np.sqrt(((average_Z - expected_value) ** 2).mean())
for i, n in enumerate(N_array):
    D_N_results[i] = D_N(n)
plt.xlabel("$N$")
plt.ylabel("expected squared-distance from true value")
plt.plot(N_array, D_N_results, lw=3,
         label="expected distance between\n\
expected value and \naverage of $N$ random variables.")
plt.plot(N_array, np.sqrt(expected_value) / np.sqrt(N_array), lw=2, ls="--",
         label=r"$\frac{\sqrt{\lambda}}{\sqrt{N}}$")
plt.legend()
plt.title("How 'fast' is the sample average converging? ");
```

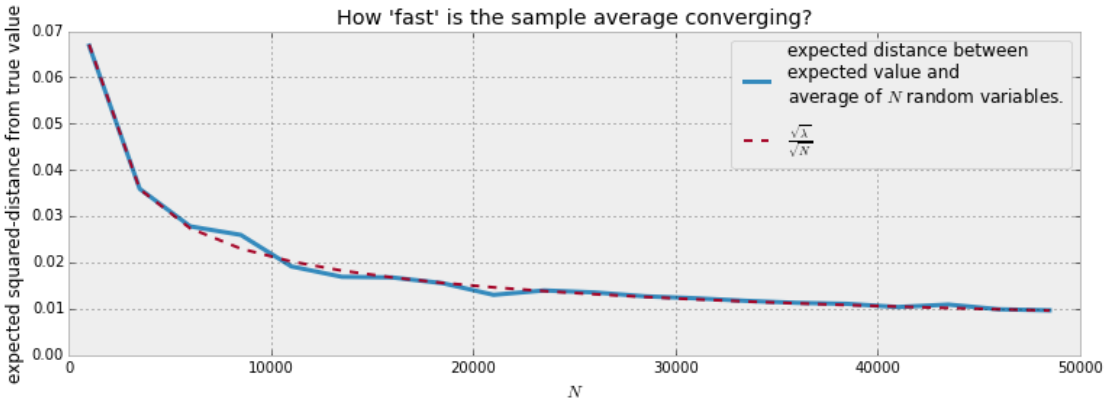


图 4.2 样本均值收敛到真实值（期望）的速度曲线

从图 4.2 中的结果和我们期望的结果差不多，随着N的增大，样本均值和实际期望值（总体分布的期望）之间的距离在减小。同时也可以从图中看到样本点从 10000 增加到 20000 时，均值和期望的距离从 0.020 降低到 0.015，减少了 0.005。当样本点从 20000 增加到 40000 时，样本均值和期望之间的距离从 0.015 降低到 0.010，仅仅减少了 0.005。说明收敛的速率在逐渐变小。

图 4.2 中的虚线表示不同N时函数$\frac{\sqrt{\lambda}}{\sqrt{N}}$值。这个计算公式不是随意选取的。在大多数情况下一个随机变量序列，假设用Z代替，按照大数定理计算$E[Z]$时的收敛速率公式为

$$\frac{\sqrt{Var(Z)}}{\sqrt{N}} \tag{4.7}$$

公式（4.7）很重要，因为对于一个给定的N，就可以知道估计出的样本平均值和真实的期望值之间的误差大小。但在贝叶斯框架看来，式（4.7）没有多大用处，因为贝叶斯分析方法本来就允许不确定性存在，它们所关心的是加入额外的精确数据后统计量如何变化。

4.1.2 如何计算$Var(Z)$

方差也是可以近似计算的。一旦知道了随机变量的期望值（通过大数定理估计的近似值，用μ表示），则方差可以使用如下公式表示

$$\frac{1}{N}\sum_{i=1}^{N}(Z_i-\mu)^2 \rightarrow E[(Z_i-\mu)^2] = Var(Z) \tag{4.8}$$

4.1.3 期望值和概率

在期望值和估计的概率之间存在一种隐性关系。

首先定义如下函数

$$1_A(x) = \begin{cases} 1 & x \epsilon A \\ 0 & else \end{cases} \tag{4.9}$$

根据大数定理，如果有样本X_i，可以估计出事件 A 的概率，用$P(A)$表示

$$\frac{1}{N}\sum_{i=1}^{N} 1_A(X_i) \rightarrow E[1_A(X)] = P(A) \tag{4.10}$$

式（4.10）显然成立，由式（4.9）知只有事件发生时$1_A(x)$才取 1 其它情况都取 0，所以只要用试验中事件发生的总次数除以试验的次数就是概率（回想一下平时我们是怎么样用频率近似概率的）。例如，假设有随机变量Z，满足分布$Z \sim Exp(0.5)$，要求Z大于 10 的概率，如果有N个样本服从$Exp(0.5)$分布，则

$$\mathrm{P}(Z > 10) = \frac{1}{N}\sum_{i=1}^{N} 1_{z>10}(Z_i) \tag{4.11}$$

```
import pymc as pm
N = 10000
print np.mean([pm.rexponential(0.5) > 10 for i in range(N)])
```

0.0069

4.1.4 贝叶斯方法的处理方式

下一章将介绍，用期望值计算贝叶斯推断中的点估计。在大多数的贝叶斯推断中，要经过多维积分计算复杂的期望值。但如果能够从后验概率中采样出样本，就可以通过对样本进行平均计算出均值。如果对均值的精度要求较高，可以画出类似图 4.2 的收敛速率图，从中判断出需要选取的N的大小，以控制合适的精度。N 越大样本的均值精度将会越高。

样本的数量如何选择？如何从后验概率中选择合适数量的样本？这就要根据实际需要决定了，同时还和样本的方差有关（高方差的数据，在计算样本均值时收敛较慢）。

我们要知道大数定理在什么情况下回失效。大数定理字如其名，只有在大样本量时才会有效，图 4.2 中已经可以看出 N 较小时计算的平均值和实际期望值差距较大。如果样本数据量不大，则无法得到图 4.2 所示的渐近线结果。了解大数定理的失效条件有利于我们判断用大数定理计算出的结果是否可信（Knowing in what situations the Law fails can give us confidence in how unconfident we should be）。下一章将介绍这部分内容。

4.2 混乱的“小数”

只有当 N 为无穷大时，大数定理的结果才是有效的，这样的条件是无法达到的。虽然大数定理是一个有效的工具，但如果不加选择的使用它，结果则会欲速则不达。下面的例子将会介绍这种情况。

4.2.1 地理学数据分析

有些数据会以某一属性进行分组。例如有些数据按照国家，县区或者城市进行分组。当然人口数量也会根据地理区域的不同而不同。如果要对某一地理区域中的一些特征进行统计平均时，必须要注意大数定理适用范围，并了解在人口数较少时它是如何失效的。

下面将用一个“玩具类型”的数据集介绍如何正确的使用大数定理。假设在数据集中包含有 5000 个县的数据。除此之外人口数量在整个国家的分布符合 100 到 1500 的均匀分布。至于人口数量是如何生成的问题和这里所要讨论的问题无关，所以不加以述及。我们关心的是每个县中人口的平均高度。我们并不知道，在每个县中的每个人的高度分布都是一样的，且并不会随着县的变化而改变，高度分布为

$$height \sim Normal(150,15) \tag{4.12}$$

把数据按照县级别进行分组，所以只能对每个县里的数据做平均。数据如下图

```
std_height = 15
mean_height = 150
n_counties = 5000
pop_generator = pm.rdiscrete_uniform
norm = pm.rnormal
# generate some artificial population numbers
population = pop_generator(100, 1500, size=n_counties)
average_across_county = np.zeros(n_counties)
for i in range(n_counties):
    # generate some individuals and take the mean
    average_across_county[i] = norm(mean_height, 1. / std_height ** 2,
                                    size=population[i]).mean()
# located the counties with the apparently most extreme average heights.
i_min = np.argmin(average_across_county)
i_max = np.argmax(average_across_county)
# plot population size vs. recorded average
plt.scatter(population, average_across_county, alpha=0.5, c="#7A68A6")
plt.scatter([population[i_min], population[i_max]],
            [average_across_county[i_min], average_across_county[i_max]],
            s=60, marker="o", facecolors="none",
            edgecolors="#A60628", linewidths=1.5,
            label="extreme heights")
plt.xlim(100, 1500)
plt.title("Average height vs. County Population")
plt.xlabel("County Population")
plt.ylabel("Average height in county")
plt.plot([100, 1500], [150, 150], color="k", label="true expected \
height", ls="--")
plt.legend(scatterpoints=1);
```

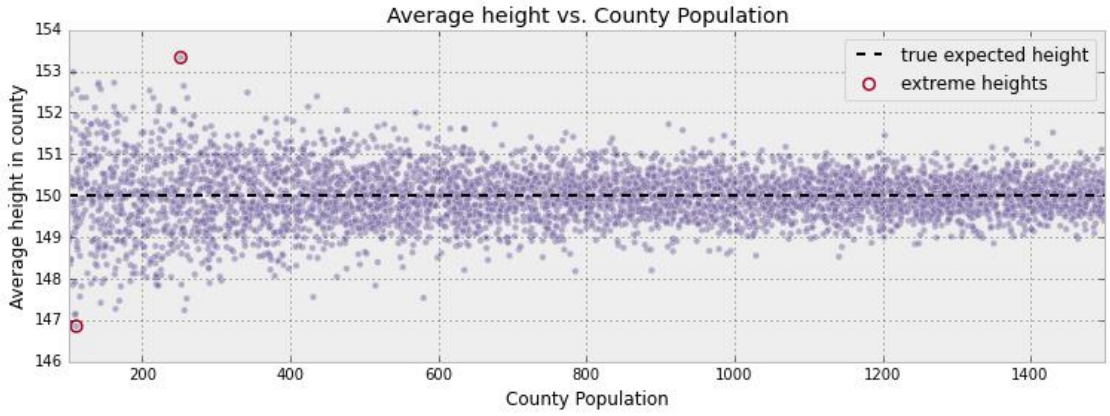


图 4.3 每个县的人口数量所对应的人口真实平均高度和个体高度对比图

从上图中可以观察到什么现象？如果不考虑人口数量大小，推断出的结果就可能存在错误，例如，如果不考虑人口数大小，我们可能会说有最高身高人口的县也有最低身高的人口，反之亦然（不考虑人口数量，图 4.3 中的 x 轴就变成一个点了，红色圆圈的点就会在一条竖线上了，所以就反映出最高人口的县也会有最低人口，最低人口的县也会有最高人口）。但是这种推断结果时不对的。因为这两个县不一定有最极端高度的人存在。错误的原因是人口数量少时计算的均值并不能反映出人口真实的期望值（真实是 $\mu = 150$）。样本的数量 N 越小，大数定理就会失效的越厉害。

下面提供一个更能说明大数定理失效的例子。上面说到人口数量均匀分布在 100 到 1500 之间。直觉告诉我们，拥有最高身高和最低身高人口县的分布也符合 100 到 1500 的均匀分布，且和县的人口数量无关。下面是拥有极端身高人群的县所具有的人口数量。

```
print "Population sizes of 10 'shortest' counties: "
print population[np.argsort(average_across_county)[:10]]
print
print "Population sizes of 10 'tallest' counties: "
print population[np.argsort(-average_across_county)[:10]]
```

```
Population sizes of 10 'shortest' counties:
[111 103 102 109 110 257 164 144 169 260]

Population sizes of 10 'tallest' counties:
[252 107 162 141 141 256 144 112 210 342]
```

从这些数字看出，县的分布根本不是 100 到 1500 的均匀分布。这完全是大数定理的错误。

4.2.2 调整 Kaggle 的人口普查回报率

下面是 2010 年美国人口普查数据，它不是按照县对数据分区，而是按照街区进行分区。本例中的数据来自于 Kaggle 机器学习竞赛中使用的数据集。竞赛的目的是为了预测各个街区对人口普查邮件的回复率，回复率的取值范围为 0 到 100，使用的预测特征包括：人口平均收入，女性数量，停车场的个数，小孩的平均个数等。下面画出人口调查邮件的回复率和街区中人口数量的关系图。

```
figsize(12.5, 6.5)
data = np.genfromtxt("./data/census_data.csv", skip_header=1,delimiter=",")
plt.scatter(data[:, 1], data[:, 0], alpha=0.5, c="#7A68A6")
plt.title("Census mail-back rate vs Population")
plt.ylabel("Mail-back rate")
plt.xlabel("population of block-group")
plt.xlim(-100, 15e3)
plt.ylim(-5, 105)
i_min = np.argmin(data[:, 0])
i_max = np.argmax(data[:, 0])
plt.scatter([data[i_min, 1], data[i_max, 1]],
            [data[i_min, 0], data[i_max, 0]],
            s=60, marker="o", facecolors="none",
            edgecolors="#A60628", linewidths=1.5,
            label="most extreme points")
plt.legend(scatterpoints=1);
```

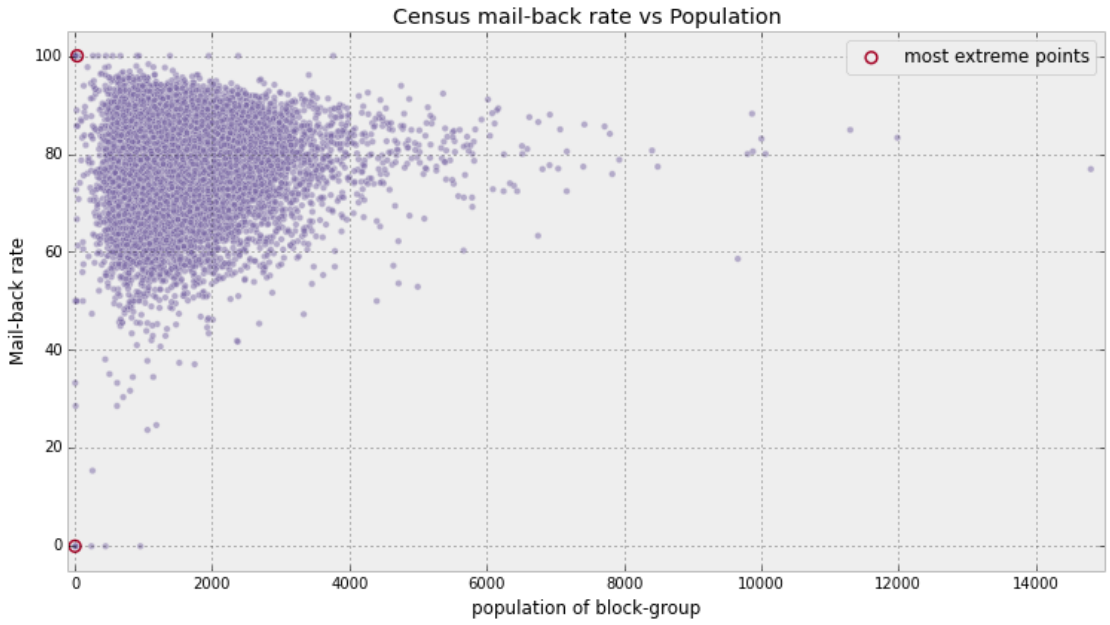


图 4.4 人口普查邮件的回复率和人口数量的关系

上图反映了典型的统计现象。这里的典型指的是散点图的形状。散点图是一个典型的三角形，随着样本数量的增加，这个三角形变得越紧实（因为越来越符合大数定理的要求）。

也许在此过分强调了数据量的问题，也许把这笔数起个名字叫“小数据所存在的问题”，不过该例中的数据确实又是一个小数据集的情况。简单的说小数据无法使用大数定理。可以毫不犹豫的将大数定理应用到大数据上。这和早前提及的处理大数据问题可以使用相对简单的算法求解看似自相矛盾。其实不然，我们要理解的是大数定理的解是稳定的，增加或者减少数据点的时候并不会影响大数定理的求解结果。另一方面，如果向一个小的数据集中添加一些数据或者从中减少一些数据，两种结果就会非常不一样。

还有一些大数定理使用时的误区，可以阅读《The Most Dangerous Equation》。

4.2.3 如何对红迪网的评论排序

也许你不会同意本章刚开始就谈到的，任何人都知道大数定理这一说法，而只是认为大数定理只是一种潜意识的决策而已。考虑在线商品的评分问题：你如何评价只有一个人，两个人或者三个人对一个商品的五星级评价结果。我们多少都会明白，这样少的评分结果无法真正的反映出商品的真实价值。

诸如这种评价信息部充分的情况给排序带来了困难。很多人都知道在线搜索图书，视频时，如果用它们的评分或者评论对返回结果进行排序，最终用户得到的结果并不会很好。通常情况下搜索结果中排在最前面的视频或评论往往有很好的评分，而这些好的评分都是由粉丝评出的。还有一些真正高质量的视频或者评论排在了稍微靠后的位置，它们的评分在 4.8 左右。我们如何对这种情况进行修正。

红迪网是一个流行的网站。它主要提供一些小说或者图片的链接，这个网站最流行的部分是它对每个链接的评论。红迪网可以对每个评论进行投票（被称为赞成票或否决票）。默认状态下红迪网将把最好的评论排在前面。

图 4.5 红迪网的评论

如何确定哪一条评论是最好的？有如下方式对评论进行排序：

1 流行度：如果一个评论有很多赞成票，则这个评论就是好的。这种评价方法的问题在于，如果一个评论得到了上百个赞成票，而同时又有上千个反对票。虽然这个评论很流行，但它却不是最好的评论（差评论也可以很流行）。

2 差异度：使用赞成票和反对票之间的差异对结果进行排序。这解决了 1 中存在的问题。但是无法处理评论所存在的时间长短问题。当一个新的链接建立以后的很长时间内都可以对其进行评论。用差异度排序时可能会将存在时间最长的评论排在前面，因为与新的评论相比旧的评论积累了更多的赞成票。

3 用时间调节排序：用 2 中的差异度除以评论所存在的时间。结果是一个比率，可以用每秒差异度或者每分钟差异度表示。例如，如果使用每秒差异度作为度量标准，一秒内就有一个好评比 100 秒有 99 个好评的排名要靠前。可以选择一个时间阈值，只有大于此时间阈值的评论采用每秒差异度来度量它的排名。但是多大的时间阈值才是最佳的？这是否意味着只要评论存在的时间大于时间阈值就是好的评论哪？不要用新的评论和旧的评论进行对比。

4 好评率：用对某一评论的好评数除以对该评论的好评和差评的总和，得到好评率，作为排名权重。这解决了 3 中的问题，如果用好评率作为排名结果，即使一个新的评论也会和老的评论一样得到较高的排名得分。在这种度量标准下，如果对一个评论的好评个数只有一个，即好评率为 1.0，而另一个评论的好评个数为 999，差评个数为 1，此时这个评论的好评率为 0.999，显然后一个评论的质量要比前一个高（因为可能前一个的评论是随机的，那 1.0 也是随机的）。

不过只有一次好评的评论有可能比有 999 次好评的评论要好。因为我们还有 999 次投票结果没看到。这 999 次的投票中也许都可能是好评，此时只有一次好评的评论就比 999 次好评的评论好，虽然这种可能性很小。

我们真正目标就是估计真实的好评率。注意，真实的好评率并不是观测到的好频率，真实的好评率隐藏起来了，我们能观测到的仅仅是每个评论的好评和差评。当有 999 个好评，1 个差评时，我们就可以说真实的好评率接近于 1，我们能这样说的原因是因为有大数定理的存在，反之，当只有一次好评时，我们无法断言真实的好评率就是 1。这听起来像个贝叶斯问题。

可以用好评率的历史分布作为定好评率的先验分布。只要对红迪网的评论进行挖掘就可以得到好评率的历史分布。计算好评率的历史分布时还应注意以下问题

1 倾斜数据：大部分的评论都没有投票数据，因此有许多评论的好评率都非常小（极端值），这就使分布大多数集中在极端值附件。可以为投票的个数设个阈值，大于该阈值的情况才予以考虑。阈值的选择还要考虑其对好评率精度的影响。

2 偏差数据：红迪网有许多不同的子页面。例如有两个页面，第一个为：r/aww 页面粘贴许多可爱动物的照片，第二个为：r/politics。用户对这两个页面的评论行为是不相同的。浏览第一个页面的人的性格会更友好，更有爱心，也就会有更多的好评（与第二个页面相比），而第二个页面的评论可能会有更多的争论或不同的观点出现。因此并不是所有的评论都是一致的。

鉴于以上的分析，我认为使用均匀分布先验分布比较好。

用均匀分布作为先验，可以得到真实好评率的后验分布。Python 脚本 comments_for_top_reddit_pic.py 将从红迪网的当前排名靠前的图片上得到评论信息。下面是图片及其评论信息。

```
from IPython.core.display import Image
# adding a number to the end of the %run call with get the ith top photo.
%run top_pic_comments.py 2

Image(top_post_url)
```

```
Title of submission:
Frozen mining truck
http://i.imgur.com/OYsHKlH.jpg

<IPython.core.display.Image at 0x1077eb850>
```

```
"""
contents: an array of the text from all comments on the pic
votes: a 2d numpy array of upvotes, downvotes for each comment.
"""
n_comments = len(contents)
comments = np.random.randint(n_comments, size=4)
print "Some Comments (out of %d total) \n----------" % n_comments
for i in comments:
    print "" + contents[i] + ""
    print"upvotes/downvotes: ", votes[i, :]
    print
```

```
Some Comments (out of 77 total)
----------
"Do these trucks remind anyone else of Sly Cooper?"
upvotes/downvotes:  [2 0]

"Dammit Elsa I told you not to drink and drive."
upvotes/downvotes:  [7 0]

"I've seen this picture before in a Duratray (the dump box supplier) brochure. If I recall it was either at Ekati or Diavik… In which case the truck could be either a Komatsu or a CAT… anyone care to comment?"
upvotes/downvotes:  [2 0]

"Actually it does not look frozen just covered in a layer of wind packed snow."
```

upvotes/downvotes: [120 18]

假设共有N个投票数，好评率为p，好评的个数就是二项随机变量，参数为p和N。构造一个函数对p进行贝叶斯推断。

```
import pymc as pm
def posterior_upvote_ratio(upvotes, downvotes, samples=20000):
    """
    This function accepts the number of upvotes and downvotes a particular comment received,
    and the number of posterior samples to return to the user. Assumes a uniform prior.
    """
    N = upvotes + downvotes
    upvote_ratio = pm.Uniform("upvote_ratio", 0, 1)
    observations = pm.Binomial("obs", N, upvote_ratio, value=upvotes, observed=True)
    # do the fitting; first do a MAP as it is cheap and useful.
    map_ = pm.MAP([upvote_ratio, observations]).fit()
    mcmc = pm.MCMC([upvote_ratio, observations])
    mcmc.sample(samples, samples / 4)
    return mcmc.trace("upvote_ratio")[:]
```

下面是后验分布结果

```
figsize(11., 8)
posteriors = []
colours = ["#348ABD", "#A60628", "#7A68A6", "#467821", "#CF4457"]
for i in range(len(comments)):
    j = comments[i]
    posteriors.append(posterior_upvote_ratio(votes[j, 0], votes[j, 1]))
    plt.hist(posteriors[i], bins=18, normed=True, alpha=.9,
             histtype="step", color=colours[i % 5], lw=3,
             label='(%d up:%d down)\n%s...' % (votes[j, 0], votes[j, 1], contents[j][:50]))
    plt.hist(posteriors[i], bins=18, normed=True, alpha=.2,
             histtype="stepfilled", color=colours[i], lw=3, )

plt.legend(loc="upper left")
plt.xlim(0, 1)
plt.title("Posterior distributions of upvote ratios on different comments");
```

```
[**********100%**********]  20000 of 20000 complete
```

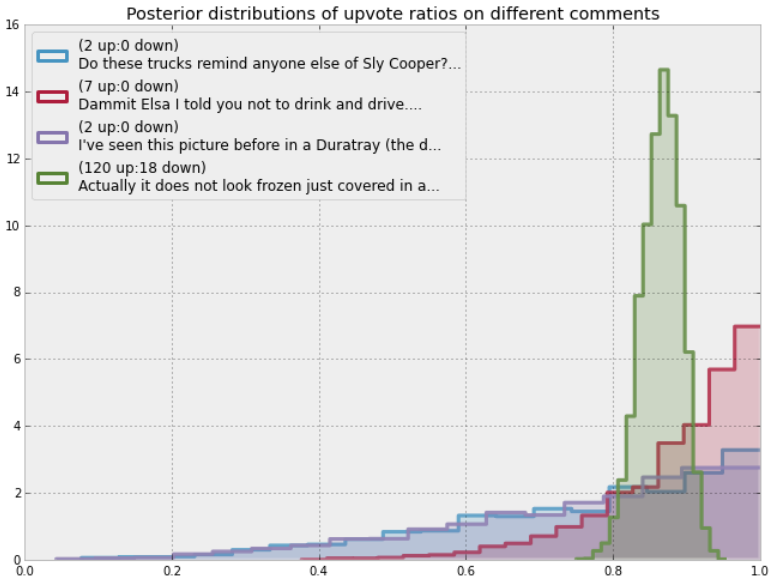


图 4.6 不同评论的后验分布

从图中可以看出，有些分布很陡峭，其它分布有较长的长尾，长尾描述了真实的好评率的不确定性。

排序

这个例子的目的就是要对评论从好到差进行排序。当然了我们不能对分布进行排序。只能对标量数字进行排序。有很多方式从分布中提取出标量信息，有一种方法是通过期望或者均值表示分布。选择用均值表示分布是一种不太好的选择。这是因为均值并没有考虑分布的不确定性。

I suggest using the 95% least plausible value, defined as the value such that there is only a 5% chance the true parameter is lower (think of the lower bound on the 95% credible region). Below are the posterior distributions with the 95% least-plausible value plotted:

```
N = posteriors[0].shape[0]
lower_limits = []

for i in range(len(comments)):
    j = comments[i]
    plt.hist(posteriors[i], bins=20, normed=True, alpha=.9,
             histtype="step", color=colours[i], lw=3,
             label='(%d up:%d down)\n%s...' % (votes[j, 0], votes[j, 1], contents[j][:50]))
    plt.hist(posteriors[i], bins=20, normed=True, alpha=.2,
             histtype="stepfilled", color=colours[i], lw=3, )
    v = np.sort(posteriors[i])[int(0.05 * N)]
    # plt.vlines( v, 0, 15 , color = "k", alpha = 1, linewidths=3 )
    plt.vlines(v, 0, 10, color=colours[i], linestyles="--", linewidths=3)
    lower_limits.append(v)
    plt.legend(loc="upper left")

plt.legend(loc="upper left")
plt.title("Posterior distributions of upvote ratios on different comments");
order = np.argsort(-np.array(lower_limits))
print order, lower_limits
```

[3 1 2 0] [0.36980613417267094, 0.68407203257290061, 0.37551825562169117, 0.8177566237850703]

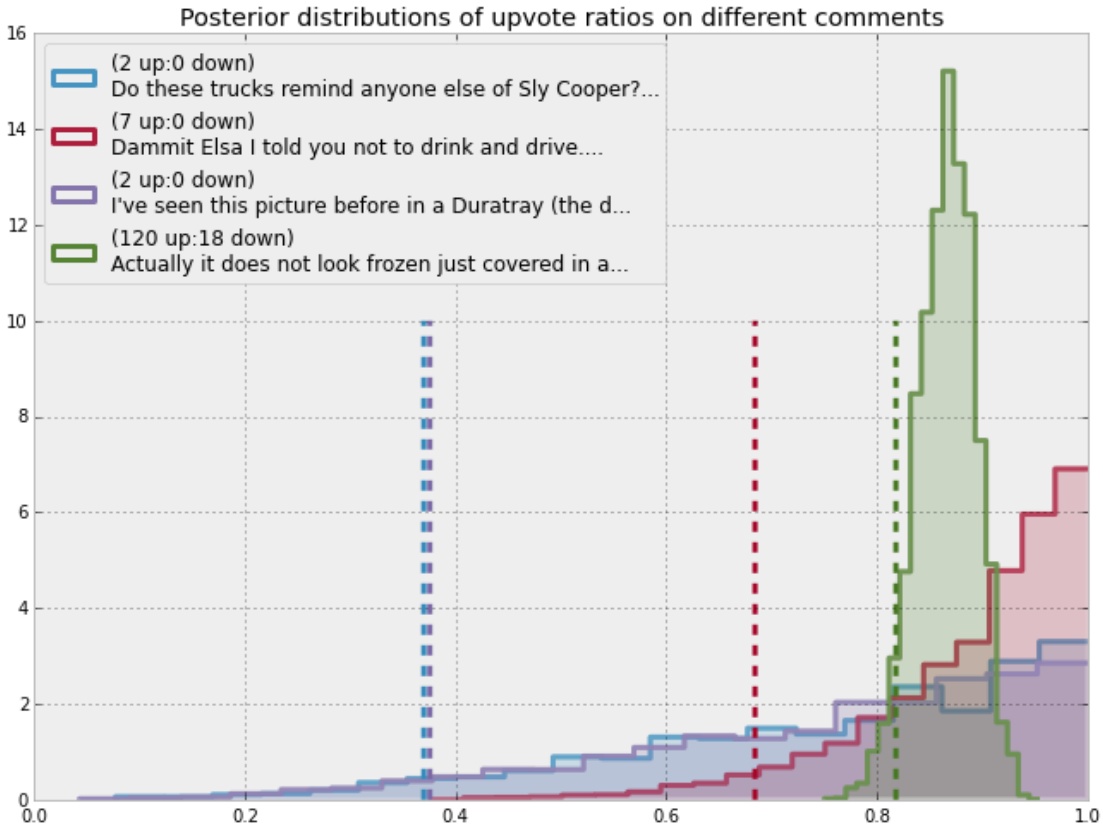


图 4.7 不同评论的好评率的后验分布

根据我们的介绍，好评率觉得一个评论的排名，即好评率越高排名越靠前。Visually those are the comments with the 95% least plausible value close to 1.

Why is sorting based on this quantity a good idea? By ordering by the 95% least plausible value, we are being the most conservative with what we think is best. That is, even in the worst case scenario, when we have severely overestimated the upvote ratio, we can be sure the best comments are still on top. Under this ordering, we impose the following very natural properties:

given two comments with the same observed upvote ratio, we will assign the comment with more votes as better (since we are more confident it has a higher ratio).

given two comments with the same number of votes, we still assign the comment with more upvotes as better.

速度达不到实时性要求

计算每个评论的后验分布是很费时间的，即使最终计算出了后验分布，可由于计算时间过长，数据又发生了变化。I delay the mathematics to the appendix,但是我建议使用下面的公式计算下限值会非常快

$$\frac{a}{a+b} - 1.65\sqrt{\frac{ab}{(a+b)^2(a+b+1)}} \qquad (4.13)$$

其中$a = 1 + \mu$，$b = 1 + d$

μ是好评的个数，d是差评的个数。计算公式（4.13）是贝叶斯推断的捷径，在第 6 章介绍先验时会具体介绍此公式。

```
def intervals(u, d):
    a = 1. + u
    b = 1. + d
    mu = a / (a + b)
    std_err = 1.65 * np.sqrt((a * b) / ((a + b) ** 2 * (a + b + 1.)))
    return (mu, std_err)

print "Approximate lower bounds:"
posterior_mean, std_err = intervals(votes[:, 0], votes[:, 1])
lb = posterior_mean - std_err
print lb
print
print "Top 40 Sorted according to approximate lower bounds:"
print
order = np.argsort(-lb)
ordered_contents = []
for i in order[:40]:
    ordered_contents.append(contents[i])
    print votes[i, 0], votes[i, 1], contents[i]
    print "-------------"
```

Approximate lower bounds:

```
[ 0.83167764  0.8041293   0.8166957   0.77375237  0.72491057  0.71705212
  0.72440529  0.73158407  0.67107394  0.6931046   0.66235556  0.6530083
  0.70806405  0.60091591  0.60091591  0.66278557  0.60091591  0.60091591
  0.53055613  0.53055613  0.53055613  0.53055613  0.53055613  0.43047887
  0.43047887  0.43047887  0.43047887  0.43047887  0.43047887  0.43047887
  0.43047887  0.43047887  0.43047887  0.43047887  0.43047887  0.43047887
  0.43047887  0.43047887  0.43047887  0.47201974  0.45074913  0.35873239
  0.3726793   0.42069919  0.33529412  0.27775794  0.27775794  0.27775794
  0.27775794  0.27775794  0.27775794  0.13104878  0.13104878  0.27775794
  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794
  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794
  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794
  0.27775794  0.27775794  0.27775794  0.27775794  0.27775794]
```

Top 40 Sorted according to approximate lower bounds:

327 52 Can you imagine having to start that? I've fired up much smaller equipment when its around 0° out and its still a pain. It would probably take a crew of guys hours to get that going. Do they have built in heaters to make it easier? You'd think they would just let them idle overnight if they planned on running it the next day though.

---

120 18 Actually it does not look frozen just covered in a layer of wind packed snow.

---

70 10 That's actually just the skin of a mining truck. They shed it periodically like snakes do.

---

76 14 The model just hasn't been textured yet!

---

21 3 No worries, [this](http://imgur.com/KeSYJud) will help.

---

7 0 Dammit Elsa I told you not to drink and drive.

---

88 23 Speaking of mining...[BAGGER 288!](http://www.youtube.com/watch?v=azEvfD4C6ow)

---

112 32 Wonder why OP has 31,944 link karma but so few submissions? /u/zkool may have the worst case of karma addiction I'm aware of.

title	points	age	/r/	comnts
[Frozen mining truck](http://www.reddit.com/r/pics/comments/1mrqvh/frozen_mining_truck/)	2507	4^mos	pics	164
[Frozen mining truck](http://www.reddit.com/r/pics/comments/1cutbw/frozen_mining_truck/)	16	9^mos	pics	4
[Frozen mining truck](http://www.reddit.com/r/pics/comments/vvcrv/frozen_mining_truck/)	439	1^yr	pics	21
[Meanwhile, in New Zealand...](http://www.reddit.com/r/pics/comments/ir1pl/meanwhile_in_new_zealand/)	39	2^yrs	pics	12
[Blizzardy day](http://www.reddit.com/r/pics/comments/1uiu3y/blizzardy_day/)	7	19^dys	pics	3

[Source: karmadecay](http://karmadecay.com/r/pics/comments/1w454i/frozen_mining_truck/)

---

11 1 This is what it's typically like, living in Alberta.

---

6 0 That'd be a haul truck. Looks like a CAT 793. We run em at the site I work at, 240ton carrying capacity.

---

22 5 Taken in Fort Mcmurray Ab!

---

9 1 "EXCLUSIVE: First look at "Hoth" from the upcoming 'Star Wars: Episode VII'"

---

32 9 This is the most fun thing to drive in GTA V.

---

5 0 it reminds me of the movie "moon" with sam rockwell.

---

4 0 Also frozen drill rig.

---

4 0 There's just something awesome about a land vehicle so huge that it warrants a set of stairs on the front of it. I find myself wishing I were licensed to drive it.

---

4 0 Heaters all over the components needing heat:
http://www.arctic-fox.com/fuel-fluid-warming-products/diesel-fired-coolant-pre-heaters

---

4 0 Or it is just an amazing snow sculpture!

---

3 0 I have to tell people about these awful conditions… Too bad I'm Snowden.

---

3 0 Someone let it go

---

3 0 Elsa, you can't do that to people's trucks.

---

3 0 woo Alberta represent

---

3 0 Just thaw it with love

---

6 2 Looks like the drill next to it is an IR DM30 or DM45. Good rigs.

---

4 1 That's the best snow sculpture I've ever seen.

---

2 0 [These](http://i.imgur.com/xYuwk5I.jpg) are used for removing the ice.

---

2 0 Nigger

---

2 0 Please someone post frozen Bagger 288

---

2 0 It's kind of cool there are trucks so big they need both a ladder and a staircase to get into them.

---

2 0 Eight miners are just out of frame hiding inside a tauntaun.

---

2 0 http://imgur.com/gallery/Fxv3Oh7

---

2 0 BRAZZERS.

---

2 0 It would take a god damn week just to warm that thing up.

---

2 0 Maybe /r/Bitcoin can use some of their mining equipment to heat this guy up!

---

2 0 Checkmate Jackie Chan

---

2 0 I've seen this picture before in a Duratray (the dump box supplier) brochure. If I recall it was either at Ekati or Diavik… In which case the truck could be either a Komatsu or a CAT… anyone care to comment?

---

2 0 The Texas snow has really hit hard!

---

2 0 I'm going to take a wild guess and say the diesel is gelled.

---

2 0 Do these trucks remind anyone else of Sly Cooper?

---

2 0 cool

通过画出后验分布的均值和边界，并按照边界进行排序，可以看到评论的最终排名顺序， notice that the left error-bar is sorted (as we suggested this is the best way to determine an ordering), so the means, indicated by dots, do not follow any strong pattern.

```
r_order = order[::-1][-40:]
plt.errorbar(posterior_mean[r_order], np.arange(len(r_order)),
             xerr=std_err[r_order], xuplims=True, capsize=0, fmt="o",
             color="#7A68A6")
plt.xlim(0.3, 1)
plt.yticks(np.arange(len(r_order) - 1, -1, -1), map(lambda x: x[:30].replace("\n", ""),
ordered_contents));
```

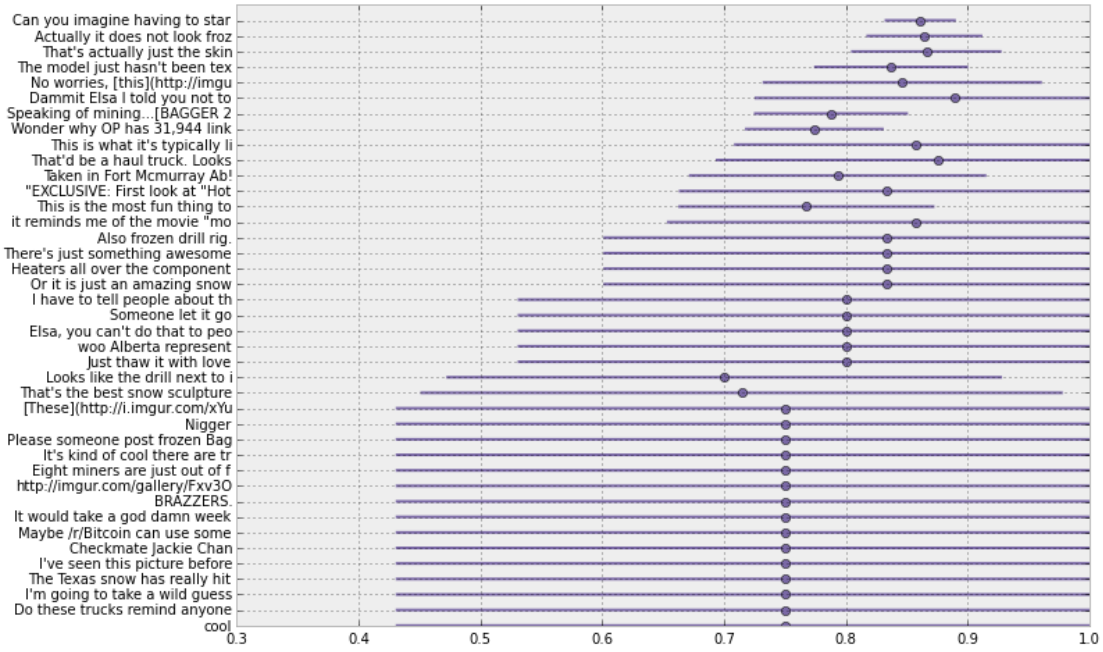


图 4.8 评论排序图

从上图可以看出，按照均值排序并不是最优的排序方案。

扩展到星级评定系统

以上排序方法对好评-差评策略很有用，但是其对星级评分系统的效果如何哪，例如 5 分制的评分系统。也可以使用类似的平均方法计算排名，也就会出现如下情况：如果一个条目有两个 5 分的评分，另一个条目有成千上万的 5 分评分，而只有一个 4 分评分，排名时前一个条目就会好于后一个条目。

可以将好评-差评问题表示为：0 表示差评，1 表示好评。一个 N 分制的评分系统就可以看成多个连续的好评-差评问题的组合，如果条目被评为n分，可以将其等价计算为$\frac{n}{N}$。例如一个 5 分评分系统，条目被评为 2 分时等价为 0.4。最好的评分是 1。这样就可以使用类似式（4.13）进行计算了，只不过a，b计算方法有所变化

$$\frac{a}{a+b} - 1.65\sqrt{\frac{ab}{(a+b)^2(a+b+1)}} \tag{4.13}$$

其中$a = 1 + S$，$b = 1 + N - S$

N是评分用户的数量，S是所有总的评分。

4.2.4 统计 Github 得分

Github 上资源的平均得分是多少？如何计算？Github 上有 6 百多万资源，这么多的数据很适合使用大数定理

4.3 结论

虽然大数定理很酷，但字如其名，只有在样本数量很大的时候它才有效。从上面的例子中已经看到推断的结果是如何受数据量的影响的。

1 在计算计算期望值时，通过画出后验分布的样本，以确保大数定理的应用的正确性。

2 在样本数量很小的贝叶斯推断中，后验分布包括更多的随机性。从后验分布中将能看出这些随机性的存在。后验分布越平坦随机性越大，越陡峭随机性越小。因此推断的结果也是可以矫正的。

3 There are major implications of not considering the sample size, and trying to sort objects that are unstable leads to pathological orderings. The method provided above solves this problem.

4.4 附录

Derivation of sorting comments formula

Basically what we are doing is using a Beta prior (with parameters a=1,b=1, which is a uniform distribution), and using a Binomial likelihood with observations u,N=u+d. This means our posterior is a Beta distribution with parameters $a' = 1+u, b' = 1+(N-u) = 1+d$. We then need to find the value, x, such that 0.05 probability is less than x. This is usually done by inverting the CDF (Cumulative Distribution Function), but the CDF of the beta, for integer parameters, is known but is a large sum [3].

We instead use a Normal approximation. The mean of the Beta is $\mu = a'/(a'+b')$ and the variance is

$$\sigma^2 = a'b'(a'+b')^2(a'+b'+1)$$

Hence we solve the following equation for x and have an approximate lower bound.

$$0.05 = \Phi((x-\mu)\sigma)$$

Φ being the cumulative distribution for the normal distribution

Exercises

1. How would you estimate the quantity E[cosX], where X~Exp(4)? What about E[cosX|X<1], i.e. the expected value given we know X is less than 1? Would you need more samples than the original samples size to be equally accurate?

```
# Enter code here
import scipy.stats as stats
exp = stats.expon(scale=4)
N = 1e5
X = exp.rvs(N)
# ...
```

2. The following table was located in the paper "Going for Three: Predicting the Likelihood of Field Goal Success with Logistic Regression" [2]. The table ranks football field-goal kickers by their percent of non-misses. What mistake have the researchers made?

Kicker Careers Ranked by Make Percentage

Rank	Kicker	Make %	Number of Kicks
1	Garrett Hartley	87.7	57
2	Matt Stover	86.8	335
3	Robbie Gould	86.2	224
4	Rob Bironas	86.1	223
5	Shayne Graham	85.4	254
…	…	…	
51	Dave Rayner	72.2	90
52	Nick Novak	71.9	64
53	Tim Seder	71.0	62
54	Jose Cortez	70.7	75
55	Wade Richey	66.1	56

In August 2013, a popular post on the average income per programmer of different languages was trending. Here's the summary chart: (reproduced without permission, cause when you lie with stats, you gunna get the hammer). What do you notice about the extremes?

Average household income by programming language

Language	Average Household Income ($)	Data Points
Puppet	87,589.29	112
Haskell	89,973.82	191
PHP	94,031.19	978
CoffeeScript	94,890.80	435
VimL	94,967.11	532
Shell	96,930.54	979
Lua	96,930.69	101
Erlang	97,306.55	168
Clojure	97,500.00	269
Python	97,578.87	2314
JavaScript	97,598.75	3443
Emacs Lisp	97,774.65	355
C#	97,823.31	665
Ruby	98,238.74	3242
C++	99,147.93	845
CSS	99,881.40	527
Perl	100,295.45	990
C	100,766.51	2120
Go	101,158.01	231
Scala	101,460.91	243
ColdFusion	101,536.70	109
Objective-C	101,801.60	562
Groovy	102,650.86	116
Java	103,179.39	1402
XSLT	106,199.19	123
ActionScript	108,119.47	113

4.5 References

1 Wainer, Howard. The Most Dangerous Equation. American Scientist, Volume 95.

2 Clarck, Torin K., Aaron W. Johnson, and Alexander J. Stimpson. "Going for Three: Predicting the Likelihood of Field Goal Success with Logistic Regression." (2013): n. page. Web. 20 Feb. 2013.

3 http://en.wikipedia.org/wiki/Beta_function#Incomplete_beta_function

```
from IPython.core.display import HTML


def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

第 5 章 损失函数

统计学家是一群思维反常的人。他们不考虑他们能获得多少，而是考虑他们损失了多少，他们认为损失的负数就是获得。如何衡量损失是一件有趣的事情。

考虑如下例子：

假设一个气象学家正在预测飓风袭击他所在城市的概率是多少。他估计飓风不袭击该城市的可能性为 99%-100%，且这一概率的可信度为 95%。气象学家非常满意他的计算精度，并建议大家不用疏散。不幸的是飓风袭击了城市，城市被淹没了。

这个典型的例子说明了单纯使用精度去度量结果时存在的缺点。Using a measure that emphasizes estimation accuracy, while an appealing and objective thing to do, misses the point of why you are even performing the statistical inference in the first place: results of inference. The author Nassim Taleb of The Black Swan and Antifragility stresses the importance of the payoffs of decisions, not the accuracy. Taleb distills this quite succinctly: "I would rather be vaguely right than very wrong."

5.1 损失函数

下面将介绍被统计学家和决策论者口中的损失函数。损失函数是一个使用已知参数估计未知参数的函数。

$$L(\theta,\hat{\theta}) = f(\theta,\hat{\theta}) \tag{5-1}$$

用损失函数度量当前估计的好坏程度：损失函数值越大，估计的结果越差。简单且常用的损失函数是均方误差损失函数，如下

$$L(\theta,\hat{\theta}) = (\theta-\hat{\theta})^2 \tag{5-2}$$

线性回归，一致最小方差无偏估计，以及机器学习领域中经常使用均方误差损失函数。还有另一种对称的均方误差损失函数

$$L(\theta,\hat{\theta}) = \begin{cases} (\theta-\hat{\theta})^2, \hat{\theta} < \theta \\ c(\theta-\hat{\theta})^2, \hat{\theta} \geq \theta, 0 < c < 1 \end{cases} \tag{5-3}$$

上式表明，损失函数更倾向于估计大于等于某一个阈值时的参数。这种损失函数形式更适合于估计下个月的网络流量，因为为了避免服务器资源分配不足情况，在对服务器资源分配进行估计时要多分配一些。

A negative property about the squared-error loss is that it puts a disproportionate emphasis on large outliers. This is because the loss increases quadratically, and not linearly, as the estimate moves away. That is, the penalty of being three units away is much less than being five units away, but the penalty is not much greater than being one unit away, though in both cases the magnitude of difference is the same:

$$\frac{1^2}{3^2} < \frac{3^2}{5^2} \quad \text{虽然} \quad 3-1 = 5-3$$

当误差较大时均方误差损失函数的结果会变得很坏。真实值和估计值的绝对值的线性损失函数稳健性较好，如下

$$L(\theta,\hat{\theta}) = |\theta-\hat{\theta}| \tag{5-4}$$

其它常用的损失函数为

- $L(\theta,\hat{\theta}) = 1_{\theta\neq\hat{\theta}}$是机器学习中常用的 0-1 损失函数。
- $L(\theta,\hat{\theta}) = -\hat{\theta}log(\theta) - (1-\hat{\theta})log(1-\theta); \hat{\theta} \in 0,1; \theta \in [0,1]$，称为 log 损失函数，也是机器学习中常用的一种损失函数。

从历史上来看，选择损失函数主要基于如下两方面：1）数学计算上的便利性，2）应用上的稳健性，即损失函数可以客观的对损失进行度量。不过第一点确实使损失函数

的使用范围受到不少限制。对数学计算的便利性没有认知的计算机来说，可以设计任意的损失函数，本章稍后部分将介绍使用计算机计算损失函数。

对于第 2）点来说，以上的损失函数确实是客观的，它们都是估计值和真值之差的函数，和估计的收益对象无关。和收益对象无关性会导致病态结果。再次考虑以上飓风的例子，统计学家也可以这样预测，飓风袭击的概率在 0%到 1%之间，如果他忽略精度把注意点集中在结果上（99%的概率没有洪水，1%的概率有洪水），他给出的建议可能截然不同。

如果不考虑实际的应用背景，或者业务问题，对损失函数的优化是纯数学问题，损失函数的最有并不一定能让被求解问题最优，所以需要更多的关注被求解的问题，损失函数只是求解手段，它的结果不一定符合物理现实（还记得经常需要去除负值解吗）。这就要求我们设计新的损失函数，新的损失函数要能够反映出结果和目标。有一些非常有趣的损失函数如下

- $L(\theta,\hat{\theta}) = \frac{|\theta-\hat{\theta}|}{\theta(\theta-\hat{\theta})}; \theta,\hat{\theta}\epsilon[0,1]$，该损失函数更倾向于估计靠近 0 或者 1 位置的参数，因为如果真值θ靠近 0 或者 1 时，损失函数将变的非常大，除非估计值$\hat{\theta}$也接近 0 或者 1。这种损失函数更适合政治评论家使用，因为他们更倾向于给出“是/否”的答案。这个损失函数表明如果参数真值接近 1（例如，一个政治结果很可能会发生），他会毫不怀疑的认为政治结果会发生。
- $L(\theta,\hat{\theta}) = 1 - \exp\left(-(\theta-\hat{\theta})^2\right)$，该损失函数的取值范围为 0 到 1，这种损失函数表明其使用者不太关心大的估计结果。这和 0-1 损失函数比较类似，但是在真值附件的惩罚程度又没有 0-1 损失函数那么强烈。
- 复杂的非线性函数可以用编程实现

```
def loss(true_value, estimate):
    if estimate*true_value > 0:
        return abs(estimate - true_value)
    else:
        return abs(estimate)*(estimate - true_value)**2
```

- 还有一个例子来自于《The Signal and The Noise》，天气预报人员喜欢使用如下损失函数

> 大家意识到对下雨预测的失败比下雨警告失败的次数要多。当预测下雨而实际上又没下的时候，大家可能会诅咒天气预报员破坏了他们的野餐，而一个意想不到的好天气将是一个意外的甜点。
> [The Weather Channel's bias] is limited to slightly exaggerating the probability of rain when it is unlikely to occur — saying there is a 20 percent change when they know it is really a 5 or 10 percent chance — covering their butts in the case of an unexpected sprinkle.

由此可见损失函数的使用可以带来好处也可以带来坏处。

5.2 现实中的损失函数

到目前为止，都有一个不太现实的假设：我们知道真实的参数。当然了，如果已经知道了真实的参数，就没有必要再对参数进行估计了。因此只有在不知道真实参数的时候损失函数才会有用。

在贝叶斯推断中，把未知参数看成是与先验和后验分布的随机变量。就后验分布而言，从后验分布得到的样本都可能是真实的参数值。有了从后验分布中采样得到的样本就可以计算与某种估计相关的损失函数值了。因为我们已经知道了未知参数的后验分布，我们更关心的是在某种估计下的期望损失。与用一个后验样本估计出的损失函数相比，期望损失对真实损失的估计结果会更稳健。

首先期望损失有利于解释贝叶斯点估计。现实世界中的系统和机器无法把先验分布作为输入参数。实际上最终关心的是估计结果，后验分布只是计算估计结果的中间步骤，如果直接给出后验分布是没有太大作用的。In the course of an individual's day, when faced with uncertainty we still act by distilling our uncertainty down to a single action。与之类似，也需要使后验分布变得陡峭，理想情况是变成一个点。如果后验分布的值取得合理，可以避免频率学派无法提供不确定性的缺陷，这种结果的信息更丰富。如果从贝叶斯后验中选择点，就是贝叶斯点估计。

假设$P(\theta|X)$是在观测数据 X 的条件下θ的后验概率，θ的期望损失$\hat{\theta}$为

$$l(\hat{\theta}) = E_{\theta}[L(\theta,\hat{\theta})] \qquad (5\text{-}5)$$

式（5-5）也称为$\hat{\theta}$的风险估计。式（5-5）中E的下标θ表示在数学期望中θ是未知的随机变量。

下面一整章将讨论如何近似计算期望值。从后验分布得到N个样本$\theta_i, i = 1,\cdots,N$，损失函数为L，可以通过大数定理，利用$\hat{\theta}$的估计值近似计算出期望损失

$$\frac{1}{N}\sum_{i=1}^{N} L(\theta,\hat{\theta}) \approx E_{\theta}[L(\theta,\hat{\theta})] = l(\hat{\theta}) \qquad (5\text{-}6)$$

与 MAP 相比，计算损失函数的期望值利用了后验分布中的更多信息，因为在 MAP 中仅仅是寻找后验分布的最大值，而忽略了后验分布的形状。MAP 中忽略掉的信息可能使你暴露在长尾部分的风险当中，如不可能发生的飓风，and leaves your estimate ignorant of how ignorant you really are about the parameter.

频率学派的目标只是最小化化误差，不考虑与误差结果相关的损失。Compound this with the fact that frequentist methods are almost guaranteed to never be absolutely accurate. Bayesian point estimates fix this by planning ahead: your estimate is going to be wrong, you might as well err on the right side of wrong.

5.2.1

第 6 章　先验知识