第三章 概率与信息论

概率论是用来表示不确定性的。直接与频率相关的叫 `频率派概率（frequentist probability）`，涉及到确定性水平叫 `贝叶斯概率（Bayesian probability）`。

频率学派需要做大量实验才能给出结论，比如为了得到抛硬币正面朝上的概率，通过抛100次硬币，用硬币正面向上的次数除以100。贝叶斯概率论假设观察者对某事件处于某个知识状态中（刚开始先验地相信一枚硬币是均匀的），之后观察者开始新的观测或实验（不断地抛硬币，发现抛了100次后，居然只有20次是正面朝上）。经过这些试验获得了一些新的观测结果，这些新的观测将影响观察者原有的信念（开始怀疑这枚硬币究竟是不是均匀的，甚至开始断定硬币并不均匀）。然而，如果要计算一个人今年高考考上北大的概率，频率学派就会遇到很大的问题，因为很难让一个人考100次，然后用考上的次数除以100。这时贝叶斯学派会找几个高考特级教师对这个人进行一下考前测验和评估，然后让这几个教师给出一个主观的可能性，比如说：这个人有9成的把握考上北大。
二派的区别主要在于对概率的定义，频率学派就是很客观的了。而贝叶斯学派认为概率就是人对一个事件发生有多大的信心。虽然二者对概率的定义不同，但是都满足概率的公理化定义所要求的条件。

因此，目前贝叶斯学派可以用于预测美国总统竞选成功的概率、巴西下届世界杯夺冠的概率、明天下雨的概率、运载火箭发射成功的概率，等等。很明显，这些事件的概率通过频率学派是很难进行估计的，因为这些事件不可能在相同条件下重复。不过，频率派的优点则是没有假设一个先验分布，因此更加客观，也更加无偏。由此，在一些保守的领域（比如制药业、法律）比贝叶斯方法更受到信任。

简单地说，频率学派与贝叶斯学派探讨「不确定性」这件事时的出发点与立足点不同。频率学派从「自然」角度出发，试图直接为「事件」本身建模，**即事件A在独立重复试验中发生的频率趋于极限p**，那么这个极限就是该事件的概率。例如想要计算抛掷一枚硬币时正面朝上的概率，就需要不断地抛掷硬币，当抛掷次数趋向无穷时正面朝上的**频率**即为正面朝上的概率。贝叶斯学派并不从试图刻画「事件」本身，而从「观察者」角度出发，而只是从「观察者知识不完备」这一出发点开始，构造一套在贝叶斯概率论的框架下可以对不确定知识做出推断的方法。频率学派下说的「随机事件」在贝叶斯学派看来，并不是「事件本身具有某种客观的随机性」，而是「观察者不知道事件的结果」而已，只是「观察者」知识状态中尚未包含这一事件的结果。贝叶斯概率论就想构建一套比较完备的框架用来描述最能服务于理性推断这一目的的「猜的过程」。因此，在贝叶斯框架下，同一件事情对于知情者而言就是「确定事件」，对于不知情者而言就是「随机事件」，随机性并不源于事件本身是否发生，而只是描述观察者对该事件的知识状态。

总的来说，贝叶斯概率论为人的知识（knowledge）建模来定义「概率」这个概念。频率学派试图描述的是「事物本体」，而贝叶斯学派试图描述的是观察者知识状态在新的观测发生后如何更新。在这一过程中，观察者无法用简单的逻辑来推断，因为观察者并没有完全的信息作为证据，因此只能采用**似真推断（plausible reasoning）**，对于各种各样可能的结果赋予一个**「合理性」**（plausibility）。例子中，根据原有的信念，抛硬币正反面朝上的概率相等这个论断合理性非常高。但是在观察到100次抛掷中只有20次正面朝上后，我们开始怀疑抛硬币正反面朝上的概率相等这个论断，认为抛硬币反面朝上的概率更大这个推断的合理性更高，而支持的证据就是刚刚实验的观测结果。

往大里说，世界观就不同，频率派认为参数是客观存在，不会改变，虽然未知，但却是固定值；贝叶斯派则认为参数是随机值，因为没有观察到，那么和是一个随机数也没有什么区别，因此参数也可以有分布。往小处说，频率派最常关心的是似然函数，而贝叶斯派最常关心的是后验分布。我们会发现，后验分布其实就是似然函数乘以先验分布再normalize一下使其积分到1。因此两者的很多方法都是相通的。贝叶斯派因为所有的参数都是随机变量，都有分布，因此可以使用一些基于采样的方法使得我们更容易构建复杂模型。

频率学派 - Frequentist - Maximum Likelihood Estimation (MLE，最大似然估计)，他们认为模型参数是个定值，希望通过类似解方程组的方式从数据中求得该未知数。这就是频率学派使用的参数估计方法-**极大似然估计（MLE）**，这种方法往往在大数据量的情况下可以很好的还原模型的真实情况。

贝叶斯学派 - Bayesian - Maximum A Posteriori (MAP，最大后验估计)。他们认为模型参数源自某种潜在分布，希望从数据中推知该分布。对于数据的观测方式不同或者假设不同，那么推知的该参数也会因此而存在差异。这就是贝叶斯派视角下用来估计参数的常用方法-**最大后验概率估计（MAP）**，这种方法在先验假设比较靠谱的情况下效果显著，随着数据量的增加，先验假设对于模型参数的主导作用会逐渐削弱，相反真实的数据样例会大大占据有利地位。极端情况下，比如把先验假设去掉，或者假设先验满足均匀分布的话，那她和极大似然估计就如出一辙了。

抽象一点来讲，频率学派和贝叶斯学派对世界的认知有本质不同：频率学派认为世界是确定的，有一个本体，这个本体的真值是不变的，我们的目标就是要找到这个真值或真值所在的范围；而贝叶斯学派认为世界是不确定的，人们对世界先有一个预判，而后通过观测数据对这个预判做调整(个人人为这个调整就是根据先验和似然得到后验，得到的这个后验就是新调整的值)，我们的目标是要找到最优的描述这个世界的概率分布。

在对事物建模时，用θ表示模型的参数，**请注意，解决问题的本质就是求θ。**那么：

- **频率学派**：存在唯一真值θ。举一个简单直观的例子—抛硬币，我们用P(head)来表示硬币的概率。抛一枚硬币100次，有20次正面朝上，要估计抛硬币正面朝上的bias P(head)=θ。在频率学派来看，θ= 20 / 100 = 0.2，很直观。当数据量趋于无穷时，这种方法能给出精准的估计；然而缺乏数据时则可能产生严重的偏差。例如，对于一枚均匀硬币，即θ= 0.5，抛掷5次，出现5次正面 (这种情况出现的概率是1/2^5=3.125%)，频率学派会直接估计这枚硬币θ= 1，出现严重错误。
- **贝叶斯学派**： θ是一个随机变量，符合一定的概率分布。在贝叶斯学派里有两大输入和一大输出，输入是先验 (prior)和似然 (likelihood)，输出是后验 (posterior)。

  先验，即P(θ)，指的是在没有观测到任何数据时对θ的预先判断，例如给我一个硬币，一种可行的先验是认为这个硬币有很大的概率是均匀的，有较小的概率是是不均匀的；似然，即P(X|θ)，是假设θ已知后我们观察到的数据应该是什么样子的；后验，即P(θ|X)，是最终的参数分布。

极大似然估计与最大后验概率估计

我们这有一个任务，就是根据已知的一堆数据样本，来推测产生该数据的模型的参数，即**已知数据，推测模型和参数**。因此根据两大派别的不同，对于模型的参数估计方法也有两类：极大似然估计与最大后验概率估计。

- **极大似然估计（MLE）**

  她是频率学派模型参数估计的常用方法。顾名思义：似然，可以简单理解为概率、可能性，也就是说要最大化该事件发生的可能性。她的含义是根据已知样本，希望通过调整模型参数来使得模型能够最大化样本情况出现的概率。

  似然（likelihood）这个词其实和概率（probability）是差不多的意思，Colins字典这么解释：The **likelihood** of something happening is how likely it is to happen. 你把likelihood换成probability，这解释也读得通。但是在统计里面，似然函数和概率函数却是两个不同的概念（其实也很相近就是了）。

  $P(x|\theta)$

  输入有两个：x表示某一个具体的数据；θ表示模型的参数。

  如果θ是已知确定的，x是变量，这个函数叫做概率函数(probability function)，它描述对于不同的样本点x，其出现概率是多少。

  如果x是已知确定的，θ是变量，这个函数叫做似然函数(likelihood function), 它描述对于不同的模型参数，出现x这个样本点的概率是多少。

在这举个猜黑球的例子：假如一个盒子里面有红黑共10个球，每次有放回的取出，取了10次，结果为7次黑球，3次红球。问拿出黑球的概率p是多少?

我们假设7次黑球，3次红球为事件A，一个理所当然的想法就是既然事件A已经发生了，那么事件A发生的概率应该最大。所以既然事件A的结果已定， 我们就有理由相信这不是一个偶然发生的事件，这个已发生的事件肯定一定程度上反映了黑球在整体中的比例。所以我们要让模型产生这个整体事件的概率最大，我们把这十次抽取看成一个整体事件A，很明显事件A发生的概率是每个子事件概率之积。我们把$P(A)$看成一个关于p的函数，求$P(A)$取最大值时的p，这就是极大似然估计的思想。具体公式化描述为$P(A) = p^7 * (1-p)^3$。接下来就是取对数转换为累加，然后通过求导令式子为0来求极值，求出p的结果。取对数得：

$$\ln(P(A)) = \ln(p^7 * (1-p)^3) = 7\ln p + 3\ln(1-p)$$

$$\text{令：}\ \ln'(P(A)) = 0$$

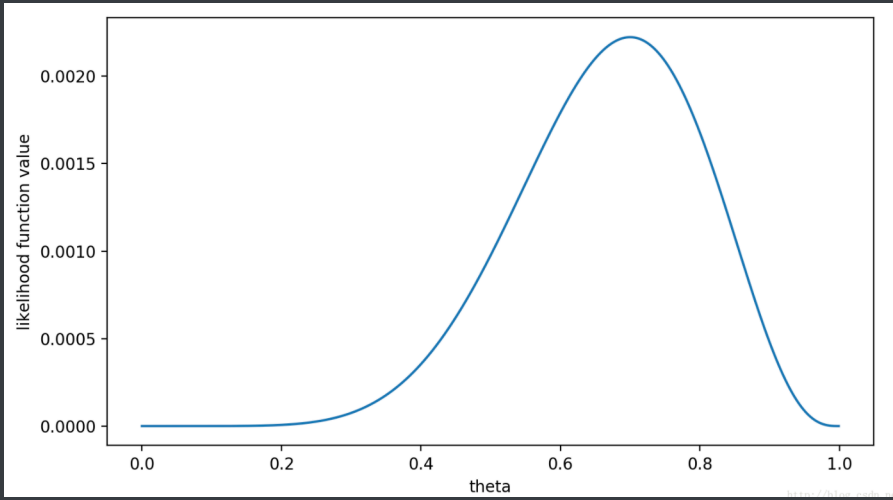
$$\text{即：}\ \frac{7}{p} + \frac{3}{1-p} = 0$$

$$\text{得：}\ p = 0.7$$

于是我们拿这枚硬币抛了10次，得到的数据（x0x0）是：反正正正正反正正正反。我们想求的正面概率θθ是模型参数，而抛硬币模型我们可以假设是二项分布。那么，出现实验结果x0x0（即反正正正正反正正正反）的似然函数是多少呢?

$$f(x_0, \theta) = (1-\theta) \times \theta \times \theta \times \theta \times \theta \times (1-\theta) \times \theta \times \theta \times \theta \times (1-\theta) = \theta^7(1-\theta)^3 = f(\theta)$$

注意，这是个只关于θθ的函数。而最大似然估计，顾名思义，就是要最大化这个函数。我们可以画出f(θ)f(θ)的图像：



通过计算，最大值是0.002223，位置在0.7处。

- **最大后验概率估计（MAP）**

  她是贝叶斯派模型参数估计的常用方法。顾名思义：就是最大化在给定数据样本的情况下模型参数的后验概率。她依然是根据已知样本，来通过调整模型参数使得模型能够产生该数据样本的概率最大，只不过对于模型参数有了一个先验假设，即模型参数可能满足某种分布，不再一味地依赖数据样例（万一数据量少或者数据不靠谱呢）。

  在这里举个掷硬币的例子：抛一枚硬币10次，有10次正面朝上，0次反面朝上。问正面朝上的概率θ。

  在频率学派来看，利用极大似然估计可以得到 $\theta = 10/10 = 1.0$。显然当缺乏数据时MLE可能会产生严重的偏差。

  如果我们利用极大后验概率估计来看这件事，先验认为大概率下这个硬币是均匀的 (例如最大值取在0.5处的Beta分布)，那么$P(\theta|X)$，是一个分布，最大值会介于0.5~1之间，而不是武断的给出$\theta = 1$。

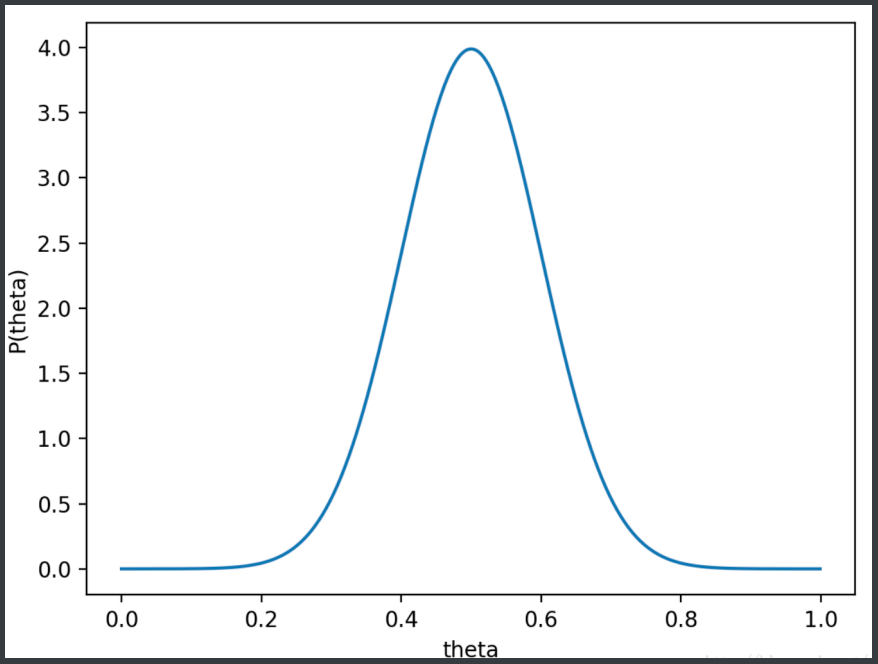
  显然，随着数据量的增加，参数分布会更倾向于向数据靠拢，先验假设的影响会越来越小。

  最大似然估计是求参数θ, 使似然函数$P(x_0|\theta)$最大。最大后验概率估计则是想求θ使$P(x_0|\theta)P(\theta)$最大。求得的θ不单单让似然函数大，θ自己出现的先验概率也得大。（这有点像正则化里加惩罚项的思想，不过正则化里是利用加法，而MAP里是利用乘法）。

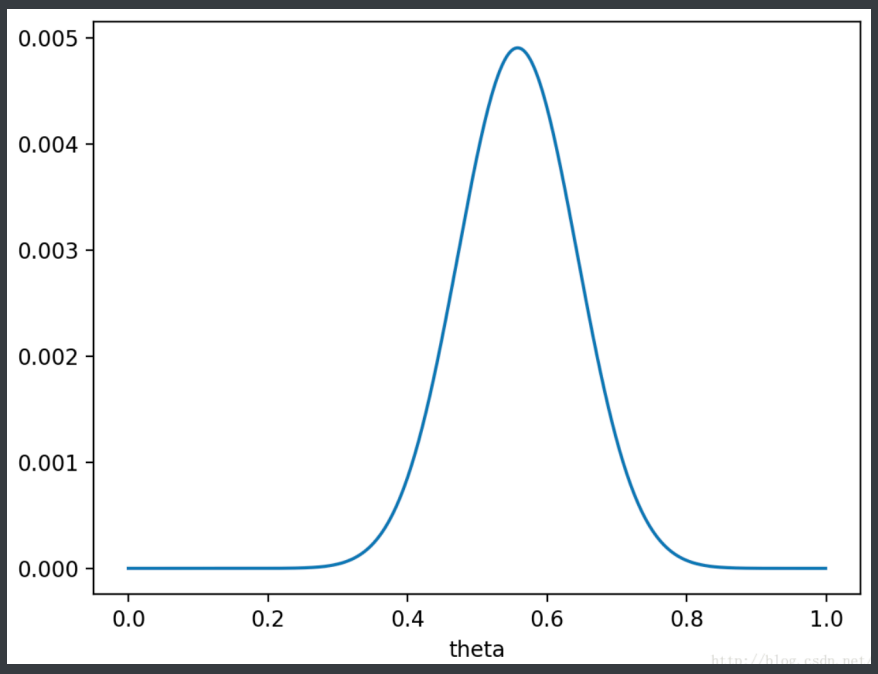
  MAP其实是在最大化$P(\theta|x_0) = \frac{P(x_0|\theta)P(\theta)}{P(x_0)}$因为$x_0$是确定的（即投出的“反正正正正反正正正反”），$P(x_0)$是一个已知值，所以去掉了分母$P(x_0)$（假设“投10次硬币”是一次实验，实验做了1000次，“反正正正正反正正正正反”出现了n次，则$P(x_0) = n/1000$。总之，这是一个可以由数据集得到的值）。最大化$P(\theta|x_0)$的意义也很明确，x_0已经出现了，要求θ取什么值使$P(\theta|x_0)$最大。$P(\theta|x_0)$这就是“最大后验概率估计”名字的由来。

  对于投硬币的例子来看，我们认为（”先验地知道“）θ取0.5的概率很大，取其他值的概率

小一些。我们用一个高斯分布来具体描述我们掌握的这个先验知识，例如假设$P(\theta)$为均值0.5，方差0.1的高斯函数，如下图：



则$P(x_0|\theta)P(\theta)$的函数图像为：



这里需要解释下。为什么会在0.5和0.7之间。因为$P(x_0|\theta)$是似然，MLE计算的0.7，$P(\theta)$是先验，拿高斯分布举例。

$$\mathcal{N}(x;\mu,\sigma^2)=\sqrt{\frac{1}{2\pi\sigma^2}}e^{\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)}$$

求$P(x_0|\theta)P(\theta)$用log方便，变成$\log P(x_0|\theta)+\log P(\theta)$再求导数，得到极值点。先验的高斯求导加上似然求导得：

$$\left(\log\sqrt{\frac{1}{2\pi\sigma^2}}e^{\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)}+\log\theta^7(1-\theta)^3\right)'=0$$

$$\frac{1}{\frac{1}{2\pi\sigma^2}e^{\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)}}\times\sqrt{\frac{1}{2\pi\sigma^2}}\times e^{\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)}\times\left(-\frac{1}{\sigma^2}(x-\mu)\right)+\left(\log\theta^7(1-\theta)^3\right)'=0$$

其中$\mu=0.5,\sigma=0.1$，如果没有似然计算的$0.7^7\times0.3^3\approx0.0022$，高斯极值在0.5，但是现在有似然项，计算的x极值在0.5～0.7之间。因此极值在0.5往右边一点。最后，那要怎样才能说服一个贝叶斯派相信$\theta=0.7$呢？你得多做点实验。。

3.2 随机变量

`随机变量（Random variable）`是可以取不同值的变量。

3.3 概率分布

`概率分布（probability distrubution）`用来描述变量或一簇随机变量在每一个可能取到的状态的可能性大小。我们描述概率分布的方式取决于取决于随机变量是离散的还是连续的。

3.3.1 离散变量和概率质量函数

离散变量的概率分布可以用`概率质量函数（probability mass function，PMF）`来描述。概率质量函数将将随机变量取得的每个状态映射到取得该变量的概率。明确写出随机变量名$P(x=x)$
概率质量函数可以同时作用于多个随机变量，多个变量的概率分布被称为`联合概率分布（joint probability distribution）`。$P(x=x,y=y)$表示x=x，y=y同时发生的概率简写$P(x,y)$。
PMF必须满足：

- P的定义域是x所有可能状态集合
- $\forall x\in x,0\leq P(x)\leq1$.不可能发生的事件概率为0。

- $\sum_{x \in x} P(x) = 1$.这条性质称为`归一化（normalized）`。例如，离散型随机变量x有k个不同的状态。我们可以假设x是`均匀分布（uniform distribution）`，PMF为：

$$P(x = x_i) = \frac{1}{k}$$

累积分布函数

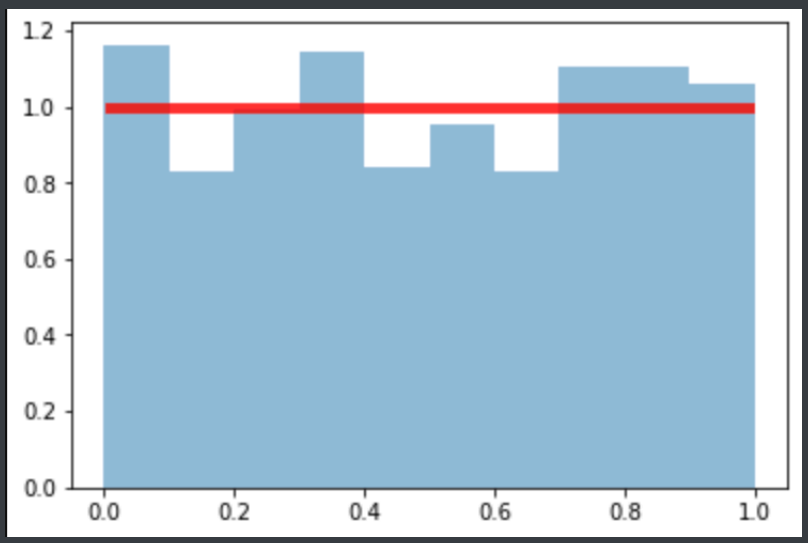
累积分布函数(Cummulative Distribution Function)表示小雨x的概率的积分：

$$CDF(x) = \int_{-\infty}^{x} p(t)dt$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import uniform

# generate samples
fix, ax = plt.subplots(1, 1)
r = uniform.rvs(loc=0, scale=1, size=1000)
# x: 指定要绘制直方图的数据；输入值，这需要一个数组或者一个序列，不需要长度相同的数组。
# bins: 指定直方图条形的个数；
# range: 指定直方图数据的上下界，默认包含绘图数据的最大值和最小值；
# density: 布尔,可选。如果"True"，返回元组的第一个元素将会将计数标准化以形成一个概率密度，也就是说，直方图下的面积（或积分）总和为1。density=ture的意思是保证该面积的积分为1，并不是概率和为1。这是通过将计数除以数字的数量来实现的观察乘以箱子的宽度而不是除以总数数量的观察。如果叠加也是“真实”的，那么柱状图被规范化为1。(替代normed)
# weights: 该参数可为每一个数据点设置权重；
# cumulative: 是否需要计算累计频数或频率；
# bottom: 可以为直方图的每个条形添加基准线，默认为0；
# histtype: 指定直方图的类型，默认为bar，除此还有’barstacked’, ‘step’, ‘stepfilled’;
# align: 设置条形边界值的对其方式，默认为mid，除此还有’left’和’right’;
# orientation: 设置直方图的摆放方向，默认为垂直方向;
# rwidth: 设置直方图条形宽度的百分比;
# log: 是否需要对绘图数据进行log变换;
```

```
    # color：设置直方图的填充色；
    # label：设置直方图的标签，可通过legend展示其图例；
    # stacked：当有多个数据时，是否需要将直方图呈堆叠摆放，默认水平摆放；
    # normed：是否将直方图的频数转换成频率；(弃用，被density替代)
    # alpha：透明度，浮点数。
    ax.hist(r, density=True, histtype='stepfilled', alpha=0.5)
    #ppf(q, loc=0, scale=1) 百分比点函数(cdf的倒数-百分位数)。
    # pdf(x, loc=0, scale=1)Probability density function. 概率密度函数
    x = np.linspace(uniform.ppf(0.01), uniform.ppf(0.99), 100)
    ax.plot(x, uniform.pdf(x), 'r-', lw=5, alpha=0.8, label='uniform pdf') #
    lw: linewidth, alpha 透明度
```



3.3.2 连续变量和概率密度函数

研究连续型随机变量用 概率密度函数（probability density function，PDF） 描述概率分布，不用概率质量函数。概率密度函数p满足：

- p的定义域是x所有可能状态集合
- $\forall x \in x, p(x) \geq 0.$，我们不要求$p(x) \leq 1$。
- $\int p(x)dx = 1.$

概率密度$p(x)$没有对特定状态给出概率，它给出了落在面积为δx的无限小的区域内的概率为$p(x)\delta x$。可以对概率密度函数求积分来获取点集的真实概率质量。单变量例子中，x落在区间$[a, b]$的概率是$\int_{[a,b]} p(x)dx$。

3.4 边缘概率

当我们知道了一组变量的联合概率分布，要了解其中一个子集的概率分布。这种定义在子集上的概率分布称为`边缘概率分布（marginal probability distribution）`。

例如有随机变量x和y，并且知道$P(x,y)$。根据一下`求和法则（sum rule）`计算P(x)：

$$\forall x \in x, P(x) = \sum_y P(x = x, y = y)$$

边缘概率来源于手算边缘概率的计算过程。$P(x,y)$每行表示不同的x值，每列表示不同的y值形成的网格，行求和结果P(x)写在每行右边纸边缘处。

对连续变量要用积分替代求和：

$$p(x) = \int p(x,y)dy$$

3.5 条件概率

给定其他事件发生时，感兴趣某个事件出现的概率，叫条件概率。给定x=x，y=y发生的条件概率记为$P(y = y|x = x)$。条件概率可以通过下面公式计算：

$$P(y = y|x = x) = \frac{P(y = y, x = x)}{P(x = x)}$$

3.6 条件概率链式法则

任何多维随机变量的联合概率分布，都可以分解成只有一个变量条件概率相乘形式。叫`链式法则(chain rule)`或`乘法法则(product rule)`。

3.7 独立性和条件独立

两个随机变量x和y，如果概率分布可以表示成两个因子乘积形式，并且一个因子只包含x另一个只包含y，则两个随机变量`相互独立(independent)`：

$$\forall x \in x, y \in y, p(x = x, y = y) = p(x = x)p(y = y)$$

如果关于x和y的条件概率分布对于z每个值都可以写成乘积形式，两个随机变量x和y在给定随机变量z时是`条件独立的(conditionally independent)：

$$\forall x \in x, y \in y, z \in z, p(x = x, y = y|z = z) = p(x = x|z = z)p(y = y|z = z)$$

$x \perp y$表示x和y相互独立，$x \perp y|z$，x和y在给定z时条件独立。

3.8 期望、方差和协方差

函数$f\ (x)$关于某分布$P(x)$的 `期望(expectation)` 或者 `期望值(expected value)` 是指，当x由P产生，f作用于x时，$f(x)$的平均值。对于离散型随机变量，可通过求和得到：

$$E_{x\sim P}[f(x)] = \sum_x P(x)f(x)$$

对于连续型随机变量可以通过求积分得到：

$$E_{x\sim P}[f(x)] = \int_x P(x)f(x)$$

可简写成$E_x[f(x)]$。进一步简写$E[f(x)]$。期望是线性的。

`方差(variance)` 衡量的是对x依据它的概率分布进行采样，随机变量x函数值会呈现多大差异：

$$Var(f(x)) = E[(f(x) - E(f(x))^2]$$

当方差很小时f(x)值形成的簇比较接近期望值。方差的平方根称为 `标准差(standard variance)` 。

`协方差(covariance)` 两个变量线性相关的强度以及这些变量的尺度：

$$Cov(f(x), g(y)) = E[(f(x) - E[f(x)])(g(y) - E[g(y)])$$

协方差用于衡量两个变量的总体误差，方差是协方差一种特殊情况。公式上看协方差表示的是两个变量总体误差的期望。变化趋势一致则协方差是正。 `相关系数(correlation)` 将每个变量的值归一化，为了只衡量变量的相关性而不受变量尺度大小影响。

独立性比零协方差的要求更强，独立性排除了非线性关系。

随机向量$x \in \mathbb{R}^n$ 协方差矩阵(covariance matrix)是$n \times n$的矩阵。满足：

$$Cov(x)_{i,j} = Cov(x_i, x_j)$$

协方差矩阵对角元素是方差：

$$Cov(x_i, x_i) = Var(x_i)$$

```
x = np.array([1,2,3,4,5,6,7,8,9])
y = np.array([9,8,7,6,5,4,3,2,1])
Mean = np.mean(x)

Var = np.var(x) # 默认总体方差
Var_unbias = np.var(x, ddof=1) # 样本方差（无偏方差）
Cov = np.cov(x, y)
Mean, Var, Var_unbias, Cov

(5.0,
 6.666666666666667,
 7.5,
 array([[ 7.5, -7.5],
        [-7.5,  7.5]]))
```

3.9 常用概率分布

3.9.1 Bernouli分布

Bernoli分布(Bernoli distribution)是单个二值随机变量分布。它由单个参数$\phi \in [0, 1]$控制，ϕ给出了随机变量等于1的概率。具有如下性质：

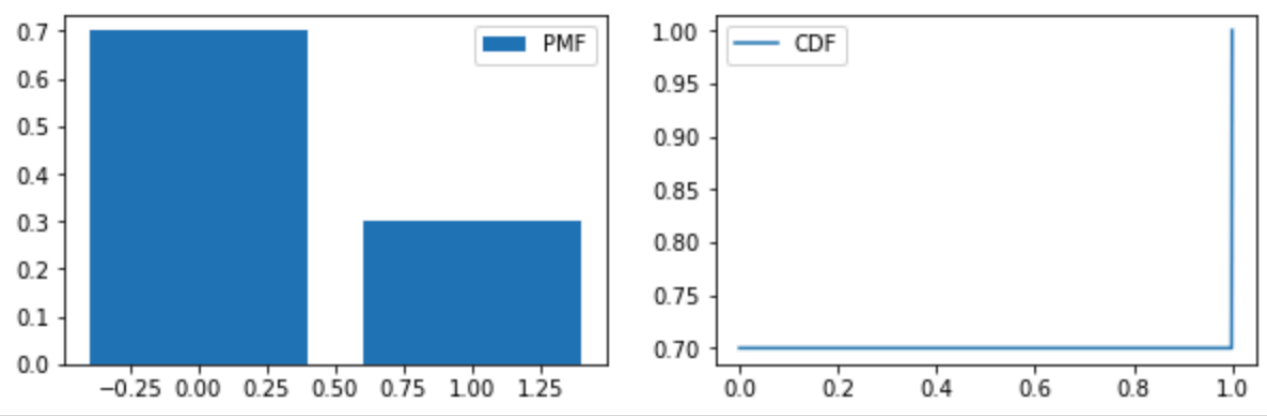
$$P(\mathrm{x} = 1) = \phi$$
$$P(\mathrm{x} = 0) = 1 - \phi$$
$$P(\mathrm{x} = x) = \phi^x (1 - \phi)^{1-x}$$
$$\mathbb{E}_{\mathrm{x}}[\mathrm{x}] = \phi$$
$$Var_{\mathrm{x}}(\mathrm{x}) = \phi(1 - \phi)$$

```
def plot_distribution(X, axes=None):
    """ 给定随机变量，绘制 PDF，PMF，CDF
        PDF：概率密度函数（probability density function）在数学中，连续型随机变量的概率密度函数是一个描述这个随机变量的输出值，在某个确定的取值点附近的可能性的函数。
        PMF : 概率质量函数（probability mass function），在概率论中，概率质量函数是离散随机变量在各特定取值上的概率。
        CDF : 累积分布函数（cumulative distribution function），又叫分布函数，是概率密度函数的积分，能完整描述一个实随机变量X的概率分布
    """
    if axes is None:
        fig, axes = plt.subplots(1, 2, figsize=(10, 3))
    x_min, x_max = X.interval(0.99)
    print(x_min, x_max)
    x = np.linspace(x_min, x_max, 1000)
    if hasattr(X.dist, 'pdf'): # 判断有没有 pdf，即是不是连续分布
        axes[0].plot(x, X.pdf(x), label="PDF")
        axes[0].fill_between(x, X.pdf(x), alpha=0.5) # alpha 是透明度，alpha=0 表示 100% 透明，alpha=100 表示完全不透明
    else: # 离散分布
        x_int = np.unique(x.astype(int))
        print(x.size)
        print(x_int)
        print(X.pmf(x_int), X.pmf(x_int).size)
        axes[0].bar(x_int, X.pmf(x_int), label="PMF") # pmf 和 pdf 是类似的
    print(X.cdf(x).size)
    axes[1].plot(x, X.cdf(x), label="CDF")
    for ax in axes:
        ax.legend()
    return axes

from scipy.stats import bernoulli
fig, axes = plt.subplots(1, 2, figsize=(10, 3)) # 画布
p = 0.3
```

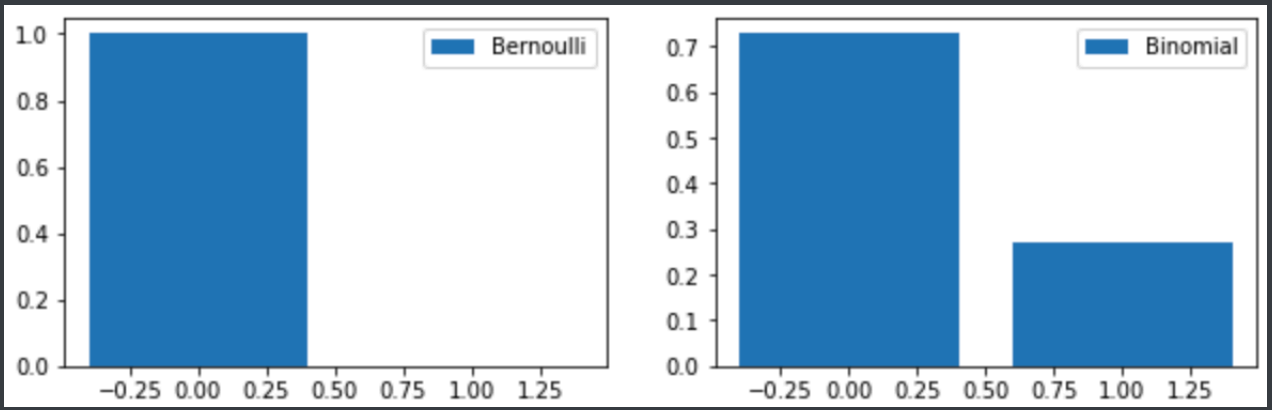
```
X = bernoulli(p) # 伯努利分布
plot_distribution(X, axes=axes)
```

```
0.0 1.0
1000
[0 1]
[0.7 0.3] 2
1000
```




```
# 生成的概率
possibility = 0.3
def trials(n_samples):
    samples = np.random.binomial(n_samples, possibility)
    proba_zero = (n_samples - samples) / n_samples
    proba_one = samples / n_samples
    return [proba_zero, proba_one]

fig, axes = plt.subplots(1, 2, figsize=(10, 3))
# 一次试验，伯努利分布
n_samples = 1
axes[0].bar([0, 1], trials(n_samples), label='Bernoulli')
# n次试验，二项分布
n_samples = 1000
axes[1].bar([0, 1], trials(n_samples), label='Binomial')
for ax in axes:
    ax.legend()
```



3.9.2 Multinoulli分布

Multinoulli分布(Multinoulli distribution)或者范畴分布(categorical distrubution)是指在具有k个不同状态的单个离散型随机变量上的分布，其中k是一个有限值。Multinoulli分布由向量$p \in [0,1]^{k-1}$参数化，其中每一个分量p_i表示第i个状态的概率。最后第k个状态概率可以通过$1-1^T p$给出。(个人理解：这里乘1矩阵是为了让p中所有概率值相加)必须限制$1^T p < 1$。Multinoulli分布经常用来表示对象分类的分布。Multinoulli分布是多项式分布(multinomial distribution)的一个特例。

伯努利分布(Bernoulli distribution)又称两点分布或0-1分布。伯努利试验是只有两种可能结果的单次随机试验。如果将试验独立重复进行n次称为n重Bernoulli分布

二项分布(binomial distribution)是n重Bernoulli试验成功次数的离散概率分布。伯努利分布是二项分布在$n=1$时的特例。

多项式分布(Multinomial Distribution)是二项式分布的推广。二项式做n次伯努利试验，规定了每次试验只有两个结果。如果做n次试验，每次试验结果可以有m多个，且m个结果发生的概率互斥，和为1。

Multinoulli分布是多项式分布的一个特例。多项式分布是$0,\ldots,n^k$中的向量分布，用于表示当对Multinoulli分布采样n次。多项式分布n=1就是Multinoulli分布。

范畴分布 (Multinoulli Distribution) 是指在具有 *k* 个不同值的单个离散型随机变量上的分布:

$$p(\mathrm{x}=x)=\prod_i \phi_i^{x_i}$$

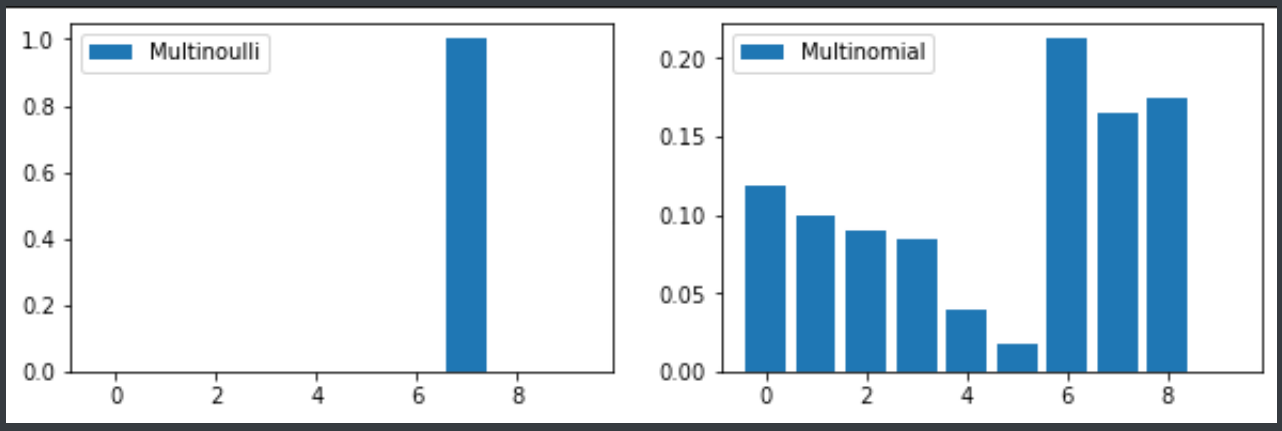
例如每次试验的结果就可以记为一个 *k* 维的向量，只有此次试验的结果对应的维度记为 1，其他记为 0。

```
def k_possibilities(k):
    """
    随机产生一组 10 维概率向量
    """
    res = np.random.rand(k)
    print(res)
    _sum = sum(res)
    print(_sum)
    for i, x in enumerate(res):
        res[i] = x / _sum
    return res

fix, axes = plt.subplots(1, 2, figsize=(10, 3))
# 一次试验,  范畴分布
k, n_samples = 10, 1
res = k_possibilities(k)
print('res:', res)
samples = np.random.multinomial(n_samples, res, size=1)[0] # n_samples=1
则只做一次试验，这次试验落在index=4位置，值为1。（若做两次试验则还会有一个值落在res
其中一个）。若size=2是上述试验执行两次。返回两个数组。
print('samples:', samples)
axes[0].bar(range(len(samples)), samples / n_samples,
label='Multinoulli')
# n次试验，多项式
n_samples = 1000
samples = np.random.multinomial(n_samples, res)
print('1000 samples:', samples)
axes[1].bar(range(len(samples)), samples / n_samples,
label='Multinomial')
for ax in axes:
    ax.legend()

[0.4980028  0.40898747 0.33920712 0.34476508 0.19825153 0.08870078
 0.96354451 0.79039993 0.60738862 0.00252534]
4.241773181122987
```

```
res: [0.11740439 0.09641899 0.07996824 0.08127853 0.04673789 0.02091125
 0.22715607 0.18633715 0.14319215 0.00059535]
samples: [0 0 0 0 0 0 0 1 0 0]
1000 samples: [119 100  90  84  39  17 212 165 174   0]
```



3.9.3 高斯分布

正态分布(normal distribution)也称高斯分布(Gaussian distribution)：

$$\mathcal{N}(x;\mu,\sigma^2)=\sqrt{\frac{1}{2\pi\sigma^2}}exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

图中心x由μ给出。峰宽度受σ控制。标准正态分布(standard normal distribution)其中$\mu=0,\sigma=1$。
μ是分布的均值：$\mathbb{E}[\mathrm{x}]=\mu$，分布标准差$\sigma$，方差$\sigma^2$
当需要对概率密度函数求值时，对σ^2取倒数。用方差倒数控制精度(presicion)：

$$\mathcal{N}(x;\mu,\beta^{-1})=\sqrt{\frac{\beta}{2\pi}}exp\left(-\frac{1}{2}\beta(x-\mu)^2\right)$$

中心极限定理(central limit theorem)说明很多独立随机变量的和近似服从正态分布。

正态分布可以推广到$\mathbb{R}^n$空间，称为多维正态分布(multivariate normal distribution)可以用正定对称矩阵$\sum$参数化：

$$\mathcal{N}(x;\mu,\sum)=\sqrt{\frac{1}{(2\pi)^n det(\sum)}}exp\left(-\frac{1}{2}(x-\mu)^T\sum^{(-1)}(x-\mu)\right)$$

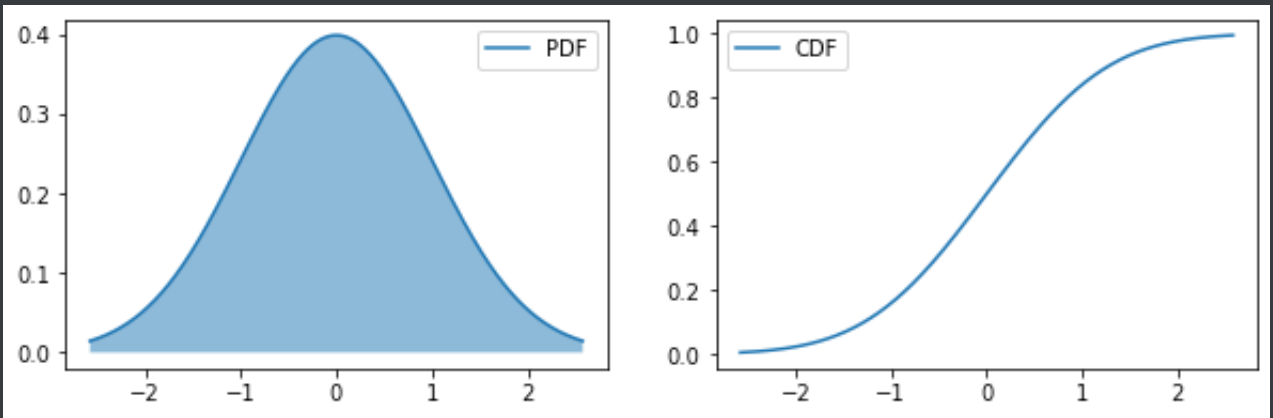
μ代表均值，只是现在是向量值。参数$\sum$给出了分布的协方差矩阵。协方差矩阵形式不高效，求$\sum$逆。用 精度矩阵(presicion matrix) 替代：

$$\mathcal{N}(x;\mu,\beta^{-1}) = \sqrt{\frac{det(\beta)}{(2\pi)^n}} exp\left(-\frac{1}{2}(x-\mu)^T\beta(x-\mu)\right)$$

常把协方差矩阵固定成对角阵。更简单形式是 各向同性(isotropic) 高斯分布，它们的协方差矩阵是标量乘以单位矩阵。

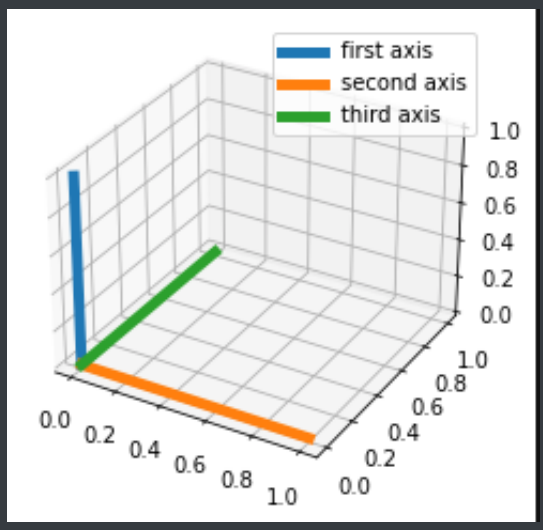
```
from scipy.stats import norm
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
mu, sigma = 0, 1
X = norm(mu, sigma) # 标准正态分布
plot_distribution(X, axes=axes)

-2.5758293035489004 2.5758293035489004
1000
```



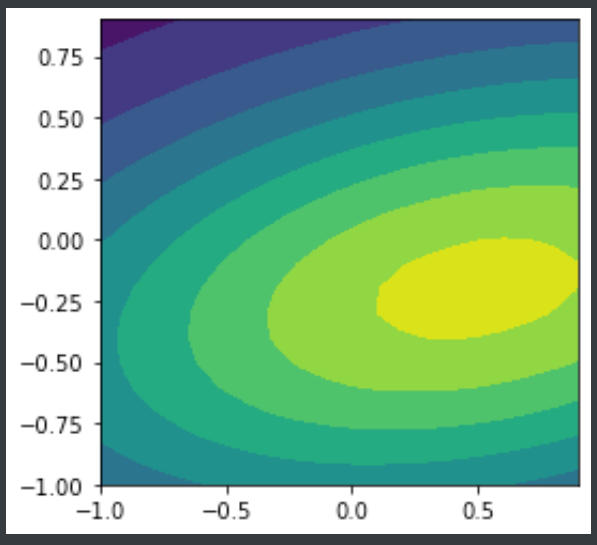

```
# 绘制多维数组拼接以及分割的三个轴方向，first axis为竖直方向，second axis为水平方向，third axis为深度方向
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = plt.axes(projection='3d')
x = [0, 0]
y = [0, 0]
z = [0, 1]
```

```
ax.plot(x, y, z, label='first axis', linewidth=5)
x = [0, 1]
y = [0, 0]
z = [0, 0]
ax.plot(x, y, z, label='second axis', linewidth=5)
x = [0, 0]
y = [0, 1]
z = [0, 0]
ax.plot(x, y, z, label='third axis', linewidth=5)
ax.legend()
# 绘制下三维度的图像，为了解释vstack hstack dstack
```




```
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
# np.mgrid[ 第1维，第2维 ，第3维 ，  …] 每一维度，a:b:c，c表示步长，为实数表示间隔；该为长度为[a,b),左开右闭。a:b:cj，cj表示步长，为复数表示点数；该长度为[a,b]，左闭右闭
x, y = np.mgrid[-1:1:0.1, -1:1:0.1]
# vstack、hstack和dstack都用于把几个小数组合并成一个大数组。它们的差别是小数组的元素在大数组中的排列顺序有所不同。a、b二者在0号轴上连接起来。
# vstack垂直合并，a=[1,2] b=[3,4] np.vstack((a,b)) => [[1 2] [3 4]]
# hstack表示轴1合并。hstack的字母h来自于horizontal，表示两个数组是水平的，hstack((a,b))将把b排在a的右边的意思。
# np.hstack([[1,2],[3]]) => [1,2,3] 没有增维，结果不像vstack，对一维数组合并的结果还是一维的。
```

```
    # a = [[1,2],
    #      [3,4]]
    # b = [[5],
    #      [6]]
    # print(np.hstack([a,b]))
    # [[1 2 5]
    #  [3 4 6]]
    # dstack是说deep stack了，它是轴2合并。dstack([a,b])把a和b摞在一起，像一摞扑克牌一样。
    # a = np.array([1,2]) b = np.array([3,4]) => np.dstack((a,b))
    # [[[1 3]
    #   [2 4]]]
    pos = np.dstack((x, y))
    fig = plt.figure(figsize=(4, 4))
    axes = fig.add_subplot(111)
    mu = [0.5, -0.2] # 均值
    sigma = [[2.0, 0.3], [0.3, 0.5]] #协方差矩阵
    X = multivariate_normal(mu, sigma)
    axes.contourf(x, y, X.pdf(pos)) # contourf 画等高线和填充等高线
```



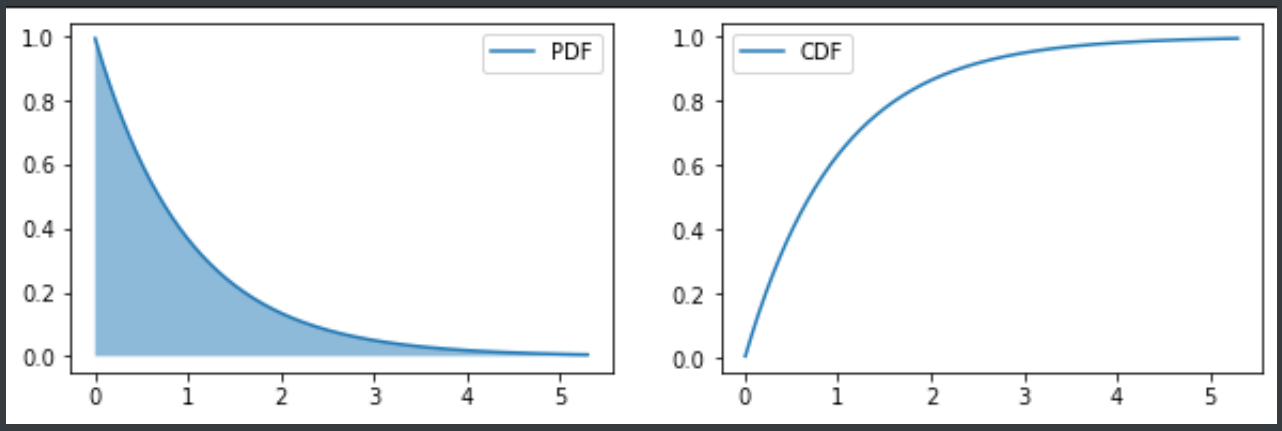
3.9.4 指数分布和Laplace分布

当需要在$x = 0$处取得边界点(sharp point)。可以用 `指数分布(exponential distribution)`：

$$p(x;\lambda) = \lambda 1_{x\geq 0} exp(-\lambda x)$$

指示函数(indicator function)$1_{x\geq 0}$使x取负值时，概率为零。是用于在 $x = 0$ 处获得最高的概率的分布，其中 $\lambda > 0$ 是分布的一个参数，常被称为率参数 (Rate Parameter)。

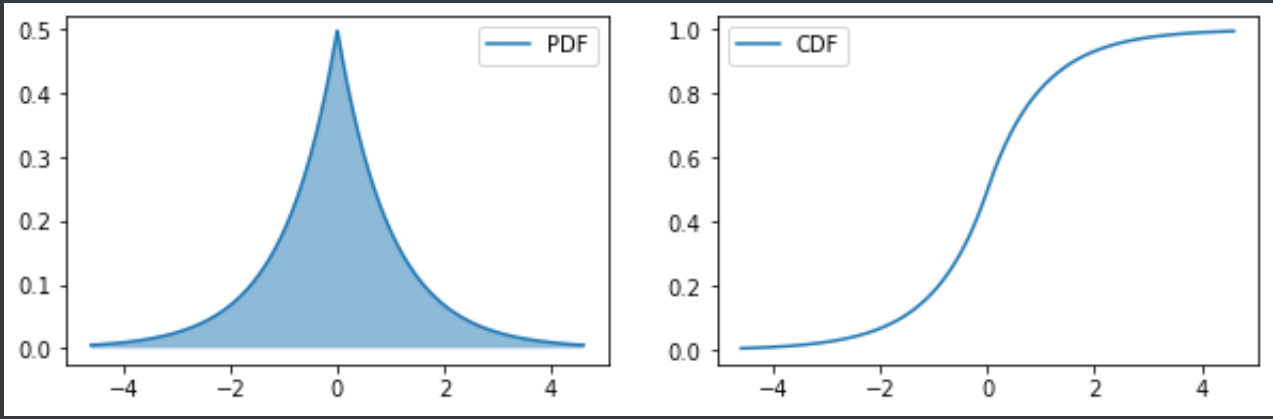
```
from scipy.stats import expon
fig, axes = plt.subplots(1, 2, figsize=(10, 3)) # 其中figsize=(a, b)用来设置图形的大小，a为图形的宽， b为图形的高
# 定义scale = 1 / lambda
X = expon(scale=1)
plot_distribution(X, axes=axes)
```



一个相似的概率分布是Laplace分布，允许在任意一点μ处设置概率质量的峰值。

$$Laplace(x;\mu,\gamma) = \frac{1}{2\gamma} exp\left(-\frac{|x-\mu|}{\gamma}\right)$$

```
from scipy.stats import laplace
fix, axes = plt.subplots(1, 2, figsize=(10, 3))
mu, gamma = 0, 1
X = laplace(loc=mu, scale=gamma)
plot_distribution(X, axes=axes)
```



3.9.5 Dirac分布和经验分布

当希望概率分布集群点围绕一个点，通过 Dirac delta函数（Dirac delta function） $\delta(x)$ 实现这样的概率密度函数:

$$p(x) = \delta(x - \mu)$$

Dirac delta函数定义成除了0以外的所有点的值都为0，但是积分为1。Dirac delta函数不像普通函数一样对x的每一个值都有一个实数值的输出，它是一种不同类型的数学对象，被称为 广义函数（generalized function） 。

通过把$p(x)$定义成δ函数左移$-\mu$个单位，我们得到$x = \mu$处具有无限窄也无限高的峰值的概率质量。
Dirac分布经常作为 经验分布(empirical distribution) 一个组成部分出现:

$$\hat{p}(x)\frac{1}{m}\sum_{i=1}^{m}\delta(x - x^{(i)})$$

经验分布将概率密度$\frac{1}{m}$赋给m个点$x^{(1)}, \ldots, x^{(m)}$中的每一个。这些点是给定的数据集或者采样集合。只有在定义连续性随机变量的经验分布时Dirac delta函数才是必要的。对于离散型随机变量，情况更加简单：经验分布也可以定义成Multinoulli分布，对于每一个可能的输入其概率可以设为，在训练集上那个输入值的 经验频率(empirical frequency) 。
在训练集上训练模型时，这个训练集上得到的经验分布指明了采样来源的分布。关于经验分布另一种观点是，它是训练数据的似然最大的那个概率密度函数(见5.5节)。

3.9.6 分布的混合

组合一些简单的概率分布定义新的概率分布。通用的组合方法是构造 混合分布(mixtrue distribution) 。混合分布由一些组件(component)分布构成。每次实验，样本是由哪个组件分布产生的取决于从一个Multinoulli分布中采样的结果：

$$P(x) = \sum_i P(c=i)P(x|c=i)$$

这里P(c)是对各组件的一个Multinoulli分布。有过混合分布的例子：实值变量的经验分布对于每一个训练实例来说，就是以Dirac分布为组件的混合分布。
混合模型是组合简单概率分布来生成更丰富的分布的一种简单策略。十六章再讨论。
混合模型使我们能一瞥以后会用到的一个非常重要的概念： 潜变量(latnet varibale) 。潜变量是不能直接观测到的随机变量。混合模型的组件标识变量c就是一个例子。潜变量在联合分布中可能和x有关，此时$P(x,c) = P(x|c)P(c)$。潜变量的分布P(x)以及关联潜变量和观测变量的条件分布$P(x|c)$共同决定了分布$P(x)$的形状。
一个强大且常见的混合模型是 高斯混合模型(Gaussion Mixtrue Model) 它的组件$p(x|c=i)$是高斯分布。每个组件都有各自的参数。均值$\mu^{(i)}$和协方差矩阵$\sum^{(i)}$。高斯混合模型参数指明了每个组件i的 先验概率(prior probability) $\alpha_i = P(c=i)$。 先验 一词表明在观测到x之前传递给模型关于c的置信度。作为对比，$P(c|x)$是 后验概率(posterior probability) 因为它是在观测x之后做的计算。高斯混合模型是概率密度的 万能近似器(universal approximator) ，在这种意义下，任何平滑的概率密度都可以用具有足够多组件的高斯混合模型以任意精度来逼近。

3.10 常用函数的有用性质

logistic sigmoid 函数：

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

logistic sigmoid函数通常用来产生Bernoulli分布中的参数ϕ(即伯努利分布中，随机变量等于1的概率)因为范围是$(0,1)$。函数在取非常大正值或负值会出现 饱和(saturate) ，对输入微小改变会变得不敏感。
softplus 函数：

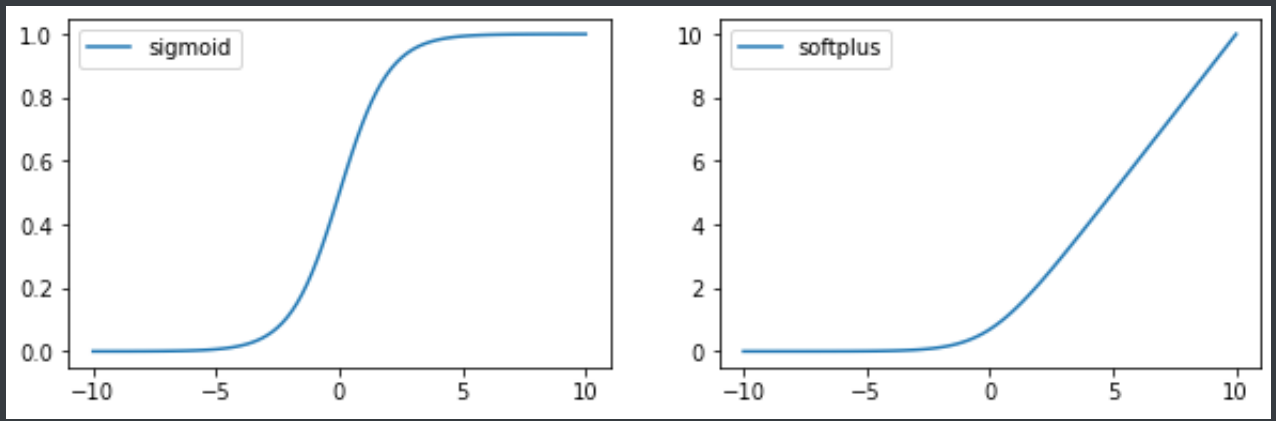
$$\zeta(x) = log(1+\exp(x))$$

softplus函数可生成正态分布的β和σ参数。因为范围是$(0, \infty)$。softplus是$x^+ = \max(0, x)$的平滑，软化。给出一些性质：

$$\sigma(x) = \frac{\exp(x)}{\exp(x) + \exp(0)}$$
$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$
$$1 - \sigma(x) = \sigma(-x)$$
$$\log \sigma(x) = -\zeta(-x)$$
$$\frac{d}{dx}\zeta(x) = \sigma(x)$$
$$\forall x \in (0, 1), \sigma^{-1}(x) = \log(\frac{x}{1 - x})$$
$$\forall x > 0, \zeta^{-1}(x) = \log(\exp(x) - 1)$$
$$\zeta(x) = \int_{-\infty}^{x} \sigma(y)dy$$
$$\zeta(x) - \zeta(-x) = x$$

函数σ^{-1}在统计学中称为 分对数(logit)，机器学习很少用。

```
x = np.linspace(-10, 10, 100)
sigmoid = 1 / (1 + np.exp(-x))
softplus = np.log(1 + np.exp(x))
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
axes[0].plot(x, sigmoid, label='sigmoid')
axes[1].plot(x, softplus, label='softplus')
for ax in axes:
    ax.legend()
```



```
def H(sentence):
    """最优编码长度"""
    entropy = 0
    # 这里有 256 个可能的 ASCII 符号
    for character_i in range(256):
        Px = sentence.count(chr(character_i)) / len(sentence)
        if Px > 0:
            entropy += -Px * math.log(Px, 2)
    return entropy
import random
import math
simple_message = "".join([chr(random.randint(0, 64)) for i in
range(500)])
print(simple_message)
H(simple_message)
-------------------------
'03-●;●.●●",●=  (;●#●●?/;,●●'1●2●●●●<1● 2●●6'#●3●@08(14 $&-'>●●●●#●?●●(●
<*+.●
<●.' ●●8/@●
●●●●●●7●1●●6●●●/(2#●(-.9
#(/@●●
"●+4●?●(2●●0=45$)'●

)*!●#●$ .'●●&8●
●●6+●$●<

6●?.7●,7?3;+8●5●96,1●●●*&<
●8-●"=2●6:●●●●6#.●%;$%●-09?●%●●!*8;●.@0
●?2●●●●<:●●,2 ●"$●! ●●60@9●●/=●5●8"●●●●"●1/9)●2●/●)●●●:;●● ●3●=●47●%/
●3●●99●;(@5●3●●(,:<'●!=●●●<●+●!+●%/?&/●/●●●●4●●5●,8<1
●68'+8?=●●●7/.●●●●4● *0# +●
●?●●.
5.918814777080752
```

3.11贝叶斯规则

经常需要在已知$P(y|x)$时计算$P(x|y)$，如果还知道$P(x)$，我们可以用 贝叶斯规则(Bayes' rule) 来实现：

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)}$$

$P(y)$可以用$P(y) = \sum_x P(y|x)P(x)$计算，so不用先知道$P(y)$。

3.12 连续型变量的细节

连续型随机变量和概率密度的深入理解需要用到数学分支 测度论(measure theory) 。

3.13 信息论

信息论是应用数学的一个分支，主要研究一个信号包含信息的多少进行量化。最初被发明用来研究在一个含有噪声的信道上用离散的字母表来发送消息，例如通过无线电来传输通信。此时，信息论告诉我们如何对消息设计最优编码以及计算消息的期望长度，这些消息是使用不同编码机制，从特定概率分布上采样得到的。
信息论基本想法是一个不太可能的事件居然发生了，要比一个非常可能的事件发生，能提供更多的信息。

- 非常可能发生的事件信息量要比较少，并且极端情况下，确保能够发生的事件应该没有信息量。
- 较不可能发生的事件更有更好的信息量。
- 独立事件应该具有增量的信息。例如，投硬币两次正面朝上传递的信息量，应该是投掷一次硬币正面朝上信息量的两倍。
  为了满足上述三个性质，我们定义一个事件$\mathrm{x} = x$的 自信息(self- information) 为：

$$I(x) = -\log P(x)$$

本书用log表示自然对数，底数是e。so我们定义$I(x)$单位是 奈特(nats) 。一奈特是$\frac{1}{e}$的概率观测到一个事件时获得的信息量。其他材料底数为2的对数，单位是 比特(bit) 或者 香农(shaonnons) ；通过比特度量的信息只是通过奈特度量信息的常数倍。
自信息只处理单个输出。我们可以用 香农熵(Shannon entropy) 对整个概率分布的不确定性总量进行量化：

$$H(x) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log P(x)]$$

也记作$H(P)$。换言之，一个分布的香农熵是指遵循这个分布的事件所产生的期望信息总量。它给出了对依据概率分布P生成的符号进行编码所需的比特数在平均意义上的下界。近似确定性的分布具有较低香农熵。接近均匀分布的概率分布具有较高的熵。当x是连续时，香农熵称为 `微分熵(differential entropy)` 。

如果我们对同一个随机变量x有两个单独的概率分布P(x)和Q(x)。可以使用 `KL散度(Kullback-Leibler(KL) divergence)` 来衡量这两个分布的差异：

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

在离散型变量的情况下，KL散度衡量的是，当我们使用一种被设计成能够使得概率分布Q产生的消息长度的最小编码，发送包含由概率分布P产生的符号的消息时，所需要的额外信息量。

KL散度非负。当P和Q在离散型变量情况下是相同的分布，KL散度为0。或者在连续变量情况下是“几乎处处”相同的。因为KL散度非负，衡量两个分布的差异，它经常被用作分布之间的某种距离。但并不是真的距离，它不是对称的。对于某些P和Q，$D_{KL}(P||Q) \neq D_{KL}(Q||P)$ 。

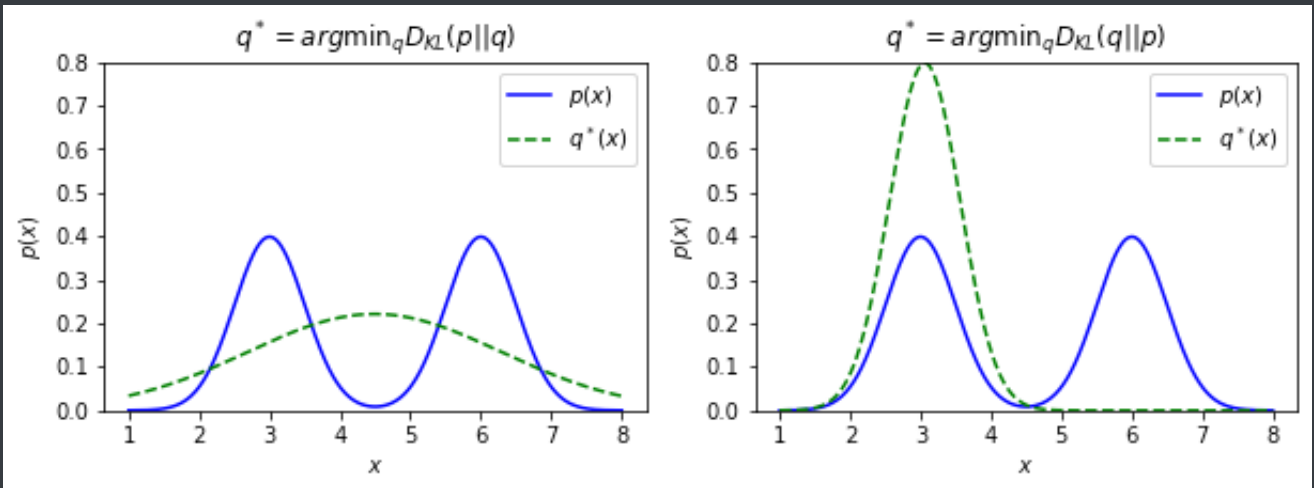
和KL散度密切联系的量是 `交叉熵(cross-entropy)` $H(P,Q) = H(P) + D_{KL}(P||Q)$，和KL散度像，但缺少左边一项：

$$H(P,Q) = -\mathbb{E}_{x\sim P}\log Q(x)$$

针对Q最小化交叉熵等价于最小化KL散度。当遇到$0\log 0$这个表达式。在信息论中，将这个表达式处理为$\lim_{x\to 0} x\log x = 0$

```
# D(P||Q) 与 D(Q||P) 比较
x = np.linspace(1, 8, 500)
y1 = norm.pdf(x, 3, 0.5) # normpdf: 正态概率密度函数 Y =
normpdf(X,mu,sigma)
y2 = norm.pdf(x, 6, 0.5)
p = y1 + y2 # 构造p(x)
# print(p)
KL_pq, KL_qp = [], []
q_list = []
for mu in np.linspace(0, 10, 50):
```

```
    for sigma in np.linspace(0.1, 5, 50): # 寻找最优的q(x)
        q = norm.pdf(x, mu, sigma)
        q_list.append(q)
        KL_pq.append(entropy(p, q))
        KL_qp.append(entropy(q, p))
KL_pq_min = np.argmin(KL_pq)
KL_qp_min = np.argmin(KL_qp)
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
axes[0].set_ylim(0, 0.8)
axes[0].plot(x, p / 2, 'b', label='$p(x)$')
axes[0].plot(x, q_list[KL_pq_min], 'g--', label='$q^*(x)$')
axes[0].set_xlabel('$x$')
axes[0].set_ylabel('$p(x)$')
axes[0].set_title('$q^*={arg\min}_q D_{KL}(p||q)$')
axes[1].set_ylim(0, 0.8)
axes[1].plot(x, p/2, 'b', label='$p(x)$')
axes[1].plot(x, q_list[KL_qp_min], 'g--', label='$q^*(x)$')
axes[1].set_xlabel('$x$')
axes[1].set_ylabel('$p(x)$')
axes[1].set_title('$q^*= {arg\min}_ q D_{KL}(q||p)$')
for ax in axes:
    ax.legend(loc='upper right')
```



3.14 结构化概率模型

机器学习算法会涉及到非常多的随机变量上的概率分布。概率分布涉及到的直接相互作用都是介于非常少的变量之间。使用单个函数来描述整个联合概率分布是非常低效的。
可以把概率分布分解成许多因子的乘积形式，而不是使用单一的函数来表示概率分布。例如假设我们有三个随机变量a, b和c，并且a影响b的取值，b影响c的取值，但是a和c在给定b时是条件独立的。可以把全部三个变量概率分布重新表示为两个变量概率分布连乘形式：

$$p(a,b,c) = p(a)p(b|a)p(c|b)$$

分解可以减少用来描述分布的参数数量。每个因子使用的参数数目是它的变量数目的指数倍。意味着，能够找到一种使每个因子分布具有更少变量的分解方法，我们能极大降低表示联合分布的成本。
可以用图来描述这种分解。由一些可以通过边互相连接的顶点的集合构成。当用图来表示这种概率分布的分解，我们把它称为 `结构化概率模型(structured probabilistic model)` 或者 `图模型(graphical model)` 。有向图和无向图都可以用G表示。其中图中每个节点对应一个随机变量。连接两个随机变量的边意味着概率分布可以表示成这两个随机变量之间的直接作用。
`有向(directed)` 模型使用带有有向边的图，他们用条件概率分布来表示分解。有向模型对于分布中的每一个随机变量x_i都包含着一个影响因子。这个组成x_i条件概率的影响因子被称为x_i的父节点，记为$P_{aG}(x_i)$:

$$p(\mathrm{x}) = \prod_i p(x_i|P_{aG}(x_i))$$

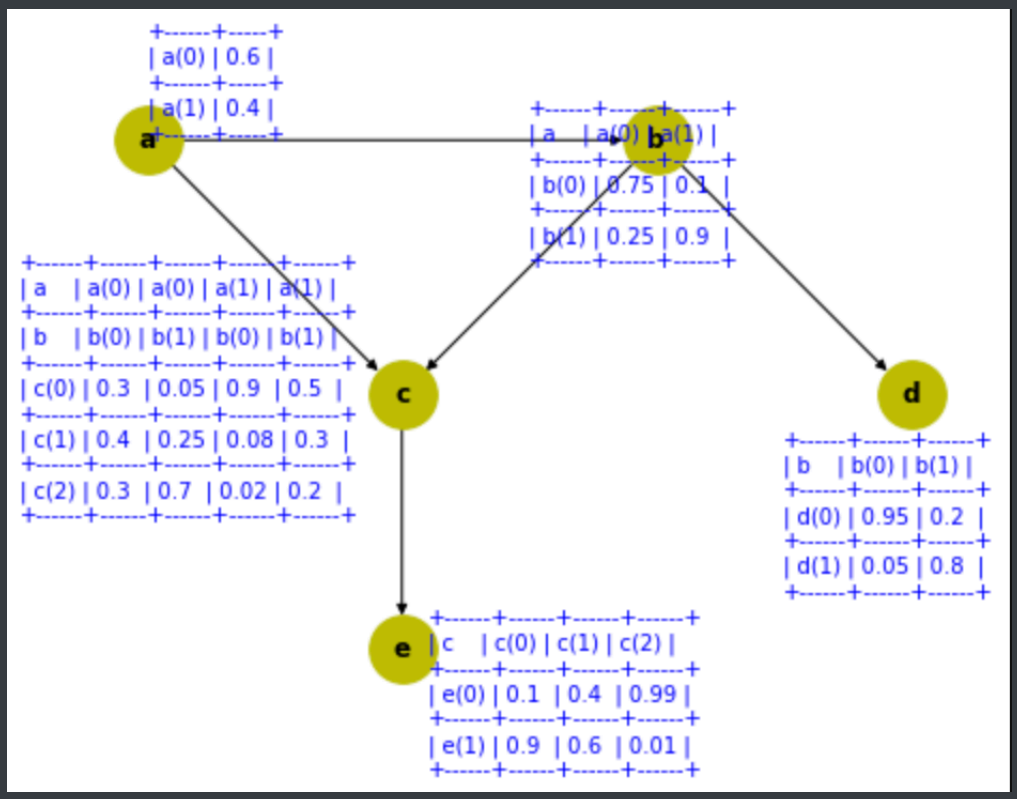
。
`无向(undirected)` 模型使用带有无向边的图，它们将分解表示成一组函数；不像有向模型那样，这些函数不是任何类型的概率分布。G中任何满足两两之间有边连接的顶点集合被称为团。无向模型中的每个团C^i都伴随着一个因子$\phi^i(C^{(i)})$。这些因子仅仅是函数，并不是概率分布。每个因子的输出必须是非负的，但并没有要求像概率分布中那样要求因子的和或者积分为1。
随机变量的联合概率与所有这些因子的乘积 `成比例(proportional)` 意味着因子的值越大则可能性越大。当然不能保证这种乘积求和为1。so要除以一个归一化常数Z来得到归一化的概率分布，归一化常数Z被定义为ϕ函数乘积的所有状态求和或积分，概率分布为：

$$p(x) = \frac{1}{Z}\prod_i \phi^{(i)}(C^{(i)})$$

图模型表示的分解仅仅是描述概率分布的一种语言。有向或者无向不是概率分布的特性；它仅是概率分布的一种特殊 描述(description) 所具有的特性，而任何概率分布都可以用这两种方式进行描述。

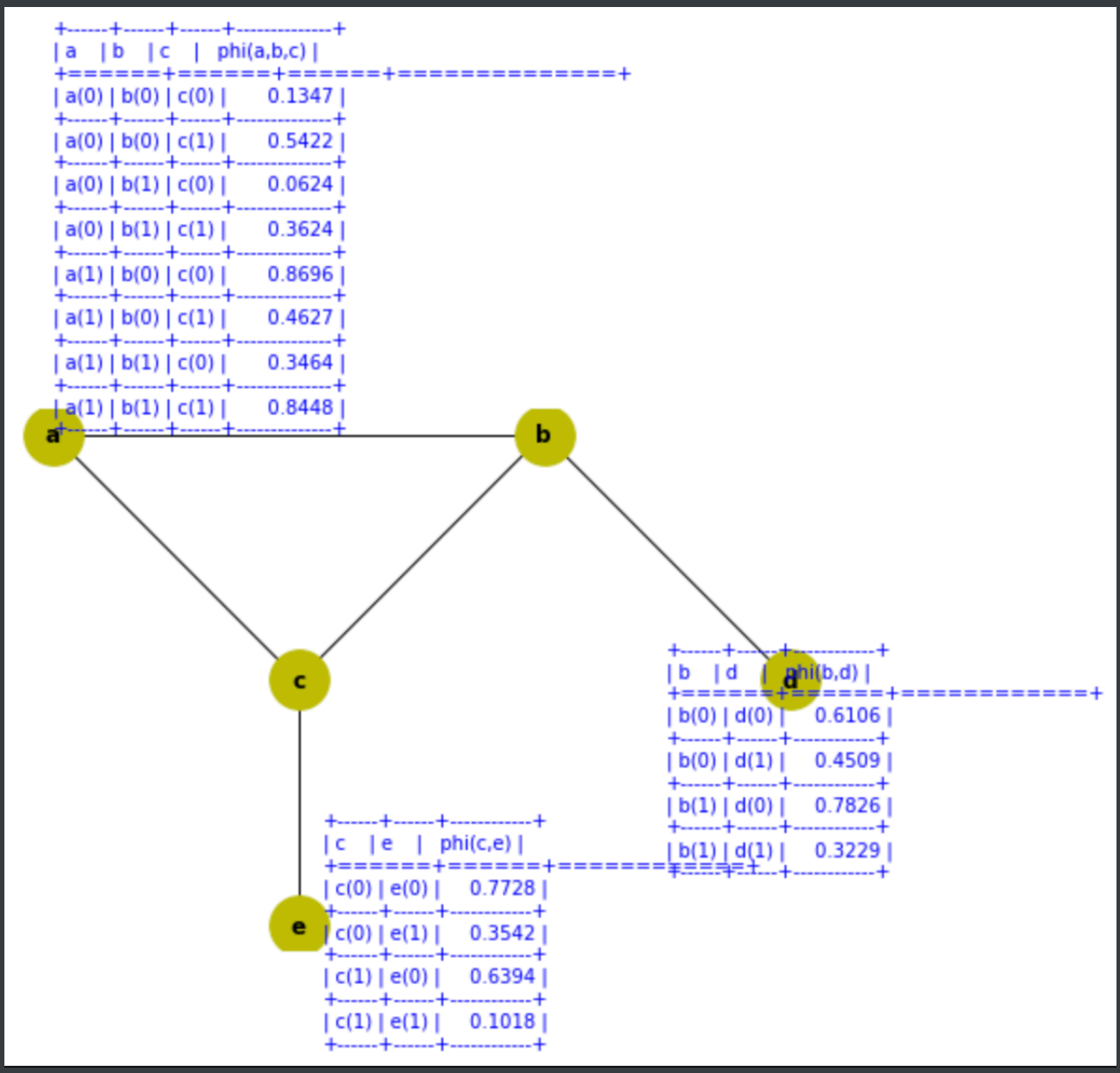
```
import networkx as nx
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
import matplotlib.pyplot as plt
# 建立一个简单贝叶斯模型框架
model = BayesianModel([('a', 'b'), ('a', 'c'), ('b', 'c'), ('b', 'd'),
('c', 'e')])
# 最顶层的父节点的概率分布表
cpd_a = TabularCPD(variable='a', variable_card=2, values=[[0.6], [0.4]])
# a: (0,1)
# 其它各节点的条件概率分布表（行对应当前节点索引，列对应父节点索引）
cpd_b = TabularCPD(variable='b', variable_card=2, # b: (0,1)
                   values=[[0.75, 0.1],
                           [0.25, 0.9]],
                   evidence=['a'],
                   evidence_card=[2])
cpd_c = TabularCPD(variable='c', variable_card=3, # c: (0,1,2)
                   values=[[0.3, 0.05, 0.9, 0.5],
                           [0.4, 0.25, 0.08, 0.3],
                           [0.3, 0.7, 0.02, 0.2]],
                   evidence=['a', 'b'],
                   evidence_card=[2, 2])
cpd_d = TabularCPD(variable='d', variable_card=2, # d: (0,1)
                   values=[[0.95, 0.2],
                           [0.05, 0.8]],
                   evidence=['b'],
                   evidence_card=[2])
cpd_e = TabularCPD(variable='e', variable_card=2, # e: (0,1)
                   values=[[0.1, 0.4, 0.99],
                           [0.9, 0.6, 0.01]],
                   evidence=['c'],
```

```
                    evidence_card=[3])
# 将各节点的概率分布表加入网络
model.add_cpds(cpd_a, cpd_b, cpd_c, cpd_d, cpd_e)
# 验证模型数据的正确性
print(u"验证模型数据的正确性:",model.check_model())
# 绘制贝叶斯图（节点 + 依赖关系）
nx.draw(model, with_labels=True, node_size=1000, font_weight='bold',
node_color='y', pos={"e":[4,3],"c":[4,5],"d":[8,5],"a":[2,7],"b":[6,7]})
plt.text(2,7,model.get_cpds("a"), fontsize=10, color='b')
plt.text(5,6,model.get_cpds("b"), fontsize=10, color='b')
plt.text(1,4,model.get_cpds("c"), fontsize=10, color='b')
plt.text(4.2,2,model.get_cpds("e"), fontsize=10, color='b')
plt.text(7,3.4,model.get_cpds("d"), fontsize=10, color='b')
plt.show()
```




```
import networkx as nx
from pgmpy.models import MarkovModel
from pgmpy.factors.discrete import DiscreteFactor
import matplotlib.pyplot as plt
```

```
# 建立一个简单马尔科夫网
model = MarkovModel([('a', 'b'), ('a', 'c'), ('b', 'c'),('b', 'd'),
('c', 'e')])
# 各团因子（参数随机选择）
factor_abc = DiscreteFactor(['a', 'b', 'c'], cardinality=[2,2,2],
values=np.random.rand(8))
factor_bd = DiscreteFactor(['b', 'd'], cardinality=[2,2],
values=np.random.rand(4))
factor_ce = DiscreteFactor(['c', 'e'], cardinality=[2,2],
values=np.random.rand(4))
# 将各团因子加入网络
model.add_factors(factor_abc,factor_bd,factor_ce)
# 验证模型数据的正确性
print(u"验证模型数据的正确性:",model.check_model())
# # 绘制贝叶斯图（节点 + 依赖关系）
nx.draw(model, with_labels=True, node_size=1000, font_weight='bold',
node_color='y', \
pos={"e":[4,3],"c":[4,5],"d":[8,5],"a":[2,7],"b":[6,7]})
plt.text(2,7,model.get_factors()[0], fontsize=10, color='b')
plt.text(7,3.4,model.get_factors()[1], fontsize=10, color='b')
plt.text(4.2,2,model.get_factors()[2], fontsize=10, color='b')
plt.show()
```


a
b
c
d
e
a	b	c	phi(a,b,c)
a(0)	b(0)	c(0)	0.1347
a(0)	b(0)	c(1)	0.5422
a(0)	b(1)	c(0)	0.0624
a(0)	b(1)	c(1)	0.3624
a(1)	b(0)	c(0)	0.8696
a(1)	b(0)	c(1)	0.4627
a(1)	b(1)	c(0)	0.3464
a(1)	b(1)	c(1)	0.8448
b	d	phi(b,d)
b(0)	d(0)	0.6106
b(0)	d(1)	0.4509
b(1)	d(0)	0.7826
b(1)	d(1)	0.3229
c	e	phi(c,e)
c(0)	e(0)	0.7728
c(0)	e(1)	0.3542
c(1)	e(0)	0.6394
c(1)	e(1)	0.1018