第一章　朴素贝叶斯原理

1.1　概述

贝叶斯分类是一类分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。

本章首先介绍贝叶斯分类算法的重点和核心——贝叶斯定理。最后，我们通过实例来讨论贝叶斯分类的中最简单的一种: 朴素贝叶斯分类。

1.2　贝叶斯思维

贝叶斯思维的全过程可以如下图所示：

1.3　条件概率

我们通过一个实际例子来解释：

已知：现有一盒巧克力，一种装了 16 块。其中黑色，白色，棕色巧克力各 4 块，红色和黄色巧克力各 2 块。

问：

（1）随机取出一块黑色巧克力的可能性是多少？

（2）随机取出一块红色巧克力的可能性是多少？

我们很容易得知，问题（1）是 $\frac{4}{16}$，问题（2）是 $\frac{2}{16}$。

我们把题目条件改动一下，把 16 块巧克力分装到 A 和 B 两个盒子中。其中 A 盒中有 3 块黑色，2 块白色，1 块棕色，1 块黄色。B 盒中有 1 块黑色，2 块白色，3 块棕色，1 块黄色，2 块红色。

问：已知巧克力出自 A 盒，取出黑色巧克力的概率为?

我们不难看出，$P(Black|Box-A)=\frac{P(Black\ and\ Box-A)}{P(Box-A)}$

其中，$P(Black\ and\ Box-A)=\frac{3}{16}, P(Box-A)=\frac{7}{16}$

所以，我们最后求得的概率为 $\frac{3}{7}$

上面就是条件概率的形式，我们接下来看一下条件概率的公式：

定理.

$$p(X=x|Y=y)=\frac{p(X=x,Y=y)}{p(Y=y)}$$

1.4　贝叶斯定理：逆概率思维

我们还是取上面的例子。但是问题发生变化。

问：已知取出一块黑色的巧克力，它来自 A 盒的概率。

我们不难得知，$P(Box-A|Black)=\frac{P(Black\ and\ Box-A)}{P(Black)}$

$$P(Black\ and\ Box-A)=P(Black|Box-A)\times P(Box-A)$$

$$P(Black)=P(Black|Box-A)\times P(Box-A)+P(Black|Box-B)\times P(Box-B)$$

具体数值不再去计算。接下来我们引入贝叶斯定理的公式：

定理. 已知：

存在 K 类 $c_1, c_2, \cdots, c_K$，给定一个新的实例 $x=(x^{(1)}, x^{(2)}, \cdots, x^{(n)})$

问：该实例归属于第 c_i 类的可能性有多大？

$$P(Y=c_i|X=x)=\frac{P(X=x|Y=c_i)P(Y=c_i)}{P(X=x)}$$

即：

$$P(Y=c_i|X=x)=\frac{P(X=x|Y=c_i)P(Y=c_i)}{\sum_{i=1}^{K}P(X=x|Y=c_i)P(Y=c_i)}$$

1.5　贝叶斯分类

定理. 已知：

存在 K 类 $c_1, c_2, \cdots, c_K$，给定一个新的实例 $x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})$

问：该实例归属于哪一类？

$$P(Y = c_i|X = x) = \frac{P(X = x|Y = c_i)P(Y = c_i)}{\sum_{i=1}^{K} P(X = x|Y = c_i)P(Y = c_i)}$$

我们的目标是：

$$\arg\max_{c_i} P(X = x|Y = c_i)P(Y = c_i)$$

1.6　朴素贝叶斯

朴素贝叶斯相对于我们上面的贝叶斯分类，其实是多了一个条件的，即：假设实例特征之间是相互独立的。

接下来我们对公式进行一定的展开：

$$\begin{aligned}
&P(X = x|Y = c_i) = \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i) \\
&\Rightarrow P(X = x) = \sum_{i=1}^{K} P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i) \\
&\Rightarrow P(Y = c_i|X = x) = \frac{P(X = x|Y = c_i)P(Y = c_i)}{\sum_{i=1}^{K} P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)} \\
&\Rightarrow P(Y = c_i|X = x) = \frac{P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)}{\sum_{i=1}^{K} P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)}
\end{aligned}$$

因此，我们得到朴素贝叶斯分类的定义如下：

定理. 已知：

存在 K 类 $c_1, c_2, \cdots, c_K$，给定一个新的实例 $x = (x^{(1)}, x^{(2)}, \cdots, x^{(n)})$

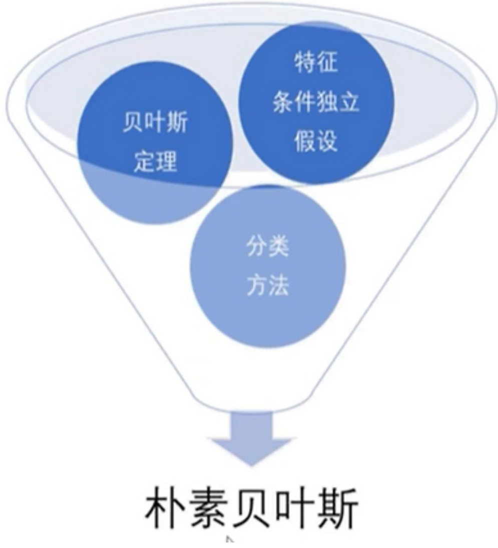
问：该实例归属于哪一类？

$$P(Y = c_i|X = x) = \frac{P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)}{\sum_{i=1}^{K} P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)}$$

我们的目标是：

$$\arg\max_{c_i} P(Y = c_i) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_i)$$

1.7　朴素贝叶斯-基本方法



1.7.1　基本方法

训练数据集：

$$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$$

输入为：$X\in R^n, x\in X$

输出为：$Y=\{c_1,c_2,\cdots,c_K\}, y\in Y$

生成方法：学习联合概率分布 $P(X,Y)$。形成的方式有：

(1) 先验概率分布：

$$P(Y=c_i), i=1,2,\cdots,K$$

(2) 条件概率分布

$$P(X=x|Y=c_i)=P(X^{(1)}=x^{(1)},\cdots,X^{(n)}=x^{(n)}|Y=c_i)$$

(3) 联合概率分布

$$P(X,Y)=P(X=x|Y=c_i)P(Y=c_i), i=1,2,\cdots,K$$

接下来我们通过一个例子来看一下以上三种方法是怎么进行计算的。

我们还举巧克力的例子。

A 盒中有三个黑色巧克力，两个白色巧克力，一个棕色巧克力以及一个黄色巧克力；B 盒中有一个黑色巧克力，两个白色巧克力，三个棕色巧克力，一个黄色巧克力以及两个红色巧克力。

这里的先验概率是：

$$P(A)=\frac{7}{16},P(B)=\frac{9}{16}$$

接下来我们看一下条件概率分布：

$$\begin{aligned}
P(black|A)&=\frac{P(black,A)}{P(A)}=\frac{3}{7}\\
P(black|B)&=\frac{P(black,B)}{P(A)}=\frac{1}{9}\\
P(white|A)&=\frac{P(white,A)}{P(A)}=\frac{2}{7}\\
P(white|B)&=\frac{P(white,B)}{P(A)}=\frac{2}{9}
\end{aligned}$$

剩下的运算方式相同，就不再赘述。

最后我们的联合概率分布为：

$P(X,Y)$	A盒	B盒
黑	$\frac{3}{7}\times\frac{7}{16}=\frac{3}{16}$	$\frac{1}{9}\times\frac{9}{16}=\frac{1}{16}$
白	$\frac{2}{7}\times\frac{7}{16}=\frac{2}{16}$	$\frac{2}{9}\times\frac{9}{16}=\frac{2}{16}$
棕	$\frac{1}{7}\times\frac{7}{16}=\frac{1}{16}$	$\frac{3}{9}\times\frac{9}{16}=\frac{3}{16}$
黄	$\frac{1}{7}\times\frac{7}{16}=\frac{1}{16}$	$\frac{1}{9}\times\frac{9}{16}=\frac{1}{16}$
红	$\frac{0}{7}\times\frac{7}{16}=\frac{0}{16}$	$\frac{2}{9}\times\frac{9}{16}=\frac{2}{16}$

1.7.2 朴素

为什么要作条件独立性假设？

我们不妨举个例子来理解。如何定义一个人是否帅气？

我们可以选取的特征为：身高、体重、脸型、鼻型。种类分为两类—帅和不帅。

假设每个特征也是两种选择，那么我们面对的情况有 $2^5=32$ 种。

我们放到一般化的例子中。假设有 n 个特征，每个特征 $x^{(j)}$ 可能的取值有 S_j 个，y 的可能取值有 K 个，那么我们面对的总个数为：$K\prod_{j=1}^{n}S_j$ 个。可能是一个非常大的数字。

因此，我们在朴素贝叶斯中加入独立性假设使得计算具有可行性，能够计算出联合概率分布。

定理. 朴素贝叶斯：

假设：实例特征之间相互独立

$$
\begin{aligned}
P(X=x|Y=c_i) &= P(X^{(1)}=x^{(1)},\cdots,X^{(n)}=x^{(n)}|Y=c_i) \\
&= \prod_{j=1}^{n} P(X^{(j)}=x^{(j)}|Y=c_i)
\end{aligned}
$$

1.8 后验概率最大化

对应的朴素贝叶斯分类如下：

定理. 朴素贝叶斯分类：

已知：

存在 K 类 $c_1,c_2,\cdots,c_K$，给定一个新的实例 $x=(x^{(1)},x^{(2)},\cdots,x^{(n)})$

问：该实例归属于哪一类？

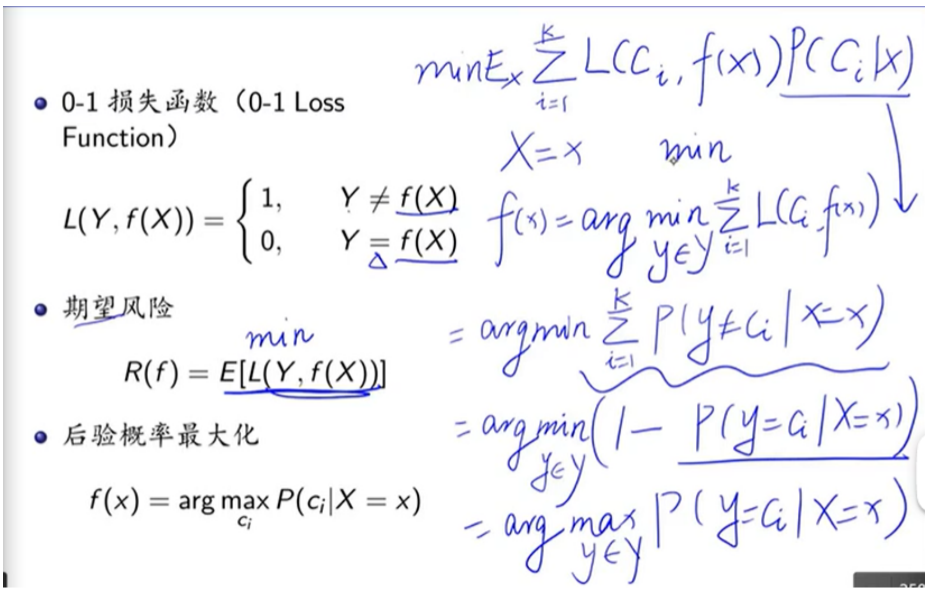
（1）后验概率

$$
P(Y=c_i|X=x)=\frac{P(Y=c_i)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_i)}{\sum_{i=1}^{K}P(Y=c_i)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_i)}
$$

（2）分类

$$
y=\arg\max_{c_i} P(Y=c_i)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_i)
$$

朴素贝叶斯法将实例分到后验概率最大的类中，这等价于期望风险最小化。具体的推导过程见《统计学习方法》。



1.9　极大似然估计

定理. 极大似然估计

原理; 使得似然函数（即联合密度函数）达到最大的参数值

假设 X 的密度函数为 $f(X,\beta)$，如果简单随机样本 $X_1,X_2,\cdots,X_N$ 相互独立，则其联合密度函数为：

$$L(x_1,\cdots,x_N;\beta)=\prod_{i=1}^{N}f(x_i,\beta)$$

当 $(X_1,X_2,\cdots,X_N)$ 取定值 $(x_1,x_2,\cdots,x_N)$ 时，$L(x_1,x_2,\cdots,x_N;\beta)$ 是 β 的函数，即样本的似然函数。

β 的极大似然估计 $\hat{\beta}$：

$$\hat{\beta}=\arg\max_{\beta\in\Theta}L(x_1,\cdots,x_N;\beta)$$

记似然函数 $L(\beta)=L(x_1,\cdots,x_N;\beta)$

- 直接通过似然函数 $L(\beta)$ 求解
  - 当 $L(\beta)$ 可微时，可通过方程组

$$\frac{\partial L(\beta)}{\partial\beta_1}=0,\frac{\partial L(\beta)}{\partial\beta_2}=0,\cdots,\frac{\partial L(\beta)}{\partial\beta_m}=0$$

  求得 $L(\beta)$ 的极大值点。
  - 当 $L(\beta)$ 不存在偏导数时，需要直接研究 $L(\beta)$，寻找最大值点。
- 通过对数似然函数 $\ln L(\beta)$ 求解 $\hat{\beta}$ 也是 $\ln L(\beta)$ 的最大值点，
  - 当 $\ln L(\beta)$ 可微时，可通过下列方程组，

$$\frac{\partial\ln L(\beta)}{\partial\beta_1}=0,\frac{\partial\ln L(\beta)}{\partial\beta_2}=0,\cdots,\frac{\partial\ln L(\beta)}{\partial\beta_m}=0$$

  求解，判断根是不是最大值点。
  - 当 $\ln L(\beta)$ 不存在偏导数时，需要研究 $\ln L(\beta)$，寻找最大值点。

在朴素贝叶斯算法中，学习意味着估计 $P(Y=c_k)$ 和 $P(X^{(j)}=x^{(j)}|Y=c_k)$。可以应用极大似然估计法估计相应的概率。先验概率 $P(Y=c_k)$ 的极大似然估计是：

$$P\ Y=c_k)=\frac{\sum_{i=1}^{N}I(y_i=c_k)}{N},k=1,2,\cdots,K$$

设第 j 个特征 $x^{(j)}$ 可能取值的集合为 $\{a_{j1},a_{j2},\cdots,a_{js_j}\}$,条件概率 $P(X^{(j)}=a_{jl}|Y=c_k)$ 的极大似然估计是:

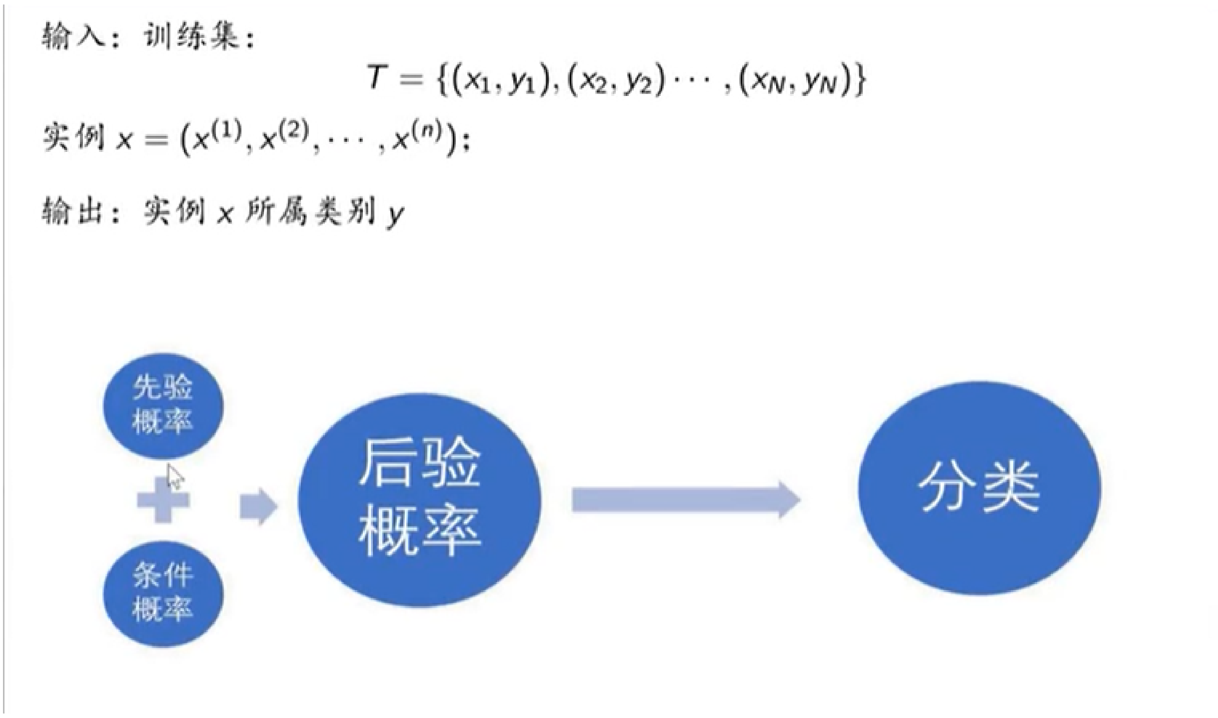
$$P(X^{(j)}=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^{N}I(x_i^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^{N}I(y_i=c_k)}$$

$$j=1,2,\cdots,n;l=1,2,\cdots,S_j;k=1,2,\cdots,K$$

上式中，$x_i^{(j)}$ 是第 i 个样本的第 j 个特征；a_{jl} 是第 j 个特征可能取的第 l 个值；I 为指示函数。

1.10 朴素贝叶斯算法

1.10.1 算法详解



具体步骤可以如下所示:

输入: 训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$,其中 $x_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})^T$，$x_i^{(j)}$ 是第 i 个样本的第 j 个特征，$x_i^{(j)}\in\{a_{j1},a_{j2},\cdots,a_{jS_j}\}$，$a_{jl}$ 是第 j 个特征可能取的第 l 个值，$j=1,2,\cdots,n; l=1,2,\cdots,S_j; y_i\in\{c_1,c_2,\cdots,c_K\}$；实例 x。

输出: 实例 x 的分类。

定理. 朴素贝叶斯算法过程

（1）计算先验概率和条件概率

$$P(Y=c_k)=\frac{\sum_{i=1}^N I(y_i=c_k)}{N},k=1,2,\cdots,K$$

$$P(X^{(j)}=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^N I(y_i=c_k)}$$

$$j=1,2,\cdots,n;l=1,2,\cdots,S_j;k=1,2,\cdots,K$$

（2）对于给定的实例 $x=(x^{(1)},x^{(2)},\cdots,x^{(n)})^T$，计算：

$$P(Y=c_k)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_k),k=1,2,\cdots,K$$

（3）确定实例 x 的类

$$y=\arg\max_{c_k}P(Y=c_k)\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_k)$$

1.10.2　例题解读

输入：训练集：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
$X^{(2)}$	S	M	M	S	S	S	M	M	L	L	L	M	M	L	L
Y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

输出：实例点 $x=(2,S)^T$ 的类标记 y.

根据算法，容易计算下列概率：

$P(Y=1)=\frac{9}{15},P(Y=-1)=\frac{6}{15}$

$P(X^{(1)}=1|Y=1)=\frac{2}{9},P(X^{(1)}=2|Y=1)=\frac{3}{9},P(X^{(1)}=3|Y=1)=\frac{4}{9}$

$P(X^{(2)}=S|Y=1)=\frac{1}{9},P(X^{(2)}=M|Y=1)=\frac{4}{9},P(X^{(2)}=L|Y=1)=\frac{4}{9}$

$P(X^{(1)}=1|Y=-1)=\frac{3}{6},P(X^{(1)}=2|Y=-1)=\frac{2}{6},P(X^{(1)}=3|Y=-1)=\frac{1}{6}$

$P(X^{(2)}=S|Y=-1)=\frac{3}{6},P(X^{(2)}=M|Y=-1)=\frac{2}{6},P(X^{(2)}=L|Y=-1)=\frac{1}{6}$

对于给定的 $x=(2,S)$ 计算：

$$P(Y=1)P(X^{(1)}=2|Y=1)P(X^{(2)}=S|Y=1)=\frac{9}{15}\times\frac{3}{9}\times\frac{1}{9}=\frac{1}{45}$$

$$P(Y=-1)P(X^{(1)}=2|Y=-1)P(X^{(2)}=S|Y=-1)=\frac{6}{15}\times\frac{2}{6}\times\frac{3}{6}=\frac{1}{15}$$

因为 $P(Y=-1)P(X^{(1)}=2|Y=-1)P(X^{(2)}=S|Y=-1)$ 最大，所以 $y=-1$。

1.11 贝叶斯估计

为什么讲了极大似然估计后还要学习贝叶斯估计呢？我们先来看一个例子。

如果我们想估计女性占总人数的比例，训练数据集选择的是女儿国，选取的人数为 N 个，而其中女性的人数为 N 个。根据极大似然估计，我们得出的女性人口的概率为 $p=\frac{N}{N}=1$。

很显然，上面的判断是不太合适的。于是我们引入了贝叶斯估计。

1.11.1 贝叶斯估计的估计方法

定理. 估计方法

先验概率的贝叶斯估计:

$$P_\lambda(Y=c_k)=\frac{\sum_{i=1}^{N} I(y_i=c_k)+\lambda}{N+K\lambda}$$

条件概率的贝叶斯估计:

$$P_\lambda(X^{(j)}=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^{N} I(x_i^{(j)}=a_{jl},y_i=c_k)+\lambda}{\sum_{i=1}^{N} I(y_i=c_k)+S_j\lambda}$$

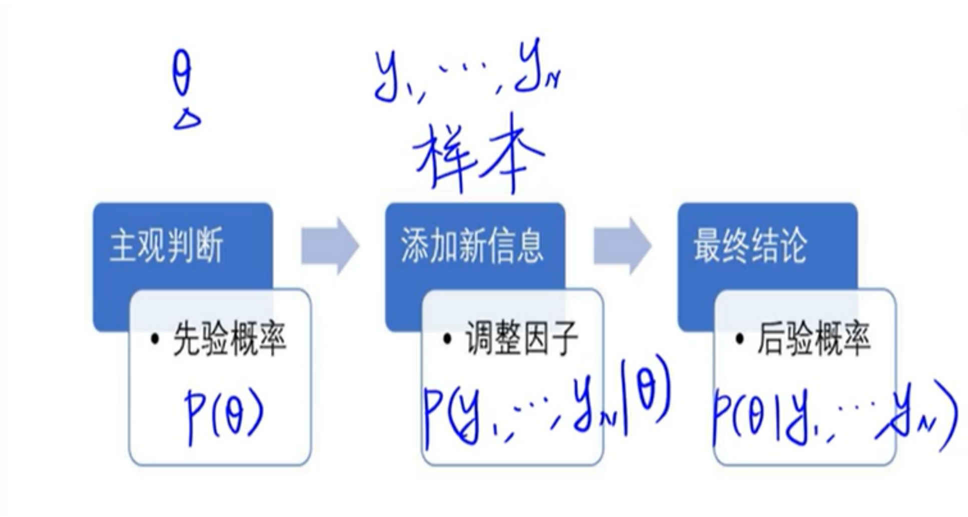
注:$\lambda\geq 0$，$\lambda=0$ 时为极大似然估计，$\lambda=1$ 时为拉普拉斯平滑（*Laplacian Smoothing*）。

显然，对任何 $l=1,2,\cdots,S_j$，$k=1,2,\cdots,K$，有:

$$P(X^{(j)}=a_{jl}|Y=c_k)>0$$
$$\sum_{l=1}^{S_j} P(X^{(j)}=a_{jl}|Y=c_k)=1$$

1.11.2　估计方法: 为什么被称作贝叶斯估计？

现在我们想要估计参数 θ，步骤可以如下图所示:



我们来看一个小例子:

我们先列出贝叶斯公式:

$$P(\theta|Y) = \frac{P(Y|\theta)P(\theta)}{P(Y)}$$

例:Y 分为两类 c_1, c_2，相应的参数为 θ_1，假设参数服从 $Beta$ 分布 $Be(\alpha, \beta)$，求参数 θ_1 的贝叶斯估计。

参数 θ_1 的先验概率为:

$$f(\theta_1; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} \theta_1^{\alpha-1}(1-\theta_1)^{\beta-1}$$

已知 θ_1 时 Y 的条件概率:

$$g(Y|\theta_1) = \begin{cases} \theta_1 & Y = c_1 \\ 1-\theta_1 & Y = c_2 \end{cases}$$

我们有这样一组样本:$\{y_1, y_2, \cdots, y_N\}$，$c_1$ 类的点的个数为 N_1 个，那么 c_2 类点的个数为 $N - N_1$ 个。

对应的先验概率为:

$$\frac{1}{B(\alpha, \beta)} \theta_1^{\alpha-1}(1-\theta_1)^{\beta-1}$$

准则是后验概率最大化，我们可以最大化分子来实现:

$$\arg\max_{\theta_1} [\theta_1^{N_1}(1-\theta_1)^{N-N_1} \frac{1}{B(\alpha, \beta)} \theta_1^{\alpha-1}(1-\theta_1)^{\beta-1}]$$

因为 $\frac{1}{B(\alpha,\beta)}$ 是一个常量，可以不用考虑。简化后得到新的等式为:

$$\begin{aligned}&\arg\max_{\theta_1}[\theta_1^{N_1}(1-\theta_1)^{N-N_1}\theta_1^{\alpha-1}(1-\theta_1)^{\beta-1}]\\&=\arg\max_{\theta_1}[\theta_1^{N_1+\alpha-1}(1-\theta_1)^{N-N_1+\beta-1}]\end{aligned}$$

为了求得最大值，我们对括号内的函数对 θ_1 求导数:

$$\begin{aligned}&\frac{d(\theta_1^{N_1+\alpha-1}(1-\theta_1)^{N-N_1+\beta-1})}{d\theta_1}\\&=(N_1+\alpha-1)\theta_1^{N_1+\alpha-2}(1-\theta_1)^{N-N_1+\beta-1}-\theta_1^{N_1+\alpha-1}\cdot(N-N_1+\beta-1)\cdot(1-\theta_1)^{N-N_1+\beta-2}\\&=0\\&\Rightarrow\theta_1^{N_1+\alpha-2}(1-\theta_1)^{N-N_1+\beta-2}\cdot[(N+\alpha-1)(1-\theta_1)-(N-N_1+\beta-1)\theta_1]=0\\&\Rightarrow\theta_1=\frac{N_1+\alpha-1}{N+(\alpha-1)+(\beta-1)}\end{aligned}$$

如果我们令 $\alpha=\beta$，$\alpha-1=\lambda$，我们便可以得到:

$$\theta_1=\frac{N+\lambda}{N+2\lambda}$$

此时如果令 $K=2$，$N_1=\sum_{i=1}^{N}I(y_i=c_i)$, 我们便可以看出这个形式就是拉普拉斯光滑。

上面我们假设参数分布服从于 $beta$ 分布，那如果参数服从均匀分布 $U(0,1)$ 呢？

已知参数 θ_1 的先验概率为:$f(\theta_1)$=1;

已知 θ_1 时 Y 的条件概率为:

$$g(Y|\theta_1)=\begin{cases}\theta_1 & Y=c_1\\1-\theta_1 & Y=c_2\end{cases}$$

我们有这样一组样本:$\{y_1,y_2,\cdots,y_N\}$，c_1 类的点的个数为 N_1 个，那么 c_2 类点的个数为 $N-N_1$ 个。

我们求 θ_1 的估计采用的式子为:

$$\begin{aligned}&\arg\max_{\theta_1}[\theta_1^{N_1}(1-\theta_1)^{N-N_1}\cdot 1]\\&\Rightarrow\frac{d(\theta_1^{N_1}(1-\theta_1)^{N-N_1}\cdot 1)}{d\theta_1}=0\\&\Rightarrow N_1\theta_1^(N_1-1)(1-\theta_1)^(N-N_1)-\theta_1^{N_1}\cdot(N-N_1)=0\\&\Rightarrow\theta_1^{N_1-1}(1-\theta_1)^{N-N_1-1}\cdot[N_1(1-\theta_1)-(N-N_1)\theta_1]=0\\&\Rightarrow\theta_1=\frac{N_1}{N}\end{aligned}$$

上面的式子就是极大似然估计。

1.11.3　平滑思想

我们知道贝叶斯估计的表达式为:

$$P_\lambda(Y=c_k)=\frac{\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+K\lambda}$$

我们对上式作一定的简化: 令 $P_\lambda(Y=c_k)=\theta_k$，$\sum_{i=1}^N I(y_i=c_k)=N_k$

$$\theta_k(N+K\lambda)=N_k+\lambda$$

$$\Rightarrow(\theta_k N-N_k)+\lambda(K\theta_k-1)=0$$

如果令第一部分 $\theta_k N-N_k$ 等于 0，我们可以得到 $\theta_k=\frac{N_k}{N}$，这是我们之前接触的极大似然估计；如果令后一部分等于 0，$\theta_k=\frac{1}{K}$，相当于对于 θ_k 的先验概率。

所以贝叶斯估计就相当于是极大似然估计和先验概率的凸组合。

我们可以回顾之前正则化的定义:

$$\min_{f\in F}\frac{1}{N}\sum_{i=1}^N L(y_i,f(x_i))+\lambda J(f)$$

对比我们的等式，后面 $\lambda(K\theta_k-1)$ 的作用相当于正则项，目的是防止出现过拟合的现象。我们不能只靠样本说话，也要有一个对于模型的假设。

λ 取值为 1 时为什么是拉普拉斯平滑呢？拉普拉斯当时是为了防止出现零概率的情况，所以直接在分子上加了一个 1，分母上加上了我们题中的 K。当 N 趋于无穷大的时候，拉普拉斯平滑已经没有什么效果了。

1.11.4　例题解说

问题痛上例，按照拉普拉斯平滑估计概率，取 $\lambda=1$

$P(Y=1)=\dfrac{10}{17},P(Y=-1)=\dfrac{7}{17}$

$P(X^{(1)}=1|Y=1)=\dfrac{3}{12},P(X^{(1)}=2|Y=1)=\dfrac{4}{12},P(X^{(1)}=3|Y=1)=\dfrac{5}{12}$

$P(X^{(2)}=S|Y=1)=\dfrac{2}{12},P(X^{(2)}=M|Y=1)=\dfrac{5}{12},P(X^{(2)}=L|Y=1)=\dfrac{5}{12}$

$P(X^{(1)}=-1|Y=1)=\dfrac{4}{9},P(X^{(1)}=2|Y=-1)=\dfrac{3}{9},P(X^{(1)}=3|Y=-1)=\dfrac{2}{9}$

$P(X^{(2)}=S|Y=-1)=\dfrac{4}{9},P(X^{(2)}=M|Y=-1)=\dfrac{3}{9},P(X^{(2)}=L|Y=-1)=\dfrac{2}{9}$

对于给定的 $x=(2,S)^T$，计算:

$$P(Y=1)P(X^{(1)}=2|Y=1)P(X^{(2)}=S|Y=1)=\frac{10}{17}\cdot\frac{4}{12}\cdot\frac{2}{12}=\frac{5}{153}=0.0327$$

$$P(Y=-1)P(X^{(1)}=2|Y=-1)P(X^{(2)}=S|Y=-1)=\frac{7}{17}\cdot\frac{3}{9}\cdot\frac{4}{9}=\frac{28}{459}=0.0610$$

由于 $P(Y=-1)P(X^{(1)}=2|Y=-1)P(X^{(2)}=S|Y=-1)$ 最大，所以 $y=-1$。

第二章　朴素贝叶斯分类实战项目

2.1　使用 Python 进行文本分类

要从文本中获取特征，需要先拆分文本。这里的特征是来自文本的词条（token），一个词条是字符的任意组合。可以把词条想象成单词。

以在线社区的留言板为例，为了不影响社区的发展，我们要屏蔽侮辱性的言论，所以要构建一个快速过滤器，如果某条留言使用了负面或者侮辱性的语言，那么就将该留言标识为内容不当。

下面是实战代码。

2.1.1　准备数据: 从文本中构建词向量

我们将把文本看成单词向量或者词条向量，也就是说将句子转换为向量。考虑出现在所有文档中的所有单词，再决定将哪些词纳入词汇表或者说所要的词汇集合，然后必须要将每一篇文档转换为词汇表上的向量。

```
def load_dataset():
    dataset = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
               ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', '
                                                stupid'],
               ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', '
                                                him'],
               ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
               ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop
                                                ', 'him'],
               ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']
                                                ]
    class_vector = [0, 1, 0, 1, 0, 1]  # 1 is abusive, 0 not
    return dataset,class_vector
```

展示数据集和数据标签为:

```
数据集为:
 [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'], ['maybe', '
                                                        not', 'take', 'him', 'to', 'dog', '
                                                        park', 'stupid'], ['my', 'dalmation'
                                                        , 'is', 'so', 'cute', 'I', 'love', '
                                                        him'], ['stop', 'posting', 'stupid',
                                                         'worthless', 'garbage'], ['mr', '
                                                        licks', 'ate', 'my', 'steak', 'how',
                                                         'to', 'stop', 'him'], ['quit', '
                                                        buying', 'worthless', 'dog', 'food',
                                                         'stupid']]
============================================================
数据标签为:
 [0, 1, 0, 1, 0, 1]
```

接下来需要创建一个包含在所有文档中出现的不重复词的列表，为此使用了 Python 的 set 数据类型。将词条列表输给 set 构造函数，set 就会返回一个不重复列表。首先，创建一个空集合，然后将每篇文档返回的新词集合添加到该集合中

```python
def create_vocab_list(dataset):
    """创建一个包含所有文档且不出现重复词的列表"""
    vocab_set = set([])  #create empty set
    for document in dataset:
        vocab_set = vocab_set | set(document) #set()去掉列表中的重复词
    return list(vocab_set)
```

我们的结果为:

```
['him', 'my', 'has', 'mr', 'take', 'licks', 'love', 'how', 'worthless', '
                                                      stupid', 'I', 'posting', 'quit', '
                                                      problems', 'not', 'is', 'stop', 'food
                                                      ', 'so', 'flea', 'help', 'maybe', '
                                                      ate', 'dog', 'to', 'dalmation', '
                                                      garbage', 'steak', 'cute', 'park', '
                                                      please', 'buying']
```

获得词汇表后，我们需要使用新的函数，输入参数为词汇表及某个文档，输出的是文档向量，向量的每一元素为 1 或者 0，分别表示词汇表的单词在输入文档中是否出现。函数首先创建一个和词汇表等长的向量，并将其元素都设置为 0。接着，遍历文档中的

所有单词，如果出现了文档中的单词，将其输出的文档向量中的对应值设为 1。

```
#词集模型
"""
输入为词汇表和文档，检查文档中的单词是否在词汇表中
采用词集模型:即对每条文档只记录某个词汇是否存在，而不记录出现的次数
创建一个与词汇表长度一致的0向量，在当前样本中出现的词汇标记为1
将一篇文档转换为词向量
"""
def set_of_words_vector(vocab_list, input_set):
    return_vector = [0]*len(vocab_list)
    for word in input_set:
        if word in vocab_list:
            return_vector[vocab_list.index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return return_vector
```

我们看一下函数的运行效果:

```
print(set_of_words_vector(my_vacab_set,dataset[0]))
```

```
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
                                          0, 0, 1, 0, 0, 0, 0]
```

该函数使用词汇表或者想要检查的所有单词作为输入，然后为其中每一个单词构建一个特征。一旦给定一篇文档，该文档就会被转换为词向量。

2.1.2 训练算法: 从词向量计算概率

现在已经知道了一个词是否出现在一篇文档中，也知道该文档所属的类别。我们重写贝叶斯准则，将之前的 x、y 替换为 ω。ω 表示一个向量，即它由多个数值组成。在这个例子中，数值个数与词汇表中的词个数相同。

$$P(c_i|\omega) = \frac{P(\omega|c_i)P(c_i)}{P(\omega)}$$

我们将使用上述公式，对每个类计算该值，然后比较这两个概率值的大小。

如何计算呢？

首先可以通过类别 i（侮辱性留言或非侮辱性留言）中文档数除以总的文档数来计算概率 $P(c_i)$。

接下来计算条件概率 $P(\omega|c_i)$，这里就要朴素贝叶斯假设。

如果将 ω 展开为一个个独立特征,那么就可以将上述概率写作 $P(\omega_0, \omega_1, \omega_2, \cdots, \omega_n|c_i)$。这里假设所有词都相互独立，该假设也称作条件独立性假设，它意味着可以使用：

$$P(\omega_0|c_i)P(\omega_1|c_i)P(\omega_2|c_i)\cdots P(\omega_N|c_i)$$

来计算上述概率，这就极大简化了计算的过程。

该函数的伪代码如下:

```
计算每个类别中的文档数目
对每篇训练文档
    对每个类别
        如果词条出现在文档中，则增加该词条的计数值
        增加所有词条的计数值
    对每个类别:
        对每个词条:
            将该词条的数目除以总词条数目得到条件概率
    返回每个类别的条件概率
```

我们实现朴素贝叶斯分类器训练函数:

```
"""
朴素贝叶斯分类器训练函数
"""
def train_native_bayes(train_matrix,train_category):
    num_train_docs=len(train_matrix)
    num_words=len(train_matrix[0])
    p=sum(train_category)/float(num_train_docs)
    p_0_num=zeros(num_words)
    p_1_num=zeros(num_words)
    p_0_denom=0.0
    p_1_denom=0.0
    for i in range(num_train_docs):
        if train_category[i]==1:
            p_1_num+=train_matrix[i]
            p_1_denom+=sum(train_matrix[i])
        else:
            p_0_num+=train_matrix[i]
            p_0_denom+=sum(train_matrix[i])
    p_1_vector=(p_1_num/p_1_denom)
    p_0_vector=(p_0_num/p_0_denom)
```

```
    return p_0_vector,p_1_vector,p
```

代码函数中的输入参数为文档矩阵 train matrix，以及由每篇文档类别标签所构成的向量 train category。首先，计算文档属于侮辱性文档（class=1）的概率，即 P（1）。因为这是一个二分类问题，所以可以通过 1-P（1）来得到 P（0）。

接下来计算 $P(\omega_i|c_1)$，$P\ \omega_i|c_2)$，需要初始化程序中的分子变量和分母变量。在 for 循环中，要遍历训练集 train matrix 中的所有文档。一旦某个词语（侮辱性或正常词语）在某一文档中出现，则该词对应的个数 $(P1num)$ 或者另一类加加 1，而且在所有文档中，该文档的总词数也相应加 1。对于两个类别都要进行同样的计算处理。

最后，对每个元素除以该类别中的总次数即可。

接下来试验一下:

```
train_mat=[]
for i in dataset:
    train_mat.append(set_of_words_vector(my_vacab_set,i))
p_0_vector,p_1_vector,p=train_native_bayes(train_mat,class_vector)
```

接下来看这些变量的内部值:

```
0.5
```

我们输出一下 P(0) 的概率如下:

```
[0.04166667 0.04166667 0.         0.04166667 0.04166667 0.
 0.         0.04166667 0.04166667 0.         0.         0.04166667
 0.04166667 0.04166667 0.         0.04166667 0.         0.04166667
 0.04166667 0.         0.04166667 0.04166667 0.         0.04166667
 0.         0.04166667 0.04166667 0.         0.125      0.04166667
 0.04166667 0.08333333]
```

首先，我们发现文档属于侮辱类的概率 p 为 0.5，该值是正确的。接下来，看一看在给定文档类别条件下词汇表中单词的出现概率，看看是否正确。词汇表中的第一个词是 cute，其在类别 0 中出现 1 次，而在类别 1 中从未出现。对应的条件概率分别为 0.041 666 67 与 0.0。该计算是正确的。

2.1.3 测试算法: 根据现实情况修改分类器

利用贝叶斯分类器对文档进行分类时，要计算多个概率的乘积以获得文档属于某个类别的概率，即计算 $P(\omega_0|1)P(\omega_1|1)P(\omega_2|1)$。如果其中一个概率值为 0，那么最后的乘积也为 0。为降低这种影响，可以将所有词的出现数初始化为 1，并将分母初始化为 2。

另一个遇到的问题是下溢出，这是由于太多很小的数相乘造成的。当计算乘积

$$P(\omega_0|c_i)P(\omega_1|c_i)P(\omega_2|c_i)\cdots P(\omega_N|c_i)$$

时，由于大部分因子都非常小，所以程序会下溢出或者得到不正确的答案。一种解决办法是对乘积取自然对数。通过求对数可以避免下溢出或者浮点数舍入导致的错误。同时，采用自然对数进行处理不会有任何损失。

我们给出 $f(x)$ 和 $\ln(f(x))$ 的曲线如下:

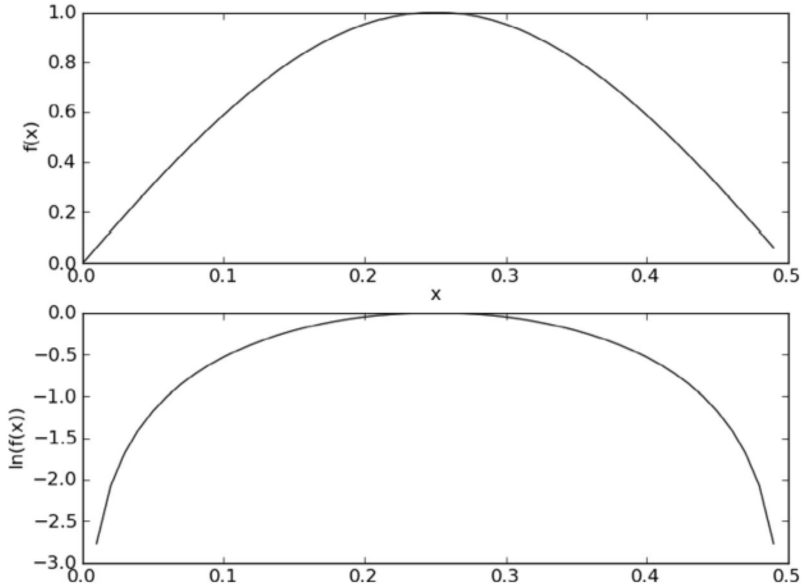


图4-4　函数f(x)与ln(f(x))会一块增大。这表明想求函数的最大值时，可以使用该函数的自然对数来替换原函数进行求解

检查这两条曲线，就会发现它们在相同区域内同时增加或者减少，并且在相同点上取到极值。它们的取值虽然不同，但不影响最终结果。

我们修改后的分类器为:

```
def train_native_bayes(train_matrix,train_category):
    num_train_docs=len(train_matrix)
    num_words=len(train_matrix[0])
    p=sum(train_category)/float(num_train_docs)
    p_0_num=ones(num_words)
    p_1_num=ones(num_words)
    p_0_denom=2.0
    p_1_denom=2.0
    for i in range(num_train_docs):
        if train_category[i]==1:
            p_1_num+=train_matrix[i]
```

```
            p_1_denom+=sum(train_matrix[i])
        else:
            p_0_num+=train_matrix[i]
            p_0_denom+=sum(train_matrix[i])
    p_1_vector=log(p_1_num/p_1_denom)
    p_0_vector=log(p_0_num/p_0_denom)
    return p_0_vector,p_1_vector,p
```

2.1.4 朴素贝叶斯分类函数

```
def classify_native_bayes(need_to_classify_vector, p_0_vector, p_1_vector,
                                        p_class):
    p_1 = sum(need_to_classify_vector * p_1_vector) + log(p_class)    #
                                            element-wise mult
    p_0 = sum(need_to_classify_vector * p_0_vector) + log(1.0 - p_class)
    if p_1 > p_0:
        return 1
    else:
        return 0
```

要分类的向量 need to classify vector 以及使用函数 train native bayes() 计算得到的三个概率。使用 NumPy 的数组来计算两个向量相乘的结果。这里的相乘是指对应元素相乘，即先将两个向量中的第 1 个元素相乘，然后将第 2 个元素相乘，以此类推。接下来将词汇表中所有词的对应值相加，然后将该值加到类别的对数概率上。最后，比较类别的概率返回大概率对应的类别标签。

```
def testing_native_bayes():
    dataset,class_vector=load_dataset()
    my_vacab_set = create_vocab_list(dataset)
    my_vacab_set.sort()
    train_mat=[]
    for i in dataset:
        train_mat.append(set_of_words_vector(my_vacab_set, i))
    p_0_vector,p_1_vector,p = train_native_bayes(array(train_mat),array(
                                            class_vector))
    test_entry = ['love','my']
    this_doc = array(set_of_words_vector(my_vacab_set, test_entry))
    print(test_entry,'classified as: ',classify_native_bayes(this_doc,
                                            p_0_vector,p_1_vector,p))
```

```
    test_entry_1 = ['stupid','garbage']
    this_doc_1 = array(set_of_words_vector(my_vacab_set, test_entry_1))
    print(test_entry_1,'classified as: ',classify_native_bayes(this_doc_1,
                                                p_0_vector,p_1_vector,p))
```

下面来看看实际结果:

```
['love', 'my'] classified as:  0
['stupid', 'garbage'] classified as:  1
```

2.1.5 准备数据：文档词袋模型

目前为止，我们将每个词的出现与否作为一个特征，这可以被描述为词集模型（set-of-words model）。如果一个词在文档中出现不止一次，这可能意味着包含该词是否出现在文档中所不能表达的某种信息，这种方法被称为词袋模型（bag-of-words model）。

在词袋中，每个单词可以出现多次，而在词集中，每个词只能出现一次。为适应词袋模型，需要对函数 set of words vector() 稍加修改，修改后的函数称为 bag of words vector()。

下面的程序给出了基于词袋模型的朴素贝叶斯代码。它与函数 set of words vector() 几乎完全相同，唯一不同的是每当遇到一个单词时，它会增加词向量中的对应值，而不只是将对应的数值设为 1。

```
def bag_word_vector(vocab_list,input_set):
    return_vector=[0]*len(vocab_list)
    for word in input_set:
        if word in vocab_list:
            return_vector[vocab_list.index(word)]+=1
    return return_vector
```

2.2 使用朴素贝叶斯过滤垃圾邮件

在前面那个简单的例子中，我们引入了字符串列表。使用朴素贝叶斯解决一些现实生活中的问题时，需要先从文本内容得到字符串列表，然后生成词向量。

下面这个例子中，我们将了解朴素贝叶斯的一个最著名的应用：电子邮件垃圾过滤。

2.2.1　准备数据：切分文本

前一节中的词向量是预先给定的，下面介绍如何从文本文档中构建自己的词列表。

对于一个文本字符串，可以使用 Python 的 string.split() 方法将其切分。但是标点符号也被当成了词的一部分。

我们可以使用正则表示式来切分句子，其中分隔符是除单词、数字外的任意字符串。但是里面的空字符串需要去掉。可以计算每个字符串的长度，只返回长度大于 0 的字符串。

我们的 Python 代码实现如下:

```
"""
函数说明:接收一个大字符串并将其解析为字符串列表
"""
def text_parse(big_string):
    # 将字符串转换为字符列表
    list_of_tokens = re.split(r"[0-9!@#$%^&*()?\n~]",big_string) # 将特殊符号作为切分标志进行字符串切分，即非字母、非数字
    return [tok.lower() for tok in list_of_tokens if len(tok) > 2] # 除了单个字母，例如大写的I，其它单词变成小写
```

2.2.2　测试算法：使用朴素贝叶斯进行交叉验证

现在来看数据集中一封完整的电子邮件的实际处理结果。该数据集放在 email 文件夹中，该文件夹又包含两个子文件夹，分别是 spam 与 ham。

我们的分类函数如下:

```
"""
函数说明:测试朴素贝叶斯分类器，使用朴素贝叶斯进行交叉验证
"""
def spam_test():
    doc_list=[]
    class_vector=[]
    full_text=[]
    for i in range(1,26): # 遍历25个txt文件
        word_list=text_parse(open('native_bayes  email dataset/spam/%d.txt'
                                  %i,'r').read()) # 读取每个垃
```

```
圾邮件，并字符串转换成字符串列表
    doc_list.append(word_list)
    full_text.extend(word_list)
    class_vector.append(1) # 标记垃圾邮件，1表示垃圾文件
    word_list=text_parse(open('native_bayes  email dataset/ham/%d.txt'%i,'r').read()) # 读取每个非垃圾邮件，并字符串转换成字符串列表
    doc_list.append(word_list)
    full_text.extend(word_list)
    class_vector.append(0) # 标记正常邮件，0表示正常文件
vocab_list=create_vocab_list(doc_list) # 创建词汇表，不重复
training_set=list(range(50))
test_set=[] # 创建存储训练集的索引值的列表和测试集的索引值的列表
for i in range(0,10): # 从50个邮件中，随机挑选出40个作为训练集,10个做测试集
    rand_index=int(random.uniform(0,len(training_set))) # 随机选取索索引值
    test_set.append(training_set[rand_index]) # 添加测试集的索引值
    del (training_set[rand_index]) # 在训练集列表中删除添加到测试集的索引值
train_mat=[]
train_class=[] # 创建训练集矩阵和训练集类别标签系向量
for doc_index in training_set: # 遍历训练集
    train_mat.append(set_of_words_vector(vocab_list,doc_list[doc_index])) # 将生成的词集模型添加到训练矩阵中
    train_class.append(class_vector[doc_index]) # 将类别添加到训练集类别标签系向量中
p_0_vector,p_1_vector,p=train_native_bayes(array(train_mat),array(train_class)) # 训练朴素贝叶斯模型
error_count=0 # 错误分类计数
for doc_index in test_set: # 遍历测试集
    word_vector=set_of_words_vector(vocab_list,doc_list[doc_index]) # 测试集的词集模型
    if classify_native_bayes(array(word_vector),p_0_vector,p_1_vector,p)!=class_vector[doc_index]: #
```

```
                                        如果分类错误
            error_count+=1 # 错误计数加1
            print('classify error:',doc_list[doc_index])
    print('the error rate is:',float(error_count)/len(test_set))
```

导入文件夹 spam 与 ham 下的文本文件，并将它们解析为词列表。接下来构建一个测试集与一个训练集，两个集合中的邮件都是随机选出的。

本例中共有 50 封电子邮件，并不是很多，其中的 10 封电子邮件被随机选择为测试集。分类器所需要的概率计算只利用训练集中的文档来完成。Python 变量 training set 是一个整数列表，其中的值从 0 到 49。接下来，随机选择其中 10 个文件。选择出的数字所对应的文档被添加到测试集，同时也将其从训练集中剔除。这种随机选择数据的一部分作为训练集，而剩余部分作为测试集的过程称为留存交叉验证（hold-out cross validation）。

假定现在只完成了一次迭代，那么为了更精确地估计分类器的错误率，就应该进行多次迭代后求出平均错误率。

我们得到的运行结果如下:

```
classify error: ['experience with biggerpenis today', ' grow ', '-inches
                                      more', 'the safest ', ' most
                                      effective methods of_penisen', '
                                      argement.', 'save your time and money
                                      ', 'bettererections with effective ma
                                      ', 'eenhancement products.', ' ma', '
                                      eenhancement supplement. trusted by
                                      millions. buy today']
classify error: ['oem adobe ', ' microsoft softwares', 'fast order and
                                      download', 'microsoft office
                                      professional plus ', 'microsoft
                                      windows ', ' ultimate ', 'adobe
                                      photoshop cs', ' extended', 'adobe
                                      acrobat ', ' pro extended', 'windows
                                      xp professional ', ' thousand more
                                      titles']
the error rate is: 0.2
```

2.2.3 K 折交叉验证

我们的代码实现如下:

```
def one_cross_validate(train_set,train_class,test_set,test_class):
    #训练模型
    p_0_vector,p_1_vector,p_c_1 = train_native_bayes(array(train_set),array
                                                      (train_class))
    error_count = 0
    #验证集进行测试
    for i in range(10):
        c = classify_native_bayes(array(test_set[i]),p_0_vector,p_1_vector,
                                                      p_c_1)
        if c != test_class[i]:
            error_count += 1
    return error_count/10

def K_Cross_Validate(train_mat,train_class_vector):  #K折交叉验证 5
    rand_index = list(range(50))
    random.shuffle(rand_index)
    error_radio = 0.0
    for i in range(5):  #5次
        index = rand_index #随机索引
        #选取训练集、验证集索引
        train_set = []
        train_cls = []
        test_set = []
        test_cls = []
        test_set_index = set(rand_index[10*i:10*i+10])  # 测试集10
        train_set_index = set(index)-test_set_index  # 验证集
        #选取训练集、验证集数据
        for idx in train_set_index:
            train_set.append(train_mat[idx])
            train_cls.append(train_class_vector[idx])
        for idx in test_set_index:
            test_set.append(train_mat[idx])
            test_cls.append(train_class_vector[idx])
        print('第%d个子集的误差率为:'%(i+1),one_cross_validate(train_set,
                                                      train_cls,test_set,test_cls))
        error_radio += one_cross_validate(train_set,train_cls,test_set,
```

```
                                          test_cls)
    return error_radio/5
```

我们最后是求五次的平均误差率。

我们导入需要的数据集并输出最终的结果为:

```
def create_dataset():
    data_set_list=[]  #全部数据集
    class_vector = []     #标签值
    #获取数据
    spam_path = "native_bayes  email dataset/spam/{}.txt"  #获取文件路径
    ham_path = "native_bayes  email dataset/ham/{}.txt"
    for i in range(1, 26):  # 两个路径各有25个文件
        document_data_1 = open(spam_path.format(i), 'r').read()
        # 使用正则进行分割，除了空格、还有标点都可以用于分割
        word_vector = text_parse(document_data_1) # \W*表示匹配多个非字母、
                                                 数字、下划线的字符
        data_set_list.append([item for item in word_vector if len(item) > 0
                                                 ])
        class_vector.append(1)
        document_data_2 = open(ham_path.format(i), 'r').read()
        # 使用正则进行分割，除了空格、还有标点都可以用于分割
        word_vector_2 = text_parse(document_data_2)  # \W*表示匹配多个非字
                                                 母、数字、下划线的字符
        data_set_list.append([item for item in word_vector_2 if len(item) >
                                                 0])
        class_vector.append(0)
    return data_set_list, class_vector
data_set_list, class_vector=create_dataset()
vocab_list = create_vocab_list(data_set_list)
trainMulList = []
for doc in data_set_list:
    trainMulList.append(set_of_words_vector(vocab_list,doc))
print('=='*30)
print('5折交叉验证的错误率为:\n',K_Cross_Validate(trainMulList,class_vector
                                      ))
```

运行结果如下:

```
第1个子集的误差率为: 0.3
第2个子集的误差率为: 0.4
第3个子集的误差率为: 0.1
第4个子集的误差率为: 0.2
第5个子集的误差率为: 0.6
5折交叉验证的错误率为:
 0.32
```

需要注意的是，因为我们对数据集的划分是随机的，所以每次运行的结果会不相同。

第三章　源代码

```
from numpy import *
import re

def load_dataset():
    dataset = [['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
               ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
               ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
               ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
               ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
               ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    class_vector = [0, 1, 0, 1, 0, 1]  # 1 is abusive, 0 not
    return dataset,class_vector
dataset,class_vector=load_dataset()
print('数据集为:\n',dataset)
print('=='*30)
print('数据标签为:\n',class_vector)
print('=='*30)

def create_vocab_list(dataset):
    """创建一个包含所有文档且不出现重复词的列表"""
    vocab_set = set([])  #create empty set
    for document in dataset:
        vocab_set = vocab_set | set(document) #set()去掉列表中的重复词
    return list(vocab_set)

my_vacab_set=create_vocab_list(dataset)
```

```python
print(my_vacab_set)

#词集模型
"""
输入为词汇表和文档，检查文档中的单词是否在词汇表中
采用词集模型:即对每条文档只记录某个词汇是否存在，而不记录出现的次数
创建一个与词汇表长度一致的0向量，在当前样本中出现的词汇标记为1
将一篇文档转换为词向量
"""
def set_of_words_vector(vocab_list, input_set):
    return_vector = [0]*len(vocab_list)
    for word in input_set:
        if word in vocab_list:
            return_vector[vocab_list.index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return return_vector
print(set_of_words_vector(my_vacab_set,dataset[0]))
"""
朴素贝叶斯词袋模型
如果一个词在文档中出现不止一次，这可能意味着包含该词是否出现中文档中所不能
                                        表达的某种信息
"""
def bag_word_vector(vocab_list,input_set):
    return_vector=[0]*len(vocab_list)
    for word in input_set:
        if word in vocab_list:
            return_vector[vocab_list.index(word)]+=1
    return return_vector
print(bag_word_vector(my_vacab_set,dataset[0]))
"""
朴素贝叶斯分类器训练函数
"""
def train_native_bayes(train_matrix,train_category):
    num_train_docs=len(train_matrix)
    num_words=len(train_matrix[0])
    p=sum(train_category)/float(num_train_docs)
    p_0_num=ones(num_words)
    p_1_num=ones(num_words)
```

```
    p_0_denom=2.0
    p_1_denom=2.0
    for i in range(num_train_docs):
        if train_category[i]==1:
            p_1_num+=train_matrix[i]
            p_1_denom+=sum(train_matrix[i])
        else:
            p_0_num+=train_matrix[i]
            p_0_denom+=sum(train_matrix[i])
    p_1_vector=log(p_1_num/p_1_denom)
    p_0_vector=log(p_0_num/p_0_denom)
    return p_0_vector,p_1_vector,p

train_mat=[]
for i in dataset:
    train_mat.append(set_of_words_vector(my_vacab_set,i))
p_0_vector,p_1_vector,p=train_native_bayes(train_mat,class_vector)
print(p)
print(p_0_vector)

def classify_native_bayes(need_to_classify_vector, p_0_vector, p_1_vector,
                                               p_class):
    p_1 = sum(need_to_classify_vector * p_1_vector) + log(p_class)     #
                                                   element-wise mult
    p_0 = sum(need_to_classify_vector * p_0_vector) + log(1.0 - p_class)
    if p_1 > p_0:
        return 1
    else:
        return 0

def testing_native_bayes():
    dataset,class_vector=load_dataset()
    my_vacab_set = create_vocab_list(dataset)
    my_vacab_set.sort()
    train_mat=[]
    for i in dataset:
        train_mat.append(set_of_words_vector(my_vacab_set, i))
    p_0_vector,p_1_vector,p = train_native_bayes(array(train_mat),array(
                                                   class_vector))
```

```
    test_entry = ['love','my']
    this_doc = array(set_of_words_vector(my_vacab_set, test_entry))
    print(test_entry,'classified as: ',classify_native_bayes(this_doc,
                                          p_0_vector,p_1_vector,p))
    test_entry_1 = ['stupid','garbage']
    this_doc_1 = array(set_of_words_vector(my_vacab_set, test_entry_1))
    print(test_entry_1,'classified as: ',classify_native_bayes(this_doc_1,
                                          p_0_vector,p_1_vector,p))
print(testing_native_bayes())

"""
函数说明:接收一个大字符串并将其解析为字符串列表
"""
def text_parse(big_string):
    # 将字符串转换为字符列表
    list_of_tokens = re.split(r"[0-9!@#$%^&*()?\n~]",big_string) # 将特殊符
                                          号作为切分标志进行字符串切分，即
                                          非字母、非数字
    return [tok.lower() for tok in list_of_tokens if len(tok) > 2] # 除了单
                                          个字母，例如大写的I，其它单词变成
                                          小写

"""
函数说明:测试朴素贝叶斯分类器，使用朴素贝叶斯进行交叉验证
"""
def spam_test():
    doc_list=[]
    class_vector=[]
    full_text=[]
    for i in range(1,26): # 遍历25个txt文件
        word_list=text_parse(open('native_bayes  email dataset/spam/%d.txt'
                                          %i,'r').read()) # 读取每个垃
                                          圾邮件，并字符串转换成字符串
                                          列表
        doc_list.append(word_list)
        full_text.extend(word_list)
        class_vector.append(1) # 标记垃圾邮件，1表示垃圾文件
        word_list=text_parse(open('native_bayes  email dataset/ham/%d.txt'%
                                          i,'r').read()) # 读取每个非垃
```

```
圾邮件，并字符串转换成字符串列表
        doc_list.append(word_list)
        full_text.extend(word_list)
        class_vector.append(0) # 标记正常邮件，0表示正常文件
    vocab_list=create_vocab_list(doc_list) # 创建词汇表，不重复
    training_set=list(range(50))
    test_set=[] # 创建存储训练集的索引值的列表和测试集的索引值的列表
    for i in range(0,10): # 从50个邮件中，随机挑选出40个作为训练集,10个做测试集
        rand_index=int(random.uniform(0,len(training_set))) # 随机选取索索引值
        test_set.append(training_set[rand_index]) # 添加测试集的索引值
        del (training_set[rand_index]) # 在训练集列表中删除添加到测试集的索引值
    train_mat=[]
    train_class=[] # 创建训练集矩阵和训练集类别标签系向量
    for doc_index in training_set: # 遍历训练集
        train_mat.append(set_of_words_vector(vocab_list,doc_list[doc_index])) # 将生成的词集模型添加到训练矩阵中
        train_class.append(class_vector[doc_index]) # 将类别添加到训练集类别标签系向量中
    p_0_vector,p_1_vector,p=train_native_bayes(array(train_mat),array(train_class)) # 训练朴素贝叶斯模型
    error_count=0 # 错误分类计数
    for doc_index in test_set: # 遍历测试集
        word_vector=set_of_words_vector(vocab_list,doc_list[doc_index]) # 测试集的词集模型
        if classify_native_bayes(array(word_vector),p_0_vector,p_1_vector,p)!=class_vector[doc_index]: # 如果分类错误
            error_count+=1 # 错误计数加1
            print('classify error:',doc_list[doc_index])
    print('the error rate is:',float(error_count)/len(test_set))

spam_test()
```

```
def one_cross_validate(train_set,train_class,test_set,test_class):
    #训练模型
    p_0_vector,p_1_vector,p_c_1 = train_native_bayes(array(train_set),array
                                        (train_class))
    error_count = 0
    #验证集进行测试
    for i in range(10):
        c = classify_native_bayes(array(test_set[i]),p_0_vector,p_1_vector,
                                            p_c_1)
        if c != test_class[i]:
            error_count += 1
    return error_count/10

def K_Cross_Validate(train_mat,train_class_vector):  #K折交叉验证 5
    rand_index = list(range(50))
    random.shuffle(rand_index)
    error_radio = 0.0
    for i in range(5):  #5次
        index = rand_index #随机索引
        #选取训练集、验证集索引
        train_set = []
        train_cls = []
        test_set = []
        test_cls = []
        test_set_index = set(rand_index[10*i:10*i+10])  # 测试集10
        train_set_index = set(index)-test_set_index  # 验证集
        #选取训练集、验证集数据
        for idx in train_set_index:
            train_set.append(train_mat[idx])
            train_cls.append(train_class_vector[idx])
        for idx in test_set_index:
            test_set.append(train_mat[idx])
            test_cls.append(train_class_vector[idx])
        print('第%d个子集的误差率为:'%(i+1),one_cross_validate(train_set,
                                            train_cls,test_set,test_cls))
        error_radio += one_cross_validate(train_set,train_cls,test_set,
                                            test_cls)
    return error_radio/5
```

```
def create_dataset():
    data_set_list=[]  #全部数据集
    class_vector = []     #标签值
    #获取数据
    spam_path = "native_bayes  email dataset/spam/{}.txt"  #获取文件路径
    ham_path = "native_bayes  email dataset/ham/{}.txt"
    for i in range(1, 26):  # 两个路径各有25个文件
        document_data_1 = open(spam_path.format(i), 'r').read()
        # 使用正则进行分割，除了空格、还有标点都可以用于分割
        word_vector = text_parse(document_data_1) # \W*表示匹配多个非字母、数字、下划线的字符
        data_set_list.append([item for item in word_vector if len(item) > 0])
        class_vector.append(1)
        document_data_2 = open(ham_path.format(i), 'r').read()
        # 使用正则进行分割，除了空格、还有标点都可以用于分割
        word_vector_2 = text_parse(document_data_2)  # \W*表示匹配多个非字母、数字、下划线的字符
        data_set_list.append([item for item in word_vector_2 if len(item) > 0])
        class_vector.append(0)
    return data_set_list, class_vector
data_set_list, class_vector=create_dataset()
vocab_list = create_vocab_list(data_set_list)
trainMulList = []
for doc in data_set_list:
    trainMulList.append(set_of_words_vector(vocab_list,doc))
print('=='*30)
print('5折交叉验证的错误率为:\n',K_Cross_Validate(trainMulList,class_vector))
```