

# 贝叶斯分类器

---

-----BY YANGZHONGXIU 2020.02

# 内容

---

- 贝叶斯决策论
- 极大似然估计
- 朴素贝叶斯分类器
- 半朴素贝叶斯分类器
- EM算法
- 朴素贝叶斯应用实例

# 贝叶斯决策论

---

贝叶斯决策论(Bayesian decision theory)是概率框架下实施决策的基本方法。对分类任务来说，在所有相关概率都已知的理想情形下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记。下面我们以多分类任务为例来解释其基本原理。

假设有 $N$ 种可能的类别标记，即  $\mathbf{y} = \{c_1, c_2, \dots, c_N\}$ ， $\lambda_{ij}$  是将一个真实标记为 $c_j$ 的样本误分类为 $c_i$ 所产生的损失。基于后验概率 $P(c_i | \mathbf{x})$ 可获得将样本 $\mathbf{x}$ 分类为 $c_i$ 所产生的**期望损失(expected loss)**，即在样本 $\mathbf{x}$ 上的**"条件风险"**(conditional risk)

$$R(c_i | \mathbf{x}) = \sum_{j=1}^N \lambda_{ij} P(c_j | \mathbf{x}) . \quad (7.1)$$

我们的任务是寻找一个判定准则  $h: X \rightarrow Y$  以最小化总体风险。

$$R(h) = \mathbb{E}_{\mathbf{x}} [R(h(\mathbf{x}) \mid \mathbf{x})] . \quad (7.2)$$

显然，对每个样本  $\mathbf{x}$ ，若  $h$  能最小化条件风险  $R(h(\mathbf{x}) \mid \mathbf{x})$ ，则总体风险  $R(h)$  也将被最小化。这就产生了 **贝叶斯判定准则(Bayes decision rule)**: 为最小化总体风险，只需在每个样本上选择那个能使条件风险  $R(c \mid \mathbf{x})$  最小的类别标记，即

$$h^*(\mathbf{x}) = \arg \min_{c \in \mathcal{Y}} R(c \mid \mathbf{x}) , \quad (7.3)$$

此时， $h^*$  称为 **贝叶斯最优分类器(Bayes optimal classifier)**，与之对应的总体风险  $R(h^*)$  称为 **贝叶斯风险(Bayes risk)**。  $1 - R(h^*)$  反映了分类器所能达到的最好性能，即通过机器学习所能产生的模型精度的理论上限。

具体来说，若目标是最小化分类错误率，则误判损失 $\lambda_{ij}$  可写为

$$\lambda_{ij} = \begin{cases} 0, & \text{if } i = j; \\ 1, & \text{otherwise,} \end{cases} \quad (7.4)$$

此时条件风险

$$R(c | \mathbf{x}) = 1 - P(c | \mathbf{x}) , \quad (7.5)$$

于是，最小化分类错误率的贝叶斯最优分类器为

$$h^*(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} P(c | \mathbf{x}) , \quad (7.6)$$

即对每个样本 $\mathbf{x}$ ， 选择能使后验概率 $P(c | \mathbf{x})$  最大的类别标记。

不难看出，欲使用贝叶斯判定准则来最小化决策风险，首先要获得后验概率 $P(c | x)$ 。然而，在现实任务中这通常难以直接获得。从这个角度来看，机器学习所要实现的是基于有限的训练样本集尽可能准确地估计出后验概率 $P(c | x)$ 。

大体来说，主要有两种策略：

**判别模型(discriminative models)**：给定 $x$ ，可通过直接建模 $P(c | x)$ 来预测 $c$ ；

**生成式模型(generative models)**：先对联合概率分布 $P(x | c)$ 建模，然后再由此获得 $P(c | x)$ 。

显然，前面介绍的决策树、BP 神经网络、支持向量机等，都可归入判别式模型的范畴。

对生成式模型来说，必然考虑

$$P(c | \mathbf{x}) = \frac{P(\mathbf{x}, c)}{P(\mathbf{x})} . \quad (7.7)$$

基于贝叶斯定理,  $P(c | \mathbf{x})$ 可写为

$$P(c | \mathbf{x}) = \frac{P(c) P(\mathbf{x} | c)}{P(\mathbf{x})} , \quad (7.8)$$

其中:

$P(c)$  是类“先验”(prior)概率;

$P(\mathbf{x} | c)$  是样本 $\mathbf{x}$  相对于类标记 $c$  的类条件概率(conditional probability) , 或称为“似然” (likelihood);

$P(\mathbf{x})$  是用于归一化的“证据” (evidence) 因子。

对给定样本 $x$ ，证据因子 $P(x)$ 与类标记无关，因此估计 $P(c | x)$ 的问题就转化为如何基于训练数据 $D$ 来估计先验 $P(c)$ 和似然 $P(x | c)$ 。

类先验概率 $P(c)$ 表达了样本主 $R$ 问中各类样本所占的比例，根据大数定律，当训练集包含充足的独立同分布样本时， $P(c)$ 可通过各类样本出现的频率来进行估计。

对类条件概率 $P(x | c)$ 来说，由于它涉及关于 $x$ 所有属性的联合概率，直接根据样本出现的频率来估计将会遇到严重的困难。例如，假设样本的 $d$ 个属性都是二值的，则样本空间将有 $2^d$ 种可能的取值，在现实应用中，这个值往往远大于训练样本数 $m$ ，也就是说，很多样本取值在训练集中根本没有出现，直接使用频率来估计 $P(x | c)$ 显然不可行，因为“未被观测到”与“出现概率为零”通常是不同的。



# 极大似然估计

---

估计类条件概率的一种常用策略是先假定其具有某种确定的概率分布形式，再基于训练样本对概率分布的参数进行估计。具体地，记关于类别 $c$ 的类条件概率为 $P(x | c)$ ，假设 $P(x | c)$ 具有确定的形式并且被参数向量。 $c$ 唯一确定，则我们的任务就是利用训练集 $D$ 估计参数 $\theta_c$ 。为明确起见，我们将 $P(x | c)$ 记为 $P(x | \theta_c)$

事实上,概率模型的训练过程就是参数估计(parameter estimation) 过程。  
对于参数估计, 统计学界的两个学派分别提供了不同的解决方案:

**频率主义学派(Frequentist)**认为参数虽然未知, 但却是客观存在的固定值, 因此, 可通过优化似然函数等准则来确定参数值;

**贝叶斯学派(Bayesian)**则认为参数是未观察到的随机变量, 其本身也可有分布, 因此, 可假定参数服从一个先验分布, 然后基于观测到的数据来计算参数的后验分布。

本节介绍源自频率主义学派的极大似然估计(Maximum Likelihood Estimation, 简称MLE), 这是根据数据采样来估计概率分布参数的经典方法。

令 $D_c$  表示训练、集 $D$  中第 $c$  类样本组成的集合，假设这些样本是独立同分布的，则参数 $\theta_c$  对于数据集 $D_c$  的似然是

$$P(D_c | \theta_c) = \prod_{x \in D_c} P(x | \theta_c) . \quad (7.9)$$

对 $\theta_c$  进行极大似然估计，就是去寻找能最大化似然 $P(D_c | \theta_c)$  的参数值 $\hat{\theta}_c$ 。直观上看，极大似然估计是试图在 $\theta_c$  所有可能的取值中，找到一个能使数据出现的“可能性”最大的值。

式(7.9)中的连乘操作易造成下溢, 通常使用对数似然(log-likelihood)

$$\begin{aligned} LL(\boldsymbol{\theta}_c) &= \log P(D_c \mid \boldsymbol{\theta}_c) \\ &= \sum_{\boldsymbol{x} \in D_c} \log P(\boldsymbol{x} \mid \boldsymbol{\theta}_c) , \end{aligned} \quad (7.10)$$

此时参数  $\boldsymbol{\theta}_c$  的极大似然估计  $\hat{\boldsymbol{\theta}}_c$  为

$$\hat{\boldsymbol{\theta}}_c = \arg \max_{\boldsymbol{\theta}_c} LL(\boldsymbol{\theta}_c) . \quad (7.11)$$

例如, 在连续属性情形下, 假设概率密度函数  $p(\mathbf{x} | c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\sigma}_c^2)$ , 则参数  $\boldsymbol{\mu}_c$  和  $\boldsymbol{\sigma}_c^2$  的极大似然估计为

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} \mathbf{x} , \quad (7.12)$$

$$\hat{\boldsymbol{\sigma}}_c^2 = \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} (\mathbf{x} - \hat{\boldsymbol{\mu}}_c)(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)^T . \quad (7.13)$$

也就是说, 通过极大似然法得到的正态分布均值就是样本均值, 方差就是  $(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)^T$  的均值, 这显然是一个符合直觉的结果. 在离散属性情形下, 也可通过类似的方式估计类条件概率.



需注意的是，这种参数化的方法虽能使类条件概率估计变得相对简单，但**估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实数据分布**。在现实应用中，欲做出能较好地接近潜在真实分布的假设，往往需在一定程度上利用关于应用任务本身的经验知识，否则若仅凭"猜测"来假设概率分布形式，很可能产生误导性的结果。

# 朴素贝叶斯分类器

---

不难发现，基于贝叶斯公式(7.8)来估计后验概率 $P(c | x)$ 的主要困难在于：类条件概率 $P(x | c)$ 是所有属性上的联合概率，难以从有限的训练样本直接估计而得。

为避开这个障碍，朴素贝叶斯分类器(naïve Bayes classifier) 采用了**“属性条件独立性假设” (attribute conditional independence assumption)**：对已知类别，假设所有属性相互独立。换言之，假设每个属性独立地对分类结果发生影响。

基于属性条件独立性假设, 式(7.8)可重写为

$$P(c | \mathbf{x}) = \frac{P(c) P(\mathbf{x} | c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i | c) , \quad (7.14)$$

其中  $d$  为属性数目,  $x_i$  为  $\mathbf{x}$  在第  $i$  个属性上的取值.

由于对所有类别来说  $P(\mathbf{x})$  相同, 因此基于式(7.6)的贝叶斯判定准则有

$$h_{nb}(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i | c) , \quad (7.15)$$

这就是朴素贝叶斯分类器的表达式.



显然，朴素贝叶斯分类器的训练过程就是基于训练集 $D$ 来估计类先验概率 $P(c)$ ，并为每个属性估计条件概率 $P(x_i | c)$ 。

令 $D_c$ 表示训练集 $D$ 中第 $c$ 类样本组成的集合，若有充足的独立同分布样本，则可容易地估计出类先验概率

$$P(c) = \frac{|D_c|}{|D|} . \quad (7.16)$$

对离散属性而言，令 $D_{c, x_i}$ ，表示 $D_c$ 中在第 $i$ 个属性上取值为 $x_i$ 的样本组成的集合，则条件概率 $P(x_i | c)$ 可估计为

$$P(x_i | c) = \frac{|D_{c, x_i}|}{|D_c|} . \quad (7.17)$$

对连续属性可考虑概率密度函数, 假定  $p(x_i | c) \sim \mathcal{N}(\mu_{c,i}, \sigma_{c,i}^2)$ , 其中  $\mu_{c,i}$  和  $\sigma_{c,i}^2$  分别是第  $c$  类样本在第  $i$  个属性上取值的均值和方差, 则有

$$p(x_i | c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp \left( -\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2} \right) . \quad (7.18)$$

下面我们用西瓜数据集 3.0 训练一个朴素贝叶斯分类器, 对测试例 “测 1” 进行分类:

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
测 1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	?

首先估计类先验概率  $P(c)$ , 显然有

$$P(\text{好瓜} = \text{是}) = \frac{8}{17} \approx 0.471 ,$$

$$P(\text{好瓜} = \text{否}) = \frac{9}{17} \approx 0.529 .$$

然后, 为每个属性估计条件概率  $P(x_i | c)$ :

$$P_{\text{青绿}|\text{是}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{是}) = \frac{3}{8} = 0.375 ,$$

$$P_{\text{青绿}|\text{否}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333 ,$$

$$P_{\text{蜷缩}|\text{是}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{是}) = \frac{5}{8} = 0.375 ,$$

$$P_{\text{蜷缩}|\text{否}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333 ,$$

$$P_{\text{浊响}|\text{是}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 ,$$

$$P_{\text{浊响}|\text{否}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{否}) = \frac{4}{9} \approx 0.444 ,$$

$$P_{\text{清晰}|\text{是}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{是}) = \frac{7}{8} = 0.875 ,$$

$$P_{\text{清晰}|\text{否}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222 ,$$

$$P_{\text{凹陷}|\text{是}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 ,$$

$$P_{\text{凹陷}|\text{否}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222 ,$$

$$P_{\text{硬滑}|\text{是}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750 ,$$

$$P_{\text{硬滑}|\text{否}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{否}) = \frac{6}{9} \approx 0.667 ,$$

$$p_{\text{密度: 0.697}|\text{是}} = p(\text{密度} = 0.697 | \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.129} \exp \left( -\frac{(0.697 - 0.574)^2}{2 \cdot 0.129^2} \right) \approx 1.959 ,$$

$$p_{\text{密度: 0.697}|\text{否}} = p(\text{密度} = 0.697 | \text{好瓜} = \text{否})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.195} \exp \left( -\frac{(0.697 - 0.496)^2}{2 \cdot 0.195^2} \right) \approx 1.203 ,$$

$$p_{\text{含糖: 0.460}|\text{是}} = p(\text{含糖率} = 0.460 | \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.101} \exp \left( -\frac{(0.460 - 0.279)^2}{2 \cdot 0.101^2} \right) \approx 0.788 ,$$

$$p_{\text{含糖: 0.460}|\text{否}} = p(\text{含糖率} = 0.460 | \text{好瓜} = \text{否})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.108} \exp \left( -\frac{(0.460 - 0.154)^2}{2 \cdot 0.108^2} \right) \approx 0.066 .$$

于是, 有

$$\begin{aligned} &P(\text{好瓜} = \text{是}) \times P_{\text{青绿}|\text{是}} \times P_{\text{蜷缩}|\text{是}} \times P_{\text{浊响}|\text{是}} \times P_{\text{清晰}|\text{是}} \times P_{\text{凹陷}|\text{是}} \\ &\quad \times P_{\text{硬滑}|\text{是}} \times p_{\text{密度: 0.697}|\text{是}} \times p_{\text{含糖: 0.460}|\text{是}} \approx 0.038, \\ &P(\text{好瓜} = \text{否}) \times P_{\text{青绿}|\text{否}} \times P_{\text{蜷缩}|\text{否}} \times P_{\text{浊响}|\text{否}} \times P_{\text{清晰}|\text{否}} \times P_{\text{凹陷}|\text{否}} \\ &\quad \times P_{\text{硬滑}|\text{否}} \times p_{\text{密度: 0.697}|\text{否}} \times p_{\text{含糖: 0.460}|\text{否}} \approx 6.80 \times 10^{-5}. \end{aligned}$$

由于  $0.038 > 6.80 \times 10^{-5}$ , 因此, 朴素贝叶斯分类器将测试样本“测 1”判别为“好瓜”.

需注意, 若某个属性值在训练集中没有与某个类同时出现过, 则直接基于式(7.17)进行概率估计, 再根据式(7.15) 进行判别将出现问题. 例如, 在使用西瓜数据集 3.0 训练朴素贝叶斯分类器时, 对一个“敲声=清脆”的测试例, 有

$$P_{\text{清脆}|\text{是}} = P(\text{敲声} = \text{清脆} | \text{好瓜} = \text{是}) = \frac{0}{8} = 0 ,$$

由于式(7.15)的连乘式计算出的概率值为零, 因此, 无论该样本的其他属性是什么, 哪怕在其他属性上明显像好瓜, 分类的结果都将是“好瓜=否”, 这显然不太合理.

为了避免其他属性携带的信息被训练集中未出现的属性值“抹去”，在估计概率值时通常要进行“平滑”(smoothing)，常用“拉普拉斯修正”(Laplacian correction)。具体来说，令  $N$  表示训练集  $D$  中可能的类别数， $N_i$  表示第  $i$  个属性可能的取值数，则式(7.16)和(7.17)分别修正为

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N}, \quad (7.19)$$

$$\hat{P}(x_i | c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}. \quad (7.20)$$

例如，在本节的例子中，类先验概率可估计为

$$\hat{P}(\text{好瓜} = \text{是}) = \frac{8 + 1}{17 + 2} \approx 0.474, \quad \hat{P}(\text{好瓜} = \text{否}) = \frac{9 + 1}{17 + 2} \approx 0.526.$$

类似地， $P_{\text{青绿}|\text{是}}$  和  $P_{\text{青绿}|\text{否}}$  可估计为

$$\hat{P}_{\text{青绿}|\text{是}} = \hat{P}(\text{色泽} = \text{青绿} | \text{好瓜} = \text{是}) = \frac{3 + 1}{8 + 3} \approx 0.364,$$



显然, 拉普拉斯修正避免了因训练集样本不充分而导致概率估值为零的问题, 并且在训练集变大时, 修正过程所引入的先验(prior)的影响也会逐渐变得可忽略, 使得估值渐趋向于实际概率值.

在现实任务中朴素贝叶斯分类器有多种使用方式. 例如, 若任务对预测速度要求较高, 则对给定训练集, 可将朴素贝叶斯分类器涉及的所有概率估值事先计算好存储起来, 这样在进行预测时只需“查表”即可进行判别; 若任务数据更替频繁, 则可采用“懒惰学习”(lazy learning)方式, 先不进行任何训练, 待收到预测请求时再根据当前数据集进行概率估值; 若数据不断增加, 则可在现有估值基础上, 仅对新增样本的属性值所涉及的概率估值进行计数修正即可实现增量学习.

# 半朴素贝叶斯分类器

---

为了降低贝叶斯公式(7.8)中估计后验概率  $P(c | \mathbf{x})$  的困难, 朴素贝叶斯分类器采用了属性条件独立性假设, 但在现实任务中这个假设往往很难成立. 于是, 人们尝试对属性条件独立性假设进行一定程度的放松, 由此产生了一类称为“半朴素贝叶斯分类器” (semi-naïve Bayes classifiers)的学习方法.

半朴素贝叶斯分类器的基本想法是适当考虑一部分属性间的相互依赖信息, 从而既不需进行完全联合概率计算, 又不至于彻底忽略了比较强的属性依赖关系. “独依赖估计” (One-Dependent Estimator, 简称 ODE) 是半朴素贝叶斯分类器最常用的一种策略. 顾名思义, 所谓“独依赖”就是假设每个属性在类别之外最多仅依赖于一个其他属性, 即

$$P(c | \mathbf{x}) \propto P(c) \prod_{i=1}^d P(x_i | c, pa_i), \quad (7.21)$$

其中  $pa_i$  为属性  $x_i$  所依赖的属性, 称为  $x_i$  的父属性. 此时, 对每个属性  $x_i$ , 若其父属性  $pa_i$  已知, 则可采用类似式(7.20) 的办法来估计概率值  $P(x_i | c, pa_i)$ . 于是, 问题的关键就转化为如何确定每个属性的父属性, 不同的做法产生不同的独依赖分类器.

最直接的做法是假设所有属性都依赖于同一个属性, 称为“超父”(super-parent), 然后通过交叉验证等模型选择方法来确定超父属性, 由此形成了 SPODE (Super-Parent ODE)方法. 例如, 在图 7.1(b)中,  $x_1$  是超父属性.

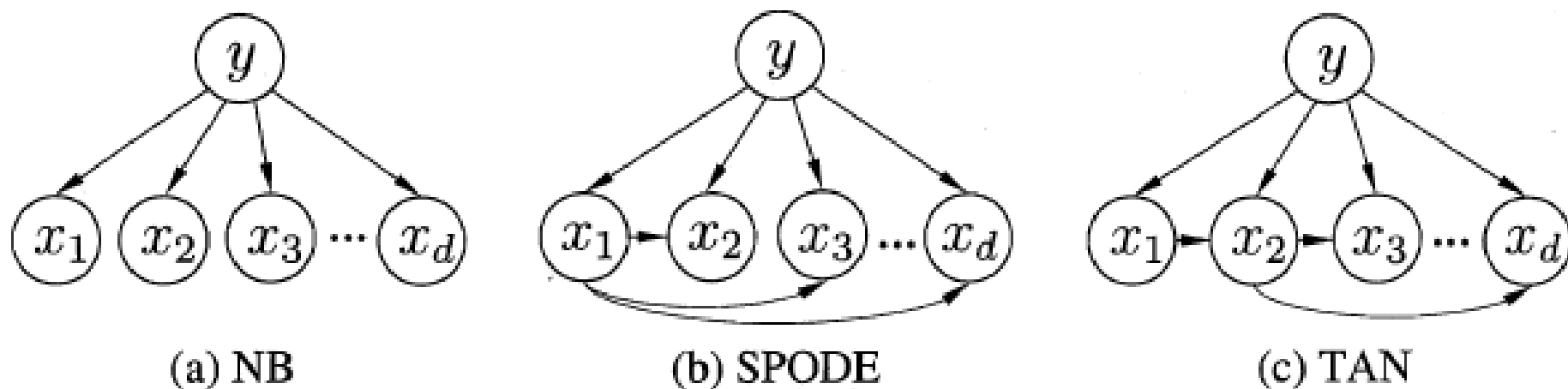


图 7.1 朴素贝叶斯与两种半朴素贝叶斯分类器所考虑的属性依赖关系

TAN (Tree Augmented naïve Bayes) [Friedman et al., 1997] 则是在最大带权生成树(maximum weighted spanning tree)算法 [Chow and Liu, 1968] 的基础上, 通过以下步骤将属性间依赖关系约简为如图 7.1(c) 所示的树形结构:

- (1) 计算任意两个属性之间的条件互信息(conditional mutual information)

$$I(x_i, x_j | y) = \sum_{x_i, x_j; c \in \mathcal{Y}} P(x_i, x_j | c) \log \frac{P(x_i, x_j | c)}{P(x_i | c)P(x_j | c)} ; \quad (7.22)$$

- (2) 以属性为结点构建完全图, 任意两个结点之间边的权重设为  $I(x_i, x_j | y)$ ;

- (3) 构建此完全图的最大带权生成树, 挑选根变量, 将边置为有向;

- (4) 加入类别结点  $y$ , 增加从  $y$  到每个属性的有向边.

容易看出，条件互信息 $I(x_i, x_j | y)$ 刻画了属性 $x_i$ 和 $x_j$ 均在已知类别情况下的相关性，因此，通过最大生成树算法， TAN 实际上仅保留了强相关属性之间的依赖性.

AODE (Averaged One-Dependent Estimator) [Webb et al., 2005] 是一种基于集成学习机制、更为强大的独依赖分类器. 与 SPODE 通过模型选择确定超父属性不同, AODE 尝试将每个属性作为超父来构建 SPODE, 然后将那些具有足够训练数据支撑的 SPODE 集成起来作为最终结果, 即

$$P(c | \mathbf{x}) \propto \sum_{\substack{i=1 \\ |D_{x_i}| \geq m'}}^d P(c, x_i) \prod_{j=1}^d P(x_j | c, x_i), \quad (7.23)$$

其中  $D_{x_i}$  是在第  $i$  个属性上取值与  $x_i$  相同的样本的集合,  $m'$  为阈值常数.

显然，AODE 需估计  $P(c, x_i)$  和  $P(x_j | c, x_i)$ 。类似式(7.20)，有

$$\hat{P}(c, x_i) = \frac{|D_{c,x_i}| + 1}{|D| + N_i}, \quad (7.24)$$

$$\hat{P}(x_j | c, x_i) = \frac{|D_{c,x_i,x_j}| + 1}{|D_{c,x_i}| + N_j}, \quad (7.25)$$

其中  $N_i$  是第  $i$  个属性可能的取值数,  $D_{c,x_i}$  是类别为  $c$  且在第  $i$  个属性上取值为  $x_i$  的样本集合,  $D_{c,x_i,x_j}$  是类别为  $c$  且在第  $i$  和第  $j$  个属性上取值分别为  $x_i$  和  $x_j$  的样本集合. 例如, 对西瓜数据集 3.0 有

$$\hat{P}_{\text{是,浊响}} = \hat{P}(\text{好瓜} = \text{是}, \text{敲声} = \text{浊响}) = \frac{6 + 1}{17 + 3} = 0.350,$$

$$\hat{P}_{\text{凹陷}|\text{是,浊响}} = \hat{P}(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{是}, \text{敲声} = \text{浊响}) = \frac{3 + 1}{6 + 3} = 0.444.$$



不难看出,与朴素贝叶斯分类器类似,AODE的训练过程也是“计数”,即在训练数据集上对符合条件的样本进行计数的过程.与朴素贝叶斯分类器相似,AODE无需模型选择,既能通过预计算节省预测时间,也能采取懒惰学习方式在预测时再进行计数,并且易于实现增量学习.

# EM算法

---

在前面的讨论中，我们一直假设训练样本所有属性变量的值都已被观测到，即训练样本是"完整"的.但在现实应用中往往会遇到"不完整"的训练样本，例如由于西瓜的根蒂已脱落，无法看出是"蜷缩"还是"硬挺"，则训练样本的"根蒂"属性变量值未知.在这种存在"未观测"变量的情形下，是否仍能对模型参数进行估计呢？

未观测变量的学名是“隐变量”(latent variable). 令  $\mathbf{X}$  表示已观测变量集,  $\mathbf{Z}$  表示隐变量集,  $\Theta$  表示模型参数. 若欲对  $\Theta$  做极大似然估计, 则应最大化对数似然

$$LL(\Theta | \mathbf{X}, \mathbf{Z}) = \ln P(\mathbf{X}, \mathbf{Z} | \Theta) . \quad (7.34)$$

然而由于  $\mathbf{Z}$  是隐变量, 上式无法直接求解. 此时我们可通过对  $\mathbf{Z}$  计算期望, 来

EM (Expectation-Maximization) 算法 [Dempster et al., 1977] 是常用的估计参数隐变量的利器, 它是一种迭代式的方法, 其基本想法是: 若参数  $\Theta$  已知, 则可根据训练数据推断出最优隐变量  $\mathbf{Z}$  的值 (E 步); 反之, 若  $\mathbf{Z}$  的值已知, 则可方便地对参数  $\Theta$  做极大似然估计 (M 步).

于是, 以初始值  $\Theta^0$  为起点, 对式(7.35), 可迭代执行以下步骤直至收敛:

- 基于  $\Theta^t$  推断隐变量  $\mathbf{Z}$  的期望, 记为  $\mathbf{Z}^t$ ;
- 基于已观测变量  $\mathbf{X}$  和  $\mathbf{Z}^t$  对参数  $\Theta$  做极大似然估计, 记为  $\Theta^{t+1}$ ;

这就是 EM 算法的原型.

进一步, 若我们不是取  $\mathbf{Z}$  的期望, 而是基于  $\Theta^t$  计算隐变量  $\mathbf{Z}$  的概率分布  $P(\mathbf{Z} | \mathbf{X}, \Theta^t)$ , 则 EM 算法的两个步骤是:

- **E 步 (Expectation):** 以当前参数  $\Theta^t$  推断隐变量分布  $P(\mathbf{Z} | \mathbf{X}, \Theta^t)$ , 并计算对数似然  $LL(\Theta | \mathbf{X}, \mathbf{Z})$  关于  $\mathbf{Z}$  的期望

$$Q(\Theta | \Theta^t) = \mathbb{E}_{\mathbf{Z}|\mathbf{X}, \Theta^t} LL(\Theta | \mathbf{X}, \mathbf{Z}) . \quad (7.36)$$

- **M 步 (Maximization):** 寻找参数最大化期望似然, 即

$$\Theta^{t+1} = \arg \max_{\Theta} Q(\Theta | \Theta^t) . \quad (7.37)$$

简要说来, EM 算法使用两个步骤交替计算: 第一步是期望(E)步, 利用当前估计的参数值来计算对数似然的期望值; 第二步是最大化(M)步, 寻找能使 E 步产生的似然期望最大化的参数值. 然后, 新得到的参数值重新被用于 E 步, ……直至收敛到局部最优解.

事实上, 隐变量估计问题也可通过梯度下降等优化算法求解, 但由于求和的项数将随着隐变量的数目以指数级上升, 会给梯度计算带来麻烦; 而 EM 算法则可看作一种非梯度优化方法.

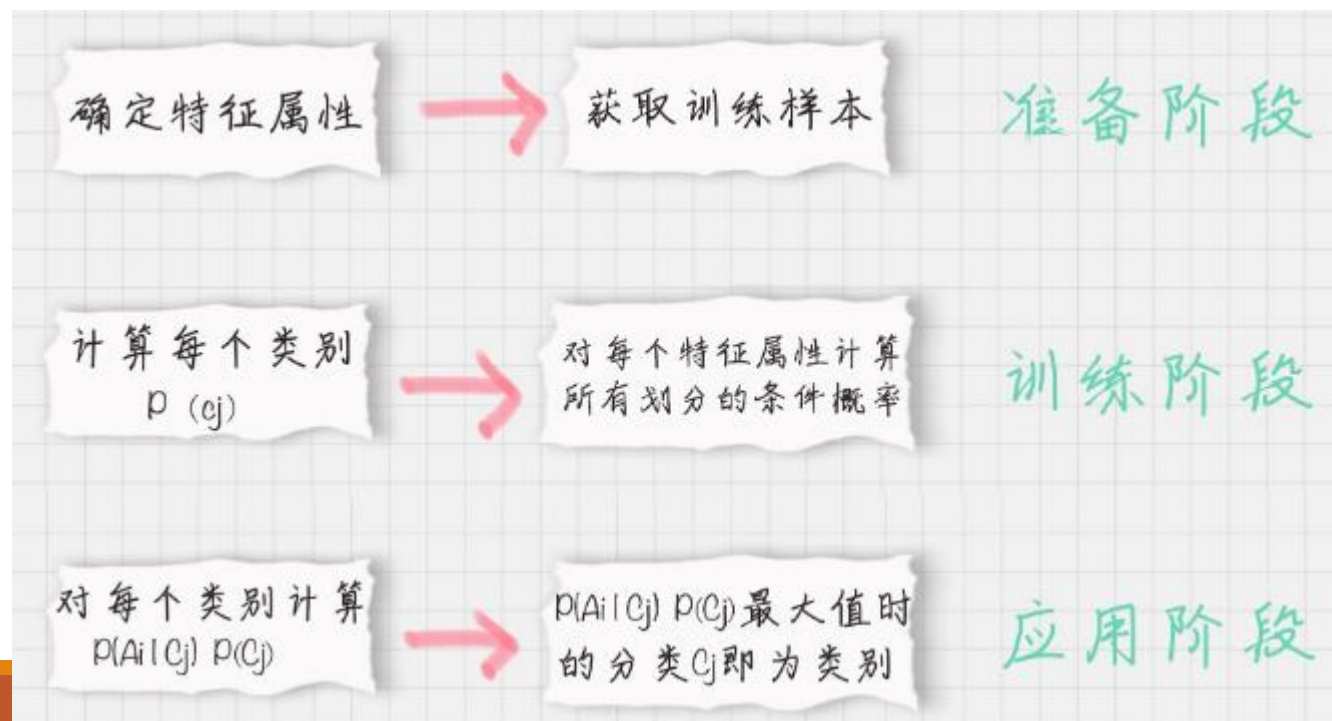
# 朴素贝叶斯应用实例

---

# 朴素贝叶斯分类器工作流程

朴素贝叶斯分类器工作流程：

朴素贝叶斯分类常用于文本分类，尤其是对于英文等语言来说，分类效果很好。它常用于垃圾文本过滤、情感预测、推荐系统等。





# 朴素贝叶斯分类器工作流程

---

## 第一阶段：准备阶段

在这个阶段我们需要确定特征属性，比如上面案例中的“身高”、“体重”、“鞋码”等，并对每个特征属性进行适当划分，然后由人工对一部分数据进行分类，形成训练样本。

这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

## 第二阶段：训练阶段

这个阶段就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率。

输入是特征属性和训练样本，输出是分类器。

## 第三阶段：应用阶段

这个阶段是使用分类器对新数据进行分类。输入是分类器和新数据，输出是新数据的分类结果。



# 朴素贝叶斯分类器应用--如何对文档进行分类

---

sklearn 机器学习包

sklearn 的全称叫 Scikit-learn，它给我们提供了 3 个朴素贝叶斯分类算法，分别是高斯朴素贝叶斯（GaussianNB）、多项式朴素贝叶斯（MultinomialNB）和伯努利朴素贝叶斯（BernoulliNB）。

这三种算法适合应用在不同的场景下

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

**高斯朴素贝叶斯**：特征变量是连续变量，符合高斯分布，比如说人的身高，物体的长度。

**多项式朴素贝叶斯**：特征变量是离散变量，符合多项分布，在文档分类中特征变量体现在一个单词出现的次数，或者是单词的 TF-IDF 值等。

**伯努利朴素贝叶斯**：特征变量是布尔变量，符合 0/1 分布，在文档分类中特征是单词是否出现。伯努利朴素贝叶斯是以文件为粒度，如果该单词在某文件中出现了即为 1，否则为 0。而多项式朴素贝叶斯是以单词为粒度，会计算在某个文件中的具体次数。而高斯朴素贝叶斯适合处理特征变量是连续变量，且符合正态分布（高斯分布）的情况。比如身高、体重这种自然界的现象就比较适合用高斯朴素贝叶斯来处理。而文本分类是使用多项式朴素贝叶斯或者伯努利朴素贝叶斯。

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 什么是 **TF-IDF** 值

TF-IDF 是一个统计方法，用来评估某个词语对于一个文件集或文档库中的其中一份文件的重要程度。

TF-IDF 实际上是两个词组 Term Frequency 和 Inverse Document Frequency 的总称，两者缩写为 TF 和 IDF，分别代表了词频和逆向文档频率。

词频 TF 计算了一个单词在文档中出现的次数，它认为一个单词的重要性和它在文档中出现的次数呈正比。

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

逆向文档频率 **IDF**，是指一个单词在文档中的区分度。它认为一个单词出现在的文档数越少，就越能通过这个单词把该文档和其他文档区分开。**IDF** 越大就代表该单词的区分度越大。

所以 **TF-IDF** 实际上是词频 **TF** 和逆向文档频率 **IDF** 的乘积。这样我们倾向于找到 **TF** 和 **IDF** 取值都高的单词作为区分，即这个单词在一个文档中出现的次数多，同时又很少出现在其他文档中。这样的单词适合用于分类。

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

TF-IDF 如何计算

$$\text{词频 TF} = \frac{\text{单词出现的次数}}{\text{该文档的总单词数}}$$

$$\text{逆向文档频率 IDF} = \log \frac{\text{文档总数}}{\text{该单词出现的文档数} + 1}$$

为什么 IDF 的分母中，单词出现的文档数要加 1 呢？因为有些单词可能不会存在文档中，为了避免分母为 0，统一给单词出现的文档数都加 1。

$$\text{TF-IDF} = \text{TF} * \text{IDF}$$

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

假设一个文件夹里一共有 10 篇文档，其中一篇文档有 1000 个单词，“this”这个单词出现 20 次，“bayes”出现了 5 次。“this”在所有文档中均出现过，而“bayes”只在 2 篇文档中出现过。我们来计算一下这两个词语的 TF-IDF 值。

针对“this”，计算 TF-IDF 值：

$$\text{词频 TF} = \frac{20}{1000} = 0.02$$

$$\text{TF-IDF} = 0.02 * (-0.0414) = -8.28e-4$$

$$\text{逆向文档频率 IDF} = \log \frac{10}{10 + 1} = -0.0414$$



# 朴素贝叶斯分类器应用--如何对文档进行分类

假设一个文件夹里一共有 10 篇文档，其中一篇文档有 1000 个单词，“this”这个单词出现 20 次，“bayes”出现了 5 次。“this”在所有文档中均出现过，而“bayes”只在 2 篇文档中出现过。我们来计算一下这两个词语的 TF-IDF 值。

针对“bayes”，计算 TF-IDF 值：

$$\text{词频 TF} = \frac{5}{1000} = 0.005$$

$$\text{TF-IDF} = 0.005 * 0.5229 = 2.61e-3。$$

$$\text{逆向文档频率 IDF} = \log \frac{10}{2 + 1} = 0.5229$$

很明显“bayes”的 TF-IDF 值要大于“this”的 TF-IDF 值。这就说明用“bayes”这个单词做区分比单词“this”要好。

# 朴素贝叶斯分类器应用--如何对文档进行分类

## 如何求TF-IDF

在 sklearn 中我们直接使用 **TfidfVectorizer** 类，它可以帮我们计算单词 TF-IDF 向量的值。在这个类中，取 sklearn 计算的对数  $\log$  时，底数是  $e$ ，不是 10。

TfidfVectorizer 类的创建：

**TfidfVectorizer(stop\_words=stop\_words, token\_pattern=token\_pattern)**

两个构造参数，可以自定义停用词 **stop\_words** 和规律规则 **token\_pattern**。需要注意的是传递的数据结构，停用词 **stop\_words** 是一个列表 List 类型，而过滤规则 **token\_pattern** 是正则表达式。

什么是停用词？停用词就是在分类中没有用的词，这些词一般词频 TF 高，但是 IDF 很低，起不到分类的作用。为了节省空间和计算时间，我们把这些词作为停用词 **stop words**，告诉机器这些词不需要帮我计算。

参数表	作用
stop_words	自定义停用词表，为列表List类型
token_pattern	过滤规则，正则表达式，如r"(?u)\b\w+\b"

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

当我们创建好 TF-IDF 向量类型时，可以用 **fit\_transform** 帮我们计算，返回给我们文本矩阵，该矩阵表示了每个单词在每个文档中的 **TF-IDF 值**。

方法表	作用
<code>fit_transform(X)</code>	拟合模型，并返回文本矩阵

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

在我们进行 `fit_transform` 拟合模型后，我们可以得到更多的 TF-IDF 向量属性，比如，我们可以得到词汇的对应关系（字典类型）和向量的 IDF 值，当然也可以获取设置的停用词 `stop_words`。

属性表	作用
<code>vocabulary_</code>	词汇表；字典类型
<code>idf_</code>	返回idf值
<code>stop_words_</code>	返回停用词表

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

举例： 文档 1 : this is the bayes document ;

文档 2 : this is the second second document ;

文档 3 : and the third one ;

文档 4 : is this the document。

现在想要计算文档里都有哪些单词，这些单词在不同文档中的 TF-IDF 值是多少呢？

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

创建 **TfidfVectorizer** 类

```
from sklearn.feature_extraction.text import TfidfVectorizer  
tfidf_vec = TfidfVectorizer()
```

创建 4 个文档的列表 **documents**，并让创建好的 **tfidf\_vec** 对 **documents** 进行拟合，得到 TF-IDF 矩阵：

```
documents = [  
    'this is the bayes document',  
    'this is the second second document',  
    'and the third one',  
    'is this the document'  
]  
tfidf_matrix = tfidf_vec.fit_transform(documents)
```

# 朴素贝叶斯分类器应用--如何对文档进行分类


---

输出文档中所有不重复的词:

```
print('不重复的词:', tfidf_vec.get_feature_names())
```

运行结果

不重复的词: ['and', 'bayes', 'document', 'is', 'one', 'second', 'the', 'third', 'this']



按照字母  
序排列



# 朴素贝叶斯分类器应用--如何对文档进行分类

---

输出每个单词对应的 id 值:

```
print('每个单词的 ID:', tfidf_vec.vocabulary_)
```

运行结果

每个单词的 ID: {'this': 8, 'is': 3, 'the': 6, 'bayes': 1, 'document': 2, 'second': 5, 'and': 0, 'third': 7, 'one': 4}

# 朴素贝叶斯分类器应用--如何对文档进行分类

输出每个单词在每个文档中的 TF-IDF 值，向量里的顺序是按照词语的 id 顺序来的：

```
1 每个单词的 tfidf 值: [[0.          0.63314609 0.40412895 0.40412895 0.          0.
2   0.33040189 0.          0.40412895]
3  [0.          0.          0.27230147 0.27230147 0.          0.85322574
4   0.22262429 0.          0.27230147]
5  [0.55280532 0.          0.          0.          0.55280532 0.
6   0.28847675 0.55280532 0.          ]
7  [0.          0.          0.52210862 0.52210862 0.          0.
8   0.42685801 0.          0.52210862]]
```

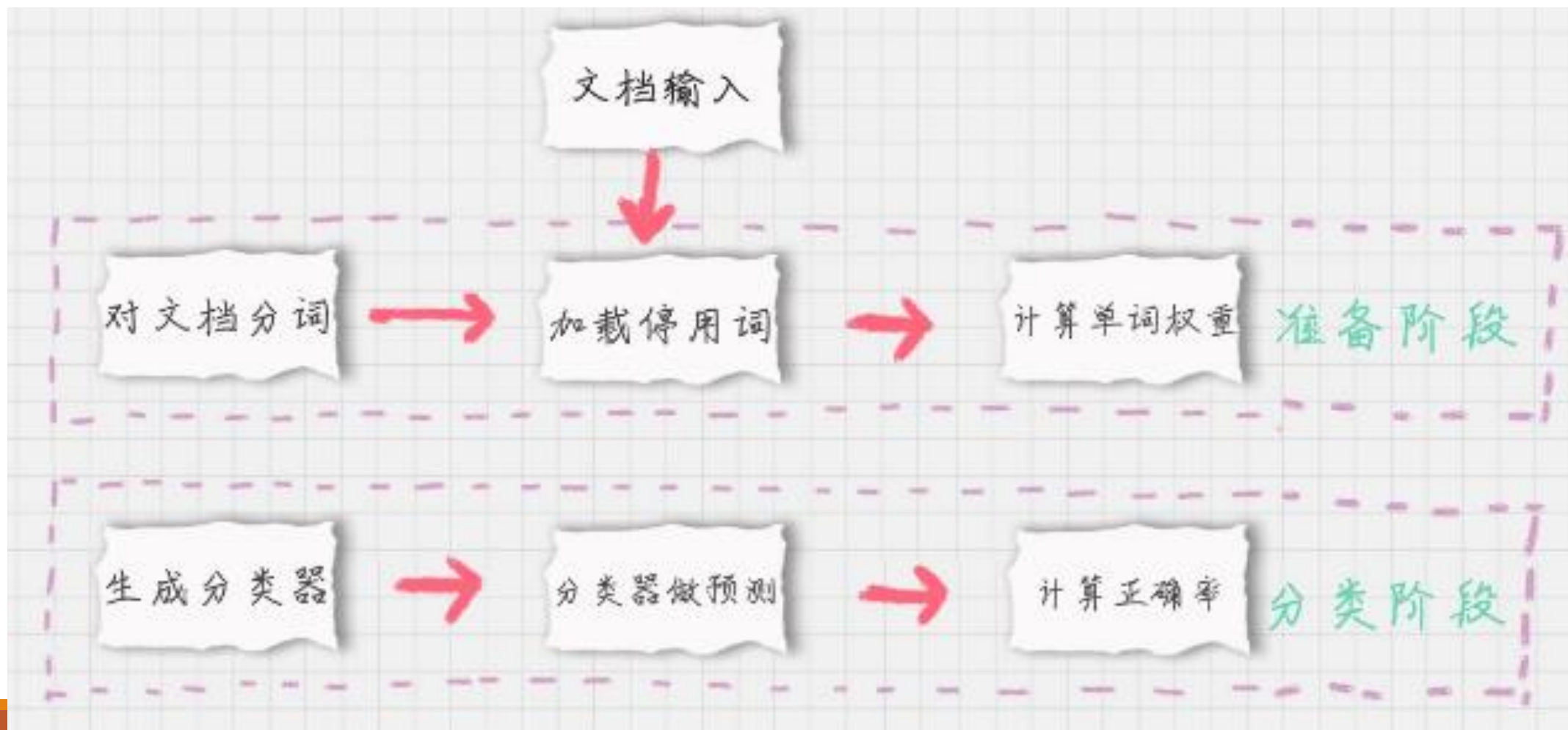
# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 如何对文档进行分类

- 基于分词的数据准备，包括分词、单词权重计算、去掉停用词；
- 应用朴素贝叶斯分类进行分类，首先通过训练集得到朴素贝叶斯分类器，然后将分类器应用于测试集，并与实际结果做对比，最终得到测试集的分类准确率。

# 朴素贝叶斯分类器应用--如何对文档进行分类



# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 1: 对文档进行分词

在准备阶段里，最重要的就是分词。那么如果给文档进行分词呢？英文文档和中文文档所使用的分词工具不同。

在英文文档中，最常用的是 NTLK 包。NTLK 包中包含了英文的停用词 stop words、分词和标注方法。

```
import nltk  
word_list = nltk.word_tokenize(text) # 分词  
nltk.pos_tag(word_list) # 标注单词的词性
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

在中文文档中，最常用的是 jieba 包。jieba 包中包。jieba 包中包含了中文的停用词 stop words 和分词方法。

```
import jieba
```

```
word_list = jieba.cut (text) # 中文分词
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 2: 加载停用词表

需要自己读取停用词表文件，从网上可以找到中文常用的停用词保存在 `stop_words.txt`，然后利用 Python 的文件读取函数读取文件，保存在 `stop_words` 数组中。

```
stop_words = [line.strip().decode('utf-8') for line in  
io.open('stop_words.txt').readlines()]
```

```
stop_words = [line.strip() for line in io.open('text  
classification/stop/stopword.txt',encoding='utf-  
8',errors='ignore').readlines()]
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 3: 计算单词的权重

直接创建 `TfidfVectorizer` 类，然后使用 `fit_transform` 方法进行拟合，得到 TF-IDF 特征空间 `features`，你可以理解为选出来的分词就是特征。我们计算这些特征在文档上的特征向量，得到特征空间 `features`。

```
tf = TfidfVectorizer(stop_words=stop_words, max_df=0.5)
```

```
features = tf.fit_transform(train_contents)
```

这里 `max_df` 参数用来描述单词在文档中的最高出现率。假设 `max_df=0.5`，代表一个单词在 50% 的文档中都出现过了，那么它只携带了非常少的信息，因此就不作为分词统计。一般很少设置 `min_df`，因为 `min_df` 通常都会很小。



# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 4：生成朴素贝叶斯分类器

我们将特征训练集的特征空间 `train_features`，以及训练集对应的分类 `train_labels` 传递给贝叶斯分类器 `clf`，它会自动生成一个符合特征空间和对应分类的分类器。

这里我们采用的是多项式贝叶斯分类器，其中 `alpha` 为平滑参数。为什么要使用平滑呢？因为如果一个单词在训练样本中没有出现，这个单词的概率就会被计算为 0。但训练集样本只是整体的抽样情况，我们不能因为一个事件没有观察到，就认为整个事件的概率为 0。为了解决这个问题，我们需要做平滑处理。

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

当  $\alpha=1$  时，使用的是 Laplace 平滑。Laplace 平滑就是采用加 1 的方式，来统计没有出现过的单词的概率。这样当训练样本很大的时候，加 1 得到的概率变化可以忽略不计，也同时避免了零概率的问题。

当  $0<\alpha<1$  时，使用的是 Lidstone 平滑。对于 Lidstone 平滑来说， $\alpha$  越小，迭代次数就越多，精度越高。我们可以设置  $\alpha$  为 0.001。

## # 多项式贝叶斯分类器

```
from sklearn.naive_bayes import MultinomialNB
```

```
clf = MultinomialNB(alpha=0.001).fit(train_features, train_labels)
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 5：使用生成的分类器做预测

首先我们需要得到测试集的特征矩阵

方法是用训练集的分词创建一个 `TfidfVectorizer` 类，使用同样的 `stop_words` 和 `max_df`，然后用这个 `TfidfVectorizer` 类对测试集的内容进行 `fit_transform` 拟合，得到测试集的特征矩阵 `test_features`。

```
test_tf = TfidfVectorizer(stop_words=stop_words, max_df=0.5,  
vocabulary=train_vocabulary)
```

```
test_features=test_tf.fit_transform(test_contents)
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

然后用训练好的分类器对新数据做预测。

方法是使用 `predict` 函数，传入测试集的特征矩阵 `test_features`，得到分类结果 `predicted_labels`。`predict` 函数做的就是求解所有后验概率并找出最大的那个。

```
predicted_labels=clf.predict(test_features)
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

---

## 模块 6: 计算准确率

计算准确率实际上是对分类模型的评估。我们可以调用 sklearn 中的 metrics 包，在 metrics 中提供了 accuracy\_score 函数，方便我们对实际结果做对比，给出模型的准确率。

使用方法如下：

```
from sklearn import metrics
```

```
print metrics.accuracy_score(test_labels, predicted_labels)
```

# 朴素贝叶斯分类器应用--如何对文档进行分类

## 数据挖掘神器 sklearn

从数据挖掘的流程来看，一般包括了获取数据、数据清洗、模型训练、模型评估和模型部署这几个过程。

sklearn 中包含了大量的数据挖掘算法，比如三种朴素贝叶斯算法，只需要了解不同算法的适用条件，以及创建时所需的参数，就可以用模型帮我们进行训练。在模型评估中，sklearn 提供了 metrics 包，帮我们对预测结果与实际结果进行评估。

在文档分类的项目中，针对文档的特点，给出了基于分词的准备流程。一般来说 NTLK 包适用于英文文档，而 jieba 适用于中文文档。我们可以根据文档选择不同的包，对文档提取分词。这些分词就是贝叶斯分类中最重要的特征属性。基于这些分词，我们得到分词的权重，即特征矩阵。

通过特征矩阵与分类结果，就可以创建出朴素贝叶斯分类器，然后用分类器进行预测，最后预测结果与实际结果做对比即可以得到分类器在测试集上的准确率。

# 朴素贝叶斯分类

## sklearn工具

### 三种朴素贝叶斯分类算法

高斯朴素贝叶斯：GaussianNB

多项式朴素贝叶斯：MultinomialNB

贝努力朴素贝叶斯：BernoulliNB

### TF-IDF值

概念：词频TF，逆向文档频率IDF

使用sklearn求TF-IDF：TfidfVectorizer类

## 如何对文档进行分类

准备阶段：对文档分词、加载停用词、计算单词权重

分类阶段：生成分类器、分类器做预测、计算准确率