概率与信息论

朱明超
Email: deityrayleigh@gmail.com
Github: github.com/MingchaoZhu/DeepLearning

1 概率

1.1 概率与随机变量

- **频率学派概率** (Frequentist Probability)：认为概率和事件发生的频率相关。
- **贝叶斯学派概率** (Bayesian Probability)：认为概率是对某件事发生的确定程度，可以理解成是确信的程度。
- **随机变量** (Random Variable)：一个可能随机取不同值的变量。例如：抛掷一枚硬币，出现正面或者反面的结果。

1.2 概率分布

1.2.1 概率质量函数

概率质量函数 (Probability Mass Function)：对于离散型变量，我们先定义一个随机变量，然后用 ~ 符号来说明它遵循的分布：$\mathrm{x} \sim P(\mathrm{x})$ ，函数 P 是随机变量 x 的 PMF。

例如, 考虑一个离散型 x 有 k 个不同的值，我们可以假设 x 是均匀分布的 (也就是将它的每个值视为等可能的)，通过将它的 PMF 设为：

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

对于所有的 i 都成立。

1.2.2 概率密度函数

当研究的对象是连续型时，我们可以引入同样的概念。如果一个函数 p 是概率密度函数 (Probability Density Function)：

- 分布满足非负性条件：$\forall x \in \mathrm{x}, p(x) \geq 0$
- 分布满足归一化条件：$\int_{-\infty}^{\infty} p(x)\, dx = 1$

例如在 (a, b) 上的均匀分布：

$$U(x; a, b) = \frac{\mathbf{1}_{ab}(x)}{b-a}$$

这里 $\mathbf{1}_{ab}(x)$ 表示在 (a, b) 内为 1，否则为 0。

1.2.3 累积分布函数

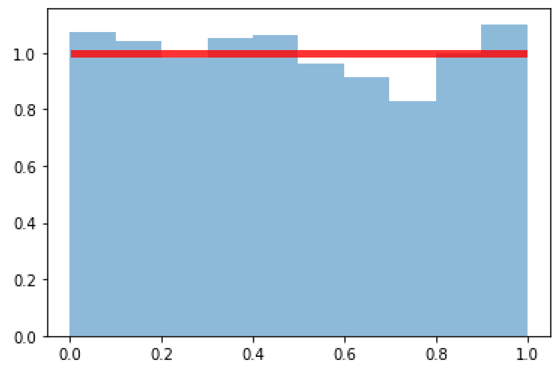
累积分布函数 (Cummulative Distribution Function) 表示对小于 x 的概率的积分：

$$\mathrm{CDF}(x) = \int_{-\infty}^{x} p(t)\, dt \tag{2}$$

[21]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import uniform
%matplotlib inline
```

[22]:
```
# 生成样本
fig, ax = plt.subplots(1, 1)
r = uniform.rvs(loc=0, scale=1, size=1000)
ax.hist(r, density=True, histtype='stepfilled', alpha=0.5)
# 均匀分布 pdf
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')
```

[22]:
```
[<matplotlib.lines.Line2D at 0x118521b38>]
```



1.3 条件概率与条件独立

边缘概率 (`Marginal Probability`)：如果我们知道了一组变量的联合概率分布，但想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被称为边缘概率分布。

$$\forall x \in \mathrm{x},\ P(\mathrm{x}=x)=\sum_{y} P(\mathrm{x}=x, \mathrm{y}=y) \tag{3}$$

条件概率 (`Conditional Probability`)：在很多情况下，我们感兴趣的是某个事件，在给定其他事件发生时出现的概率。这种概率叫做条件概率。我们将给定 $\mathrm{x}=x$，$\mathrm{y}=y$ 发生的条件概率记为 $P(\mathrm{y}=y \mid \mathrm{x}=x)$。这个条件概率可以通过下面的公式计算:

$$P(\mathrm{y}=y \mid \mathrm{x}=x)=\frac{P(\mathrm{y}=y, \mathrm{x}=x)}{P(\mathrm{x}=x)} \tag{4}$$

条件概率的链式法则 (`Chain Rule of Conditional Probability`)：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的条件概率相乘的形式:

$$P(x_1, \ldots, x_n)=P(x_1) \prod_{i=2}^{n} P(x_i \mid x_1, \ldots, x_{i-1}) \tag{5}$$

独立性 (`Independence`)：两个随机变量 x 和 y，如果它们的概率分布可以表示成两个因子的乘积形式，并且一个因子只包含 x 另一个因子只包含 y，我们就称这两个随机变量是相互独立的:

$$\forall x \in \mathrm{x}, y \in \mathrm{y},\ p(\mathrm{x}=x, \mathrm{y}=y)=p(\mathrm{x}=x) p(\mathrm{y}=y) \tag{6}$$

条件独立性 (`Conditional Independence`)：如果关于 x 和 y 的条件概率分布对于 z 的每一个值都可以写成乘积的形式，那么这两个随机变量 x 和 y 在给定随机变量 z 时是条件独立的。

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, z \in \mathrm{z},\ p(\mathrm{x}=x, \mathrm{y}=y \mid \mathrm{z}=z)=p(\mathrm{x}=x \mid \mathrm{z}=z) p(\mathrm{y}=y \mid \mathrm{z}=z) \tag{7}$$

1.4 随机变量的度量

期望 (`Expectation`)：函数 f 关于**概率分布** $P(\mathrm{x})$ 或 $p(\mathrm{x})$ 的期望表示为由概率分布产生 x，再计算 f 作用到 x 上后 $f(x)$ 的平均值。对于离散型随机变量，这可以通过求和得到:

$$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]=\sum_{x} \mathrm{P}(x) f(x) \tag{8}$$

对于连续型随机变量可以通过求积分得到:

$$\mathbb{E}_{\mathrm{x} \sim p}[f(x)]=\int P(x) f(x) d x \tag{9}$$

另外，期望是线性的:

$$\mathbb{E}_{\mathrm{x}}[\alpha f(x)+\beta g(x)]=\alpha \mathbb{E}_{\mathrm{x}}[f(x)]+\beta \mathbb{E}_{\mathrm{x}}[f(x)] \tag{10}$$

方差 (`Variance`)：衡量的是当我们对 x 依据它的概率分布进行采样时，随机变量 x 的函数值会呈现多大的差异，描述采样得到的函数值在期望上下的波动程度:

$$\operatorname{Var}(f(x))=\mathbb{E}[(f(x)-\mathbb{E}[f(x)])^2] \tag{11}$$

将方差开平方即为**标准差** (`Standard Deviation`)。

协方差 (`Covariance`)：用于衡量两组值之间的**线性相关程度**:

$$\operatorname{Cov}(f(x), g(y))=\mathbb{E}[(f(x)-\mathbb{E}[f(x)])(g(y)-\mathbb{E}[g(y)])] \tag{12}$$

注意，独立比零协方差要求更强，因为**独立还排除了非线性的相关**。

[23]:
```
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
```

[23]:
```
(5.0, 6.666666666666667, 7.5, array([[ 7.5, -7.5],
        [-7.5,  7.5]]))
```

1.5 常用概率分布

1.5.1 伯努利分布 (两点分布)

伯努利分布 (Bernoulli Distribution) 是**单个二值随机变量**的分布，随机变量只有两种可能。它由一个参数 $\phi \in [0,1]$ 控制，ϕ 给出了随机变量等于 1 的概率：

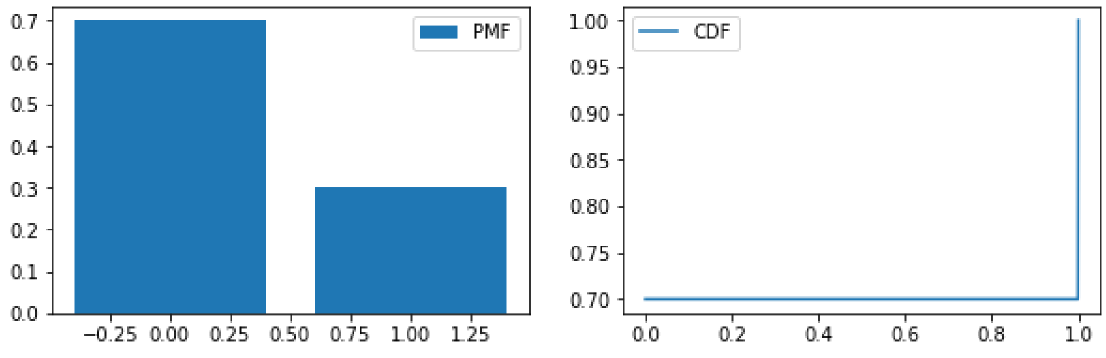
$$\begin{aligned} &P(\mathrm{x}=1)=\phi \\ &P(\mathrm{x}=0)=1-\phi \\ &P(\mathrm{x}=x)=\phi^{x}(1-\phi)^{1-x} \end{aligned} \tag{13}$$

表示一次试验的结果要么成功要么失败。

[24]:
```python
def plot_distribution(X, axes=None):
    """ 给定随机变量，绘制 PDF，PMF，CDF"""
    if axes is None:
        fig, axes = plt.subplots(1, 2, figsize=(10, 3))
    x_min, x_max = X.interval(0.99)
    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))
        axes[0].bar(x_int, X.pmf(x_int), label="PMF") # pmf 和 pdf 是类似的
    axes[1].plot(x, X.cdf(x), label="CDF")
    for ax in axes:
        ax.legend()
    return axes
```

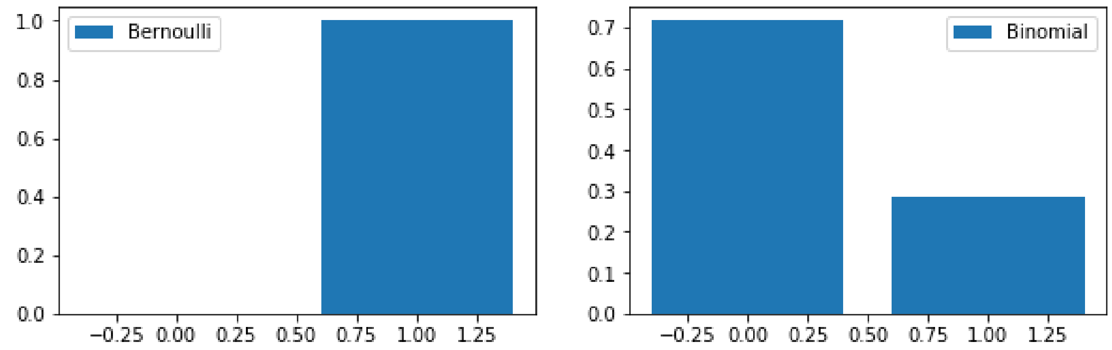
[25]:
```python
from scipy.stats import bernoulli
fig, axes = plt.subplots(1, 2, figsize=(10, 3)) # 画布
p = 0.3
X = bernoulli(p) # 伯努利分布
plot_distribution(X, axes=axes)
```

[25]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x1187d3d30>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11885ccf8>],
      dtype=object)
```



[26]:
```python
# 产生成功的概率
possibility = 0.3
def trials(n_samples):
    samples = np.random.binomial(n_samples, possibility)  # 成功的次数
    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()
```



1.5.2 范畴分布 (分类分布)

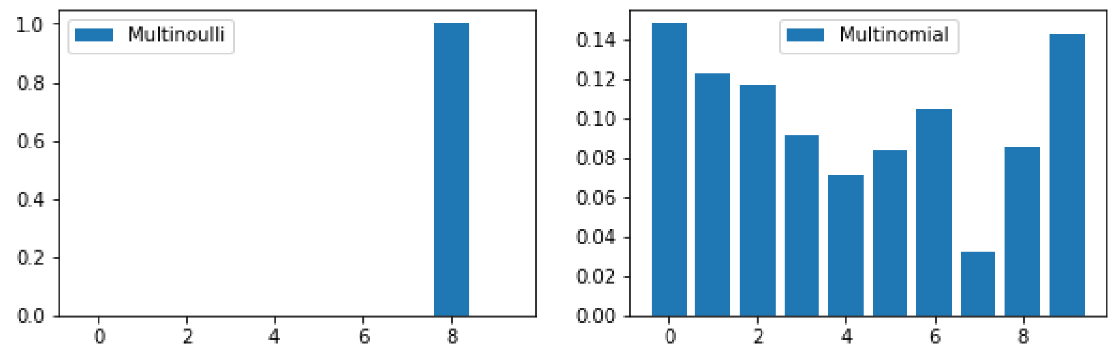
范畴分布 (`Multinoulli Distribution`) 是指在**具有 k 个不同值**的**单个离散型随机变量**上的分布:

$$p(\mathrm{x}=x)=\prod_i \phi_i^{x_i} \tag{14}$$

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。

[27]:
```
def k_possibilities(k):
    """
    随机产生一组 10 维概率向量
    """
    res = np.random.rand(k)
    _sum = sum(res)
    for i, x in enumerate(res):
        res[i] = x / _sum
    return res

fig, axes = plt.subplots(1, 2, figsize=(10, 3))
# 一次试验， 范畴分布
k, n_samples = 10, 1
samples = np.random.multinomial(n_samples, k_possibilities(k))  # 各维度“成功”的次数
axes[0].bar(range(len(samples)), samples/n_samples, label="Multinoulli")
# n 次试验， 多项分布
n_samples = 1000
samples = np.random.multinomial(n_samples, k_possibilities(k))
axes[1].bar(range(len(samples)), samples/n_samples, label="Multinomial")
for ax in axes:
    ax.legend()
```



1.5.3 高斯分布（正态分布）

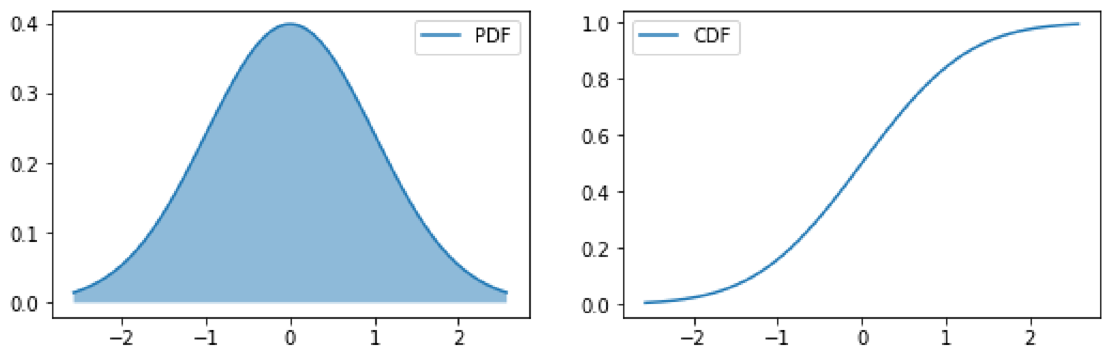
高斯分布 (Gaussian Distribution) 或正态分布 (Normal Distribution) 形式如下：

$$N(x;\mu,\sigma^2)=\sqrt{\frac{1}{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{15}$$

有时也会用 $\beta=\frac{1}{\sigma^2}$ 表示分布的精度 (precision)。中心极限定理 (Central Limit Theorem) 认为，大量的独立随机变量的和近似于一个正态分布，因此可以认为噪声是属于正态分布的。

[28]:
```
from scipy.stats import norm
fig, axes = plt.subplots(1, 2, figsize=(10, 3)) # 画布
mu, sigma = 0, 1
X = norm(mu, sigma) # 标准正态分布
plot_distribution(X, axes=axes)
```

[28]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x118b76710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x118c54978>],
      dtype=object)
```



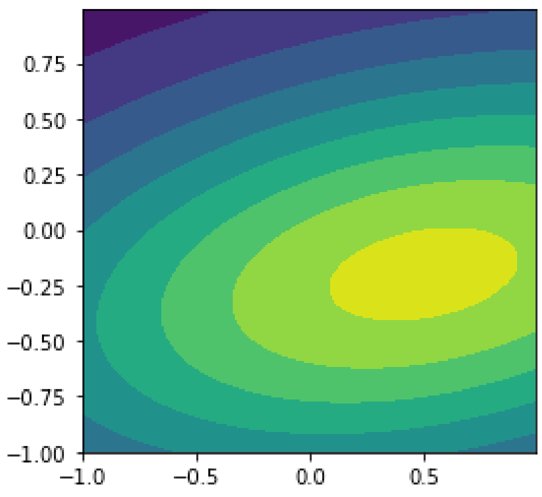
1.5.4 多元高斯分布（多元正态分布）

多元正态分布 (Multivariate Normal Distribution) 形式如下：

$$N(x;\mu,\Sigma)=\sqrt{\frac{1}{(2\pi)^n\det(\Sigma)}}\exp\left(-\frac{1}{2}(x-\mu)^\top\Sigma^{-1}(x-\mu)\right) \tag{16}$$

[29]:
```
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
x, y = np.mgrid[-1:1:.01, -1:1:.01]
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))
```

[29]: <matplotlib.contour.QuadContourSet at 0x118cd4438>



1.5.5 指数分布

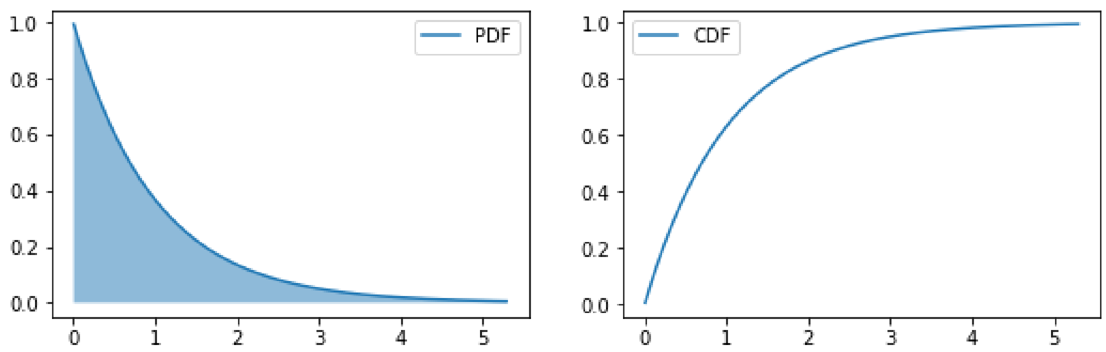
指数分布 (Exponential Distribution) 形式如下：

$$p(x;\lambda) = \lambda 1_{x\geq 0} \exp(-\lambda x) \tag{17}$$

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)。

[30]:
```python
from scipy.stats import expon
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
# 定义 scale = 1 / lambda
X = expon(scale=1)
plot_distribution(X, axes=axes)
```

[30]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x11900f438>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1190956d8>],
      dtype=object)
```



1.5.6 拉普拉斯分布

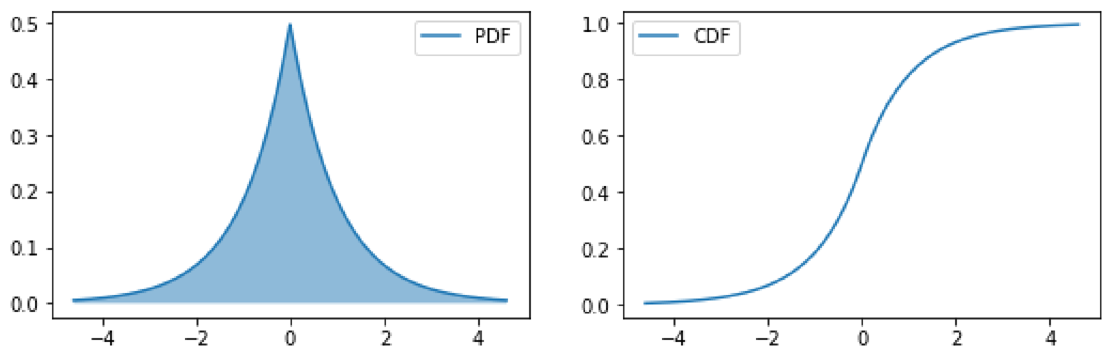
拉普拉斯分布（Laplace Distribution）形式如下：

$$\text{Laplace}(x;\mu,\gamma) = \frac{1}{2\gamma}\exp\left(-\frac{|x-\mu|}{\gamma}\right) \tag{18}$$

这也是可以在一个点获得比较高的概率的分布。

[31]:
```python
from scipy.stats import laplace
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
mu, gamma = 0, 1
X = laplace(loc=mu, scale=gamma)
plot_distribution(X, axes=axes)
```

[31]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x11913eb00>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x118a3b2b0>],
      dtype=object)
```



1.5.7 Dirac 分布

Dirac delta 函数定义为 $p(x) = \delta(x-\mu)$，这是一个泛函数。它常被用于组成**经验分布** (Empirical Distribution)：

$$\hat{p}(x) = \frac{1}{m}\sum_{i=1}^{m}\delta(x - x^{(i)}) \tag{19}$$

1.6 常用函数的有用性质

1.6.1 logistic sigmoid 函数

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{20}$$

logistic sigmoid 函数通常用来产生伯努利分布中的参数 ϕ ，因为它的范围是 $(0,1)$，处在 ϕ 的有效取值范围内。sigmoid 函数在变量取绝对值非常大的正值或负值时会出现饱和 (`Saturate`) 现象，意味着函数会变得很平，并且对输入的微小改变会变得不敏感。

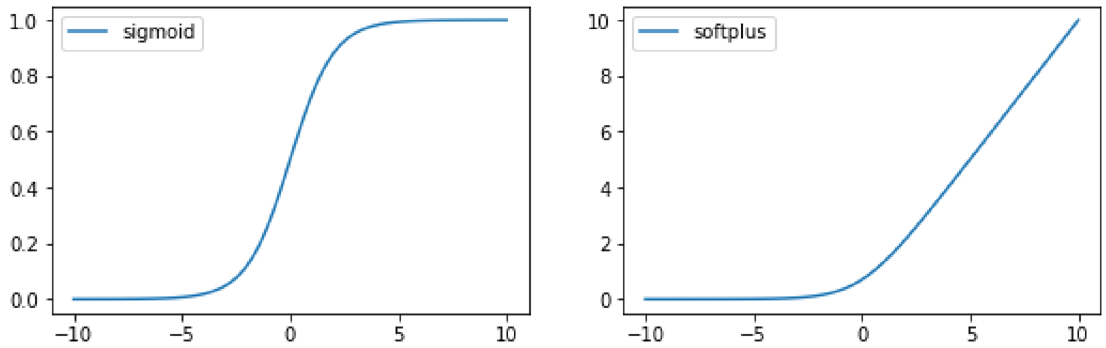
1.6.2 softplus 函数

$$\zeta(x) = \log(1+\exp(x)) \tag{21}$$

softplus 函数可以用来产生正态分布的 β 和 σ 参数，因为它的范围是 $(0,\infty)$。当处理包含 sigmoid 函数的表达式时它也经常出现。softplus 函数名来源于它是另外一个函数的平滑 (或 “软化”) 形式，这个函数是:

$$x^+ = \max(0,x) \tag{22}$$

[32]:
```
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()
```



2 信息论

信息论背后的思想：一件不太可能的事件比一件比较可能的事件更有信息量。

信息 (`Information`) 需要满足的三个条件：

- 比较可能发生的事件的信息量要少。
- 比较不可能发生的事件的信息量要大。
- 独立发生的事件之间的信息量应该是可以叠加的。例如，投掷的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的信息量的两倍。

自信息 (`Self-Information`)：对事件 $x = x$，我们定义：

$$I(x) = -\log\ P(x) \tag{23}$$

自信息满足上面三个条件，单位是奈特 (nats) （底为 e）

香农熵 (`Shannon Entropy`)：上述的自信息只包含一个事件的信息，而**对于整个概率分布** P，不确定性可以这样衡量：

$$\mathbb{E}_{x\sim P}[\mathrm{I}(x)] = -\mathbb{E}_{x\sim P}[\log P(x)] \tag{24}$$

也可以表示成 H(P)。香农熵是编码原理中最优编码长度。

多个随机变量：

- **联合熵** (`Joint Entropy`)：表示同时考虑多个事件的条件下（即考虑联合分布概率）的熵。

$$H(X,Y) = -\sum_{x,y} P(x,y)\log(P(x,y)) \tag{25}$$

- **条件熵** (Conditional Entropy)：表示某件事情已经发生的情况下，另外一件事情的熵。

$$H(X|Y) = -\sum_{y} P(y) \sum_{x} P(x|y) \log (P(x|y)) \tag{26}$$

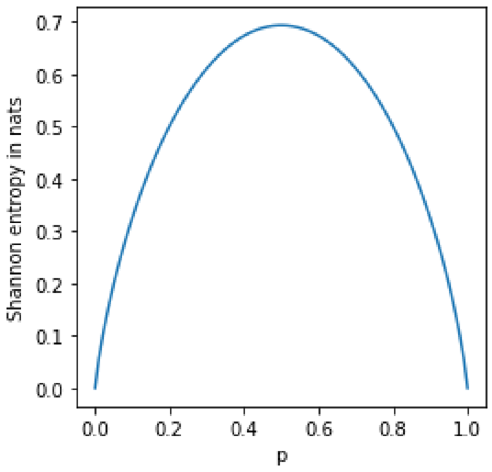
- **互信息** (Mutual Information)：表示两个事件的信息**相交**的部分。

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{27}$$

- **信息变差** (Variation of Information)：表示两个事件的信息**不相交**的部分。

$$V(X,Y) = H(X,Y) - I(X,Y) \tag{28}$$

[33]:
```
p = np.linspace(1e-6,1-1e-6,100)
entropy = (p-1)*np.log(1-p)-p*np.log(p)
plt.figure(figsize=(4,4))
plt.plot(p,entropy)
plt.xlabel('p')
plt.ylabel('Shannon entropy in nats')
plt.show()
```



[34]:
```
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) # 注:log 以 2 为底
    return entropy
```

[35]:
```
import random
import math
# 只用 64 个字符
simple_message = "".join([chr(random.randint(0,64)) for i in range(500)])
print(simple_message)
H(simple_message)
```

```
-.218?
        2>  -=?$'     <$<
:&5<>=>
/'+#)>@((931":< > 5)
4$%79@
)$6#63(?1.,&4'%%<
7 )1,
<%.)*4 :57+ ? 3#
```

[35]: 5.939286791378741

KL 散度 (Kullback-Leibler Divergence) 用于衡量两个分布 $P(\mathrm{x})$ 和 $Q(\mathrm{x})$ 之间的差距：

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)] \tag{29}$$

注意 $\mathrm{D_{KL}(P||Q)} \neq \mathrm{D_{KL}(Q||P)}$，不满足对称性。

交叉熵 (Cross Entropy)：

$$H(P,Q) = H(P) + D_{\mathrm{KL}}(P||Q) = -\mathbb{E}_{x\sim P}[\log Q(x)] \tag{30}$$

假设 P 是真实分布，Q 是模型分布，那么最小化交叉熵 H(P,Q) 可以让模型分布逼近真实分布。

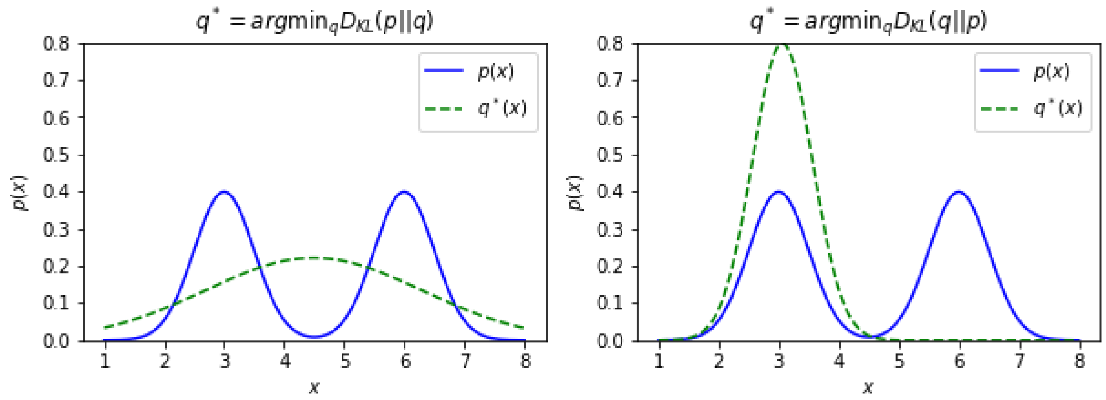
[36]:
```python
# KL 定义
from scipy.stats import entropy  # 内置 kl
def kl(p, q):
    """
    D(P || Q)
    """
    p = np.asarray(p, dtype=np.float)
    q = np.asarray(q, dtype=np.float)
    return np.sum(np.where(p != 0, p * np.log(p / q), 0))
```

[37]:
```python
# 测试
p = [0.1, 0.9]
q = [0.1, 0.9]
print(entropy(p, q) == kl(p, q))
```

```
True
```

[38]:
```python
# D(P||Q) 与 D(Q||P) 比较
x = np.linspace(1, 8, 500)
y1 = norm.pdf(x, 3, 0.5)
y2 = norm.pdf(x, 6, 0.5)
p = y1 + y2  # 构造 p(x)
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 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 图模型

机器学习算法会涉及到非常多的随机变量上的概率分布。利用分解可以减少表示联合分布的成本，于是用图来表示概率分布的分解，这称为结构化概率模型 (Structured Probabilistic Model) 或者图模型 (Graphical Model)。

3.1 有向图模型（Directed Model）

有向图模型的概率可以因子分解 $P(x) = P(x_1, ...x_i, ...) = \prod_i P(x_i \mid \mathrm{PA}(x_i))$，其中 $\mathrm{PA}(x_i)$ 是 x_i 的父节点，单个因子 $P(x_i|\mathrm{PA}(x_i))$ 称为条件概率分布 (CPD)。示例如下图所示，有：

$$P(a, b, c, d, e) = P(a)P(b \mid a)P(c \mid a, b)P(d \mid b)P(e \mid c)$$

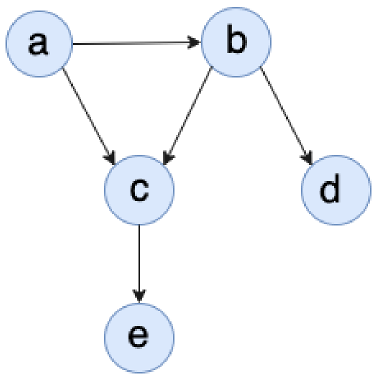


图 1. 有向图示例

有向图的代表是**贝叶斯网**。

贝叶斯网与朴素贝叶斯模型建立在相同的直观假设上：**通过利用分布的条件独立性来获得紧凑而自然的表示**。贝叶斯网核心是一个有向无环图 (DAG)，其节点为论域中的随机变量，节点间的有向箭头表示这两个节点的依赖关系。

贝叶斯网可以看作是**各特征节点间的依赖关系图** (有向无环图表示) 和**各特征节点相对其依赖节点的条件概率表**。

有向无环图可以由如下 3 种元结构构成：

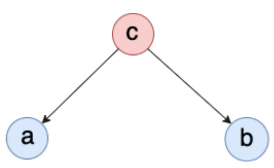
- 同父结构。例如在上图中，若不考虑节点 a ，则 c 和 d 有同一父节点 b，于是 $P(b, c, d) = P(b)P(c \mid b)P(d \mid b)$。
- V 型结构。例如在上图中，若不考虑节点 a 到 b 的依赖关系，则 $P(a, b, c) = P(a)P(b)P(c \mid a, b)$。
- 顺序结构。例如在上图中，若仅考虑节点 a、b、d，则有 $P(a, b, d) = P(a)P(b \mid a)P(d \mid b)$。

基于此，我们对简化的条件概率分布 (如 $P(c \mid a, b)$) 获取条件概率表。同时，也可以求得联合概率分布 $P(a, b, c, d, e) = P(a)P(b \mid a)P(c \mid a, b)P(d \mid b)P(e \mid c)$。

3.1.1 贝叶斯网的独立性

贝叶斯网的基本独立性体现在

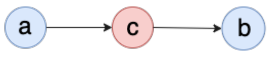
- **局部独立性**：给定父节点条件下，每个节点都独立于它的非后代节点。例如，给定父节点 c 时，e 与网中其他节点条件独立 $\implies (e \perp a, b, d \mid c)$
- **全局独立性** (d - 分离)：d - 分离是用来判断变量是否条件独立的图形化方法。它常见于以下三种条件独立的情况：
  1. tail-to-tail



$\implies$ 可以得知 $P(a, b, c) = P(a \mid c)P(b \mid c)P(c)$。

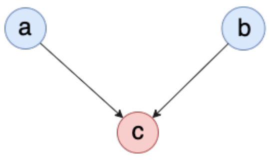
$\Longrightarrow$ 若 c 不作为观察点，可得 $P(a,b)=\sum_c P(a\mid c)P(b\mid c)P(c)\neq P(a)P(b)$，于是 a 和 b 不是 c 条件独立的。
$\Longrightarrow$ 若 c 作为观察点，可得 $P(a,b|c)=\frac{P(a,b,c)}{P(c)}=P(a\mid c)P(b\mid c)$，于是 a 和 b 是 c 条件下独立的。

2. head-to-tail



$\Longrightarrow$ 可以得知 $P(a,b,c)=P(b\mid c)P(c\mid a)P(a)$。
$\Longrightarrow$ 若 c 不作为观察点，可得 $P(a,b)=P(a)\sum_c P(c\mid a)P(b|c)=P(a)P(b\mid a)$，于是 a 和 b 不是 c 条件独立的。
$\Longrightarrow$ 若 c 作为观察点，可得 $P(a,b\mid c)=\frac{P(a,b,c)}{P(c)}=\frac{P(a)P(c|a)P(b|c)}{P(c)}=P(a\mid c)P(b\mid c)$，于是 a 和 b 是 c 条件下独立的。

3. head-to-head



$\Longrightarrow$ 可以得知 $P(a,b,c)=P(a)P(b)P(c\mid a,b)$。
$\Longrightarrow$ 若 c 不作为观察点，可得 $P(a,b)=P(a)P(b)\sum_c P(c\mid a,b)=P(a)P(b)$，于是 a 和 b 是 c 条件独立的。
$\Longrightarrow$ 若 c 作为观察点，可得 $P(a,b|c)=\frac{P(a,b,c)}{P(c)}=\frac{P(a)P(b)P(c|a,b)}{P(c)}\neq P(a\mid c)P(b\mid c)$，于是 a 和 b 不是 c 条件下独立的。

从而我们考虑复杂的有向无环图 (DAG)，如果 A,B,C 是三个集合 (可以是单独的节点或者是节点的集合)。为了判断 A 和 B 是否是 C 条件独立的，我们考虑图中所有 A 和 B 之间的路径。对于其中的一条路径，如果它满足以下两个条件中的任意一条，则称这条路径是阻塞 (`block`) 的：

1. 路径中存在某个节点 X 是 head-to-tail 或者 tail-to-tail 节点，并且 X 是包含在 C 中的；
2. 路径中存在某个节点 X 是 head-to-head 节点，并且 X 或 X 的儿子是不包含在 C 中的。

如果 A,B 间所有的路径都是阻塞的，那么 A,B 就是关于 C 条件独立的；否则 A,B 不是关于 C 条件独立的。

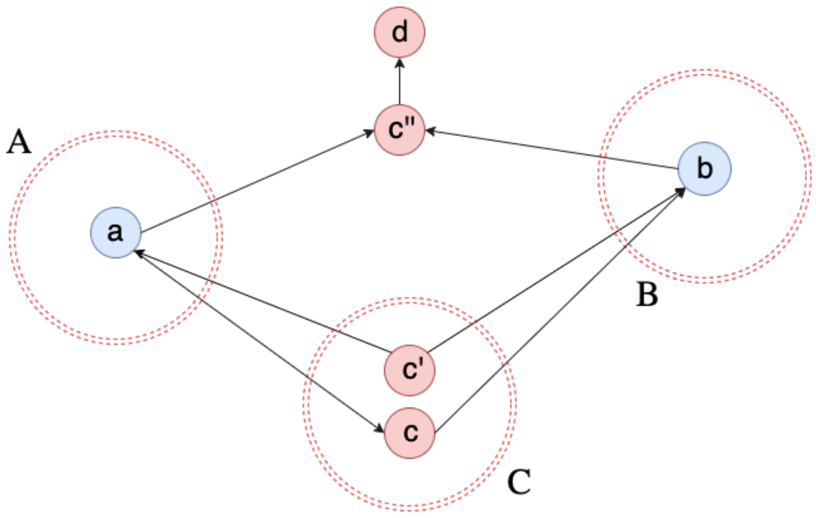


图 2. 考虑集合下的有向无环图

我们形象化阐述上面的结论：如图 2 所示，集合 x_A, x_B 和 x_C，我们希望 x_A 和 x_B 在给定 x_C 下条件独立或所有路径阻塞，即 $x_A\perp x_B\mid x_C$。对于集合中的 a，b 和 c，如果我们希望 $a{\to}c{\to}b$ 的路径阻塞，当路径为 tail-to-tail 或者 head-to-tail，则 c 应当在集合 C 中，这便是条件 1；当路径为 head-to-head，则 c 以及其后代 d 不能在集合 C 中，这便是条件 2。

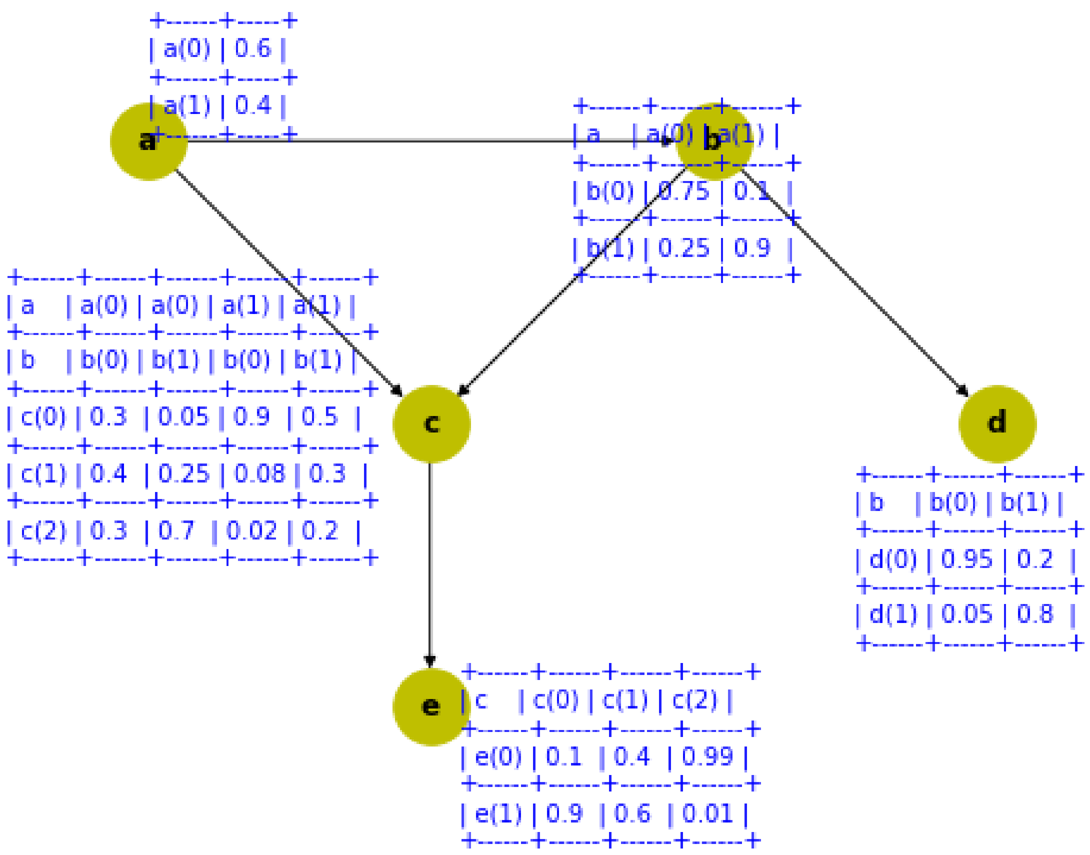
例如，我们再考虑图 1 中的例子，判断 a 和 d 是否是 e 下条件独立的：可以看出 a 到 d 有两条路径 $a{\to}c{\to}b{\to}d$ 以及 $a{\to}b{\to}d$。考虑路径 $a{\to}b{\to}d$，b 是 head-to-tail 类型的，但不在条件集合中，因此该路径不阻塞。考虑路径 $a{\to}c{\to}b{\to}d$，c 是 head-to-head 类型的，且它的儿子 e 在条件集合中，因此该路径也不阻塞。因此，得到结论：a 和 d 不是 e 下条件独立的。

[39]:
```
import networkx as nx
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
import matplotlib.pyplot as plt
%matplotlib inline

# 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()
```

验证模型数据的正确性: True



3.2 无向图模型 (Undirected Model)：

无向图模型的概率可以记作 $P(\boldsymbol{x}) = \frac{1}{Z}\prod_{C\in\mathbf{Q}}\Phi_C\boldsymbol{x}_C$。其中，我们将所有节点都彼此联通的集合称作团 (Clique，C)，Φ 称作因子 (factor)，每个因子和一个团 C 相对应，Z 是归一化常数。示例如下图所示，有 $P(a,b,c,d,e) = \frac{1}{Z}\Phi^{(1)}(a,b,c)\Phi^{(2)}(b,d)\Phi^{(3)}(c,e)$。

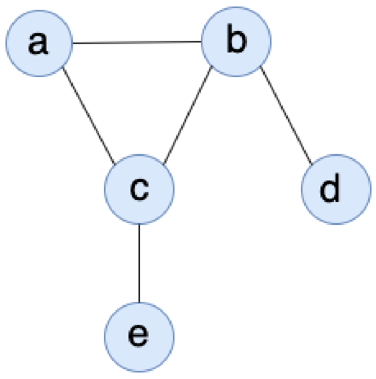


图 3. 无向图示例

有向图的代表是**马尔可夫网**。

贝叶斯网是根据节点依赖关系构成有向无环图，进而引申出每个节点的条件概率分布来表征其对父节点的依赖。但**马尔可夫网节点间的依赖关系是无向的**（相互平等的关系），无法用条件概率分布来表示，为此为引入**极大团**概念，进而为每个极大团引入一个**势函数**作为因子，然后将联合概率分布表示成这些因子的乘积再归一化，归一化常数被称作**配分函数**。

团: 假设一个特征集的任何两个特征都相互关联，那么这个特征集的联合概率分布是无法简化的，我们称这样的特征集为团。

极大团: 如果一个团不能被其他团包含，那么我们称这个团为极大团。

对于具有 n 个特征变量 $\boldsymbol{x} = (x_1, ..., x_n)$ 的马尔可夫网的所有极大团构成的集合 $\mathbf{Q}$，与极大团 $\mathrm{C} \in \mathbf{Q}$ 对应的属性变量集合记作 $\boldsymbol{x}_\mathrm{C}$，那么马尔可夫网 $\mathrm{P}(\boldsymbol{x})$ 可以写成因子分解的形式：

$$P(\boldsymbol{x}) = \frac{1}{\mathrm{Z}} \prod_{\mathrm{C} \in \mathbf{Q}} \Phi_\mathrm{C}(\boldsymbol{x}_\mathrm{C}) \qquad \mathrm{Z} = \sum_{\boldsymbol{x}} \prod_{\mathrm{C} \in \mathbf{Q}} \Phi_\mathrm{C}(\boldsymbol{x}_\mathrm{C}) \tag{31}$$

其中 Φ_C 就是极大团 C 对应的势函数 (因子)，用于对极大团 C 内的特征变量关系进行建模，必须为正。Z 为归一化因子 (配分函数)，就是对势函数乘积的所有属性变量求和求积分，使 P 成为概率。此时势函数可以写作 $\Phi(\mathbf{x}_\mathrm{C}) = \exp(-E(\mathbf{x}_\mathrm{C}))$，其中 E 为能量函数，我们也称 $P(\boldsymbol{x})$ 是由因子集 $\{\Phi_\mathrm{C} \mid \mathrm{C} \in \mathbf{Q}\}$ 参数化的吉布斯分布 (`Gibbs Distribution`) 或玻尔兹曼分布 (`Boltzmann Distribution`)。于是，示例的马尔可夫网的联合概率分布可写成：

$$P(a,b,c,d,e) = \frac{1}{\mathrm{Z}} \Phi_{a,b,c}(a,b,c) \Phi_{b,d}(b,d) \Phi_{c,e}(c,e)$$

马尔可夫网的条件独立性

前面介绍了贝叶斯网的元结构及其条件独立性，而马尔可夫网的有向分离，同样能够引出条件独立性 (相对分离集)，这就是全局马尔可夫性。

如果将 $\{a,b,c\}$ 视作团 A，$\{b,d\}$ 视作团 B，$\{c,e\}$ 视作团 C。特别地，我们可以验证最简单情况下的全局马尔可夫性：

$$\begin{aligned}
\mathrm{P}(x_A, x_B \mid x_C) &= \frac{\mathrm{P}(x_A, x_B, x_C)}{\mathrm{P}(x_C)} && \text{条件概率的定义} \\
&= \frac{\frac{1}{Z}\Phi_{A,C}(x_A,x_C)\Phi_{B,C}(x_B,x_C)}{\sum_{x'_A,x'_B} \mathrm{P}(x'_A, x'_B, x_C)} && \text{分子极大团分解} \\
& && \text{分母概率分布边缘求和 (积分)} \\
&= \frac{\frac{1}{Z}\Phi_{A,C}(x_A,x_C)\Phi_{B,C}(x_B,x_C)}{\sum_{x'_A,x'_B} \frac{1}{Z}\Phi_{A,C}(x'_A,x_C)\Phi_{B,C}(x'_B,x_C)} && \text{分母局部做极大团分解} \\
&= \frac{\Phi_{A,C}(x_A,x_C)}{\sum_{x'_A}\Phi_{A,C}(x'_A,x_C)} \frac{\Phi_{B,C}(x_B,x_C)}{\sum_{x'_B}\Phi_{B,C}(x'_B,x_C)} && \text{简单的代数操作} \\
&= \mathrm{P}(x_A \mid x_C)\mathrm{P}(x_B \mid x_C) && \text{利用边缘求和和极大团分解} \\
& && \text{而}P(x_A,x_B,x_C)\text{作为中间分布}
\end{aligned} \tag{32}$$

由全局马尔可夫性可以容易推导出两个推论：

- 局部马尔可夫性：将节点 $v \in \mathrm{V}$ 的所有邻接节点集作为分离集 $\mathrm{N}(v) \subset \mathrm{V}$，于是该节点 v 与被邻接变量集分离的剩余变量集是条件独立的 (相对 $\mathrm{N}(v)$ 而言)。

$$x_v \perp \boldsymbol{x}_{V \setminus N^*(v)} \mid \boldsymbol{x}_{N(v)}, \quad N^*(v) = N(v) \cup \{v\} \tag{33}$$

- 成对马尔可夫性：两个非邻接节点 $u, v \in \mathrm{V}$，必然可以被其他所有节点构成的集 $\boldsymbol{x}_{\mathrm{V} \setminus \{u,v\}}$ 分离，进而 u, v 也具有条件独立性 (相对前面指定的节点集)。

$$x_u \perp x_v \mid \boldsymbol{x}_{\mathrm{V} \setminus \{u,v\}}, \quad \{u,v\} \notin E, \quad E\text{是边集} \tag{34}$$

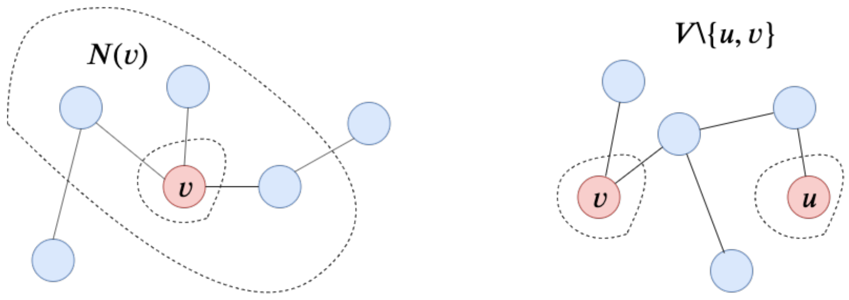
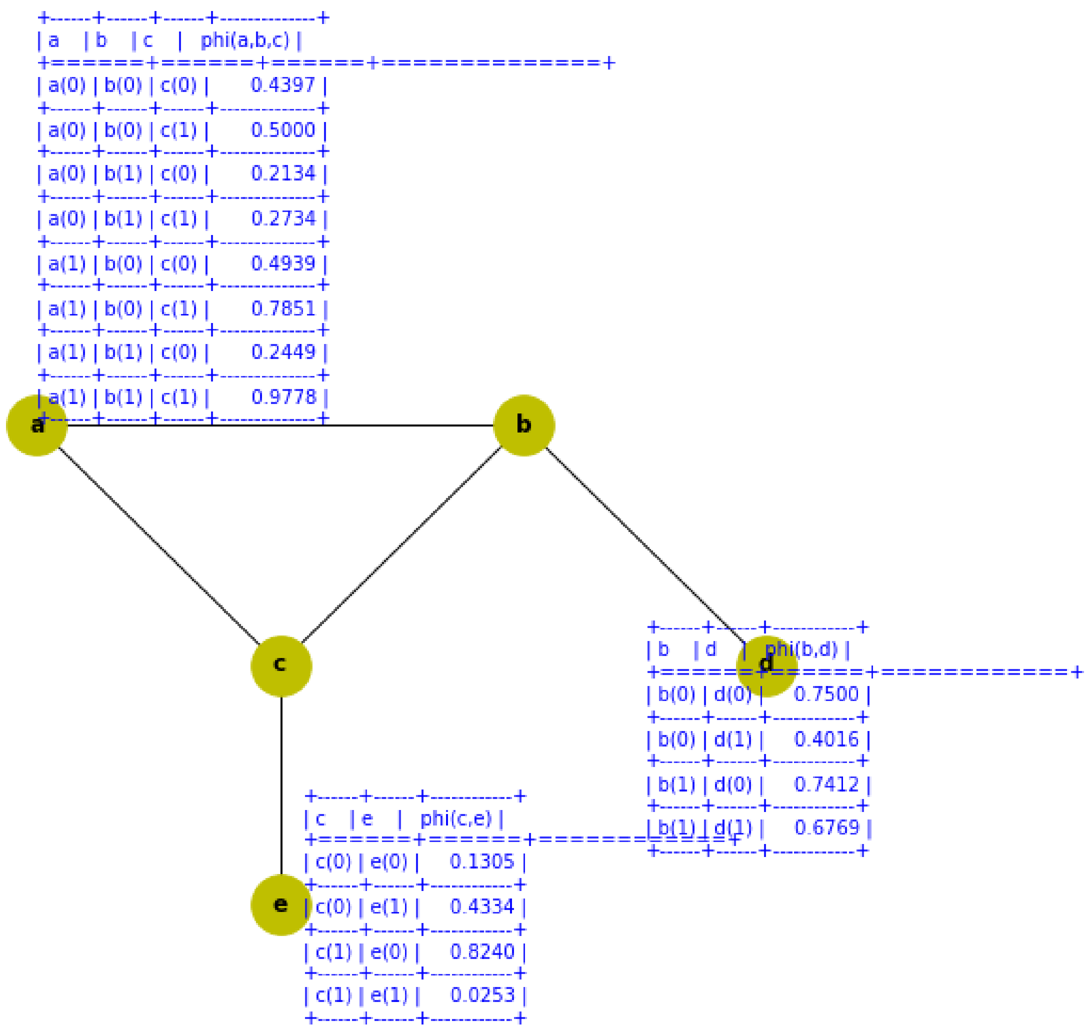


图 4. 局部马尔可夫性与成对马尔可夫性

[40]:
```python
import networkx as nx
from pgmpy.models import MarkovModel
from pgmpy.factors.discrete import DiscreteFactor
import matplotlib.pyplot as plt
%matplotlib inline
# 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()
```

验证模型数据的正确性: True



有关于图模型的更多内容会在第十六章呈现。

[41]:
```
import numpy, scipy, matplotlib, networkx, pgmpy
print("numpy:", numpy.__version__)
print("scipy:", scipy.__version__)
print("matplotlib:", matplotlib.__version__)
print("networkx:", networkx.__version__)
print("pgmpy:", pgmpy.__version__)
```

```
numpy: 1.14.5
scipy: 1.3.1
matplotlib: 3.1.1
networkx: 2.4
pgmpy: 0.1.10
```