李航《统计学习方法》笔记

——从原理到实现：基于 R

作者：渣君

前言

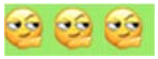
渣君最近一个半月开始真正接触机器学习了，作为一个 ML 小白，我在决定入手入门参考书籍时便在度娘的帮助下毫不费劲地查到了李航大牛的《统计学习方法》，据说是国内很好的入门书籍，简单易懂，对 ML 小白比较友好。我从 7 月初开始认真看这本书，才发现，这本书基本是干货，逻辑思路流畅，但并非十分简单易懂。全书下来基本是公式，感觉作者惜字如金。书中有些地方公式的推导并非十分详细，可能是书本篇幅限制；我在第一遍看的时候，的确是无法领会其中一些公式所蕴含的深意，然后在网上也找不到相关的学习笔记可以参考（主要是关于那些公式的证明），第一遍花了十多天看完，感觉自己是懵的，脑袋里没留下多少东西。然后第二、第三遍我开始尝试去推一下里面省略掉的较为简单的数学证明，比如 SVM 的 SMO 算法中最优值上下界的证明、CART 算法中剪枝算法的理解、最大熵模型中改进的迭代尺度法中的牛顿法迭代公式的推导、Adaboost 算法中用负梯度作为提升回归树算法中残差的近似值的原因、隐马尔科夫模型中前向算法/后向算法/Baunm-Welch 算法中一些公式的证明等等。一如既往，渣君的书本写满了笔记，现在想把这些笔记整理出来分享，也是作为自己第四/五遍看这本书的一个方式。

笔记中的证明仅代表个人观点，望读者谨慎参阅。（2017/8/10）

阅读说明

本笔记对应的是《统计学习方法》2017 年 3 月的重印版，对于笔记中证明所涉及的定理内容或其他书上的证明过程均不再罗列，只给出相关的定理名称及所在原文页数，证明/个人理解所用的符号公式的含义与原书一致，向量的乘法均表示内积。最后，大部分笔记将以“请证明***—证明***”或“关键词***—说明***”的模式展开。

另外，对于一些有显式解或较简单的算法/案例我通过编写 R 函数进行了实现，这对于本渣而言有点难度，但也很有乐趣。所谓真理来源于实践。

相信我，看完这份笔记后，你不会后悔的。 

阅读建议

建议大家看完“请证明***”或“关键词***”先别看“证明***”和“说明***”，可以自己先思考一下，或许会得出更好的答案。

目录

1.6.2 泛化误差上界（P16-P17）

请证明 1： $$P\left(R(f)-\hat{R}(f)\geq\varepsilon\right)\leq\exp\left(-2N\varepsilon^2\right)$$

证明 1：

$$R(f)=E[\mathrm{L}(Y,f(X))]$$

$$\hat{R}(f)=\frac{1}{N}\sum_{i=1}^{N}L(y_i,f(x_i))=\sum_{i=1}^{N}\frac{1}{N}L(y_i,f(x_i))$$

把$\sum_{i=1}^{N}L(y_i,f(x_i))$理解成N个独立同分布的随机变量$L(y_i,f(x_i))$的和，则有：

$$E\left[\hat{R}(f)\right]=E\left[\mathrm{L}(Y,f(X))\right]=R(f)$$

$$P\left(R(f)-\hat{R}(f)\geq\varepsilon\right)=P\left(E\hat{R}(f)-\hat{R}(f)\geq\varepsilon\right)$$

又因为$\hat{R}(f)$实际上为N个$\frac{1}{N}L(y_i,f(x_i))$的和，且有：

$$\frac{1}{N}L(y_i,f(x_i))\in\left[0,\frac{1}{N}\right]$$

于是根据，Hoeffding不等式可得：

$$P\left(R(f)-\hat{R}(f)\geq\varepsilon\right)=\exp\left\{\frac{-2\varepsilon^2}{\sum_{i=1}^{N}\left(\frac{1}{N}-0\right)^2}\right\}=\exp\left(-2N\varepsilon^2\right)$$

■

请证明 2： $P\left(R(f)-\hat{R}(f)<\varepsilon\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$

证明 2：

思考：$P\left(R(f)-\hat{R}(f)<\varepsilon\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$并非看起来那么理所当然，$P\left(R(f)-\hat{R}(f)<\varepsilon\right)$针对某个单独的 f，而 $P\left(\exists f\in F:R(f)-\hat{R}(f)\geq\varepsilon\right)$是针对集合 F 而言的，因此

$$1-P\left(R(f)-\hat{R}(f)\geq\varepsilon\right)=P\left(R(f)-\hat{R}(f)<\varepsilon\right)$$
$$\neq 1-P\left(\exists f\in F:R(f)-\hat{R}(f)\geq\varepsilon\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$$

但确实又有 $P\left(R(f)-\hat{R}(f)<\varepsilon\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$，渣君认为这里面一定是有故事的，于是尝试着推导一下。

首先将原书证明过程的等式改写一下，得：

$$P\left(\exists f\in F:R(f)-\hat{R}(f)\geq\varepsilon\right)=P\left(\bigcup_{f\in F}\left\{R(f)-\hat{R}(f)\geq\varepsilon\right\}\right)=P\left(\bigcup_{i=1}^{d}A_{fi}\right)$$
$$\text{注：}A_{fi}=\left\{R(f_i)-\hat{R}(f_i)\geq\varepsilon\right\},\ i=1,2,...,d.$$

则根据德摩根定律知，对于任意 $k\in\{1,2,...,d\}$ 有：

$$P\left(R(f_k)-\hat{R}(f_k)<\varepsilon\right)=P\left(\overline{A_{fk}}\right)\geq P\left(\bigcap_{i=1}^{d}\overline{A_{fi}}\right)=P\left(\overline{\bigcup_{i=1}^{d}A_{fi}}\right)$$
$$=1-P\left(\bigcup_{i=1}^{d}A_{fi}\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$$

因此 $P\left(R(f)-\hat{R}(f)<\varepsilon\right)\geq 1-d\exp\left(-2N\varepsilon^2\right)$得证。

■

1.4.2 过拟合与模型选择（P11）

关键词 1： 过拟合/训练误差/测试误差/例 1.1 的 R 实现

说明 1：

过拟合是指学习时选择的模型所包含的参数过多，以致于出现这一模型对已知数据预测得很好，但对未知数据预测得很差的现象。在这里，本渣尝试着通过 R 语言实现例 1.1，并计算不同模型对应的留一交叉验证法的均方误差 MSPE 和训练误差 TRAIN。这个代码我参考了《R 语言编程艺术》,只是修改了一小段。函数分别为 polyfit()、print.polyreg()、plot.polyreg()、powers()、lvoneout().主要是利用 polyfit()、print.polyreg()、plot.polyreg()另外两个是辅助函数。（详细代码参见附录 1）

函数 polyfit(y,x, maxdeg)中 y 是因变量，x 是自变量，maxdeg 是指 x 的最大阶数；polyfit 将 x 的 1 到 maxdeg 不同阶数的模型一次性拟合。print.polyreg()输出 MSPE/TRAIN 的计算结果并返回一个矩阵（直接调用 print 即可）；plot.polyreg()执行后可以按照 console 框的提示输入想绘制的模型的拟合图。

我们来模拟一个样本量为 60 的数据,并将模拟的最大阶数设为 12（尽量大），以便观察 MSPE 和 TRAIN 随阶数的增大的变动情况。

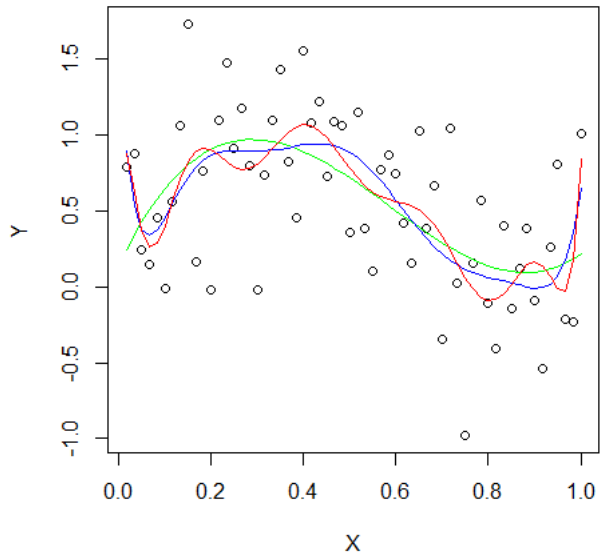
代码操作如下。

```
#### 多项式模拟 ####
n<-60
x<-(1:n)/n
y<-vector(length = n)
for(i in 1:n) y[i]<-sin((3*pi/2)*x[i])+x[i]^2+rnorm(1,0,.5)
dg<-12#指定最大阶数为 12
lmo<-polyfit(y,x,dg);lmo
error<-print(lmo)
plot(lmo)#由于编写的 plot.polyreg()是泛函 plot 的子函数，直接调用 plot 即可
```

首先我们看一下 plot(lmo)的情况，我按照 console 的提示分别键入 3（绿）,9（蓝）,12（红）,便得到阶数各自对应的拟合曲线。操作如下。

```
> plot(lmo)
RETURN for CV fit for degree 1 or type degree or q for quit:3
RETURN for CV fit for degree 4 or type degree or q for quit:9
RETURN for CV fit for degree 10 or type degree or q for quit:12
```

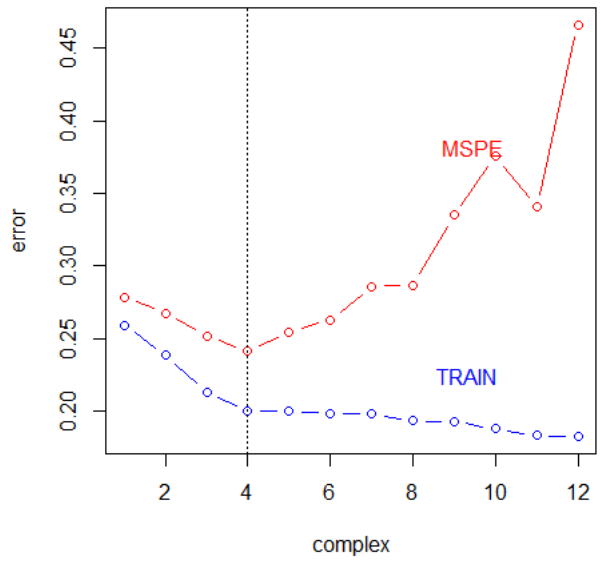
图形结果如下：



然后我们看一下 print(lmo)的结果，并将其绘制成图，代码如下。

```
error<-print(lmo)
plot(error["TRAIN"],ylab="error",xlab="complex",
     ylim=c(min(error),max(error)),col="blue",type="b")
points(error["MSPE"],type="b",col="red")
abline(v=4,lty=3)
text(locator(),"MSPE",col="red")
text(locator(),"TRAIN",col="blue")
```

稍加修饰便可以得到以下图形：



mean squared prediction errors,by degree

	MSPE	TRAIN
[1,]	0.2786117	0.2593698
[2,]	0.2673140	0.2384004
[3,]	0.2516644	0.2132075
[4,]	0.2416086	0.1998308
[5,]	0.2543939	0.1996281
[6,]	0.2628561	0.1983103
[7,]	0.2858330	0.1978605
[8,]	0.2868034	0.1931861
[9,]	0.3348163	0.1930854
[10,]	0.3755319	0.1877315
[11,]	0.3403323	0.1832570
[12,]	0.4662283	0.1823727

显然，由上述结果可以看出，模型的预测误差 MSPE 随着模型的复杂度是先下降后上升的，而模型的训练误差则是随着模型复杂度的增加而不断下降的，过大的模型复杂度造成了“过拟合”现象。

■

第二章　感知机

2.3.1 感知机学习算法的原始形式（P28-P29）

关键词 1：　　（随机）梯度下降法

说明 1：（参考《高等数学》同济第六版，下册，P104）

设函数 $f(x,y)$ 在 (x_0,y_0) 点处可微分，l 代表某个方向，$\ell_l=(\cos\alpha,\cos\beta)$ 是与 l 方向相同的单位向量 ，则有方向导数：

$$\begin{aligned}\frac{\partial f}{\partial l}\Big|_{(x_0,y_0)} &= f_x(x_0,y_0)\cos\alpha + f_y(x_0,y_0)\cos\beta \\ &= gradf\,(x_0,y_0)\cdot\ell_l = \left|gradf\,(x_0,y_0)\right|\cdot\left|\ell_l\right|\cdot\cos\theta\end{aligned}$$

其中 $\theta=\widehat{(gradf\,(x_0,y_0),\ell_l)}$ 是两个向量的夹角，$\cos\theta\le 1$

当 $\theta=0$ 时，$gradf$ 与 $\ell_l=(\cos\alpha,\cos\beta)$ 方向相同，$\frac{\partial f}{\partial l}\Big|_{(x_0,y_0)}$ 的值最大，此时，$f(x,y)$ 沿梯度 $gradf$ 的方向增加最快；反之，当 $\theta=\pi$ 时，$f(x,y)$ 沿负梯度 $-gradf$ 的方向下降最快。

■

请证明 1：　　随机梯度下降法能逐步降低损失函数的值

证明 1：

不失一般性，设 (x_1,y_1) 为误分类点，(w_0,b_0) 为初始参数，则有：$y_1(w_0\cdot x_1+b_0)\le 0$。其中 $x_1\in R^n, w_0\in R^n$ 均为列向量。令：

$$w_1 \leftarrow w_0+\eta y_1 x_1$$

$$b_1 \leftarrow b_0+\eta y_1$$

则有：

$$y_1(w_1\cdot x_1+b_1)=y_1(w_0\cdot x_1+b_0)+\eta y_1^2(x_1\cdot x_1+1)$$
$$=y_1(w_0\cdot x_1+b_0)+\eta(x_1\cdot x_1+1)>y_1(w_0\cdot x_1+b_0)$$
注：$\eta>0,(x_1\cdot x_1+1)\geq 1, y_1^2=1$

由上式可知，相比 $y_1(w_0\cdot x_1+b_0)\leq 0$， $y_1(w_1\cdot x_1+b_1)$ 的值大于等于 0 的机会变高了，也就是随机梯度下降法使得 (x_1,y_1) 更加接近被正确分类时的情况（即 $y_1(w\cdot x_1+b)\geq 0$）。同样，在依旧未正确分类的情况下，有：

$$y_1(w_0\cdot x_1+b_0)<y_1(w_1\cdot x_1+b_1)\leq 0\Rightarrow |y_1(w_1\cdot x_1+b_1)|<|y_1(w_0\cdot x_1+b_0)|$$
$$\text{or }\Rightarrow\ -y_1(w_1\cdot x_1+b_1)\ <\ -y_1(w_0\cdot x_1+b_0)$$

损失函数的形式为：$L(w,b)=-y_1(w\cdot x_1+b)-\sum_{x_i\in M\setminus\{x_1\}}y_i(w\cdot x_i+b)$，显然随机梯度下降法使得损失函数值变小了。

■

2.3.2 算法的收敛性（Novikoff 定理）（P31-P33）

请证明 2： 一定能够取得 $\hat{w}_{opt}$ 满足，$\|\hat{w}_{opt}\|=1$，且超平面 $\hat{w}_{opt}\bullet\hat{x}=w_{opt}\bullet x+b_{opt}=0$ 能够将线性可分数据集完全正确划分。

证明 2：

由于训练集是线性可分集，一定存在能将其完全正确划分的超平面。设 $\bar{w}_{opt}$ 满足 $\|\bar{w}_{opt}\|\neq 0$，且 $\forall i, y_i(\bar{w}_{opt}\bullet x_i)>0$，则：

$$\forall i, y_i(\bar{w}_{opt}\bullet x_i)>0\Rightarrow\forall i, y_i\left(\frac{\bar{w}_{opt}}{\|\bar{w}_{opt}\|}\bullet x_i\right)>0, \text{令}\hat{w}_{opt}=\frac{\bar{w}_{opt}}{\|\bar{w}_{opt}\|},\ \text{显然}\|\hat{w}_{opt}\|=1.$$

■

请证明 3：

$$\left\|\hat{w}_k\right\|^2=\left\|\hat{w}_{k-1}\right\|^2+2\eta y_i\hat{w}_{k-1}\bullet\hat{x}_i+\eta^2\left\|\hat{x}_i\right\|^2$$

注：$\hat{w}_k=\hat{w}_{k-1}+\eta y_i\hat{x}_i$，$\hat{w}_k$、$\hat{x}_i$为$n+1$维向量

证明 3：（数学归纳法）

（这是个L_2向量范数的性质，因为后面 SVM 的一个小证明要用到，我尝试推导一下）

要证明上式，等价于证明：

$$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2,\gamma\in R^n,\alpha\in R^n,n\text{有限.}$$

当$n=1$时，上式显然成立。

当$n=2$时，设$\gamma=a=(a_1,a_2),\alpha=b=(b_1,b_2)$，则：

$$\begin{aligned}\|a+b\|^2&=(a_1+b_1)^2+(a_2+b_2)^2\\&=(a_1{}^2+a_2{}^2)+(b_1{}^2+b_2{}^2)+2(a_1b_1+a_2b_2)\\&=\|a\|^2+\|b\|^2+2a\bullet b\end{aligned}$$

因此，当$n=2$时，$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2$成立。

假设，当$n=k$时，$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2$成立。（$k\geq2$）

则需证明当$n=k+1$时，$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2$也成立。

令$x_1,y_1\in R^{k+1}$为 k+1 维列向量，$x,y\in R^k$为x_1,y_1的前 k 维子向量，即：

$$x_1=\begin{pmatrix}x\\x_1{}^{k+1}\end{pmatrix}=\begin{pmatrix}x_1{}^1\\x_1{}^2\\...\\x_1{}^k\\x_1{}^{k+1}\end{pmatrix},\qquad y_1=\begin{pmatrix}y\\y_1{}^{k+1}\end{pmatrix}=\begin{pmatrix}y_1{}^1\\y_1{}^2\\...\\y_1{}^k\\y_1{}^{k+1}\end{pmatrix}$$

则有：（上标 k 表示第 k 个分量）

$$\begin{aligned}\|x_1+y_1\|^2&=(x_1{}^1+y_1{}^1)^2+(x_1{}^2+y_1{}^2)^2+...+(x_1{}^k+y_1{}^k)^2+(x_1{}^{k+1}+y_1{}^{k+1})^2\\&=\|x+y\|^2+(x_1{}^{k+1}+y_1{}^{k+1})^2=\|x\|^2+\|y\|^2+2x\bullet y+(x_1{}^{k+1}+y_1{}^{k+1})^2\\&=\left(\|x\|^2+(x_1{}^{k+1})^2\right)+\left(\|y\|^2+(y_1{}^{k+1})^2\right)+2(x\bullet y+x_1{}^{k+1}y_1{}^{k+1})\\&=\|x_1\|^2+\|y_1\|^2+2x_1\bullet y_1\end{aligned}$$

因此当$n=k+1$时，$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2$也成立。根据$y_1^2=1$知等式$\left\|\hat{w}_k\right\|^2=\left\|\hat{w}_{k-1}\right\|^2+2\eta y_i\hat{w}_{k-1}\bullet\hat{x}_i+\eta^2\left\|\hat{x}_i\right\|^2$成立。

■

（后记：这个小问题如果不纠结的话，跳过这里，定理剩余证明容易看懂。思考一下，1 维时的结果显而易见，就是两项和的平方展开即可；但 n 维呢，是否也可以这样展开，这就不是一目了然的了。）

请证明 4： $\hat{w}_k\cdot\hat{w}_{opt}\le\left\|\hat{w}_k\right\|\cdot\left\|\hat{w}_{opt}\right\|$

证明 4：

$$\hat{w}_k\cdot\hat{w}_{opt}\le\left\|\hat{w}_k\right\|\cdot\left\|\hat{w}_{opt}\right\|\cdot\cos\theta\le\left\|\hat{w}_k\right\|\cdot\left\|\hat{w}_{opt}\right\|=\left\|\hat{w}_k\right\|$$

$$注:\cos\theta\le1,\theta=\widehat{\left(\hat{w}_k,\hat{w}_{opt}\right)},\left\|\hat{w}_{opt}\right\|=1$$

■

2.3.3 感知机学习算法的对偶形式（P33-P34）

关键词 2：

$$w\leftarrow w+\eta y_ix_i$$
$$b\leftarrow b+\eta y_i$$
$$\alpha_i=n_i\eta,\sum_{i=1}^{N}n_i=n$$

说明 2：

$w\leftarrow w+\eta y_ix_i$，且设点$(x_i,y_i)$被误分了$n_i$次，$\sum_{i=1}^{N}n_i=n$，则：

当$n_i=0$时，说明点(x_i,y_i)在迭代过程中从来没有被误分过。

当$n_i\ge1$时，说明点(x_i,y_i)在迭代过程中被误分了n_i次，此时$w\leftarrow w+\eta y_ix_i$迭代了$n_i$次，因此对于所有样本点$(x_i,y_i)$，$i\in\{1,2,...,N\}$有：

$$w=\sum_{i=1}^{N} n_i \eta y_i x_i=\sum_{i=1}^{N} \alpha_i y_i x_i$$

$$b=\sum_{i=1}^{N} n_i \eta y_i=\sum_{i=1}^{N} \alpha_i y_i$$

注：$w_0=0, b_0=0$

■

2.3.1 感知机算法的原始形式（P28-P29）

关键词 3： 算法 2.1 的 R 实现/例 2.1 的 R 程序求解

（采用随机梯度下降法，线性可分/不可分感知机）

说明 3：（算法思路）

输入：训练数据集

$$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}, x_i \in R^n, y_i \in \{-1,+1\}, i=1,2,...,N; 0<\eta \leq 1$$

输出：w,b；感知机模型：$f(x)=sign(w\cdot x+b)$

（1）选取初值 w_0,b_0

（2）在训练集中选取数据(x_i,y_i)（这里选误分类最严重的点）

（3）如果对于 $x_i \in M$，有 $y_i(w\cdot x_i+b)\leq 0$，则选取误分点为：

$$(x,y)=\max_{xi\in M}\left|y_i(w\cdot x_i+b)\right|=\min_{xi\in M}\{y_i(w\cdot x_i+b)\}$$

$$w \leftarrow w+\eta yx$$

$$b \leftarrow b+\eta y$$

（4）转至（2），直至训练集中没有误分点。

我利用 R 编写了函数 linePercept()、print.linePercept()、plot.linePercept()、preClinePercept()、predict.linePercept()这几个函数来实现感知机的原始算法，其中 linePercept()是求解函数，可以用来求解线性可分感知机、也可以用来求解线性不可分感知

机。plot.linePercept()可以将二维的感知机转换成图形。函数predict.linePercept()可用来预测。其他函数是辅助函数。

首先来介绍一下这函数 linePercept()的作用。然后用这个函数来求解例 2.1.（详细代码参加附录 2）

linePercept(cls="y",atr=c("x1","x2"),data=NULL,aita=1,endure=0,maxiter=1000,w0=rep(0,length(atr)),b0=0). cls 指的是“class”类别属性，也就是因变量；atr 指的是“attribute”特征，是一个包含特征名称的字符串向量，特征的顺序是可以任意的，函数输出的结果会根据特征的顺序稍有变动；data 在这里只能是数据框，且列名必须包含 cls 和 atr 指定的变量; aita 是学习效率（$0<\eta\leq 1$）；endure=是容忍度，endure=0 指的是线性可分机，当 endure>0 时是线性不可分情形；maxiter=1000 指定最大迭代次数；w0=rep(0,length(atr)),b0=0 指定权重的初始值，默认为 0；但注意，当 endure>0 时，w0 的元素和 b0 不能全为 0.

predict.linePercept(lPobj,cls="y",atr=c("x1","x2"),atr_value=NULL)函数用来实现预测。lPobj 指 linePercept()训练出来的模型，cls、atr 的含义与 linePercept()中的相同，atr_value=必须输入一个只包括特征信息的数据框，而且特征名称要与 atr 参数指定的相同，顺序可以不同。对于二维特征，predict 还会随便打印出预测图，高于二维的不打印。

好了，现在我们首先利用这些函数来求解例 2.1.书本原图如下。

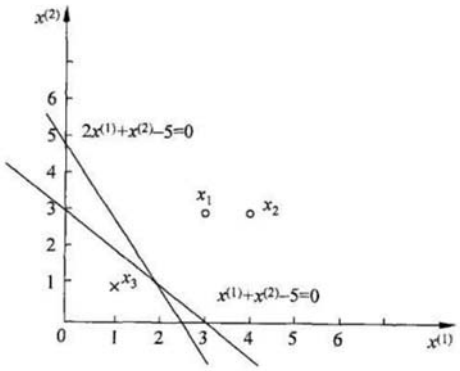


图 2.2　感知机示例

在这里，我们尝试训练完模型后，一次性预测多个新实例。先将训练样本数据及新实例的特征都存入数据框。

```
#### 例 2.1 线性可分感知机 ####
x1<-c(3,4,1);x2<-c(3,3,1);y<-c(1,1,-1)#样本数据
dataB2.1<-data.frame(x1=x1,x2=x2,y=y)#把样本数据存入数据框
data_atr<-data.frame(x1=c(0,2,1,3),x2=c(1,1,3,2))#把将要预测的特征数据存入数据框
```

求解例 2.1，代码如下。

```
percept<-linePercept(data=dataB2.1,cls="y",atr=c("x1","x2"))#其余参数用默认值
percept
names(percept)
```

输出结果如下：

```
>percept<-linePercept(data=dataB2.1,cls="y",atr=c("x1","x2"))
The Final sign_min is :  1
> percept
$Finalweight
     x1 x2  b
[1,]  1  1 -3

$iteration
[1] 7
```

```
> names(percept)
 [1] "Finalweight"
 [2] "weight"
 [3] "iteration"
 [4] "miss"
 [5] "origindata"
 [6] "atrdata"
 [7] "clsdata"
 [8] "endure"
 [9] "aita"
[10] "w0"
[11] "b0"
```

可见，percept 中存储了许多有用的结果，但是在 console 只打

印其中的两个组件，即 Finalweight、iteration.这是由于我编写了函数 print.linePercept()对 console 的输出指定的方式，只输出必要的结果，保持界面简洁。现在，我们看到模型求解出来的最终权重和书本求解出来的结果是一样的，而且迭代次数 iteration 也和书本的相同。那问题来了，函数 linePercept()训练出来的模型的迭代情况是怎样的，每一步迭代选择的误分类点是什么？不用担心，渣君把这些信息分别存储在了 weight 组件和 miss 组件中。现在，我们来提取这些信息，并将训练出来的模型绘制成图。代码如下。

```
percept[1:4]#提取前 4 个组件
plot(percept)#绘图
```

结果如下：

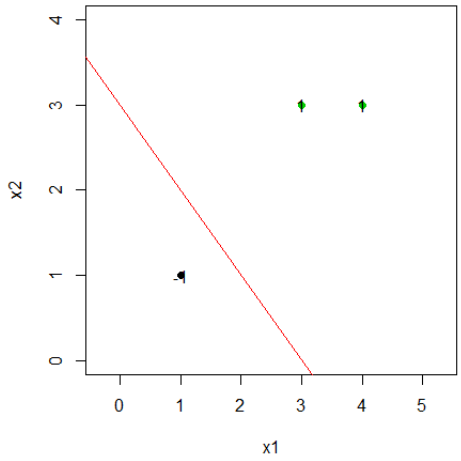
```
> percept[1:4]
$Finalweight
     x1 x2  b
[1,]  1  1 -3

$weight
    iter0 iter1 iter2 iter3 iter4 iter5 iter6 iter7
x1      0     3     2     1     0     3     2     1
x2      0     3     2     1     0     3     2     1
b       0     1     0    -1    -2    -1    -2    -3

$iteration
[1] 7

$miss
[1] 1 3 3 3 1 3 3
```

weight 记录每次迭代权重更改的情况，miss 记录了每次迭代所选取的误分样本点是第几个样本点。与书本结果一致。

表 2.1　例 2.1 求解的迭代过程

迭代次数	误分类点	w	b	$w \cdot x+b$
0		0	0	0
1	x_1	$(3,3)^T$	1	$3x^{(1)}+3x^{(2)}+1$
2	x_3	$(2,2)^T$	0	$2x^{(1)}+2x^{(2)}$
3	x_3	$(1,1)^T$	−1	$x^{(1)}+x^{(2)}-1$
4	x_3	$(0,0)^T$	−2	−2
5	x_1	$(3,3)^T$	−1	$3x^{(1)}+3x^{(2)}-1$
6	x_3	$(2,2)^T$	−2	$2x^{(1)}+2x^{(2)}-2$
7	x_3	$(1,1)^T$	−3	$x^{(1)}+x^{(2)}-3$
8	0	$(1,1)^T$	−3	$x^{(1)}+x^{(2)}-3$

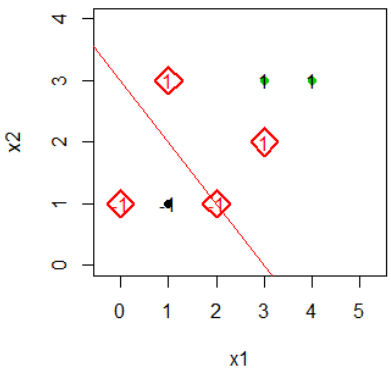
将模型绘制成图，得：



好了，现在我们尝试一下利用训练出来的模型进行预测。

```
predict(percept,cls="y",atr=c("x1","x2"),atr_value = data_atr)#直接调用 predict 即可
```

结果如下：



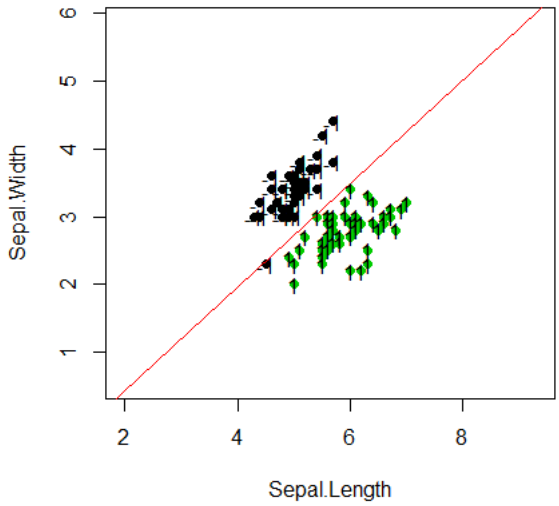
	x1	x2	y
1	0	1	-1
2	2	1	-1
3	1	3	1
4	3	2	1

最后，为了检查这几个函数的适用性，我利用 R 内置数据集 iris 进行了测试。在测试前要进行一些数据处理，主要是 linePercept()是按照书本的伪代码编的，这就要求二分类属性变量“y”必须是{+1，-1}而且是数值型的，而不是 factor。因为算法设计过程中涉及了关于“y”的数学计算，而且通过“y”的符号来设计损失函数。我从 iris 数据集的前 100 个样本中（两类）随机抽取了 90 个作为训练集，10 个作为预测集。

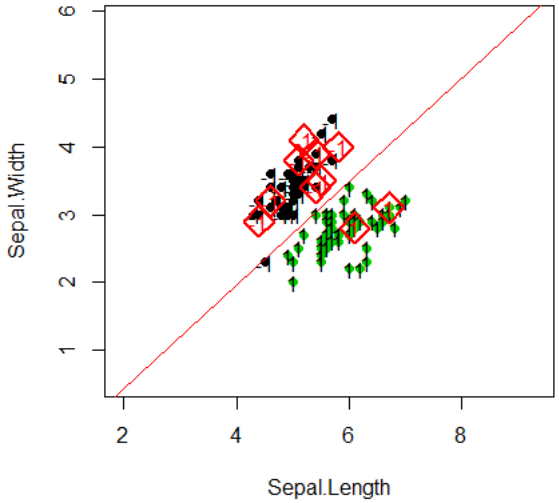
代码如下。

```
#### iris 数据集测试 ####
datatest<-iris[1:100,]
datatest$Species<-as.factor(as.character(datatest$Species))
datatest$Species<-as.factor(as.numeric(datatest$Species))
datatest$Species<-as.numeric(datatest$Species)
datatest$Species<-ifelse(datatest$Species==1,-1,1)#转换成可计算的+1，-1
lab<-sample(1:100,90)
data_iris<-datatest[lab,]#训练集
data_iris_atr<-datatest[-lab,c(1,2)]#测试集
### 训练模型 ###
percept_iris<-linePercept(cls="Species",atr=c("Sepal.Length","Sepal.Width"),data=data_iris,
                          endure = 1,w0=c(1,1))
percept_iris
plot(percept_iris)
### 模型预测 ###
predict(percept_iris,cls="Species",atr=c("Sepal.Length","Sepal.Width"),
atr_value =data_iris_atr )
```

（查看下一页）




```
> percept_iris#模型训练结果
$Finalweight
        Sepal.Length Sepal.Width     b
[1,]            12.4        -16.2 -18

$iteration
[1] 258
```



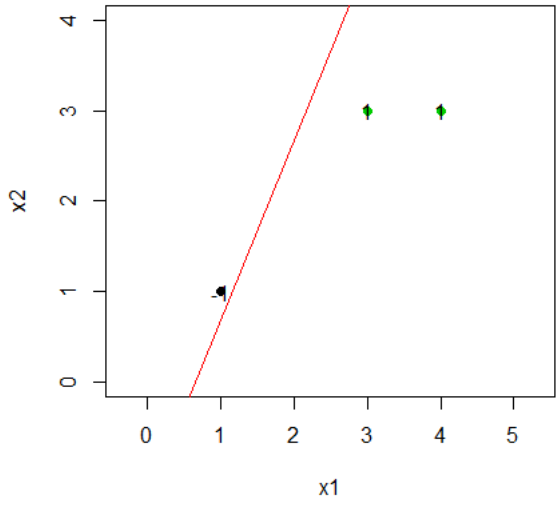

```
> predict(percept_iris,cls="Species",
atr=c("Sepal.Length","Sepal.Width"),
atr_value =data_iris_atr )#测试结果
    Sepal.Length Sepal.Width Species
6            5.4         3.9      -1
9            4.4         2.9      -1
15           5.8         4.0      -1
20           5.1         3.8      -1
32           5.4         3.4      -1
33           5.2         4.1      -1
37           5.5         3.5      -1
48           4.6         3.2      -1
74           6.1         2.8       1
87           6.7         3.1       1
```

小伙伴们还可以尝试在训练模型时，更改参数，如初始值 w0，容忍度 endure 等，对于 iris 这个数据集，会得出稍有不同但大致相同的拟合模型。另外，原书上有讲到，对于线性可分情形，不同的初始值会导致不同的感知机，这也可以通过代码体现。如下。

改变 w0 的初始值：

```
percept1<-linePercept(data=dataB2.1,cls="y",atr=c("x1","x2"),w0=c(10,1))
percept1[1:4]
plot(percept1)
```

结果如下，得到不同之前的感知机。




```
> percept1[1:4]
$Finalweight
      x1 x2  b
[1,]  6 -3 -4

$weight
    iter0 iter1 iter2 iter3 iter4
x1     10     9     8     7     6
x2      1     0    -1    -2    -3
b       0    -1    -2    -3    -4

$iteration
[1] 4

$miss
[1] 3 3 3 3
```

■

2.3.3 感知机学习算法的对偶形式（P33-P34）

关键词 4： 例 2.2 的 R 实现/感知机对偶算法的 R 实现（线性可分机）

说明 4：（算法思路）

输入：训练数据集

$$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\},x_i\in R^n,y_i\in\{-1,+1\},i=1,2,...,N;0<\eta\le 1$$

输出：α,b；感知机模型：$f(x)=sign\left(\sum_{j=1}^{N}\alpha_j y_j x_j\cdot x+b\right)$

（1）选取初值$\alpha \leftarrow 0$，$b \leftarrow 0$

（2）在训练集中选取数据(x_i, y_i)（这里选误分类最严重的点）

（3）如果对于$x_i \in M$，有$y_i\left(\sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i + b\right) \leq 0$，则选取误分点为：

$$(x, y) = \max_{xi \in M}\left|y_i\left(\sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i + b\right)\right| = \min_{xi \in M}\left\{y_i\left(\sum_{j=1}^{N}\alpha_j y_j x_j \cdot x_i + b\right)\right\}$$

$$\alpha_i \leftarrow \alpha_i + \eta$$

$$b \leftarrow b + \eta y$$

（4）转至（2），直至训练集中没有误分点。

最后为了简单起见，对于“对偶算法”我只编写了仅适用于线性可分情况的感知机函数 DualPercept().另外，我利用了 R 中 S3 类的继承性质，不再对 DualPercept()函数输出的对象“DualPercept” 另外编写打印、作图、预测等函数（class 类属性为“DualPercept”，父类属性为“linePercept”）由于继承性质的作用，我们可以直接在 R 中调用 plot()来实现二维特征时的作图，直接调用 predict()来实现预测。

DualPercept(cls="y",atr=c("x1","x2"),data=NULL,aita=1, maxiter=1000,alpha0=rep(0,nrow(data)),b0=0)所有参数的含义与 linePercept()中阐述的相同，alpha0 指定默认初始值为 0。

DualPercept()能完成任务几乎和 linePercept 一样，所不同的是，DualPercept()函数是按照对偶算法编写的，求解的思路不一样。（详细代码参见附录 2）

现在，我们利用对偶算法函数 DualPercept()来求解例 2.2.

代码如下。

```
### 例 2.2 的 R 实现 ###
###  使用了 S3 类的继承性质 ###
perpectdual<-DualPercept(cls="y",atr = c("x1","x2"),data=dataB2.1)
plot(perpectdual)
perpectdual
class(perpectdual)
perpectdual[1:5]
predict(perpectdual,cls="y",atr = c("x1","x2"),atr_value = data_atr)
```

结果如下：

```
> perpectdual[1:5]
$Finalweight
     x1 x2  b
[1,]  1  1 -3

$Finalalpha
     alpha1 alpha2 alpha3  b
[1,]      2      0      5 -3

$iteration
[1] 7

$Alpha
       iter0 iter1 iter2 iter3 iter4 iter5 iter6 iter7
alpha1     0     1     1     1     1     2     2     2
alpha2     0     0     0     0     0     0     0     0
alpha3     0     0     1     2     3     3     4     5
b          0     1     0    -1    -2    -1    -2    -3

$miss
[1] 1 3 3 3 1 3 3
```

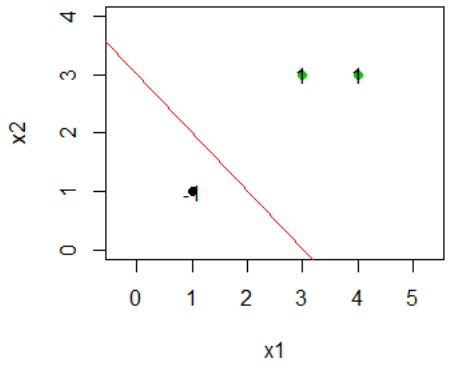
稍作对比，发现与书本求解的结果完全一致。

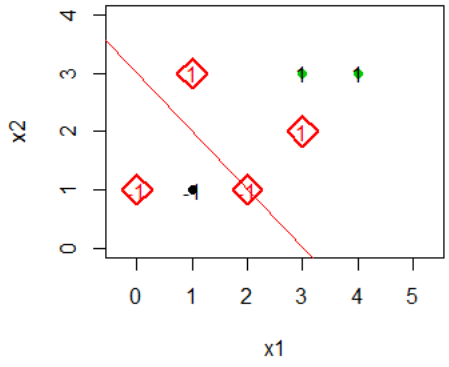
表 2.2　例 2.2 求解的迭代过程

k	0	1	2	3	4	5	6	7
		x_1	x_3	x_3	x_3	x_1	x_3	x_3
α_1	0	1	1	1	2	2	2	2
α_2	0	0	0	0	0	0	0	0
α_3	0	0	1	2	2	3	4	5
b	0	1	0	−1	0	−1	−2	−3

在此，本渣只罗列一下输出的结果，对于函数 DualPercept()的其他操作，小伙伴们可以自己去尝试，比如利用在例 2.1 中建立的数据集 data_iris（线性可分）来测试一下函数 DualPercept().

与书中的结果一样，例 2.2 利用对偶算法求出的感知机和例 2.1 利用原始算法求出的感知机一样。如图。





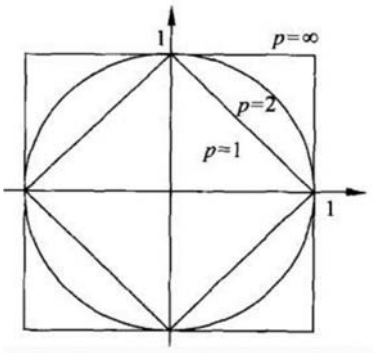

```
> class(perpectdual)
[1] "DualPercept"#子类属性
[2] "linePercept"#父类属性
> names(perpectdual)
 [1] "Finalweight"
 [2] "Finalalpha"
 [3] "iteration"
 [4] "Alpha"
 [5] "miss"
 [6] "atrdata"
 [7] "clsdata"
 [8] "aita"
 [9] "alpha0"
[10] "b0"
> plot(perpectdual)
> predict(perpectdual,
cls="y",atr = c("x1","x2"),
atr_value = data_atr)
  x1 x2  y
1  0  1 -1
2  2  1 -1
3  1  3  1
4  3  2  1
```

■

第三章　K 近邻法

3.2.2 距离度量（P39）

请证明 1：　　图 3.2 对应的 3 种距离及 $L_\infty=\max\limits_{l\in\{1,2,\cdots,n\}}\left|x_i^{(l)}-x_j^{(l)}\right|$



证明 1：

图 3.2 给出了二维空间中 p 取不同值时，与原点的 L_p 距离为 1（$L_p=1$）的点的图形. 对于两点 x_i、x_j 设：

$$L_p=\left(\sum_{l=1}^{2}\left|x_i^{(l)}-x_j^{(l)}\right|^p\right)^{\frac{1}{p}}=\left(\left|x_i^{(1)}-x_j^{(1)}\right|^p+\left|x_i^{(2)}-x_j^{(2)}\right|^p\right)^{\frac{1}{p}}=\left(|x|^p+|y|^p\right)^{\frac{1}{p}}=1$$

$$注: x=x_i^{(1)}-x_j^{(1)}, y=x_i^{(2)}-x_j^{(2)}$$

于是：

当 $p=1$ 时，显然有 $|x|+|y|=1.$

当 $p=2$ 时，显然有 $x^2+y^2=1.$

当 $p=\infty$ 时，不失一般性设 $|x|\le|y|=\max\limits_l\left|x_i^{(l)}-x_j^{(l)}\right|$. 则有：

$$\left(|y|^p\right)^{\frac{1}{p}}\le\left(|x|^p+|y|^p\right)^{\frac{1}{p}}\le\left(2|y|^p\right)^{\frac{1}{p}}, p\ge1\Rightarrow|y|\le\left(|x|^p+|y|^p\right)^{\frac{1}{p}}\le2^{\frac{1}{p}}|y|$$

$$\Rightarrow|y|\le\lim_{p\to\infty}\left(|x|^p+|y|^p\right)^{\frac{1}{p}}\le\lim_{p\to\infty}2^{\frac{1}{p}}|y|=|y|$$

$$\Rightarrow\lim_{p\to\infty}\left(|x|^p+|y|^p\right)^{\frac{1}{p}}=|y|=\max_l\left|x_i^{(l)}-x_j^{(l)}\right|=1,\quad L_\infty=\max_l\left|x_i^{(l)}-x_j^{(l)}\right|=1$$

由于，若$|x|\le|y|$，则有$|x|\le|y|\le 1$（$|x|\le|y|=1$）.

实际上当在n维空间时，按照上述方法容易证明$L_\infty=\max\limits_{l\in\{1,2,..,n\}}\left|x_i^{(l)}-x_j^{(l)}\right|$.

对于$x_i=\left(x_i^{(1)},x_i^{(2)},...,x_i^{(n)}\right)^T$，$x_j=\left(x_j^{(1)},x_j^{(2)},...,x_j^{(n)}\right)^T$，不妨设，

$$y_l=x_i^{(l)}-x_j^{(l)},l=1,2,...,n.$$

$$|y|=\max_{l\in\{1,2,...,n\}}\left|x_i^{(l)}-x_j^{(l)}\right|=\max_{l\in\{1,2,...,n\}}|y_l|$$

则有：

$$\left(|y|^p\right)^{\frac{1}{p}}\le\left(\sum_{l=1}^{n}|y_l|^p\right)^{\frac{1}{p}}\le\left(n|y|^p\right)^{\frac{1}{p}},p\ge1,n\le\infty\Rightarrow|y|\le\left(\sum_{l=1}^{n}|y_l|^p\right)^{\frac{1}{p}}\le n^{\frac{1}{p}}|y|$$

$$\Rightarrow|y|\le\lim_{p\to\infty}\left(\sum_{l=1}^{n}|y_l|^p\right)^{\frac{1}{p}}\le\lim_{p\to\infty}n^{\frac{1}{p}}|y|=|y|$$

$$\Rightarrow\lim_{p\to\infty}\left(\sum_{l=1}^{n}|y_l|^p\right)^{\frac{1}{p}}=|y|=\max_l\left|x_i^{(l)}-x_j^{(l)}\right|$$

$$\Rightarrow L_\infty=\max_l\left|x_i^{(l)}-x_j^{(l)}\right|$$

■

请证明 2： $L_p=\left(\sum\limits_{l=1}^{n}\left|x_i^{(l)}-x_j^{(l)}\right|^p\right)^{\frac{1}{p}}$ 关于 p 单调递减.

证明 2：

思考：之所以想到要证明这个问题原因是原书例 3.1 给出如下一组计算结果.

$$L_1(x_1,x_3)=6>L_2(x_1,x_3)=4.24>L_3(x_1,x_3)=3.78>L_4(x_1,x_3)=3.57$$

小伙伴发现了没有，L_p 随着 p 的增加不断递减，难道这仅仅是一种巧合？不，渣君拒绝巧合，渣君相信一切都是上天最好的安排。于是，我又尝试推导了一下。

证明 $L_p=\left(\sum\limits_{l=1}^{n}\left|x_i^{(l)}-x_j^{(l)}\right|^p\right)^{\frac{1}{p}}$ 关于 p 单调递减等价于证明，

$$f(p)=\left(\sum_{i=1}^{n}x_i{}^p\right)^{\frac{1}{p}},x_i\geq 0,p\geq 1.\text{关于}p\text{单调递减.}$$

令 $x_k=\max\{x_1,x_2,\ldots,x_n\}\neq 0$，则有：

$$f(p)=\left(\sum_{i=1}^{n}x_i{}^p\right)^{\frac{1}{p}}=x_k\left(\sum_{i=1}^{n}\left(\frac{x_i}{x_k}\right)^p\right)^{\frac{1}{p}}$$

$$=x_k\left(1+\sum_{i\neq k}^{n}\left(\frac{x_i}{x_k}\right)^p\right)^{\frac{1}{p}}=x_k\left(1+\sum_{i\neq k}^{n}(a_i)^p\right)^{\frac{1}{p}},0\leq a_i=\left(\frac{x_i}{x_k}\right)\leq 1,a_k=1.$$

显然，$\sum_{i\neq k}^{n}(a_i)^p$ 关于 p 单调减，$(1+\bullet)^{\frac{1}{p}}$ 关于 p 单调减，所以：

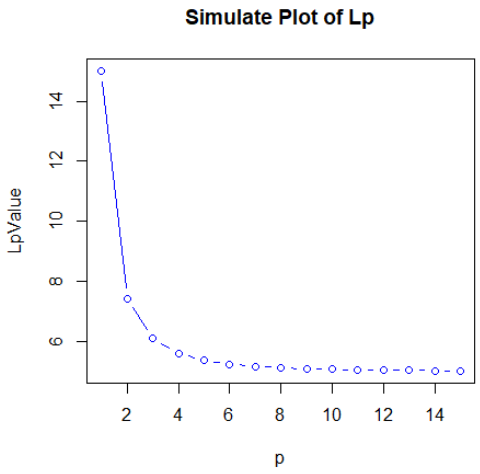
$f(p)=\left(\sum_{i=1}^{n}x_i{}^p\right)^{\frac{1}{p}}$ 关于 p 单调递减，于是 $L_p=\left(\sum_{l=1}^{n}\left|x_i^{(l)}-x_j^{(l)}\right|^p\right)^{\frac{1}{p}}$ 关于 p 单调减.

为了更直观的认识这个性质，我们可以利用 R 做一个模拟，看看 L_p 范数是否真的如同上述证明那样是递减，并且 $L_\infty=\max_l\left|x_i^{(l)}-x_j^{(l)}\right|$.

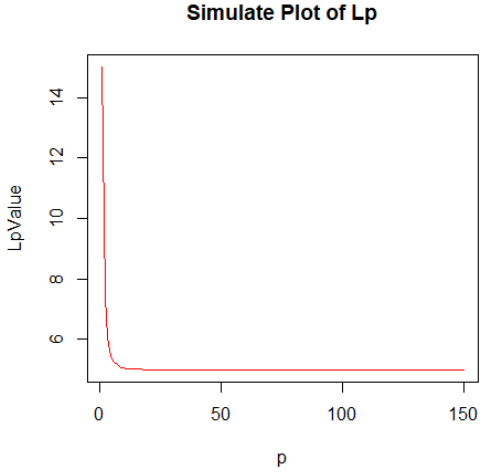
代码如下：

```
####   Lp 范数递减性模拟  ####
LpSim.Plot<-function(number,maxp=1){
  if(any(number<0)) stop("The number must not smaller than 0.")
  max_num<-max(number)
  LpVec<-vector(length = maxp)
  for(i in 1:maxp) LpVec[i]<-(sum(number^i))^(1/i)
  tye<-ifelse(maxp<=20,"b","l")
  cols<-ifelse(maxp<=20,"blue","red")
  plot(1:maxp,LpVec,type = tye,ylab="LpValue",xlab="p",col=cols,
       main="Simulate Plot of Lp")
  list(maxnumber=max_num,minLp=min(LpVec),LpValue=LpVec)
}
LpSim.Plot(number = 1:5,maxp=15)
LpSim.Plot(number = 1:5,maxp=150)#增大 p 值
LpSim.Plot(number = sample(1:50,10),maxp=15)
```

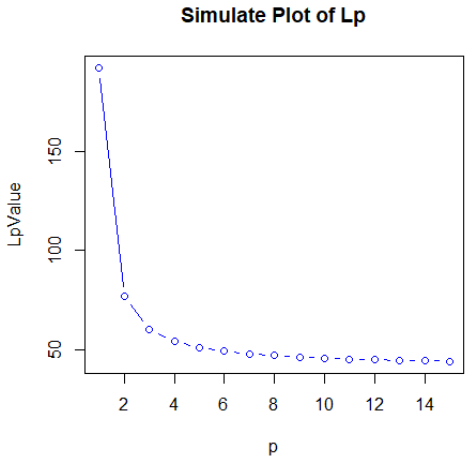
得到三个输出结果分别如下：




```
> LpSim.Plot(number = 1:5,maxp=15)
$maxnumber
[1] 5

$minLp
[1] 5.011692

$LpValue
 [1] 15.000000  7.416198  6.082202
 [4]  5.593655  5.360220  5.232131
 [7]  5.155656  5.107345  5.075576
[10]  5.054056  5.039144  5.028628
```




```
> LpSim.Plot(number = 1:5,maxp=150)
$maxnumber
[1] 5

$minLp
[1] 5

$LpValue#省略大部分输出
   [1] 15.000000  7.416198
   [3]  6.082202  5.593655
   [5]  5.360220  5.232131
   [7]  5.155656  5.107345
   [9]  5.075576  5.054056
  [11]  5.039144  5.028628
```




```
> LpSim.Plot(number =
 sample(1:50,10),maxp=15)
$maxnumber
[1] 49

$minLp
[1] 49.63413

$LpValue
 [1] 292.00000 101.13358
 [3]  73.29027  63.24317
 [5]  58.30388  55.46587
```

3.3.1 构造 *kd* 树（P41-P42）

关键词 1： 例 3.2　2 维平衡 *kd* 树构造

$$T=\left\{(2,3)^T,(5,4)^T,(9,6)^T,(4,7)^T,(8,1)^T,(7,2)^T\right\}.$$

说明 1：

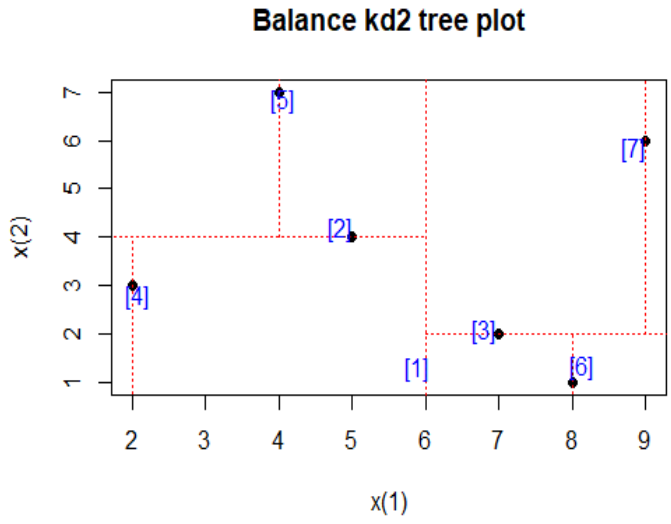
书本的解答中说选择 $x^{(1)}$ 轴作为第一个结点的划分轴，并且 $x^{(1)}$ 中 6 个数的中位数是 7. 但其实这是错的，中位数不是 7，而是 6. 于是我按照中位数是 6，将 $x^{(1)}=6$ 作为第一个结点重新做了一下这个题。

求解中位数结点的迭代过程如下：

$$[1]\begin{pmatrix}x^{(1)}:2,4,5,7,8,9\\ x^{(2)}:3,7,4,2,1,6\end{pmatrix}\Rightarrow x^{(1)}=\frac{5+7}{2}=6$$

$$[2]\begin{pmatrix}x^{(1)}:2,5,4\\ x^{(2)}:3,4,7\end{pmatrix}\Rightarrow x^{(2)}=4,\quad [3]\begin{pmatrix}x^{(1)}:8,7,9\\ x^{(2)}:1,2,6\end{pmatrix}\Rightarrow x^{(2)}=2$$

$$[4]\begin{pmatrix}x^{(1)}:2\\ x^{(2)}:3\end{pmatrix}\Rightarrow x^{(1)}=2,[5]\begin{pmatrix}x^{(1)}:4\\ x^{(2)}:7\end{pmatrix}\Rightarrow x^{(1)}=4,[6]\begin{pmatrix}x^{(1)}:8\\ x^{(2)}:1\end{pmatrix}\Rightarrow x^{(1)}=8,[7]\begin{pmatrix}x^{(1)}:9\\ x^{(2)}:6\end{pmatrix}\Rightarrow x^{(1)}=9$$



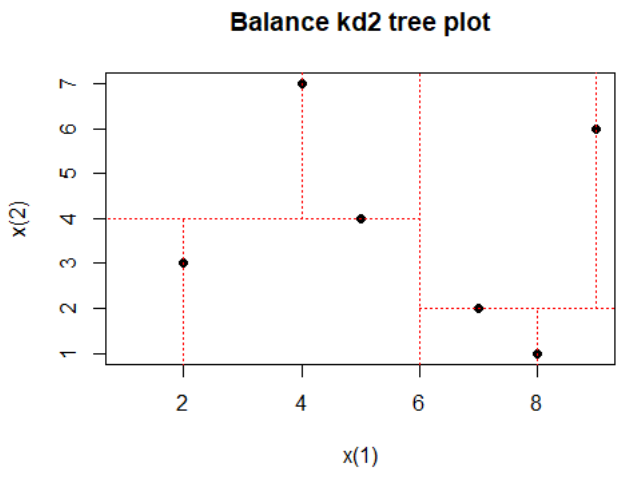
最后为了满足本渣自娱自乐的需求，我用 R 编写了两个函数来实现离散特征的2维平衡*kd*树，分别是kd_tie(x1,x2)和kd_plot(x1,x2)kd_tie(x1,x2)是用来计算迭代过程中生成的不同深度的结点以及该结点对应的训练实例集，kd_plot(x1,x2)负责将 kd_tie(x1,x2)的结果转换为图形。（为保持简洁，函数代码见附录 3）代码的含义大家就自己去理解啦，本渣就不再一一阐述了，主要是利用了 list 的性质循环地装载计算出来的结点和子集。

好了，现在我们利用函数 kd_tie(x1,x2)和 kd_plot(x1,x2)来求解一下书本的例题 3.2，调用以下代码：

（kd_plot(x1,x2)内部调用了 kd_tie(x1,x2)）

```
x1<-c(2,4,5,7,8,9);x2<-c(3,7,4,2,1,6)
kd_plot(x1=x1,x2=x2)
```

运行代码后得出的结果如下，与例题求解的结果一致。

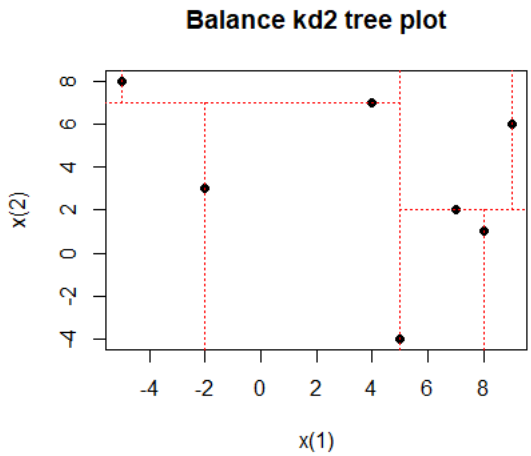



```
>kd_plot(x1=x1,x2=x2)
$tielist
$tielist[[1]]
[1] 6

$tielist[[2]]
[1] 4 2

$tielist[[3]]
[1] 2 4 8 9
```

现在我们改变一下样本，比如说，对于有负值的样本。

```
x1<-c(-2,4,5,7,8,9,-5);x2<-c(3,7,-4,2,1,6,8)
kd_plot(x1=x1,x2=x2)
```

运行代码后结果如下：

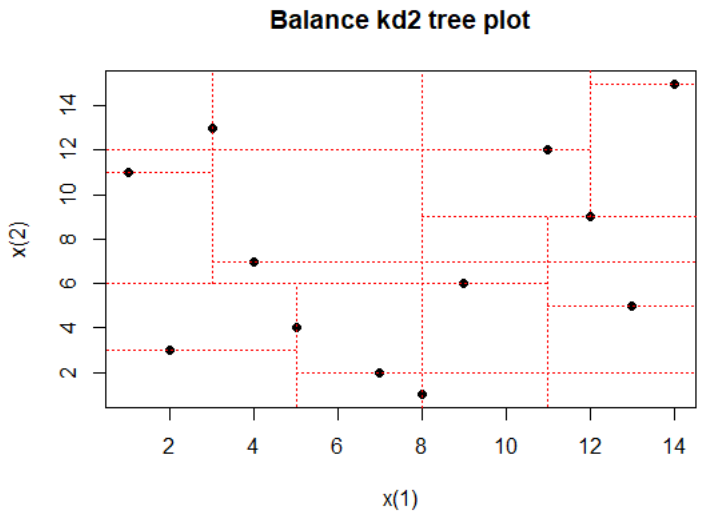



```
> kd_plot(x1=x1,x2=x2)
$tielist
$tielist[[1]]
[1] 5

$tielist[[2]]
[1] 7 2

$tielist[[3]]
[1] -2 -5  8  9
```

我们再增加一些样本量，比如：

```
x1<-c(2,4,5,7,8,9,12,13,11,1,3,14)
x2<-c(3,7,4,2,1,6,9,5,12,11,13,15)
kd_plot(x1=x1,x2=x2)
```

运行代码后结果如下：




```
> kd_plot(x1=x1,x2=x2)
$tielist
$tielist[[1]]
[1] 8

$tielist[[2]]
[1] 6 9

$tielist[[3]]
[1]  5  3 11 12

$tielist[[4]]
[1]  3  2 11  7  6  5 12 15
```

■

关键词 2： 基于线性扫描方法的 k 近邻的 R 实现

说明 2：

由于本渣实在是编不出来高维的 kd 树，所以在不考虑程序性能的前提下，我编写了基于 Lp 距离及线性扫描法的 k 近邻函数，包括 LpCalculate(),lineKnn(),print.lineKnn(),plot.lineKnn().

其中 LpCalculate()是辅助函数，会在 lineKnn()内部被调用，在编写过程中用到了 dprep 包中的 moda()函数，因此要成功运行 lineKnn(),需要先安装 dprep 包。我仅简单介绍 lineKnn()函数的用法。plot.lineKnn()可对二维情形做预测图。（详细代码参见附录 3）

LineKnn(cls=NULL,atr=NULL,dataTrain=NULL,dataTest=NULL,k=3,p=2)其中 cls 指分类变量，atr 指特征；dataTrain 是训练集，包括 cls,atr 中指代的变量数据；dataTest 是预测集，仅包括 atr 中指代的数据，分类变量允许是非数值型变量；k 指近邻数，默认为 3；p 指距离类型，默认 p=2 为欧氏距离。

值得注意的是 k 近邻的训练和预测是同时进行的。现在，我们利用函数 lineKnn()来实践一下 Knn 算法。这里，我们只用 iris 数据集来试一下。

首先，我们来整理一下数据集。

```
####  iris 数据集的 knn 实例 ####
lab<-sample(1:150,130)
dataKnn_iris<-iris[lab,]#训练集
dataKnn_iris_test<-iris[-lab,]#测试集
dataKnn_iris_atr<-iris[-lab,-5]#测试特征集
```

然后开始用 k 近邻方法预测，代码如下。

```
Knn_iris<-lineKnn(cls="Species",atr=c("Sepal.Length","Sepal.Width"),
                  dataTrain = dataKnn_iris,dataTest = dataKnn_iris_atr,k=3,p=10)
####训练及预测，我们采用其中两个特征做 Knn 预测
Knn_iris#预测结果
#### 预测正确率 ####
miss<-cbind(as.character(dataKnn_iris_test[,5]),as.character(Knn_iris$FinalPredict))
misslab<-which(!apply(miss,1,function(x)ifelse(length(unique(x))==1,T,F)))
missp<-length(misslab)/20;1-missp
```

结果如下：

```
> Knn_iris$PredictMat#预测结果
    Sepal.Length Sepal.Width    Species
4            4.6         3.1     setosa
10           4.9         3.1     setosa
36           5.0         3.2     setosa
45           5.1         3.8     setosa
52           6.4         3.2  virginica
54           5.5         2.3 versicolor
57           6.3         3.3  virginica
61           5.0         2.0 versicolor
63           6.0         2.2 versicolor
65           5.6         2.9 versicolor
84           6.0         2.7 versicolor
86           6.0         3.4  virginica
91           5.5         2.6 versicolor
109          6.7         2.5 versicolor
112          6.4         2.7  virginica
123          7.7         2.8  virginica
128          6.1         3.0 versicolor
136          7.7         3.0  virginica
145          6.7         3.3  virginica
149          6.2         3.4  virginica
```

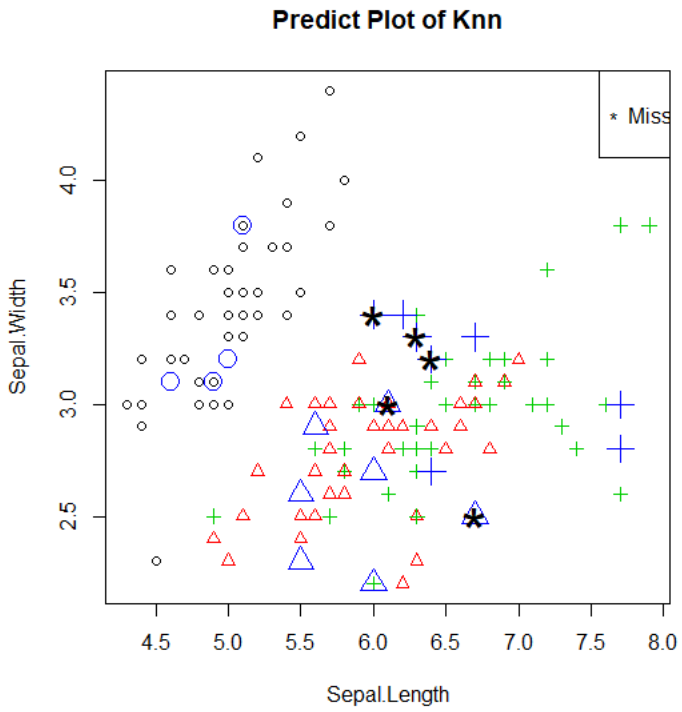
```
      [,1]         [,2]
 [1,] "setosa"     "setosa"
 [2,] "setosa"     "setosa"
 [3,] "setosa"     "setosa"
 [4,] "setosa"     "setosa"
 [5,] "versicolor" "virginica"
 [6,] "versicolor" "versicolor"
 [7,] "versicolor" "virginica"
 [8,] "versicolor" "versicolor"
 [9,] "versicolor" "versicolor"
[10,] "versicolor" "versicolor"
[11,] "versicolor" "versicolor"
[12,] "versicolor" "virginica"
[13,] "versicolor" "versicolor"
[14,] "virginica"  "versicolor"
[15,] "virginica"  "virginica"
[16,] "virginica"  "virginica"
[17,] "virginica"  "versicolor"
[18,] "virginica"  "virginica"
[19,] "virginica"  "virginica"
[20,] "virginica"  "virginica"
```

```
> missp<-length(misslab)/20;1-missp#预测正确率
[1] 0.75
```

可见，当 k=3，p=10 时，有 5 个样本被预测错误，预测准确率为 0.75.我们还可以利用 plot()函数将结果预测绘制成图。代码如下。

```
plot(Knn_iris)#蓝色是预测点，不同形状代表不同的类
points(Knn_iris$dataTest[,1][misslab],Knn_iris$dataTest[,2][misslab],col="black",
       pch="*",cex=3)
legend("topright",legend = "Miss",pch="*",col="black")
```

结果如图：



更多的尝试，就留给小伙伴们了，哈哈。

■

第四章　朴素贝叶斯算法

4.1.1 基本方法（P47-P48）

关键词 1：　　（条件独立性假设）

$$P\left(X=x\,|\,Y=c_k\right)=P\left(X^{(1)}=x^{(1)},...,X^{(n)}=x^{(n)}\,|\,Y=c_k\right)$$
$$=\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}\,|\,Y=c_k\right).$$

说明 1：

朴素贝叶斯算法依赖于条件独立性假设，这说明我们应该注意，在利用朴素贝叶斯法进行分类预测时应该对特征进行相关性检验，对于不同类别属性，如果特征的相关性高，那这样的训练样本可能并不适合使用朴素贝叶斯法。应该考虑其他方法或者对特征进行处理，使其满足特征的条件独立性要求。（另外注意到书本举的例子都是针对离散特征的）

■

4.1.2 后验概率最大化的含义（P48-49）

请证明 1：　　$\min\limits_{y}R_{\exp}(f)=E\left[L\left(Y,f\left(X\right)\right)\right]\hat{=}\arg\max\limits_{c_k}P\left(c_k\,|\,X=x\right)$

（即后验概率最大化原则等价于期望风险最小化原则）

证明 1：

思考与疑惑：到现在我依然没有看懂原文的证明过程，困惑尚未解决。所以我贴一下原文的证明并阐述本渣的困惑。原文的证明为：

$$R_{\exp}(f)=E\left[L\left(Y,f\left(X\right)\right)\right]=E_X\sum_{k=1}^{K}\left[L\left(c_k,f\left(X\right)\right)\right]P\left(c_k|X\right)$$

为了使期望风险最小化，只需对 $X=x$ 逐个极小化，有：

$$\begin{aligned} f(x) &= \arg\min_{y\in y}\sum_{k=1}^{K} L(c_k,y)P(c_k|X=x) \\ &= \arg\min_{y\in y}\sum_{k=1}^{K} P(y\neq c_k|X=x) = \arg\min_{y\in y}\left(1-P(y=c_k|X=x)\right) \\ &= \arg\max_{y\in y} P(y=c_k|X=x) = \arg\max_{c_k} P(c_k|X=x) \end{aligned}$$

上式中 $\arg\min_{y\in y}\sum_{k=1}^{K} L(c_k,y)P(c_k|X=x) = \arg\min_{y\in y}\sum_{k=1}^{K} P(y\neq c_k|X=x)$ 我没有理解，因为

$$\begin{aligned} \sum_{k=1}^{K} P(y\neq c_k|X=x) &= \sum_{k=1}^{K}\left(1-P(y=c_k|X=x)\right) \\ &= K-\sum_{k=1}^{K} P(y=c_k|X=x) = K-1 \end{aligned}$$

$$\Rightarrow \arg\min_{y\in y}\sum_{k=1}^{K} P(y\neq c_k|X=x) = \arg\min_{y\in y}(K-1), K-1\text{为常数}$$

$$\Rightarrow \arg\min_{y\in y}\sum_{k=1}^{K} P(y\neq c_k|X=x) \neq \arg\min_{y\in y}\sum_{k=1}^{K} L(c_k,y)P(c_k|X=x)$$

另一方面，$\arg\min_{y\in y}\sum_{k=1}^{K} P(y\neq c_k|X=x) = \arg\min_{y\in y}\left(1-P(y=c_k|X=x)\right)$ 这一步我也没有理解。于是我尝试着跳开原文的证明过程，从其他角度去理解等式 $f(x)=\arg\min_{y\in y}\sum_{k=1}^{K} L(c_k,y)P(c_k|X=x)=\arg\max_{c_k} P(c_k|X=x)$。以下是我的思路。

令 $I(y\neq c_k)=L(c_k,y)=\begin{cases}1, & y\neq c_k\\ 0, & y=c_k\end{cases}\quad k=1,2,...,K$。

又令 $\Delta=\sum_{k=1}^{K} L(c_k,y)P(c_k|X=x)=\sum_{k=1}^{K} I(y\neq c_k)P(c_k|X=x)$. 等式里面的 c_k 指真实值，y 为预测值。观察：

当 $y=c_1$ 时，有 $\Delta_1=0+\sum_{k\neq 1} P(c_k|X=x)=1-P(c_1|X=x)$

当 $y=c_2$ 时，有 $\Delta_2=\sum_{k\neq 2}^{K}P\left(c_k\middle|X=x\right)=1-P\left(c_2\middle|X=x\right)$

当 $y=c_j$ 时，有 $\Delta_j=\sum_{k\neq j}^{K}P\left(c_k\middle|X=x\right)=1-P\left(c_j\middle|X=x\right)$

于是，有下式成立：

$$\min\Delta=\min\left\{\Delta_1,\Delta_2,...,\Delta_j,...,\Delta_K\right\}=\min_j\left\{\Delta_j\right\}$$
$$=\min_{c_j}\left\{1-P\left(c_j\middle|X=x\right)\right\}=\max_{c_j}P\left(c_j\middle|X=x\right)=\max_{c_k}P\left(c_k\middle|X=x\right)$$

所以，得：$f\left(x\right)=\arg\min\Delta=\arg\max_{c_k}P\left(c_k\middle|X=x\right)$，即后验概率最大化原则等价于期望风险最小化原则。

■

4.2.1 极大似然估计（P49）

请证明 2：

$$P\left(Y=c_k\right)=\frac{\sum_{i=1}^{N}I\left(y_i=c_k\right)}{N},P\left(X^{(j)}=a_{jl}\middle|Y=c_k\right)=\frac{\sum_{i=1}^{N}I\left(x_i^{(j)}=a_{jl},y_i=c_k\right)}{\sum_{i=1}^{N}I\left(y_i=c_k\right)}$$
$$j=1,2,...,n;l=1,2,...,S_j;k=1,2,...,\mathrm{K}.$$

证明 2：

极大似然估计其实很容易证明，稍微推导即可。

设 $P_k=P\left(Y=c_k\right),V_k=\sum_{i=1}^{N}I\left(y_i=c_k\right)$，则有 $\sum_{k=1}^{K}V_k=N,\sum_{k=1}^{K}P_k=1$，其中 $\sum_{k=1}^{K}V_k=N,P_k>0$ 为已知条件，$\sum_{k=1}^{K}P_k=1$ 为最优化约束。写出 Y 的对数似然函数，如下：

$$L=\log\left(\prod_{k=1}^{K}P_k^{V_k}\right)=\sum_{k=1}^{K}V_k\cdot\log P_k$$

则，极大似然估计转化为最优化问题：

$$\max_{p}\left\{\sum_{k=1}^{K}V_k\cdot\log P_k\right\}$$
$$s.t\quad\sum_{k=1}^{K}P_k=1$$

写出拉格朗日函数，如下：

$$L(P,\alpha)=\sum_{k=1}^{K}V_k\cdot\log P_k+\alpha\left(\sum_{k=1}^{K}P_k-1\right)$$

则有：

$$\frac{\partial L}{\partial P}=\frac{V_k}{P_k}+\alpha=0\Rightarrow V_k+\alpha P_k=0\Rightarrow\sum_{k=1}^{K}V_k+\alpha\sum_{k=1}^{K}P_k=0$$
$$\Rightarrow N+\alpha=0\Rightarrow-\alpha=N\Rightarrow\frac{V_k}{P_k}=-\alpha=N$$
$$\Rightarrow P_k=\frac{V_k}{N}=\frac{\sum_{i=1}^{N}I(y_i=c_k)}{N}$$

同理，将样本数据限制在$Y=c_k$，按照相同的步骤可得：

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

■

4.2.2 学习与分类算法（P50-51）

关键词 2： 例 4.1/未知特征

说明 2：

对于一个新的实例的特征x，x的取值必须是已知特征的取值范

围内的。否则无法使用朴素贝叶斯法进行预测。比如当$x=(2,\mathrm{S})^T$，2 属于$X^{(1)}$的取值范围$\{1,2,3\}$，S 属于$X^{(2)}$的取值范围$\{\mathrm{S},\mathrm{M},\mathrm{L}\}$，因此可以进行后验概率的估计。但若$x=(4,\mathrm{H})^T$由于新的特征值不在已知特征的取值范围内，因此无法进行后验概率估计。 ■

关键词 3： 小结/离散特征朴素贝叶斯法的 R 实现

说明 3：

○对于训练集，属性 Y 是已知的，因此可以假设特征的条件独立性$P(X=x|Y=c_k)=P\left(X^{(1)}=x^{(1)},...,X^{(n)}=x^{(n)}|Y=c_k\right)=\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}|Y=c_k\right).$

通过极大似然估计对$P(Y=c_k)$和$P\left(X^{(j)}=a_{jl}|Y=c_k\right)$进行学习，从而实现对联合概率分布$P(X,Y)$的学习。

○对于预测数据，只知道特征X，而不知道Y。由于已经通过训练集完成了联合概率分布$P(X,Y)$的学习。因此，可以对给定的X，求$Y=c_k$的后验概率，并选出后验概率最大的属性作为预测样本的类别属性，即，$y=\arg\max_{c_k}P(Y=c_k)\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}|Y=c_k\right).$

○朴素贝叶斯模型的预测过程是“期望风险最小化”（后验概率最大化），但模型的学习过程并不是“期望风险最小化”而是“经验风险最小化”（极大似然估计）。

OK，最后为了把自娱自乐的精神发扬光大，我用 R 编写了四个函数来实现基于极大似然估计和贝叶斯估计的朴素贝叶斯模型的学习及预测，为 navieBayes(),print.navieBayes(),preCnavieBayes()和 predict.navieBayes().这个实现只是针对离散特征的。我先简

单讲讲这几个函数的作用，然后利用这几个函数来求解书本的例4.1 和例 4.2.（详细代码参见附录 4）

navieBayes(cls="Y",atr=c("X1","X2"),data=NULL,lmada=0)中cls 指的是“class”类别属性，也就是因变量；atr 指的是“attribute”特征，是一个包含特征名称的字符串向量，特征的顺序是可以任意的，函数输出的结果会根据特征的顺序稍有变动；data 在这里只能是数据框，且列名必须包含 cls 和 atr 指定的变量；lmada 是控制参数，lmada=0，模型采用极大似然估计法进行学习，lmada>0,模型采用贝叶斯估计法进行学习，比如 lmada=1 时，使用的就是拉普拉斯平滑法。输出的结果是一个 R 中类别属性（class）指定为“navieBayes”的 S3 类对象，里面包括了一些组件，通过 names()函数可以查看所有组件，navieBayes()函数的输出结果不会全部打印在控制台上，只会按照 print.navieBayes()（R 泛函）指定的打印方式打印部分信息，所有的组件信息可通过 names()或 str()获取。

print.navieBayes(obj)指定如何 print 类“navieBayes”。不用显式地调用 print.navieBayes(obj)只要正常打印即可，因为这个函数已经作为泛函 print 的一个子函数进入 R 系统，print 函数会根据你输入对象的类别属性（这里是“navieBayes”）自动调用函数 print.navieBayes(obj)。

preCnavieBayes(NBobj,cls=NULL,atr=NULL,atr_value=NULL)函数只能对一个实例进行预测，而且实例特征值必须在 atr_value=中

以向量的形式输入。cls,atr 参数的含义和 navieBayes()函数中的含义一样。参数 NBobj=必须是（class）属性为“navieBayes”对象，否则函数报错。读者在重现代码时，本渣不建议用这个函数进行预测。preCnavieBayes()函数只是我为了编写预测函数（R 泛函）predict.navieBayes()所写的一个辅助函数，当然也是基础的。preCnavieBayes()会在 predict.navieBayes()的内部被调用。predict.navieBayes(NBobj,cls=NULL,atr=NULL,atr_value=NULL)函数是模型学习完之后对新的样本实例进行预测的函数，也是泛函 predict()的一个子函数，predict.navieBayes()不需要显式地被调用。你只需要像做 lm 回归之后直接调用 predict()函数进行预测即可，predict()函数会根据输入对象的类型属性（“navieBayes”）自动调用子函数 predict.navieBayes()进行预测。NBobj,cls=NULL,atr=NULL,atr_value=NULL 等参数的含义与前几个函数描述的含义一致，稍有区别是，在预测时，atr_value=输入的必须是一个包含新样本实例特征信息的数据框。predict.navieBayes()可以一次性预测多个结果，并以数据框的形式打印结果。

好了，介绍完之后，我们现在来用这几个函数求解一下书本的例题 4.1 和 4.2.原题的数据如下：

表 4.1　训练数据

	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

调用以下代码：（先把要样本数据及预测数据都储存在数据框内，这

里，我们一次预测多个新样本的后验概率）

```
X1<-c(1,1,1,1,1,2,2,2,2,2,3,3,3,3,3)
X2<-c("S","M","M","S","S","S","M","M","L","L","L","M","M","L","L")
Y<-c(-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1)
dataB4.1<-data.frame(X1=X1,X2=X2,Y=Y)#将训练数据储存在数据框中
pred_var<-data.frame(X1=c(2,1,1,3,3),X2=c("S","L","S","M","L"))#需要预测的实例
```

现在，我们来求解例 4.1，代码如下。

```
####  例 4.1  lmada=0 极大似然估计 ####
plist<-navieBayes(cls="Y",atr=c("X1","X2"),data=dataB4.1,lmada = 0)#训练模型
predict(plist,cls="Y",atr=c("X1","X2"),atr_value =pred_var)#预测模型
```

在查看预测结果之前，我们先查看一下 plist 的组件和属性。

```
> plist#打印“navieBayes”类，学习结果
response = prec_var:   Y ; lmada =   0

The variables are :   X1 X2

$Y
  level prob
1    -1  0.4
2     1  0.6

$X1
     prec_var
xvar        -1         1
   1 0.5000000 0.2222222
   2 0.3333333 0.3333333
   3 0.1666667 0.4444444

$X2
     prec_var
xvar        -1         1
   L 0.1666667 0.4444444
   M 0.3333333 0.4444444
   S 0.5000000 0.1111111
```

```
> class(plist)#属性
[1] "navieBayes"
> names(plist)
[1] "Y"          "X1"          "X2"
"lmada"                  "response"
"variables"
> plist$lmada
[1] 0
> plist$variables
[1] "X1" "X2"
> sapply(plist,class)#组件的属性
                Y                X1
"data.frame"         "table"
               X2             lmada
      "table"      "numeric"
     response     variables
  "character"  "character"
```

可见，控制台上并没有打印出 plist 所有的组件，只打印包含概率结果的关键部分。通过 class()函数可以识别，plist 是类

“navieBayes”的一个对象。通过 sapply(plist,class)可以查看 plist 各个组件的类别属性。最后我们观察 plist 中的概率结果，和例题 4.1 的结果是一样的，只是分数与小数表示形式的不同而已。

好了，现在我们查看一下模型的预测结果，如下。

```
> predict(plist,cls="Y",atr=c("X1","X2"),atr_value =pred_var)#直接调用 predict()即可
The response :  Y
   X1 X2 level       post_p
1   2  S    -1 0.06666667
2   2  S     1 0.02222222
3   1  L    -1 0.03333333
4   1  L     1 0.05925926
5   1  S    -1 0.10000000
6   1  S     1 0.01481481
7   3  M    -1 0.02222222
8   3  M     1 0.11851852
9   3  L    -1 0.01111111
10  3  L     1 0.11851852
> 1/15#Y=-1
[1] 0.06666667
> 1/45#Y=1
[1] 0.02222222
```

结果与例题 4.1 一致，我们还看到当特征为其他取值组合时的预测情况，在此不一一阐述了。现在我们来求解一下例 4.2，如下。

```
## 例 4.2 lmada=1  贝叶斯估计 拉普拉斯平滑 ##
plist1<-navieBayes(cls="Y",atr=c("X1","X2"),data=dataB4.1,lmada = 1)
predict(plist1,cls="Y",atr=c("X1","X2"),atr_value =pred_var)
```

输出结果为：

（查看下一页）

```
>plist1#模型学习
response = prec_var:   Y ; lmada =   1

The variables are :   X1 X2

$Y
  level         prob
1      -1 0.4117647
2       1 0.5882353

$X1
      prec_var
xvar            -1             1
   1 0.4444444 0.2500000
   2 0.3333333 0.3333333
   3 0.2222222 0.4166667

$X2
      prec_var
xvar            -1             1
   L 0.2222222 0.4166667
   M 0.3333333 0.4166667
   S 0.4444444 0.1666667
```

```
#### 预测结果 ####
The response :   Y
   X1 X2 level        post_p
1   2  S      -1 0.06100218
2   2  S       1 0.03267974
3   1  L      -1 0.04066812
4   1  L       1 0.06127451
5   1  S      -1 0.08133624
6   1  S       1 0.02450980
7   3  M      -1 0.03050109
8   3  M       1 0.10212418
9   3  L      -1 0.02033406
10  3  L       1 0.10212418
> 28/459#Y=-1
[1] 0.06100218
> 5/153#Y=1
[1] 0.03267974
```

可见上述结果与书本例 4.2 的求解一致。读者还可以通过改变参数 lmada 的值来进行不同的贝叶斯估计，查看预测结果的变动。本渣就不在此一一阐述了。

■

5.2.2 信息增益（P60-P61）

关键词 1：　　信息增益的含义/0-1 分布$H(p)$的 R 实现

说明 1：（参考《概率论基础》复旦、李贤平，P226）

熵$H(p)$是满足以下 3 个性质的连续可导函数。

（1）$H(p)$是p_i的连续函数。

（2）对于有 n 个等概率结果的试验，$H(p)$是 n 的单调上升函数。

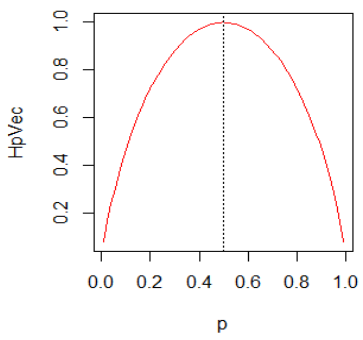
（3）一个试验分成相继的两个试验时，未分之前的$H(p)$是既分之后的H的加权和。

（香农定理）唯一满足（1）、（2）、（3）三个条件的H具有下列形式：

$$H = -C\sum_{i=1}^{n} p_i \log p_i$$

其中 C 是正常数。定理中的系数 C，可以根据方便选择，它取决于度量单位。常用的度量单位有二进制单位及十进制单位，前者对数的底数取 2，后者用常用对数。把常数 C 单位化为 1，就得到熵的定义。

然后，我们可以利用 R 来画一下 0-1 分布的$H(p)$曲线。




```
#### 0-1 分布的 H(p)曲线 ####
p<-pretty(c(0.01,0.99),100)
HpVec<-vector(length = length(p))
for(i in 1:length(p))
HpVec[i]<--p[i]*log(p[i],2)-(1-p[i])*log(1-p[i],2)
plot(p,HpVec,type="l",col="red");
abline(v=0.5,lty=3)
```

■

5.2.3 信息增益比（P63）

关键词 2： 信息增益/信息增益比/例 5.2 的 R 实现

说明 2：

书本有说，以信息增益作为选取划分训练数据集特征的依据，存在偏向于选择取值较多的特征的问题，因此采用信息增益比作为对这一问题的修正。我先罗列书本的相关公式，然后阐述一下自己的理解。

$$g(D,A)=H(D)-H(D|A)$$

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)},H_A(D)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$$

我们来看：

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

$$=-\frac{1}{|D|}\sum_{i=1}^{n}\sum_{k=1}^{K}|D_{ik}|\cdot\left[\log_2|D_{ik}|-\log_2|D_i|\right]$$

$|D|=\sum_{i=1}^{n}|D_i|$ 是常数，n 指特征 A 的取值个数；$\sum_{k=1}^{K}|D_{ik}|=|D_i|, D_{ik}=D_i\cap C_k.$ $k=1,2,...,K.$ 因此，在 $|D|$ 固定的情况下，大致有以下 3 个特点。

（1）$|D_i|$ 关于 n 单调减（非增，$n\uparrow\Rightarrow|D_i|\downarrow, i=1,...,n.$）

（2）$|D_{ik}|$ 关于 $|D_i|$ 单调增（非减，$|D_i|\uparrow\Rightarrow|D_{ik}|\uparrow, k=1,2,...,K.$）

（3）当 $|D_i|$ 增加一个单位时，$|D_{ik}|$ 的增加必然小于等于 1.（这是由于各个不同的类 $C_1,C_2,...,C_K.$ 分摊了这个增量，$\sum_{k=1}^{K}|D_{ik}|=|D_i|.$）

我们假设，$H(|D_i|)=H(D|A), |D_{ik}|=f_k(|D_i|), k=1,2,...,K.$ 是关于 $|D_i|$ 的连

续可导函数。则有：（由于$|D_i|$关于n单调减）

$$g(D,A)=H(D)-H(D|A)\text{关于}n\uparrow\Leftrightarrow H(D|A)\text{关于}n\downarrow\Leftrightarrow H(D|A)\text{关于}|D_i|\uparrow$$

则有：（当取自然对数时）

$$\frac{\partial H(D|A)}{\partial|D_i|}=-\frac{1}{|D|}\sum_{k=1}^{K}\left\{\frac{\partial f_k}{\partial|D_i|}\left[\log f_k(|D_i|)-\log|D_i|\right]+f_k(|D_i|)\left[\frac{1}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-\frac{1}{|D_i|}\right]\right\}$$

由于$f_k(|D_i|)$关于$|D_i|$单调增，$f_k(|D_i|)=|D_{ik}|\le|D_i|$于是我们粗略假设有，$\frac{\partial f_k}{\partial|D_i|}>0.$又有$\log f_k(|D_i|)-\log|D_i|<0$。因此$\frac{\partial f_k}{\partial|D_i|}\left[\log f_k(|D_i|)-\log|D_i|\right]<0$。

另一方面，$\frac{1}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-\frac{1}{|D_i|}=\frac{1}{|D_i|}\left(\frac{|D_i|}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-1\right)$。其中，

$$\frac{|D_i|}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}=\frac{\partial f_k/f_k}{\partial|D_i|/|D_i|}$$

是指f_k关于$|D_i|$的弹性，根据特点（3），我们可得下式：

$$\left(\frac{|D_i|}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-1\right)\le0\Rightarrow\frac{1}{|D_i|}\left(\frac{|D_i|}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-1\right)=\frac{1}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-\frac{1}{|D_i|}\le0$$

$$\Rightarrow f_k(|D_i|)\left[\frac{1}{f_k(|D_i|)}\frac{\partial f_k}{\partial|D_i|}-\frac{1}{|D_i|}\right]\le0$$

因此，$\frac{\partial H(D|A)}{\partial|D_i|}>0\Rightarrow\left(n\uparrow\Rightarrow|D_i|\downarrow,H(D|A)\downarrow\Rightarrow g(D,A)\uparrow\right).$

由以上阐述可知，对于同一个特征A而言， 特征A取值的个数n越大，信息增益$g(D,A)$也倾向于越大。但需要注意的是，这个对比是就特征A自身而言的。当A与其他特征（比如B）比较时，$g(D,A)$的这种特点并不一定能保证$g(D,A)>g(D,B)$。即如下：（n指特征A的取值个数）

$$n\uparrow\Rightarrow g(D,A)\uparrow$$
$$\not\Rightarrow g(D,A)>g(D,B)$$

尽管如此，$n\uparrow \Rightarrow g(D,A)\uparrow$依然会使得$g(D,A)>g(D,B)$的可能性变大了，因此，信息增益准则更加倾向于选择取值比较多的特征。

信息增益比是对信息增益准则的修正，即在$g(D,A)$的基础上添加一个惩罚参数$\frac{1}{H_A(D)}$，$H_A(D)$是指在实例集D中将A的取值作为类别属性时，实例集D的熵。当A的取值n较大时，$\frac{1}{H_A(D)}$较小，将$g(D,A)$拉小成$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$；当$A$的取值$n$较小时，$\frac{1}{H_A(D)}$较大，将$g(D,A)$扩大成$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$。因此，$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$更加偏向于取值个数较少的特征，因为$\frac{1}{H_A(D)}$将这样的特征的信息增益$g(D,A)$扩大成$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$使得相应的特征更加容易被该准则选择。

接下来，我们将使用在 R 中编辑好的 InfoGain()和 Hatr()函数来求解例 5.2 并通过例子来说明信息增益$g(D,A)$的特点。其中 Hatr()函数是辅助函数，可以用来计算单个特征所对应的经验条件熵。

InfoGain(cls=NULL,atr=NULL,method=c("info","inforate"),data=NULL)函数用来计算所有特征的信息增益或信息增益比。cls 表示类别属性，atr 表示特征；data 为包括 cls、atr 指代的所有变量的数据框;当 method="info"时,求信息增益,当 method="inforate"时，求信息增益比。（函数代码，详见附录 5）

现在，我们利用函数 InfoGain()来求解例 5.2，代码如下。

```
#### 例 5.2 的 R 实现 ####
A1<-rep(c("青年","中年","老年"),each=5)
A2<-c("否","否","是","是","否","否","否","是","否","否","否","否","是","是","否")
A3<-c("否","否","否","是","否","否","否","是","是","是","是","是","否","否","否")
A4<-c("一般","好","好","一般","一般","一般","好","好","非常好","非常好","非常好","好", "好","非常好","一般")
Y<-c("否","否","是","是","否","否","否","是","是","是","是","是","是","是","否")
dataB5.2<-data.frame(A1,A2,A3,A4,Y);dataB5.2
InfoGain(cls="Y",atr=c("A1","A2","A3","A4"),method="info",data=dataB5.2)
InfoGain(cls="Y",atr=c("A1","A2","A3","A4"),method="inforate",data=dataB5.2)
```

结果如下：

```
>InfoGain(cls="Y",atr=c("A1","A2","A3","A4"), method="info",data=dataB5.2)
#信息增益
$infogain
     A1        A2        A3        A4
0.0830075 0.3236502 0.4199731 0.3629896

$HDcls#类别属性的经验熵
[1] 0.9709506

$HatrVec#对应于各个特征的经验条件熵
     A1        A2        A3        A4
0.8879431 0.6473004 0.5509775 0.6079610

$HDatr#以各个特征作为类别属性时的熵
     A1        A2        A3        A4
1.5849625 0.9182958 0.9709506 1.5655962
```

```
>InfoGain(cls="Y",atr=c("A1","A2","A3","A4"), method="inforate",data=dataB5.2)
# 信息增益比
$infogain
     A1        A2        A3        A4
0.0523719 0.3524465 0.4325381 0.2318539

$HDcls
[1] 0.9709506

$HatrVec
     A1        A2        A3        A4
0.8879431 0.6473004 0.5509775 0.6079610

$HDatr
     A1        A2        A3        A4
1.5849625 0.9182958 0.9709506 1.5655962
```

可以发现，函数 InfoGain()所计算的各个特征信息增益与书本例 5.2 的求解结果完全一致。右边还给出了所计算的信息增益比的结果。我们知道 A1/A4 特征都有 3 个取值，而 A2/A3 特征只有 2 个取值，通过以上结果左右对比发现，取值个数多的特征受到更大的惩罚，其信息增益比小于对应的信息增益，而取值个数少的特征受到更小的惩罚，其信息增益比下降较少（这里甚至上升了）。另一方便，我们查看信

息增益的结果，发现 A1（3 个取值）的信息增益小于 A3（2 个取值）的信息增益，因此，并不是取值较多的特征的信息增益一定大于其他特征的信息增益。也就是说 $g(D,A)$ 对于取值较多的特征的偏向性是仅仅对于单个特征自身的信息增益变动而言的，并不能推导出一定会有 $g(D,A)>g(D,B)$.

最后，我们利用 InfoGain()函数来考察一下信息增益偏向于取值较多的特征的问题，即，对单个特征 A，有：（n 指特征 A 的取值个数）

$$n\uparrow \Rightarrow g(D,A)\uparrow$$

为此，我们改变一下特征的数据，假设 A1、A4 特征中的三类可以再细分为五类，代码如下。

```
A1<-rep(c("少年","青年","中年","老年","晚年"),each=3)#改为 5 类
A2<-c("否","否","是","是","否","否","否","是","否","否","否","否","是","是","否")
A3<-c("否","否","否","是","否","否","否","是","是","是","是","是","否","否","否")
A4<-c("坏","好","好","坏","一般","一般","好","好","非常好","非常好","非常好","好",
      "好","极好","一般")#改为 5 类
Y<-c("否","否","是","是","否","否","否","是","是","是","是","是","是","是","否")
dataB5.2<-data.frame(A1,A2,A3,A4,Y);dataB5.2
InfoGain(cls="Y",atr=c("A1","A2","A3","A4"),method="info",data=dataB5.2)
```

计算信息增益得到结果如下：

```
> InfoGain(cls="Y",atr=c("A1","A2","A3","A4"),method="info",data=dataB5.2)
$infogain
     A1        A2        A3        A4
0.2363139 0.3236502 0.4199731 0.4702989
```

可以发现，当特征 A1、A4 的取值个数增加时，其对应的信息增益都明显增大了，与结论 $n\uparrow \Rightarrow g(D,A)\uparrow$ 一致，此时最优特征变成了 A4 而非 A3，这就是所谓的 $g(D,A)$ 的偏向性。 ■

5.3.1 ID3 算法/C4.5 算法（P63-P65）

关键词 3： 例 5.3 的程序求解/ ID3 算法、C4.5 算法的 R 实现

说明 3：（算法思路）

输入：训练数据集 D，特征集 A，阈值 ε

输出：决策树 T

（1）若 D 中所有实例属于同一类 C_k，则 T 为单结点树，并将类 C_k 作为该结点的类标记，返回 T；

（2）若 $A=\varnothing$，则 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T；

（3）否则计算 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g；（C4.5 算法计算信息增益比）

（4）如果的 A_g 的信息增益小于阈值 ε，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T；

（5）否则对 A_g 每一可能值 a_i，依照 $A_g=a_i$ 将 D 分割为若干非空子集 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由结点及其子结点构成树 T，返回 T

（6）对第 i 个子结点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用（1）~（5）步，得到子树 T_i，返回 T_i.

由以上的算法思路我们可以发现，在 ID3/C4.5 算法中，对于离散特征情形，特征集 A 中的每一个特征会且仅会被算法利用一次，不会再迭代过程中被重复利用。以 A_g 为例，这是由于划分后的每一个子集 D_i，$i=1,2,...,n$ 中，A_g 的值是唯一的，即必有 $A_g=a_i$，因此，

对于子集D_i而言，特征A_g已经失去了划分能力。所以在子集D_i上调用信息增益准则时，我们仅计算$A-\{A_g\}$中每个特征的信息增益，不再计算A_g的信息增益。也就是说，A_g被使用一次后，就会立刻退出迭代，不再参与剩下的迭代过程。因此，对于离散特征情形而言，特征的个数不宜太少，否则，建立起来的树就会过于简单。

对于用 R 实现决策树，这次渣君是真没编出来，虽然算法看起来简单，也就几句话，但在编写过程中本渣还没想到较好的存储方法来装迭代的结果。这次只是编了一个简单的函数 subTree()用来求解例题 5.3，以后有机会再尝试一下编完整的决策树。

subTree(cls=NULL,atr=NULL,method=c("info","inforate"),data=NULL,ept=0.1)函数可用来计算以数据集 data 作为样本集的“树桩”（只有根结点及叶结点），多次运用 subTree()函数可以实现简单的决策树求解；参数 ept 是阈值。（函数代码见附录 5）

现在利用函数 subTree()来求解例 5.3，代码如下。

```
####  例 5.3 求解 ID3 信息增益 ####
stree1<-subTree(cls="Y",atr = c("A1","A2","A3","A4"),method="info",data=dataB5.2);
stree2<-lapply(stree[1:2],subTree,cls="Y",atr=c("A1","A2","A4"),method="info")
```

结果如下：

（查看下一页）

```
> stree1#第一次迭代结果
$`A3=否`
      A1 A2       A4   Y
1   青年  否    一般  否
2   青年  否      好  否
3   青年  是      好  是
5   青年  否    一般  否
6   中年  否    一般  否
7   中年  否      好  否
13 老年  是      好  是
14 老年  是 非常好  是
15 老年  否    一般  否

$`A3=是`
      A1 A2       A4   Y
4   青年  是    一般  是
8   中年  是      好  是
9   中年  否 非常好  是
10 中年  否 非常好  是
11 老年  否 非常好  是
12 老年  否      好  是

$newatr
[1] "A1" "A2" "A4"

$infoatr
[1] "A3"
```

```
> stree2#第二次迭代结果
$`A3=否`
$`A3=否`$`A2=否`
      A1    A4   Y
1   青年 一般  否
2   青年   好  否
5   青年 一般  否
6   中年 一般  否
7   中年   好  否
15 老年 一般  否

$`A3=否`$`A2=是`
      A1      A4   Y
3   青年      好  是
13 老年      好  是
14 老年 非常好  是

$`A3=否`$newatr
[1] "A1" "A4"

$`A3=否`$infoatr
[1] "A2"


$`A3=是`
$`A3=是`$origindata
      A1 A2       A4   Y
4   青年  是    一般  是
8   中年  是      好  是
9   中年  否 非常好  是
10 中年  否 非常好  是
11 老年  否 非常好  是
12 老年  否      好  是

$`A3=是`$single
[1] 是
Levels: 否 是
```

可见以上结果与书本的求解一致，首次迭代选择 A3 作为最优的划分特征，并且在“A3=是”的子集中，Y 只有一类“是”，于是这个子集作为一个叶结点。在第二次迭代中，选择 A2 作为最优特征，划分

出来的子集都是单一属性的集合，于是都作为叶结点。对于本例，采用 ID3 算法和 C4.5 算法生成的树是一样的。代码如下。

```
####  例 5.3 求解 C4.5 信息增益比 ####
stree1<-subTree(cls="Y",atr = c("A1","A2","A3","A4"),
                method="inforate",data=dataB5.2);
stree2<-lapply(stree[1:2],subTree,cls="Y",atr=c("A1","A2","A4"),method="inforate")
```

结果如下：

```
> stree1#第一次迭代结果
$`A3=否`
    A1 A2     A4  Y
1  青年 否   一般 否
2  青年 否    好 否
3  青年 是    好 是
5  青年 否   一般 否
6  中年 否   一般 否
7  中年 否    好 否
13 老年 是    好 是
14 老年 是 非常好 是
15 老年 否   一般 否

$`A3=是`
    A1 A2     A4  Y
4  青年 是   一般 是
8  中年 是    好 是
9  中年 否 非常好 是
10 中年 否 非常好 是
11 老年 否 非常好 是
12 老年 否    好 是

$newatr
[1] "A1" "A2" "A4"

$infoatr
[1] "A3"
```

```
> stree2#第二次迭代结果
$`A3=否`
$`A3=否`$`A2=否`
    A1   A4  Y
1  青年 一般 否
2  青年  好 否
5  青年 一般 否
6  中年 一般 否
7  中年  好 否
15 老年 一般 否
$`A3=否`$`A2=是`
    A1     A4  Y
3  青年    好 是
13 老年    好 是
14 老年 非常好 是
$`A3=否`$newatr
[1] "A1" "A4"
$`A3=否`$infoatr
[1] "A2"
$`A3=是`
$`A3=是`$origindata
    A1 A2     A4  Y
4  青年 是   一般 是
8  中年 是    好 是
9  中年 否 非常好 是
10 中年 否 非常好 是
11 老年 否 非常好 是
12 老年 否    好 是
$`A3=是`$single
[1] 是
Levels: 否 是
$`A3=是`$infoatr
[1] "None"
```

■

5.4 决策树的剪枝（P65-P67）

关键词 4： 损失函数的 R 实现/例 5.3 决策树剪枝

说明 4：

决策树的剪枝往往通过极小化决策树整体的损失函数来实现。设树T的叶结点个数为$|T|$，t是树T的叶结点，该叶结点有N_t个样本点，其中k类的样本点有N_{tk}个，$k=1,2,...,K$. $H_t(T)$为叶结点t上的经验熵，$\alpha \geq 0$为参数，则决策树损失函数定义为：

$$C_\alpha(T)=\sum_{t=1}^{|T|} N_t H_t(T)+\alpha|T|$$

$$H_t(T)=-\sum_{k=1}^{K}\frac{N_{tk}}{N_t}\log\frac{N_{tk}}{N_t}$$

参数α表示对模型复杂度的惩罚，较大的α促使选择的较简单的树，较小的α促使选择较复杂的树。$\alpha=0$意味着只考虑模型与训练数据的拟合程度，不考虑复杂度。决策树生成学习局部的模型，而决策树剪枝学习整体的模型。

我用编写了函数 cutTree()、Extree()及 HDfunc()来求例 5.3 所建立的决策树的损失函数值。其中 Extree()及 HDfunc()是辅助函数，Extree()负责将 subTree()计算出来的树中的子结点数据集提取出来并存在一个一层的 list 中，HDfunc()负责计算某个分类属性的经验熵，在 cutTree()内部被调用。（详细代码，参考附录 5）

cutTree(cls="Y",data=NULL,alpha=1)函数计算某颗树的损失函数值，cls=指定分类属性变量，data 输入一个装载树的各个叶结点子集数据的 list（即 Extree()处理后的结果），alpha 指定惩罚参数，

默认值为 1.我们在例 5.3 中利用 subTree()生成了两棵树，stree1 是简单树，只利用了一个特征 A3；stree2 是稍微比 stree1 复杂的树，利用了两个特征（A3，A2）。现在，我们使用 cutTree()来计算一下这

```
####  基于信息增益的剪枝 例 5.3  ####
stree1<-subTree(cls="Y",atr = c("A1","A2","A3","A4"),method="info",data=dataB5.2)
stree2<-lapply(stree1[1:2],subTree,cls="Y",atr=c("A1","A2","A4"),method="info")
le1<-Extree(list(stree1))#Extree()中的输入必须是一个双层的 list
le2<-Extree(stree2)
alp=1
cutTree(cls="Y",data=le1,alpha=alp)#简单树
cutTree(cls="Y",data=le2,alpha=alp)#复杂树
```

结果如下：

```
> cutTree(cls="Y",data=le1,alpha=alp)#简单树的损失函数值
          [,1]
[1,] 10.26466
> cutTree(cls="Y",data=le2,alpha=alp)#复杂树的损失函数值
     [,1]
[1,]    3
```

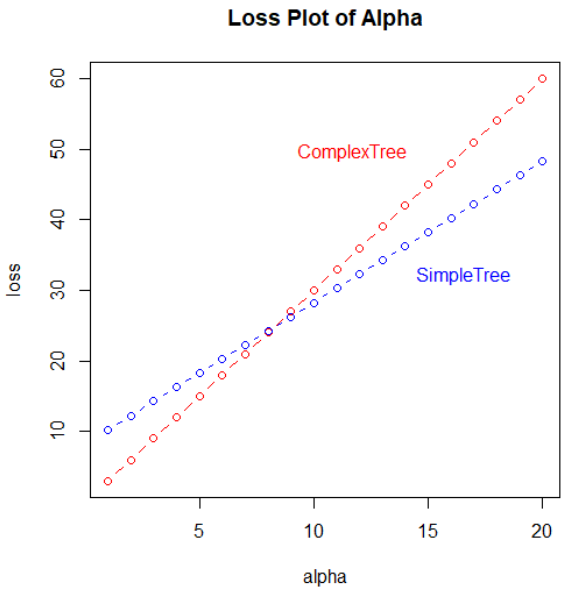
```
> le1
[[1]]
   A1 A2     A4  Y
1  青年 否   一般 否
2  青年 否    好 否
3  青年 是    好 是
5  青年 否   一般 否
6  中年 否   一般 否
7  中年 否    好 否
13 老年 是    好 是
14 老年 是 非常好 是
15 老年 否   一般 否

[[2]]
   A1 A2     A4  Y
4  青年 是   一般 是
8  中年 是    好 是
9  中年 否 非常好 是
10 中年 否 非常好 是
11 老年 否 非常好 是
12 老年 否    好 是
```

```
> le2
[[1]]
   A1     A4  Y
1  青年 一般 否
2  青年  好 否
5  青年 一般 否
6  中年 一般 否
7  中年  好 否
15 老年 一般 否

[[2]]
   A1     A4  Y
3  青年   好 是
13 老年   好 是
14 老年 非常好 是

[[3]]
   A1 A2     A4  Y
4  青年 是   一般 是
8  中年 是    好 是
9  中年 否 非常好 是
```

可见，当$\alpha=1$时，简单树要比复杂树具有更大的损失，此时应选择复杂树。另外根据书本的说明α取值的大小会影响决策树的剪枝行为，较大的α促使选择的较简单的树，较小的α促使选择较复杂的树。我们现在可以利用 cutTree()函数来模拟一下。代码如下。

```
#### 模拟 alpha 的变动对剪枝的影响 ####
ysimple<-vector(length = 20);ycomplex<-vector(length = 20)
for(i in 1:20){
  ysimple[i]<-as.vector(cutTree(cls="Y",data=le1,alpha=i))#简单树
  ycomplex[i]<-as.vector(cutTree(cls="Y",data=le2,alpha=i))#复杂树
}
plot(1:20,ycomplex,type="b",col="red",xlab="alpha",ylab = "loss",
     main="Loss Plot of Alpha")
points(1:20,ysimple,type="b",col="blue")
text(locator(),"SimpleTree",col="blue")
text(locator(),"ComplexTree",col="red")
```

结果如下：



由上图可见，当α较小时，简单树的损失更大，这时不剪枝，选择复杂树；当α较大时，复杂树的损失更大，这时剪枝，选择简单树。这种规律和书本描述的一致。 ■

5.5.1 CART 生成（P68-P71）

关键词 5： 基尼系数$Gini(p)$与熵 $H(p)$的关系及其 R 模拟

说明 5：

我们先写出基尼系数和熵的计算公式，如下：

$$Gini(p)=\sum_{k=1}^{K}p_k(1-p_k)$$

$$H(p)=-\sum_{k=1}^{K}p_k\log p_k$$

我们先来看一下上式中$(1-p_k)$与$-\log p_k$的关系，设函数：

$$f(p_k)=-\log p_k-(1-p_k)=-\log p_k-1+p_k,0<p_k\le 1$$

则有：（假设$f(p_k)$关于p_k连续可导）

$$\left.\begin{array}{c}\frac{\partial f}{\partial p_k}=-\frac{1}{p_k}+1=\frac{p_k-1}{p_k}<0,0<p_k<1.\\ f(1)=0\end{array}\right\}\Rightarrow f(p_k)>f(1)=0\Rightarrow -\log p_k>1-p_k$$

$$\Rightarrow H(p)>Gini(p)$$

另一方面，我们令$g(p_k)=-\log p_k,0<p_k\le 1.$将其在$p_k=1$处一阶及二阶泰勒展开有：

$$g(p_k)=-1\cdot(p_k-1)+o(p_k-1)=1-p_k+o(p_k-1)$$

$$g(p_k)=-1\cdot(p_k-1)+1\cdot(p_k-1)^2+o\left((p_k-1)^2\right)=1-p_k+(1-p_k)^2+o\left((p_k-1)^2\right)$$

我们看到，从一阶泰勒展开可以发现，如果忽略高阶无穷小部分，有$-\log p_k\approx 1-p_k$，因此基尼系数与熵大致上有近似的含义，可以用来表示变量的不确定性，基尼系数越大，变量的不确定性就越大。我们来看二阶泰勒展开式有:

$$\frac{-\log p_k}{1-p_k}=\frac{g(p_k)}{1-p_k}=1+(1-p_k)+\frac{o\left((p_k-1)^2\right)}{1-p_k}=2+\left[\frac{o\left((p_k-1)^2\right)}{1-p_k}-p_k\right]\approx 2$$

因此，大致上有（非常粗略）：

$$H(p)\approx 2Gini(p)\Rightarrow \frac{1}{2}H(p)\approx Gini(p)$$

这就是本渣理解的所谓的熵之半近似等于基尼系数的说法了。

那么，问题来了，究竟什么时候会有$\frac{1}{2}H(p)\approx Gini(p)$呢？我们知道，当变量的取值（特征的取值）个数$K$较多时，变量的熵就会越大，基尼系数也会变大。但是基尼系数的上限是 1，而熵的上限是$\log(K)$，熵之半的上限则为$\log\left(\sqrt{K}\right)$。显然只有当$\sqrt{K}$较小时，熵之半的上限与基尼系数的上限才会比较接近。我们可以利用 R 模拟$Gini(p)$和 $H(p)$的关系，看看是否能验证这种猜测。代码如下。

数据生成及处理：

```
####   Gini(p)与 1/2H(p)关系模拟  ####
#p 是一个概率向量所有元素的和为 1
HpSim<-function(p){#熵
  lp<-ifelse(p==0,0,log(p,2))
  hp<--p%*%lp
  return(hp)
}
GpSim<-function(p){#基尼系数
  p2<-1-p
  gp<-p%*%p2
  return(gp)
}
#随机生成概率分布 p#
pCreate<-function(l=10,chs=1000){
  num<-sample(0:chs,l)
  p<-num/sum(num)
  return(p)
}
```

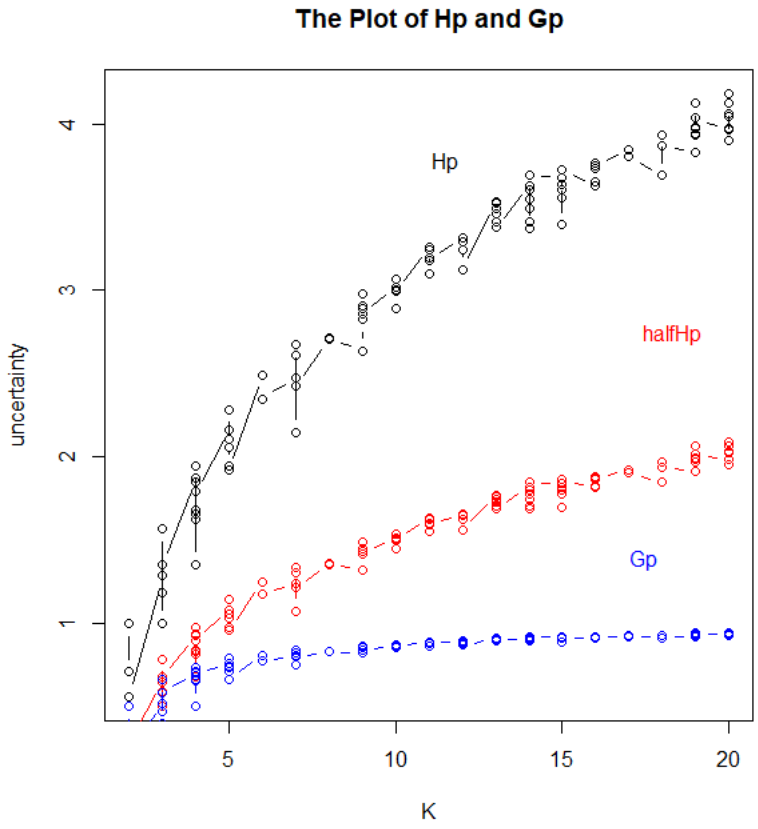
```
plist<-list()
lst<-vector(length = 100)
chlst<-vector(length = 100)
for(i in 1:100){
  l<-sample(2:20,1)
  chs<-sample(30:1000,1)
  lst[i]<-l
  chlst[i]<-chs
  plist[[i]]<-pCreate(l=l,chs=chs)
}
all(sapply(plist,sum)==1)#检查和为 1
hpvec<-sapply(plist,HpSim)
gpvec<-sapply(plist,GpSim)
dataHG<-
data.frame(K=lst,halfHp=hpvec/2,
           Gp=gpvec,Hp=hpvec)
datahg<-dataHG[order(dataHG[,1]),]
```

datahg 数据框装载了按照K值大小排序后的数据。

将 datahg 数据框中的数据绘制成图：

```
plot(datahg$K,datahg$Hp,type="b",col="black",xlab = "K",
     ylab="uncertainty",main="The Plot of Hp and Gp")
points(datahg$K,datahg$halfHp,type = "b",col="red")
points(datahg$K,datahg$Gp,type = "b",col="blue")
text(locator(),"Gp",col="blue")
text(locator(),"halfHp",col="red")
text(locator(),"Hp",col="black")
```

结果如下：

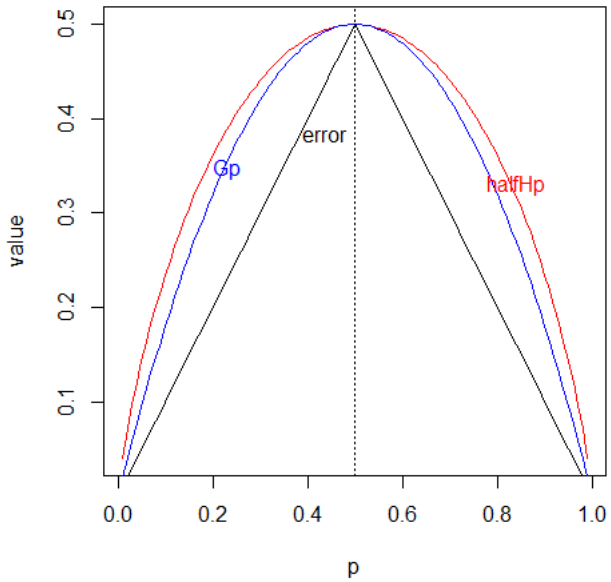


我们可以看到熵是一直大于基尼系数的，而且只有当变量取值个数 K 较小时（这里大致上 $K \le 5$），才会有 $\frac{1}{2}H(p) \approx Gini(p)$；而当 K 较大时，明显地有 $\frac{1}{2}H(p) > Gini(p)$. 结果验证了上述猜想。

另外，由课本我们可以看到，图 5.7 中变量是 0-1 变量，只有两个取值，所以此时，熵之半与基尼系数是非常接近的。我们可以用一

小段代码，实现图 5.7.

```
####  图 5.7 的实现 ####
p<-pretty(c(0.01,0.99),100)
HpVec<-vector(length = length(p))
GpVec<-vector(length = length(p))
for(i in 1:length(p)){
  HpVec[i]<--p[i]*log(p[i],2)-(1-p[i])*log(1-p[i],2)
  GpVec[i]<-2*p[i]*(1-p[i])
}
error<-ifelse(p<.5,p,1-p)
plot(p,HpVec/2,type="l",col="red",xlab="p",ylab="value")
lines(p,GpVec,type="l",col="blue")
lines(p,error,type = "l",col="black")
abline(v=0.5,lty=3)
text(locator(),"Gp",col="blue")
text(locator(),"halfHp",col="red")
text(locator(),"error",col="black")
```



关键词 6： CART 生成/条件基尼系数的 R 实现/例 5.4 的程序求解

说明 6：

（1）条件基尼指数

对于给定的样本集合 D，其基尼系数为：

$$Gini(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2$$

这里，C_k 是 D 中属于第 k 类的样本子集，K 是类的个数。

如果样本集合 D 根据特征 A 是否取某一可能值 a 被分割成 D_1 和 D_2 两部分，即；

$$D_1=\{(x,y)\in D|A(x)=a\},\quad D_2=D-D_1$$

则在特征 A 的条件下，集合 D 的基尼指数定义为：

$$Gini(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2)$$

$$Gini(D_1)=1-\sum_{k=1}^{K}\left(\frac{|D_{1k}|}{|D_1|}\right)^2, D_{1k}=C_k\bigcap D_1, k=1,2,...,K.$$

$$Gini(D_2)=1-\sum_{k=1}^{K}\left(\frac{|D_{2k}|}{|D_2|}\right)^2, D_{2k}=C_k\bigcap D_2, k=1,2,...,K.$$

基尼指数 $Gini(D,A)$ 表示经 $A=a$ 分割后集合 D 的不确定性，基尼指数越大，样本集合的不确定性就越大。因此，挑选特征时，条件基尼指数越小越好。

（2）CART 生成

算法步骤书本上有，本渣不再阐述了，在这里本渣阐述一下所发现的 CART 与 ID3/C4.5 的不同点。首先，CART 算法每一步迭代生成的是二叉树，而 ID3/C4.5 则不一定生成二叉树，依据特征取值个数的多少，ID3/C4.5 生成的一般是多枝的树。

第二点，由于生成的是二叉树，并且二叉树对应的两个子集的分配规则为：$D_1=\{(x,y)\in D|A(x)=a\}$，$D_2=D-D_1$，所以在 CART 算法中，（离散特征情形）特征 A 是有可能被重复利用的，特征 A 的取值越多，其被重复使用的可能性就越大。而在 ID3/C4.5 算法中（离散特征情形）特征 A 不会在迭代中重复被使用。对于 CART 算法中划分的 D_1 和 D_2，其中 D_1 恒有 $A=a$，因此对于 D_1 而言，特征 A 已经失去了划分能力，在对子集 D_1 进行下一步迭代时不再使用特征 A（仅仅使用剩余特征）；而对于子集 D_2 而言，如果 A 仅仅是二分类特征，则在子集 D_2 上 A 的值也是固定的，特征 A 失去划分能力，在对子集 D_2 进行下一步迭代时同样不再使用特征 A。但是，大部分情况下，A 是多分类特征，在子集 D_2 上 A 有多种取值可能，此时特征 A 仍然具有划分能力，因此，在 D_2 的下一步迭代中，仍然会使用到特征 A。所以，CART 算法中，特征 A，只要取值个数大于 2，就有被重复利用的可能。按照这样的想法，可以推测，CART 更加适应特征个数较少时（特征取值个数较多）的情况，而 ID3/C4.5 算法在特征个数较少时建立起来的树可能会过于简单。

为了求解例 5.4，我用 R 编写了函数 GiniCART()、GiniSingle() 和 GpSim(). 其中 GiniSingle() 和 GpSim() 是辅助函数，会在 GiniCART() 内部被调用，GiniSingle() 函数可以用来求一个特征所对应的基尼指数。GpSim() 在模拟熵和基尼指数的关系时使用过。（函数详细代码请参加附录 5）。

GiniCART(cls=NULL,atr=NULL,data=NULL) 函数可用来求某个样

本集单次 CART 迭代所生成的二叉树。多次使用 GiniCART()函数，可以求解简单 CART 分类树。现在，我们利用这个函数来求解例 5.4.

代码如下：

```
####  例 5.4 的程序求解  ####
cart1<-GiniCART(cls="Y",atr=c("A1","A2","A3","A4"),data=dataB5.2);cart1
cart2<-lapply(cart1[5:6],GiniCART,cls="Y",atr=c("A1","A2","A4"));cart2
```

结果如下：

```
> cart1[1:4]#部分结果
$Finalabel
[1] "A3" "否"

$FinalGini
        A3
0.2666667

$GiniMat
     A1   A2        A3   A4
[1,] 1.00 1.00 1.0000000 3.00
[2,] 0.44 0.32 0.2666667 0.32

$Ginilst
$Ginilst$A1
老年 青年 中年
0.44 0.44 0.48

$Ginilst$A2
  否   是
0.32 0.32

$Ginilst$A3
        否         是
0.2666667 0.2666667

$Ginilst$A4
   非常好        好       一般
0.3636364 0.4740741 0.3200000
```

```
> cart2[[1]]#部分结果
$Finalabel
[1] "A2" "否"

$FinalGini
A2
 0

$GiniMat
          A1 A2        A4
[1,] 1.0000000  1 3.0000000
[2,] 0.3333333  0 0.2666667

$Ginilst
$Ginilst$A1
      老年       青年       中年
0.3333333 0.4333333 0.3809524

$Ginilst$A2
否 是
 0  0

$Ginilst$A4
   非常好        好       一般
0.3333333 0.4000000 0.2666667
```

由上面的输出我们可以看到，所计算的基尼指数与最终的生成的树和书本的结果是一致的。在第一次迭代中挑选了特征 A_3 作为划分变量，A_3="否"作为划分点（由于 A_3 是二分类变量，使用 A_3="否"作为划分点与使用 A_3="是"作为划分点没有任何区别），在第二次迭代中挑选了特征 A_2 作为划分变量，A_2="否"作为划分点。但是，注意到，第二次迭代中 A_2 的基尼指数都为 0，这是为什么呢？是不是计算错误了？嗯，其实小伙伴们可以用 table()函数来验证一下这样的结果。

代码如下：

```
da1<-cart1[[5]]#提取第一次树的叶结点的样本实例集 D1
table(da1$A2,da1$Y)
```

结果如下：

```
> table(da1$A2,da1$Y)

    否 是
否  6  0
是  0  3
```

从以上结果可知，第二次迭代中，根据 A_2 划分样本集后，每一个子集的类属性都是唯一确定的，不具有不确定性，因此计算出来的基尼指数会为 0.

树 cart1 和 cart2 的结果还有更多的输出，小伙伴在 R 中查看完整的输出后，会发现，这里例 5.4 利用 CART 算法生成的树和 ID3/C4.5 生成的树是一样的。本渣就不在这里浪费篇幅展示结果了。

■

5.5.2 CART 剪枝（P72-P73）

关键词 7： 理解 CART 剪枝算法

说明 7：

其实这个剪枝算法渣君也还没完全理解，都已经看了那么多遍了啊？！无奈之下，本渣只能把自己理解了的那一部分写下来。算法内容小伙伴们可以查看书本，在此不再重复，仅先贴出几个重要公式。

具体地，从整体树T_0开始剪枝，对T_0的任意内部结点t，以t为单结点树的损失函数是：（也就是以t为叶结点）

$$C_\alpha(t)=C(t)+\alpha \quad \rightarrow SimpleTree$$

以t为根结点的子树T_t的损失函数是：（也就是内部结点t下还有内部结点和叶结点）

$$C_\alpha(T_t)=C(T_t)+\alpha|T_t| \quad \rightarrow ComplexTree$$

其中$|T_t|$表示t下叶结点的个数。

（1）当$\alpha=0$及α充分小时，有不等式：$C_\alpha(T_t)<C_\alpha(t)$

（2）当α增大时，在某一α有：

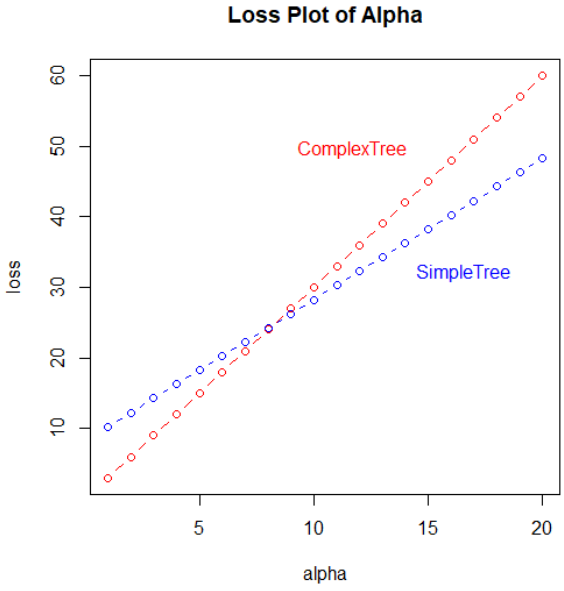
$$C_\alpha(T_t)=C_\alpha(t)\Rightarrow\alpha=\frac{C(t)-C(T_t)}{|T_t|-1}$$

（3）当α再增大时，有不等式：$C_\alpha(T_t)>C_\alpha(t)$.

只要满足$\alpha=\frac{C(t)-C(T_t)}{|T_t|-1}$，简单树$t$与复杂树$T_t$有相同的损失函数，而$t$的结点比$T_t$的结点少，因此$t$更可取，于是进行剪枝。

关键是如何理解上述三种情况，直观点理解就是，α表示对树复杂度的惩罚，当α较小时，复杂树更优（特别当$\alpha=0$时，整体树最优），

此时复杂的树会有更小的损失函数值，于是$C_\alpha(T_t)<C_\alpha(t)$；相反，当$\alpha$较大时，简单树更优（特别当$\alpha=+\infty$时，单结点树最优），这是简单树会有更小的损失函数值，于是$C_\alpha(T_t)>C_\alpha(t)$. 而且，比较容易看出，$C_\alpha(t)=C(t)+\alpha$是一条关于$\alpha$斜率为 1 的直线（$C(t)$关于$\alpha$为常数）；而$C_\alpha(T_t)=C(T_t)+\alpha|T_t|$则是一条关于$\alpha$斜率为$|T_t|$的直线（$C(T_t)$关于$\alpha$为常数）。我们知道$|T_t|>1, C_0(T_t)<C_0(t)$，因此，当$\alpha$增大时，必然在某一个$\alpha$使得，$C_\alpha(T_t)=C_\alpha(t)$；当越过这个界限时，就会有$C_\alpha(T_t)>C_\alpha(t)$；当小于这个界限时有$C_\alpha(T_t)<C_\alpha(t)$. 事实上，我们还可以用在“关键词 4”中所模拟的一幅关于简单树/复杂树损失函数值与α的关系的图来体现这种规律。如下：



嗯，这样看起来就一目了然了。理解了这一点之后，我们来看看究竟如何剪枝。书中说道，对T_0中每一内部结点t，计算：

$$\alpha_t=g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}$$

它表示剪枝后整体损失函数减少的程度。在T_0中减去$\alpha_t = g(t)$最小的T_t，将得到的最小的子树作为T_1，同时将最小的$\alpha_t = g(t)$记为α_1，T_1为区间$[\alpha_1, \alpha_2)$的最优子树。其实，渣君没有理解的有两点，其一，为什么$\alpha_t = g(t)$可以表示剪枝后整体损失函数减少的程度；其二是，为什么不是减去“损失函数减少程度”最大的反而是减去最小的呢？减去“最大”的，不是使得损失函数下降得最快吗？本渣觉得李航大牛的这一段写得有点含糊，实在是理解不透。于是，我尝试跳过第一点，直接去理解第二点，阐述如下。

我们知道，T_0中内部结点t个数不止一个，假设为T个，用$t_1, t_2, ..., t_T$表示不同的内部结点，而每个内部结点可以按照公式$g(t)$计算出对应的$\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tT}$。于是对于每一棵子树$T_{t1}, T_{t2}, ..., T_{tT}$有：

$$\alpha \geq \alpha_{t1} \Rightarrow \quad cut \quad T_{t1}$$
$$\alpha \geq \alpha_{t2} \Rightarrow \quad cut \quad T_{t2}$$
$$.....$$
$$\alpha \geq \alpha_{tT} \Rightarrow \quad cut \quad T_{tT}$$

可见，只要，α大于$\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tT}$中任意一个，都有必要减去其对应的子树。那么，究竟那一棵树应该最先被减去呢？我们先来看看书本描述的剪枝操作。Breiman等人证明：可以用递归的方法对树进行剪枝。将α从小增大，$0 = \alpha_0 < \alpha_1 < ... < \alpha_n < +\infty$，产生一系列相互嵌套的区间$[\alpha_i, \alpha_{i+1}), i = 1, 2, ..., n$剪枝得到的子树序列$\{T_1, T_2, ..., T_n\}$与$[\alpha_i, \alpha_{i+1})$一一对应。我们看到，剪枝的基本操作是令$\alpha$从小增大并在对应的区间进行剪枝，得到相应的子树。那么在α从小增大的过程中，首先要大于的必然是$\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tT}$中最小的那个，于是$\min\{\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tT}\}$所对应的树枝

是最先被剪掉的。所以，会有第二点中，“减去”$\alpha_t = g(t)$最小的树枝。

Ok，解决了上面这个问题之后，我们再来看一下其他的问题。书中提到，按照 CART 剪枝算法得到的一系列$\{\alpha_1,\alpha_2,...,\alpha_n\}$是从小到大的，而且每个区间$[\alpha_i,\alpha_{i+1}),i=1,2,...,n$内对应的最优子树$T_i$是唯一的。为什么会这样呢？渣君在此阐述一下自己的直观理解。首先，我们来理解为什么$\{\alpha_1,\alpha_2,...,\alpha_n\}$是从小到大的。

假设原始树是T_0，由原始树经过一次 CART 剪枝减掉内部结点t_0后得到子树T_1和α_1。然后再把T_1作为原始树，由T_1经过一次 CART 剪枝减掉内部结点t_1后得到子树T_2和α_2，同理，经过相同的步骤得到T_3和α_3.即：

$$T_0 \xrightarrow{cut\ t_0} T_1,\alpha_1;(\alpha_1,t_0)=\min_{\alpha,t}\{\alpha_{t1},\alpha_{t2},...,\alpha_{tr.}\}$$

$$SimpleTree: C_\alpha(t_0)=C(t_0)+\alpha; ComplexTree: C_\alpha(T_0)=C(T_0)+\alpha|T_0|$$

$$T_1 \xrightarrow{cut\ t_1} T_2,\alpha_2;(\alpha_2,t_1)=\min_{\alpha\notin\{\alpha_1\},t\notin\{t_1\}}\{\alpha_{t1},\alpha_{t2},...,\alpha_{tr.}\}$$

$$SimpleTree: C_\alpha(t_1)=C(t_1)+\alpha; ComplexTree: C_\alpha(T_1)=C(T_1)+\alpha|T_1|$$

$$T_2 \xrightarrow{cut\ t_2} T_3,\alpha_3;(\alpha_3,t_2)=\min_{\alpha\notin\{\alpha_1,\alpha_2\},t\notin\{t_1,t_2\}}\{\alpha_{t1},\alpha_{t2},...,\alpha_{tr.}\}$$

$$SimpleTree: C_\alpha(t_2)=C(t_2)+\alpha; ComplexTree: C_\alpha(T_2)=C(T_2)+\alpha|T_2|$$

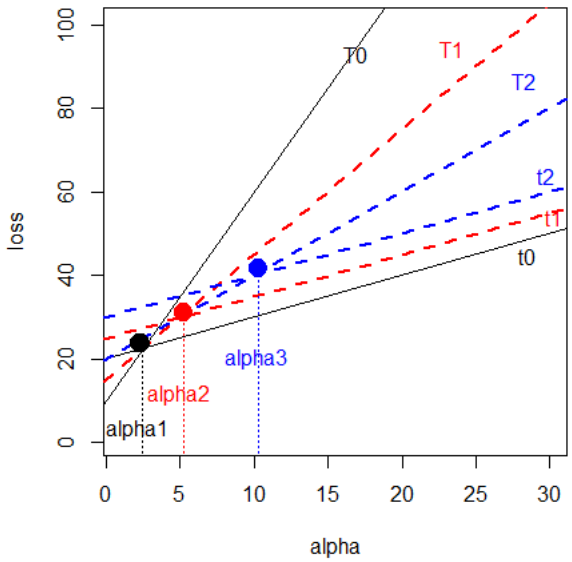
$$|T_0|>|T_1|>|T_2|,\ C(T_0)<C(T_1)<C(T_2), C(t_0)<C(t_1)<C(t_2)$$

我们知道，每一次剪枝后都会导致新的子树比上一次剪枝的子树的叶结点更少，因此$|T_0|>|T_1|>|T_2|$.同时由于剪枝后的树会越来越简单，其对应的训练误差则会越大越大，因此$C(T_0)<C(T_1)<C(T_2)$；而以t_0,t_1,t_2为单结点（叶结点）的树只是各自比T_0,T_1,T_2少一些内部结点，但是由于$|T_0|>|T_1|>|T_2|$，于是大致上有$|t_0|>|t_1|>|t_2|$，因此$C(t_0)<C(t_1)<C(t_2)$.又由于每次 CART 只是剪去一个结点，因此可以认为，$C(T_0)/C(T_1)/C(T_2)$

各自与$C(t_0)/C(t_1)/C(t_2)$差别不会太大（并且渣君在这里假设有$C(T_0)-C(T_1)\approx C(t_0)-C(t_1);C(T_1)-C(T_2)\approx C(t_1)-C(t_2)$）。为了直观地理解$\alpha_1,\alpha_2,\alpha_3$之间的大小关系，我们可以利用 R 把上述 3 种情况的复杂树/简单树的损失函数模拟到同一张图上面。代码如下。

```
plot(seq(1,30,length=100),1:100,type="n",xlab = "alpha",ylab="loss")
abline(a=10,b=5)#T1
abline(a=20,b=1)#t1
abline(a=15,b=3,col="red",lty=2,lwd=2)#T2
abline(a=25,b=1,col="red",lty=2,lwd=2)#t2
abline(a=20,b=2,col="blue",lty=2,lwd=2)#T3
abline(a=30,b=1,col="blue",lty=2,lwd=2)#t3
lines(c(2.475627, 2.475627),c(-5,22.97046),lty=3)#alpha1
lines(c(5.211,5.211),c(-5,29.33042),col="red",lty=3)#alpha2
lines(c(10.26247,10.26247),c(-5,42.05034),col="blue",lty=3)#alpha3
text(locator(),"alpha1")
text(locator(),"alpha2",col="red")
text(locator(),"alpha3",col="blue")
```

结果如图：



由上图可以看出，应该有，$\alpha_1<\alpha_2<\alpha_3$。当然，这种理解是相当粗糙的，严格的数学证明渣君就无能为力了，小伙伴伴们可以去尝试一下或者

查一查 Breiman 定理的证明。至于为什么在区间$[\alpha_i,\alpha_{i+1}),i=1,2,...,n.$内对应的最优子树是唯一的？我的理解是，举个例子，我们知道$[\alpha_1,\alpha_2)$区间对应的最优子树为T_1，当α开始从α_1向α_2增大时，在这个过程中，α大于α_1因此必然会剪枝产生T_1；但是又一直有α严格小于α_2，于是不会发生另一次剪枝，所以T_1是在区间$[\alpha_1,\alpha_2)$上是唯一的。

■

6.1.3 逻辑斯蒂回归模型的参数估计（P79）

关键词 1：　基于梯度下降法的极大似然估计及其 R 实现

说明 1：

书本中给出了需要极大化对数似然函数，如下：

$$L(w)=\sum_{i=1}^{N}\left[y_i(w\cdot x_i)-\log\left(1+\exp(w\cdot x_i)\right)\right]$$

$$-\infty<L(w)<0\Leftarrow\begin{cases}L(w)=\log\left(\prod_{i=1}^{N}\left[\pi(x_i)\right]^{y_i}\left[1-\pi(x_i)\right]^{1-y_i}\right)\\0\le\prod_{i=1}^{N}\left[\pi(x_i)\right]^{y_i}\left[1-\pi(x_i)\right]^{1-y_i}<1\end{cases}$$

我们来尝试利用梯度下降法求解逻辑斯蒂回归的参数向量，首先，令：

$$g(w)=-L(w)=\sum_{i=1}^{N}\left[\log\left(1+\exp(w\cdot x_i)\right)-y_i(w\cdot x_i)\right]$$

$$0<g(w)<+\infty$$

现在的目标是极小化$g(w)$，则有：

$$\nabla_w g(w)=\sum_{i=1}^{N}\left[\frac{\exp(w\cdot x_i)}{1+\exp(w\cdot x_i)}x_i-y_i\cdot x_i\right]=\sum_{i=1}^{N}\left[\pi(x_i)-y_i\right]\cdot x_i$$

$$\text{其中：}\ \pi(x_i)=\frac{\exp(w\cdot x_i)}{1+\exp(w\cdot x_i)},\ i=1,2,...,N.$$

写出求解极大似然估计的梯度下降算法：

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$，其中

$$x_i\in R^{n+1},x_i=\left(x_i^{(1)},x_i^{(2)},...,x_i^{(n)},1\right)^T,y_i\in\{0,1\},i=1,2,...,N;0<\eta$$

输出：$w=(w_1,w_2,...,w_n,b)$，逻辑斯蒂模型$p(Y=1|x)=\frac{\exp(w\cdot x_i)}{1+\exp(w\cdot x_i)}$

（1）指定初始参数：$w_0=0,\eta>0,\varepsilon>0$，计算

$$\pi_i^{(0)}=\frac{\exp\left(w^{(0)}\cdot x_i\right)}{1+\exp\left(w^{(0)}\cdot x_i\right)},i=1,2,...,N.$$

（2）对于$w^{(k)}$，计算$\nabla_w g\left(w^{(k)}\right)$，即：

$$\text{计算}\begin{cases}\pi_i^{(k)}=\dfrac{\exp\left(w^{(k)}\cdot x_i\right)}{1+\exp\left(w^{(k)}\cdot x_i\right)},i=1,2,...,N.\\ \nabla_w g\left(w^{(k)}\right)=\sum_{i=1}^{N}\left[\pi_i^{(k)}-y_i\right]\cdot x_i\end{cases}$$

若$\left\|\nabla_w g\left(w^{(k)}\right)\right\|<\varepsilon$，则停止，置$w=w^{(k)}$.否则，计算：

$$w^{(k+1)}\leftarrow w^{(k)}-\eta\cdot\nabla_w g\left(w^{(k)}\right)$$

若$\left\|g\left(w^{(k+1)}\right)-g\left(w^{(k)}\right)\right\|<\varepsilon$或$\left\|w^{(k+1)}-w^{(k)}\right\|<\varepsilon$，则停止，置$w=w^{(k+1)}$

（3）否则，置$k=k+1$，转（2）.

由以上算法的推导我们可知，迭代的参数结果w使得$g(w)$越接近0越优，越接近于极大化对数似然函数$L(w)$的目标，因为：

$$g(w)\to 0\Leftrightarrow L(w)\to 0\Rightarrow\prod_{i=1}^{N}\left[\pi(x_i)\right]^{y_i}\left[1-\pi(x_i)\right]^{1-y_i}\to 1$$

好了，小伙伴们脑补完数学后，是时候进入codetime阶段了。我利用 R 编写了函数 gradLogistic()、predict.gradLogistic()以及函数 print.gradLogistic()来实现逻辑斯蒂回归的基于梯度下降算法的极大似然参数估计。和之前一样，这次编写函数也是使用了简单的S3类。函数gradLogistic()用来求解模型参数（训练模型），predict.gradLogistic()进行预测及回测，print.gradLogistic()指定类“gradLogistic”的打印方式。（详细代码参见附录6）

```
gradLogistic(cls=NULL,atr=NULL,data=NULL,scale=TRUE,
```

w0=rep(0,length(atr)+1),aita=1,ept=1e-5,maxiter=100000)函数中 cls 指定二分类变量，atr 指定自变量，data 输入一个至少包括 cls 及 atr 指定变量数据的数据框；scale=TRUE 表示对自变量数据进行 0-1 标准化，默认值为 TRUE（标准化之后数据消除了量纲，提高了迭代效率）；w0 指定初始权重，默认为 0；aita 为学习步长，默认为 1，人为指定；ept 为阈值；maxiter 为最大迭代数，默认 100000 次。

好了，现在让我们来用 R 的内置数据集 mtcars 来测试一下这个算法。我们利用原始数据训练模型，并用 predict()进行回测。

我们来看第一个模型，代码如下：

```
####  测试模型 1 ####
dataB6.1<-mtcars;
dataB6.1_pred<-mtcars[,c("mpg","cyl","disp","hp")]#回测集
gradlog1<-gradLogistic(data=dataB6.1,cls="am",
                       atr=c("mpg","cyl","disp","hp"))#训练模型
predLog1<-predict(gradlog1,atr=c("mpg","cyl","disp","hp"),
                       atr_value = dataB6.1_pred)#模型回测
miss<-data.frame(newG=predLog1$FinalPredict,G=mtcars$am)
tbl1<-table(miss$newG,miss$G);tbl1;sum(diag(tbl1))/sum(tbl1)#正确率
```

训练模型的参数结果如下：

```
> gradlog1#模型 1 训练结果
The stoprule is :  abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept
iteration :  6603
formula :  am ~ mpg+cyl+disp+hp
$weight
          mpg          cyl         disp           hp            b
 52.67829541  20.77315732 -61.66076364  53.61224304  -7.30205294

$minusLogkplus1
[1] 0.0722225391

$minusLogk
[1] 0.0722325389
```

渣君先来解释一下这些输出的结果，“stoprule”是指迭代过程是在哪个条件的约束下停止的，这里：

abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept

指的是停止条件$\left\|g\left(w^{(k+1)}\right)-g\left(w^{(k)}\right)\right\|<\varepsilon$或$\left\|w^{(k+1)}-w^{(k)}\right\|<\varepsilon$.而停止条件$\left\|\nabla_w g\left(w^{(k)}\right)\right\|<\varepsilon$的“stoprule”输出为：sqrt(sum(gradfvec^2))<=ept

迭代上限条件的“stoprule”输出为：iterk>=maxiter.

iteration 指迭代次数，formula 指模型形式，weight 是参数结果；而 minusLogkplus1 和 minusLogk 则是指最后两次迭代中对应的$g(w)$（即$-L(w)$）的值。通过 names()函数可以查看 gradlog1 所有的组件，包括未在 console 台打印的。

现在我们来看一下结果，发现迭代结束后，$g(w)$的值比较接近于0，这是比较理想的，这说明概率$\prod_{i=1}^{N}\left[\pi(x_i)\right]^{y_i}\left[1-\pi(x_i)\right]^{1-y_i}$接近 1，这正是极大似然估计法想要的结果。然后，我们来看模型的回测效果。

```
> miss<-data.frame(newG=predLog1$FinalPredict,G=mtcars$am)
> tbl1<-table(miss$newG,miss$G);tbl1

   0  1
0 19  0
1  0 13
> sum(diag(tbl1))/sum(tbl1)#正确率
[1] 1
```

可见，模型的训练效果很好，回测准确率为 100%（这里就不讨论泛化能力啦，我估计可能是过拟合了）。小伙伴们还可以查看 predLog1 的所有结果，我就不在这里铺张了。

接下来，我们再利用数据集 mtcars 来尝试两个模型。我们先来改变一下自变量集及二分类变量，看看效果如何。代码如下：

```
#### 测试模型 2 ####
dataB6.1_pred2<-dataB6.1[,c("mpg","cyl","disp","hp","drat","wt","qsec")]#修改特征
gradlog2<-gradLogistic(data=dataB6.1,cls="vs", #更改分类变量
                       atr=c("mpg","cyl","disp","hp","drat","wt","qsec"))
predLog2<-predict(gradlog2,atr=c("mpg","cyl","disp","hp","drat","wt","qsec"),
                  atr_value = dataB6.1_pred2)
miss2<-data.frame(newG=predLog2$FinalPredict,G=mtcars$vs)
tbl2<-table(miss2$newG,miss2$G);tbl2
sum(diag(tbl2))/sum(tbl2)#正确率
```

模型训练结果：

```
> gradlog2
The stoprule is :  abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept
iteration :  626
formula :  vs ~ mpg+cyl+disp+hp+drat+wt+qsec
$weight
          mpg           cyl          disp            hp          drat            wt
 4.11828488 -4.37808447 -8.63872495 -5.42528123 -5.85690363  1.09865139
         qsec             b
22.33565325 -2.67325992

$minusLogkplus1
[1] 0.00868628451

$minusLogk
[1] 0.00869627844
```

回测结果：

```
> miss2<-data.frame(newG=predLog2$FinalPredict,G=mtcars$vs)
> tbl2<-table(miss2$newG,miss2$G);tbl2

   0  1
0 18  0
1  0 14
> sum(diag(tbl2))/sum(tbl2)#正确率
[1] 1
```

可见这次模型拟合结果也比较理想，而且收敛速度很快。

针对模型 2，我们随机删掉一些特征再训练一个新的模型，代码如下。

```
####   测试模型 3 ####
dataB6.1_pred3<-dataB6.1[,c("mpg","cyl","drat","wt")]#减少了一些特征
gradlog3<-gradLogistic(data=dataB6.1,cls="vs",
                       atr=c("mpg","cyl","drat","wt"))
predLog3<-predict(gradlog3,atr=c("mpg","cyl","drat","wt"),
                  atr_value = dataB6.1_pred3)
miss3<-data.frame(newG=predLog3$FinalPredict,G=mtcars$vs)
tbl3<-table(miss3$newG,miss3$G);tbl3
sum(diag(tbl3))/sum(tbl3)#正确率
```

模型训练结果：

```
The stoprule is :    iterk>=maxiter
iteration :    1e+05
formula :    vs ~ mpg+cyl+drat+wt
$weight
          mpg          cyl         drat           wt            b
 0.669383624 -8.849971436 -2.815718400  4.232913463 -0.912329789

$minusLogkplus1
[1] 12.2239643

$minusLogk
[1] 8.21384372
```

回测结果：

```
> miss3<-data.frame(newG=predLog3$FinalPredict,G=mtcars$vs)
> tbl3<-table(miss3$newG,miss3$G);tbl3

    0  1
  0 14  3
  1  4 11
> sum(diag(tbl3))/sum(tbl3)#正确率
[1] 0.78125
```

可能是由于不小心删除了重要特征的原因，对于同一个二分类变量“vs”，模型 3 的拟合效果显然不如模型 2，这里回测正确率只有 0.7815.

另一方面，随着迭代次数的增加，从算法上讲$g(w)$的值应该是渐次下降并不断逼近于 0 的。我们可以通过控制 gradLogistic()函数中的参数 maxiter 来实现这种模拟。代码如下。

```
####  迭代效果模拟：迭代 50/100/1000/1 万/10 万/100 万次 ####
gradLogistic(data=dataB6.1,cls="am",
                atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 50)
gradLogistic(data=dataB6.1,cls="am",
                atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 100)
gradLogistic(data=dataB6.1,cls="am",
                atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 1000)
gradLogistic(data=dataB6.1,cls="am",
               atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 10000)
gradLogistic(data=dataB6.1,cls="am",
               atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 100000)
gradLogistic(data=dataB6.1,cls="am",
               atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 1000000)
```

结果如下：（我删掉了一些不必要的展示结果）

```
> gradLogistic(data=dataB6.1,cls="am",
+                 atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 50)
$minusLogkplus1
[1] 7.24964021

$minusLogk
[1] 9.81320946
> gradLogistic(data=dataB6.1,cls="am",
+                 atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 100)
$minusLogkplus1
[1] 0.953077851

$minusLogk
[1] 1.03098142
> gradLogistic(data=dataB6.1,cls="am",
+                 atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 1000)
$minusLogkplus1
[1] 0.334450536

$minusLogk
[1] 0.334672513
```

```
> gradLogistic(data=dataB6.1,cls="am",
+                     atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 10000)
$minusLogkplus1
[1] 0.0491650871

$minusLogk
[1] 0.0491696979
> gradLogistic(data=dataB6.1,cls="am",
+                     atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 100000)
$minusLogkplus1
[1] 0.00524132108

$minusLogk
[1] 0.00524137308
> gradLogistic(data=dataB6.1,cls="am",
+                     atr=c("mpg","cyl","disp","hp"),ept = 1e-10,maxiter = 1000000)
$minusLogkplus1
[1] 0.0005276212

$minusLogk
[1] 0.0005276217
```

可见，随着迭代次数的增加，$g(w)$的值确实是渐次下降并不断逼近于 0 的。这符合了我们对梯度下降法的预期。通过梯度下降法求解的尝试就写到这里啦，接下来再写写怎么用拟牛顿法求解逻辑斯蒂回归的参数。

■

关键词 2： 基于拟牛顿法的极大似然估计及其 R 实现

说明 2:

参考书本附录 DFP 算法，我们把这个算法用在逻辑斯蒂回归的参数求解上。已知：

$$g(w)=-L(w)=\sum_{i=1}^{N}\left[\log\left(1+\exp\left(w\cdot x_i\right)\right)-y_i\left(w\cdot x_i\right)\right]$$

$$0<g(w)<+\infty$$

现在的目标是极小化$g(w)$，则有：

$$\nabla_w g(w)=\sum_{i=1}^{N}\left[\frac{\exp\left(w\cdot x_i\right)}{1+\exp\left(w\cdot x_i\right)}x_i-y_i\cdot x_i\right]=\sum_{i=1}^{N}\left[\pi\left(x_i\right)-y_i\right]\cdot x_i$$

$$\text{其中：}\ \pi\left(x_i\right)=\frac{\exp\left(w\cdot x_i\right)}{1+\exp\left(w\cdot x_i\right)},\ i=1,2,...,N.$$

写出 DFP 算法：

输入：训练数据集$T=\left\{\left(x_1,y_1\right),\left(x_2,y_2\right),...,\left(x_N,y_N\right)\right\}$，其中

$$x_i\in R^{n+1},x_i=\left(x_i^{(1)},x_i^{(2)},...,x_i^{(n)},1\right)^T,y_i\in\left\{0,1\right\},i=1,2,...,N;0<\eta$$

输出：$w=\left(w_1,w_2,...,w_n,b\right)$，逻辑斯蒂模型$p\left(Y=1\middle|x\right)=\frac{\exp\left(w\cdot x_i\right)}{1+\exp\left(w\cdot x_i\right)}$

（1）指定初始参数：$w_0=0,\eta>0,\varepsilon>0$，$G_0=I_{n+1}$,并计算

$$\pi_i^{(0)}=\frac{\exp\left(w^{(0)}\cdot x_i\right)}{1+\exp\left(w^{(0)}\cdot x_i\right)},i=1,2,...,N.$$

（2）对于$w^{(k)}$，计算$\nabla_w g\left(w^{(k)}\right)$，即：

$$\text{计算}\begin{cases}\pi_i^{(k)}=\dfrac{\exp\left(w^{(k)}\cdot x_i\right)}{1+\exp\left(w^{(k)}\cdot x_i\right)},i=1,2,...,N.\\ \nabla_w g\left(w^{(k)}\right)=\sum_{i=1}^{N}\left[\pi_i^{(k)}-y_i\right]\cdot x_i\end{cases}$$

若$\left\|\nabla_w g\left(w^{(k)}\right)\right\|<\varepsilon$，则停止，置$w=w^{(k)}$. 否则，计算：

$$w^{(k+1)} \leftarrow w^{(k)}-\eta \cdot G_k \nabla_w g\left(w^{(k)}\right)$$

$$\text{计算}\begin{cases} G_{k+1}=G_k+\dfrac{\delta_k \cdot \delta_k{}^T}{\delta_k{}^T \cdot y_k}-\dfrac{G_k y_k y_k{}^T G_k}{y_k{}^T G_k y_k} \\ y_k=\nabla_w g\left(w^{(k+1)}\right)-\nabla_w g\left(w^{(k)}\right) \\ \delta_k=w^{(k+1)}-w^{(k)} \end{cases}$$

（3）置$k=k+1$，转（2）.

（4）可考虑的附加停止条件：

若$\left\|g\left(w^{(k+1)}\right)-g\left(w^{(k)}\right)\right\|<\varepsilon$或$\left\|w^{(k+1)}-w^{(k)}\right\|<\varepsilon$，则停止，置$w=w^{(k+1)}$

由以上算法的推导我们可知，迭代的参数结果w使得$g(w)$越接近0越优，越接近于极大化对数似然函数$L(w)$的目标，因为：

$$g(w)\to 0 \Leftrightarrow L(w)\to 0 \Rightarrow \prod_{i=1}^{N}\left[\pi\left(x_i\right)\right]^{y_i}\left[1-\pi\left(x_i\right)\right]^{1-y_i}\to 1$$

另外，在编写程序时，我尝试了对学习率η使用基于牛顿法的一维搜索，推导过程如下。对于$w^{(k)}$，令：

$$\begin{aligned} H(\eta) &= g\left(w^{(k)}-\eta \cdot \Delta_k\right) \\ &= \sum_{i=1}^{N}\left\{\log\left(1+\exp\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]\right)-y\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]\right\}, \Delta_k=G_k \nabla_w g\left(w^{(k)}\right). \end{aligned}$$

则有：

$$H'(\eta)=\frac{dH(\eta)}{d\eta}=\sum_{i=1}^{N}\left\{y_i-\frac{\exp\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]}{1+\exp\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]}\right\} \cdot \Delta_k x_i$$

$$H''(\eta)=\frac{d^2 H(\eta)}{d\eta^2}=\sum_{i=1}^{N}\left\{\frac{\exp\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]}{\left(1+\exp\left[\left(w^{(k)}-\eta \cdot \Delta_k\right) \cdot x_i\right]\right)^2}\right\} \cdot \left(\Delta_k x_i\right)^2$$

于是可以建立迭代公式：

$$\eta^{(l+1)}=\eta^{(l)}-\frac{H'\left(\eta^{(l)}\right)}{H''\left(\eta^{(l)}\right)}$$

按照此公式，可求得 DFP 算法第k次迭代最终学习率：

$$\eta \leftarrow \max\left\{\eta_{search},\varepsilon\right\};\varepsilon>0.$$

好了，接下来就让我们使用 DFP 算法来实现逻辑斯蒂回归吧。渣君这次编的函数代码稍微长了一点，小伙伴们在看代码的时候可能需要些耐心了。其实代码还可以优化，但本渣是懒得改了，这是个牵一发而动全身的活。我编写了 DFPLogistic()、print.DFPLogistic()以及 predict.DFPLogistic()函数来实现 DFP 算法。其中 DFPLogistic()函数用于参数求解，predict.DFPLogistic()函数用于预测及回测，print.DFPLogistic()函数指定了类“DFPLogistic”的打印方式。（详细代码参见附录 6）

DFPLogistic(cls=NULL,atr=NULL,data=NULL,scale=TRUE, ept=1e-5,G0=diag(rep(1,length(atr)+1)),MoreStopRule=FALSE, w0=rep(0,length(atr)+1),aita=.1,maxiter=10000,SearchAita=F, maxsearch=1000)函数中 cls,atr,data，ept 含义和之前所有函数中的意义一样；scale=TRUE，默认对自变量数据进行 0-1 标准化;G0 指定初始正定矩阵，默认为单位矩阵；w0 指定初始权重，默认为 0；MoreStopRule=FALSE 默认不使用更多的停止条件，如果为 TRUE，则上述 DFP 算法中第（4）点提到的停止条件将会被使用，这样会加大迭代收敛的可能，但不能使得迭代的结果更优；aita 指学习率；

maxiter 为 DFP 算法默认的最大迭代次数；SearchAita=F 控制是否对学习率 aita 进行一维搜索，SearchAita=T 时，迭代中每一步的 aita 将会被重新计算，不再是常数；maxsearch=1000 指定一维搜索的最大迭代次数。

值得一提的是，拟牛顿法的收敛速度比较快，但是对初始值及初始正定矩阵比较敏感，有时候不一定能够收敛到最优解上，特别是原始数据自变量量纲差异过大的时候，不收敛的几率就会变大。因此最好先将数据标准化，消除量纲，这样可以提高算法的学习效率。本渣这次编写的预测函数是按照标准化的数据来预测的，因此，只有用 scale=TRUE 训练出来的模型才能与预测函数对应。另外，在编写函数时遇到了计算困难，由于计算的是指数数值 $\exp\left(w^{(k)}\cdot x_i\right)$，很容易就计算出超大的数，这时 R 无法处理这样的数，会默认将它强制转换为 Inf/-Inf/NaN.我在程序中设置了如果遇到这种情况，函数会自动重设初始值，重新进行迭代，最终结果会输出重设初始值的次数。数据标准化能够较好地避免这种情况的发生。Ok，现在我们利用 R 中的内置数据集 mtcars 及 iris 来测试一下这个函数。代码如下。

```
####  mtcars 数据集  模型 1：多停止条件  ####
dataDFPLog<-mtcars
dataDFPLog_pred<-mtcars[,c("mpg","cyl","disp","hp")]#回测自变量数据集
DFPLog1<-DFPLogistic(data=dataDFPLog,cls="am",
                     atr=c("mpg","cyl","disp","hp"),ept=1e-3,
                     maxiter = 10000,MoreStopRule = T);DFPLog1#训练模型
predDFPLog1<-predict(DFPLog1,atr=c("mpg","cyl","disp","hp"),
                     atr_value = dataDFPLog_pred)#模型回测
miss<-data.frame(newG=predDFPLog1$FinalPredict,G=mtcars$am)
tbl1<-table(miss$newG,miss$G);tbl1
sum(diag(tbl1))/sum(tbl1)#正确率
```

结果如下：

```
> DFPLog1
The stoprule is :   abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept
iteration :   96
formula :   am ~ mpg+cyl+disp+hp
$weight
            mpg             cyl            disp              hp               b
 77.68700489   29.71398805 -90.83788571   83.73133664   -8.79243159

$minusLogkplus1
[1] 0.00959671176

$minusLogk
[1] 0.0105517855

$LpGradf
[1] 0.00239348524

$changW
[1] 0

$changeG
[1] 0

$changeAita
[1] 0
```

上述输出的前几个的含义和 gradLogistic()中输出的含义是一样的，不重复了，仅解释一下其他几个。LpGradf 组件是指对应于最终的 weight，目标函数的梯度的范数值(2 范)，这个是越小越好；changW、changeG、changeAita 分别对应 w,G,aita 被函数自动重设的次数，如果初始值合适，重设次数就会为 0.现在来看看结果，这次使用多停止条件，迭代 96 次就收敛到较理想的结果了；我们来对比一下 gradLogistic()的“测试模型 1”，在同样的训练数据及停止条件下，使用梯度下降法的迭代次数是 6603.这么说来，拟牛顿法的效率真是

可以甩梯度下降法九条街了。

现在，我们来看看模型的回测结果。

```
> predDFPLog1<-predict(DFPLog1,atr=c("mpg","cyl","disp","hp"),
+                              atr_value = dataDFPLog_pred)#模型回测
> miss<-data.frame(newG=predDFPLog1$FinalPredict,G=mtcars$am)
> tbl1<-table(miss$newG,miss$G);tbl1

   0  1
0 19  0
1  0 13
> sum(diag(tbl1))/sum(tbl1)#回测正确率
[1] 1
```

可以看到，模型的回测结果是比较理想的，没有回测错误。接下来，我们再多看几个例子（这里就不再做回测了）。代码如下：

```
#### mtcars 数据集  模型 2：更改分类变量，单停止条件，进行一维搜索 ####
DFPLog2<-DFPLogistic(data=dataDFPLog,cls="vs",
                     atr=c("mpg","cyl","disp","hp"),ept=1e-3,
                     maxiter = 100000,MoreStopRule = F,SearchAita = T);DFPLog2
#### 对比：不进行一维搜索 ####
DFPLogistic(data=dataDFPLog,cls="vs",
            atr=c("mpg","cyl","disp","hp"),ept=1e-3,
            maxiter = 100000,MoreStopRule = F,SearchAita = F)#不进行一维搜索
```

结果如下：（仅展示部分）

```
> DFPLog2#进行一维搜索
The stoprule is :  sqrt(sum(gradfvec^2))<=ept
iteration :  115
formula :  vs ~ mpg+cyl+disp+hp
$weight
          mpg          cyl         disp           hp            b
-1.301272522 -2.930874042  0.884727024 -3.304966165 -1.318467479

$minusLogkplus1
[1] 7.4830148

$minusLogk
[1] 7.48301488

$LpGradf
[1] 0.000987232872
```

```
> DFPLogistic(data=dataDFPLog,cls="vs",#不进行一维搜索
+                     atr=c("mpg","cyl","disp","hp"),ept=1e-3,
+                     maxiter = 100000,MoreStopRule = F,SearchAita = F)
The stoprule is :   sqrt(sum(gradfvec^2))<=ept
iteration :  80
formula :   vs ~ mpg+cyl+disp+hp
$weight
            mpg             cyl            disp              hp               b
-1.300957715 -2.930247911   0.885196876 -3.304373286 -1.317871983

$minusLogkplus1
[1] 7.4830151

$minusLogk
[1] 7.48301527

$LpGradf
[1] 0.000947073443
```

可见两次的参数结果差异不大，一维搜索在这个例子中并没有提高收敛的效率。即使使用单停止条件，两次的迭代次数都很低，说明 DFP 算法的效率还是较高的。

对于数据集 mtcars 的尝试就到此为止了，这个数据集样本量太少，不好划分预测集。我们接下来利用 iris 数据集来进行测试，看一下基于 DFP 算法的逻辑斯蒂回归的预测效率。代码如下：

数据准备：（50 样本作为训练集，另外 50 个作为测试集）

```
#### iris 数据集  数据准备  ####
dataDFPLog_iris<-iris[1:100,]
dataDFPLog_iris$Species<-ifelse(dataDFPLog_iris$Species=="setosa",1,0)
trainlab<-sample(100,50)
dataDFPiris_train<-dataDFPLog_iris[trainlab,]#训练集
dataDFPiris_test<-dataDFPLog_iris[-trainlab,]#测试集
dataDFPiris_test_atr<-dataDFPiris_test[,-5]#测试特征集
```

模型代码：

```
#### iris 数据集   模型 1   ####
DFPLogiris1<-DFPLogistic(cls="Species",data=dataDFPiris_train,
                         atr=c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width"),
                         ept=1e-3,MoreStopRule = F,maxiter = 10000);DFPLogiris1
predDFPLogiris1<-predict(DFPLogiris1,atr_value  =  dataDFPiris_test_atr,# 模 型 预 测
                         atr=c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width"))
miss<-data.frame(newG=predDFPLogiris1$FinalPredict,G=dataDFPiris_test$Species)
tbl<-table(miss$newG,miss$G);tbl
sum(diag(tbl))/sum(tbl)#预测正确率
```

模型训练结果如下：

```
> DFPLogiris1
The stoprule is :   sqrt(sum(gradfvec^2))<=ept
iteration :   91
formula :   Species ~ Sepal.Length+Sepal.Width+Petal.Length+Petal.Width
$weight
  Sepal.Length     Sepal.Width    Petal.Length     Petal.Width                b
   6.195851206     0.915403312 -13.812358004 -11.340559547    -5.349128919

$minusLogkplus1
[1] 0.000418712313

$minusLogk
[1] 0.0004592794

$LpGradf
[1] 0.000980541667

$changW
[1] 0

$changeG
[1] 0

$changeAita
[1] 0
```

模型预测结果如下：

```
> predDFPLogiris1<-predict(DFPLogiris1,atr_value = dataDFPiris_test_atr,#模型预测
+                          atr=c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width"))
> miss<-data.frame(newG=predDFPLogiris1$FinalPredict,G=dataDFPiris_test$Species)
> tbl<-table(miss$newG,miss$G);tbl

   0  1
0 25  0
1  0 25
> sum(diag(tbl))/sum(tbl)#预测正确率
[1] 1
```

可见，模型的预测效率也是比较理想的。100%的预测准确率，我也是震惊了，吓得我赶紧再做了一个例子。代码如下：

```
#### iris 数据集  模型 3 ： 另外两类 ####
#数据准备
dataDFPLog_iris2<-iris[51:150,]
dataDFPLog_iris2$Species<-ifelse(dataDFPLog_iris2$Species=="versicolor",1,0)
trainlab2<-sample(1:100,50)
dataDFPiris_train2<-dataDFPLog_iris2[trainlab2,]#训练集
dataDFPiris_test2<-dataDFPLog_iris2[-trainlab2,]#测试集
dataDFPiris_test_atr2<-dataDFPiris_test2[,-5]#测试特征集

DFPLogiris2_1<-DFPLogistic(cls="Species",data=dataDFPiris_train2,#训练模型
                           atr=c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width"),
                              ept=1e-3,MoreStopRule = F,maxiter = 10000);DFPLogiris2_1
predDFPLogiris2_1<-predict(DFPLogiris2_1,
                           atr=c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width"),
                           atr_value = dataDFPiris_test_atr2)#模型预测
miss<-data.frame(newG=predDFPLogiris2_1$FinalPredict,G=dataDFPiris_test2$Species)
tbl1<-table(miss$newG,miss$G);tbl1
sum(diag(tbl1))/sum(tbl1)#预测正确率
```

这次，我们更换训练数据，换成 iris 数据集中的另外两类，并且减少一个自变量。现在我们来看看，效果如何。

模型训练结果：

```
> DFPLogiris2_1#训练模型
The stoprule is :  sqrt(sum(gradfvec^2))<=ept
iteration :  115
formula :  Species ~ Sepal.Length+Sepal.Width+Petal.Length+Petal.Width
$weight
Sepal.Length  Sepal.Width Petal.Length  Petal.Width                   b
 38.35718235    5.55639785 -86.02951808 -47.33838836 -19.99674837

$minusLogkplus1
[1] 0.00152466968

$minusLogk
[1] 0.00167710074

$LpGradf
[1] 0.000948014995
```

模型预测结果：

```
> miss<-data.frame(newG=predDFPLogiris2_1$FinalPredict,G=dataDFPiris_test2$Species)
> tbl1<-table(miss$newG,miss$G);tbl1

    0  1
  0 21  4
  1  0 25
> sum(diag(tbl1))/sum(tbl1)#正确率 0.92
[1] 0.92
```

预测准确率也有 0.92,总体而言，效果还是较为理想的。DFP 算法就写到这里了，有兴趣的小伙伴还可以利用拟牛顿算法中的 BFGS 算法来实现基于极大似然的逻辑斯蒂回归，思路过程与实现 DFP 算法差不多，应该不会太难。本渣就不再铺张笔墨了。

■

6.2.3 最大熵模型的学习（P83-P85）

请证明 1：

$$\frac{\partial L(p,w)}{\partial P(y|x)}=\sum_{x,y}\tilde{P}(x)\left(\log P(y|x)+1\right)-\sum_{y}w_0-\sum_{x,y}\left(\tilde{P}(x)\sum_{i=1}^{n}w_i f_i(x,y)\right)$$

$$=\sum_{x,y}\tilde{P}(x)\left(\log P(y|x)+1-w_0-\sum_{i=1}^{n}w_i f_i(x,y)\right)$$

证明 1：

这是由于：$\sum_{x,y}\tilde{P}(x)\cdot w_0=\sum_{x}\tilde{P}(x)\left(\sum_{y}w_0\right)=\sum_{y}w_0,\ \ \sum_{x}\tilde{P}(x)=1.$

■

请证明 2： $P_w(y|x)=\frac{1}{Z_w}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right),\ Z_w=\sum_{y}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)$

证明 2：

由于有$\sum_{y}P_w(y|x)=1$，于是：

$$P_w(y|x)=P_w(y|x)/1=P_w(y|x)\Big/\sum_{y}P_w(y|x)$$

$$=\frac{\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)\Big/\exp(1-w_0)}{\sum_{y}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)\Big/\exp(1-w_0)}$$

$$=\frac{\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)}{\sum_{y}\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)}=\frac{\exp\left(\sum_{i=1}^{n}w_i f_i(x,y)\right)}{Z_w(x)}$$

■

6.2.4 极大似然估计（P87）

请证明 3： $L_{\tilde{P}}(P_w)=\log\prod_{x,y}P(y|x)^{\tilde{P}(x,y)}=\sum_{x,y}\tilde{P}(x,y)\log P(y|x)$

证明 3：

这公式看第一眼，很自然啊，没毛病啊，对数似然函数不就是长这样的吗？为什么还要弄个证明呢。好吧，那我们来盯着 $\prod_{x,y}P(y|x)^{\tilde{P}(x,y)}$ 看第二眼，咦……真没问题啊…….

在渣君的记忆里，$\prod_{x,y}P(y|x)^{\tilde{P}(x,y)}$ 右上角标红的部分应该是一个频数才对，而不应该是一个频率 $\tilde{P}(x,y)$. 那么，这里是频率 $\tilde{P}(x,y)$ 的话有没有错呢？其实也没有错，因为 $\log\prod_{x,y}P(y|x)^{\tilde{P}(x,y)}$ 与对数似然函数是等价的。简单推导如下：

令 $v(x,y)$ 表示 (x,y) 出现的频数， N 表示样本的个数，则有：

$$\tilde{P}(x,y)=\frac{v(x,y)}{N}\Rightarrow v(x,y)=N\cdot\tilde{P}(x,y)$$

于是似然函数可以写成：

$$\prod_{x,y}P(y|x)^{v(x,y)}=\prod_{x,y}P(y|x)^{N\cdot\tilde{P}(x,y)}=\prod_{x,y}\left[P(y|x)^{\tilde{P}(x,y)}\right]^N$$

$$\Rightarrow\log\prod_{x,y}\left[P(y|x)^{\tilde{P}(x,y)}\right]^N=N\cdot\sum_{x,y}\tilde{P}(x,y)\log P(y|x)$$

$$\Rightarrow\arg\max_w\left(N\sum_{x,y}\tilde{P}(x,y)\log P_w(y|x)\right)\triangleq\arg\max_w\left(\sum_{x,y}\tilde{P}(x,y)\log P_w(y|x)\right)$$

$$\Rightarrow L_{\tilde{P}}(P_w)=\sum_{x,y}\tilde{P}(x,y)\log P(y|x)$$

■

6.3.1 改进的迭代尺度算法（P89-P91）

请证明 4： $\delta_i = \frac{1}{M}\log\frac{E_{\tilde{P}}(f_i)}{E_P(f_i)}$，$\delta_i^{(k+1)} = \delta_i^{(k)} - \frac{g\left(\delta_i^{(k)}\right)}{g'\left(\delta_i^{(k)}\right)}$

证明 4：

如果有 $f^{\#}(x,y) = M$ ，则：

$$\exp(\delta_i M)\cdot E_P(f_i) = E_{\tilde{P}}(f_i) \Rightarrow \delta_i = \frac{1}{M}\log\frac{E_{\tilde{P}}(f_i)}{E_P(f_i)}$$

如果 $f^{\#}(x,y)$不是常数，则：

$$g(\delta_i) = \exp(\delta_i M)\cdot E_P(f_i) - E_{\tilde{P}}(f_i) \Rightarrow find \quad \delta_i^{*} \ s.t. \ g(\delta_i) = 0$$

给定$\delta_i^{(k)}$， $g(\delta_i)$在$\delta_i^{(k)}$处一阶泰勒展开有：

$$g(\delta_i) = g\left(\delta_i^{(k)}\right) + g'\left(\delta_i^{(k)}\right)\left(\delta_i - \delta_i^{(k)}\right)$$

假定$\delta_i^{(k+1)}$使得$g\left(\delta_i^{(k+1)}\right) = 0$ ，则有：

$$g\left(\delta_i^{(k+1)}\right) = g\left(\delta_i^{(k)}\right) + g'\left(\delta_i^{(k)}\right)\left(\delta_i^{(k+1)} - \delta_i^{(k)}\right) = 0$$

$$\Rightarrow \delta_i^{(k+1)} = \delta_i^{(k)} - \frac{g\left(\delta_i^{(k)}\right)}{g'\left(\delta_i^{(k)}\right)}$$

小结：这次不打算编写函数实现最大熵模型，这是由于最大熵模型中的特征函数是不明确的，书本中仅仅强调特征函数是表征(x,y)满足某种条件时的取值，但没有指明这些条件具体是什么。也就是说根据不同的约束条件，最大熵模型会有所不同，因此只能根据实际情况编写不同的最大熵模型，可能没有所谓统一的最大熵模型。最大熵模型的实现在利用拟牛顿法时与逻辑斯蒂回归相似；但是最大熵模型可用于“有约束分类”的情况。

■

7.1.3 间隔最大化（P101）

关键词 1：　　最大间隔分离超平面的存在唯一性定理

说明 1：

这里主要写写书本唯一性证明过程中的几个关键的等式及不等式。在唯一性证明中假设优化问题存在两个最优解(w^*_1,b^*_1)和(w^*_2,b^*_2)。显然$\|w^*_1\|=\|w^*_2\|=c$，其中c是一个常数. 令：

$$w=\frac{w^*_1+w^*_2}{2},b=\frac{b^*_1+b^*_2}{2}$$

则易知(w,b)是优化的问题的可行解，从而有：

$$c\le\|w\|\le\frac{1}{2}\|w^*_1\|+\frac{1}{2}\|w^*_2\|=c$$

在这里，左边的不等号成立的原因是最小化$\frac{1}{2}\|w\|^2$与最小化$\|w\|$是等价的，而且$\|w^*_1\|=\|w^*_2\|=c$，因此c是$\|w\|$的一个下界，因此有$c\le\|w\|$；右边不等号的成立则是因为范数的三角不等式。因此有：

$$\left\|\frac{w^*_1+w^*_2}{2}\right\|=\|w\|=\frac{1}{2}\|w^*_1\|+\frac{1}{2}\|w^*_2\|\Rightarrow\|w^*_1+w^*_2\|=\|w^*_1\|+\|w^*_2\|$$

$$\Rightarrow w^*_1=\lambda w^*_2$$

当假设这里使用的是 2 范时，利用第二章感知机笔记中证明的一个结果，很容易证明上面的最后一个等式。在第二章，我们证明了，对于 L2 范数，有：

$$\|\gamma+\alpha\|^2=\|\gamma\|^2+2\gamma\bullet\alpha+\|\alpha\|^2,\gamma\in R^n,\alpha\in R^n,n\text{有限.}$$

因此有：

$$\|w^*_1+w^*_2\|=\|w^*_1\|+\|w^*_2\|\Rightarrow\|w^*_1+w^*_2\|^2=\left(\|w^*_1\|+\|w^*_2\|\right)^2$$
$$\Rightarrow\|w^*_1\|^2+\|w^*_2\|^2+2w^{*T}_1\cdot w^*_2=\|w^*_1\|^2+\|w^*_2\|^2+2\|w^*_1\|\cdot\|w^*_2\|$$
$$\Rightarrow w^{*T}_1\cdot w^*_2=\|w^*_1\|\cdot\|w^*_2\|\Rightarrow\|w^*_1\|\cdot\|w^*_2\|=w^{*T}_1\cdot w^*_2\le\|w^*_1\|\cdot\|w^*_2\|\cdot\cos\theta\le\|w^*_1\|\cdot\|w^*_2\|$$
$$\Rightarrow\cos\theta=1,\theta=\left(\widehat{w^*_1,w^*_2}\right),\theta=0\ \text{or}\pi\Rightarrow w^*_1=\lambda w^*_2,|\lambda|=1.$$

于是接下来易证，$w^*_1=w^*_2$，即$\lambda=1$. 因此w是唯一的。

由此可以把两个最优解写成$(w^*,b^*_1),(w^*,b^*_2)$，再证$b^*_1=b^*_2$. 假设x'_1和x'_2是集合$\{x_i|y_i=+1\}$中分别对应(w^*,b^*_1)和(w^*,b^*_2)使得问题不等式等号成立的点，x''_1和x''_2是集合$\{x_i|y_i=-1\}$分别对应(w^*,b^*_1)和(w^*,b^*_2)使得问题不等式等号成立的点。则有：

$$b^*_1=-\frac{1}{2}\left(w^*\cdot x'_1+w^*\cdot x''_1\right),\ b^*_2=-\frac{1}{2}\left(w^*\cdot x'_2+w^*\cdot x''_2\right)$$

这是由于，在$\{x_i|y_i=+1\}$中，x'_1使得：

$$y'_1\left(w^*\cdot x'_1+b^*_1\right)=1\Rightarrow\left(w^*\cdot x'_1+b^*_1\right)=y'_1=1$$

这是由于，在$\{x_i|y_i=-1\}$中，x''_1使得：

$$y''_1\left(w^*\cdot x''_1+b^*_1\right)=1\Rightarrow\left(w^*\cdot x''_1+b^*_1\right)=y''_1=-1$$

于是有：$\left(w^*\cdot x'_1+b^*_1\right)+\left(w^*\cdot x''_1+b^*_1\right)=0\Rightarrow b^*_1=-\frac{1}{2}\left(w^*\cdot x'_1+w^*\cdot x''_1\right)$

同理有：$\left(w^*\cdot x'_2+b^*_2\right)+\left(w^*\cdot x''_2+b^*_2\right)=0\Rightarrow b^*_2=-\frac{1}{2}\left(w^*\cdot x'_2+w^*\cdot x''_2\right)$

因此可得：$b^*_1-b^*_2=-\frac{1}{2}\left[w^*\left(x'_1-x'_2\right)+w^*\left(x''_1-x''_2\right)\right]$

又由于$(w^*,b^*_1),(w^*,b^*_2)$对应的分类器将数据集完全正确划分，因此：

对于(w^*,b^*_1)而言有：$w^*\cdot x'_2+b^*_1\ge1=w^*\cdot x'_1+b^*_1\Rightarrow w^*\cdot\left(x'_1-x'_2\right)\le0$

对于(w^*,b^*_2)而言有：$w^*\cdot x'_1+b^*_2\ge1=w^*\cdot x'_2+b^*_2\Rightarrow w^*\cdot\left(x'_1-x'_2\right)\ge0$

因此有：$w^*\cdot\left(x'_1-x'_2\right)=0$，同理有，$w^*\cdot\left(x''_1-x''_2\right)=0$.

所以，$b^*_1=b^*_2$.

■

7.1.4 学习的对偶算法（P104）

关键词 2： 理解 $w=\sum_{i=1}^{N}\alpha_i y_i x_i$

说明 2：

注意到 w 和 x_i 都是多维的，假设是 n 维。我们来把式 $w=\sum_{i=1}^{N}\alpha_i y_i x_i$ 展开，这样能更好理解 w 和 x_i 的关系。展开如下：（$\alpha_i y_i$ 是一个数）

$$w=\begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_n \end{pmatrix}=\sum_{i=1}^{N}(\alpha_i y_i)\begin{pmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \cdots \\ x_i^{(n)} \end{pmatrix}=\begin{pmatrix} \sum_{i=1}^{N}\alpha_i y_i x_i^{(1)} \\ \sum_{i=1}^{N}\alpha_i y_i x_i^{(2)} \\ \cdots \\ \sum_{i=1}^{N}\alpha_i y_i x_i^{(n)} \end{pmatrix}$$

可见 w 和 x_i 的累加关系是在每一分量上一一对应的。

■

7.2.3 支持向量（P113）

关键词 3： 软间隔 SVM 的支持向量

说明 3：

总结：在线性不可分的情况下，对应 $\alpha_i>0$ 的样本点为支持向量。

$$\begin{cases} (1)\alpha_i<C,\xi_i=0\Rightarrow y_i\left(w^*\cdot x_i+b^*\right)=1\Rightarrow \text{在支持边界上} \\ (2)\alpha_i=C,0<\xi_i<1\Rightarrow 1>y_i\left(w^*\cdot x_i+b^*\right)=1-\xi_i>0\Rightarrow \text{分类正确} \\ (3)\alpha_i=C,\xi_i=1\Rightarrow y_i\left(w^*\cdot x_i+b^*\right)=1-\xi_i=0\Rightarrow \text{在分类器上} \\ (4)\alpha_i=C,\xi_i>1\Rightarrow y_i\left(w^*\cdot x_i+b^*\right)=1-\xi_i<0\Rightarrow \text{错误分类} \end{cases}$$

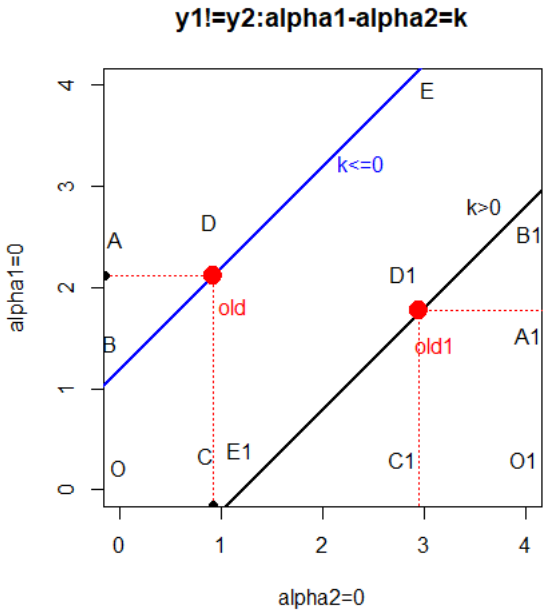
■

7.4 序列最小最优化算法（P126）

请证明 1：

$$\begin{cases} if \quad y_1^* \neq y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, C + \alpha_2^{*(k)} - \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} - \alpha_1^{*(k)}\right) \\ if \quad y_1^* = y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, \alpha_2^{*(k)} + \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} + \alpha_1^{*(k)} - C\right) \end{cases}$$

证明 1：



我们先来看看 $y_1 \neq y_2$ 的情形，上图纵轴表示 α_2 的取值大小，在最底部有 α_2=0；横轴表示 α_1 的取值大小，在最左边有 α_1=0. 忽略掉 R 画图的刻度，两条直线的斜率均是 1. 由于有 $\alpha_1 - \alpha_2 = k, \alpha_2 = \alpha_1 - k$. 设 $\alpha_2^{(new)}$ 为满足条件的可行点，即在上面的其中一条直线上； $\alpha_1^{(old)}, \alpha_2^{(old)}$ 为初始可行解，即上图标红的点对应的坐标值。为区别起见，这里将初始上限设为小写字母 c，则有：

$y_1 \neq y_2$：

$$(1)k>0 \Rightarrow \begin{cases} 0 \le \alpha_2^{(new)} \le O_1B_1 = A_1B_1 + A_1O_1 \\ A_1B_1 = O_1C_1, A_1O_1 = D_1C_1 \\ \alpha_2^{(old)} = D_1C_1, \alpha_1^{(old)} = OC_1 = c - O_1C_1 \end{cases}$$

$$\Rightarrow 0 \le \alpha_2^{(new)} \le c + \alpha_2^{(old)} - \alpha_1^{(old)}$$

$$(2)k \le 0 \Rightarrow \begin{cases} OB \le \alpha_2^{(new)} \le c, OB = AO - AB \\ AO = DC, AB = OC \\ \alpha_2^{(old)} = DC, \alpha_1^{(old)} = OC \end{cases}$$

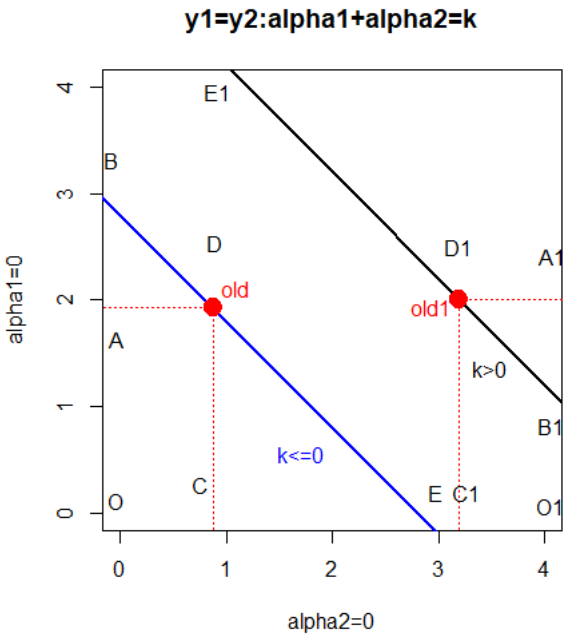
$$\Rightarrow \alpha_2^{(old)} - \alpha_1^{(old)} \le \alpha_2^{(new)} \le c$$

因此$y_1 \neq y_2$时有：

$$\max\left(0, \alpha_2^{(old)} - \alpha_1^{(old)}\right) \le \alpha_2^{(new)} \le \min\left(c, c + \alpha_2^{(old)} - \alpha_1^{(old)}\right)$$

$$\Rightarrow L = \max\left(0, \alpha_2^{(old)} - \alpha_1^{(old)}\right), H = \min\left(c, c + \alpha_2^{(old)} - \alpha_1^{(old)}\right)$$

同理$y_1 = y_2$时有：



忽略掉 R 画图的刻度，两条直线的斜率均是−1. $\alpha_1+\alpha_2=k, \alpha_2=-\alpha_1+k.$

$y_1 = y_2$：

$$(1)k > 0 \Rightarrow \begin{cases} O_1B_1 \le \alpha_2^{(new)} \le c, O_1B_1 = A_1O_1 - A_1B_1 \\ A_1B_1 = A_1D_1 = O_1C_1, A_1O_1 = D_1C_1 \\ \alpha_2^{(old)} = D_1C_1, \alpha_1^{(old)} = c - O_1C_1 \end{cases}$$

$$\Rightarrow \alpha_2^{(old)} + \alpha_1^{(old)} - c \le \alpha_2^{(new)} \le c$$

$$(2)k \le 0 \Rightarrow \begin{cases} 0 \le \alpha_2^{(new)} \le OB, OB = AO + AB \\ AO = DC, AB = AD = OC \\ \alpha_2^{(old)} = DC, \alpha_1^{(old)} = OC \end{cases}$$

$$\Rightarrow 0 \le \alpha_2^{(new)} \le \alpha_2^{(old)} + \alpha_1^{(old)}$$

因此 $y_1 = y_2$ 时有：

$$\max\left(0, \alpha_2^{(old)} + \alpha_1^{(old)} - c\right) \le \alpha_2^{(new)} \le \min\left(c, \alpha_2^{(old)} + \alpha_1^{(old)}\right)$$

$$\Rightarrow L = \max\left(0, \alpha_2^{(old)} + \alpha_1^{(old)} - c\right), H = \min\left(c, \alpha_2^{(old)} + \alpha_1^{(old)}\right)$$

综上所述便有：

$$\begin{cases} if \quad y_1^* \ne y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, C + \alpha_2^{*(k)} - \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} - \alpha_1^{*(k)}\right) \\ if \quad y_1^* = y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, \alpha_2^{*(k)} + \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} + \alpha_1^{*(k)} - C\right) \end{cases}$$

■

关键词 4： 基于 SMO 算法的支持向量机的 R 实现

说明 4：

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$ 及精度 ε.

其中 $x_i \in R^n, y_i \in y=\{-1,+1\}, i=1,2,...,N.$

输出：近似解 $\hat{\alpha}$.

（1）选取初始值 $\alpha^{(0)}=\left(\alpha_1^{(0)},\alpha_2^{(0)},...,\alpha_N^{(0)}\right)=0,\ b^{(0)}=0,\ \varepsilon>0, C>0$.

令 $k=0$，并选定合适的核函数 $K(x_i,x_j)$. C 越高，说明越不能容忍出现误差,容易过拟合。C 越小，容易欠拟合。

（2）选取优化变量 $\alpha_1^{*(k)},\alpha_2^{*(k)}$，进行以下计算并得到 $\alpha_1^{*(k+1)},\alpha_2^{*(k+1)}$，更新 α 为 $\alpha^{(k+1)}$.

计算步骤如下：

[1]变量选择的外层循环：

对于 $\alpha^{(k)}=\left(\alpha_1^{(k)},\alpha_2^{(k)},...,\alpha_N^{(k)}\right)$，计算:

$$\begin{cases} g(x_i)^{(k)}=\sum_{j=1}^{N}\alpha_j^{(k)}y_jK(x_i,x_j)+b^{(k)} \\ E^{(k)}=\left(E_1^{(k)},E_2^{(k)},...,E_N^{(k)}\right) \\ E_i^{(k)}=g(x_i)^{(k)}-y_i,\ i=1,2,...,N. \end{cases}$$

对于每一 (x_i,y_i) 检验是否满足条件：（在精度 ε 范围内）

$$\begin{cases} \alpha_i^{(k)}=0 \Leftrightarrow y_ig(x_i)^{(k)}\geq 1 \\ 0<\alpha_i^{(k)}<C \Leftrightarrow y_ig(x_i)^{(k)}=1 \\ \alpha_i^{(k)}=C \Leftrightarrow y_ig(x_i)^{(k)}\leq 1 \end{cases}$$

将所有不满足条件样本点对应的 $\alpha^{(k)}$ 的集合记为 M，则选取违反 *KKT* 条件最严重的样本点对应的 $\alpha^{(k)}$ 作为第一个优化变量 $\alpha_1^{*(k)}$，即：

$$\alpha_1^{*(k)} = \arg\max_{\alpha_i^{(k)} \in M} \left| y_i g\left(x_i\right)^{(k)} - 1 \right|$$

[2]变量选择的内层循环：

找到$\alpha_1^{*(k)}$对应的$E^{(k)}$即$E_1^{*(k)}$，则判断：

$$\begin{cases} if \quad E_1^{*(k)} > 0 \quad \Rightarrow \quad E_2^{*(k)} = \min E^{(k)} \\ if \quad E_1^{*(k)} \le 0 \quad \Rightarrow \quad E_2^{*(k)} = \max E^{(k)} \end{cases}$$

于是找到$E_2^{*(k)}$对应的$\alpha^{(k)}$作为$\alpha_2^{*(k)}$.

[3]参数更新：

找到$\alpha_1^{*(k)}$及$\alpha_2^{*(k)}$对其采用 SMO 法更新为$\alpha_1^{*(k+1)}, \alpha_2^{*(k+1)}$. 步骤如下：

1. 首先，找到$\alpha_1^{*(k)}$，$\alpha_2^{*(k)}$对应的y_1^*, y_2^*. 并计算下界及上界：

$$\begin{cases} if \quad y_1^* \ne y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, C + \alpha_2^{*(k)} - \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} - \alpha_1^{*(k)}\right) \\ if \quad y_1^* = y_2^* \quad \Rightarrow \quad H^{(k)} = \min\left(C, \alpha_2^{*(k)} + \alpha_1^{*(k)}\right), \mathrm{L}^{(k)} = \max\left(0, \alpha_2^{*(k)} + \alpha_1^{*(k)} - C\right) \end{cases}$$

2. 找到$\alpha_1^{*(k)}$，$\alpha_2^{*(k)}$对应的x_1^*, x_2^*，并计算：$\eta^{(k)} = K^*_{11} + K^*_{22} - 2K^*_{12}$.

3. 计算：

$$\alpha_2^{*(k+1).unc} = \alpha_2^{*(k)} + \frac{y_2^*\left(E_1^{*(k)} - E_2^{*(k)}\right)}{\eta^{(k)}}$$

4. 更新$\alpha_1^{*(k)}, \alpha_2^{*(k)}$为$\alpha_1^{*(k+1)}, \alpha_2^{*(k+1)}$：

$$\alpha_2^{*(k+1)} = \begin{cases} H^{(k)}, & \alpha_2^{*(k+1).unc} > H^{(k)} \\ \alpha_2^{*(k+1).unc}, & 0 \le \alpha_2^{*(k+1).unc} \le H^{(k)} \\ \mathrm{L}^{(k)}, & \alpha_2^{*(k+1).unc} < L^{(k)} \end{cases}$$

$$\alpha_1^{*(k+1)} = \alpha_1^{*(k)} + y_1^* y_2^*\left(\alpha_2^{*(k)} - \alpha_2^{*(k+1)}\right)$$

将α更新为$\alpha^{(k+1)} = \left(\alpha_1^{(k+1)}, \alpha_2^{(k+1)}, ..., \alpha_N^{(k+1)}\right)$.

5. 更新阈值$b^{(k)}$为$b^{(k+1)}$：

$$\begin{cases} if\ \ 0<\alpha_1^{*(k+1)}<C \ \Rightarrow\ b^{(k+1)}=b_1^{(k+1)} \\ \qquad\qquad =-E_1^{*(k)}-y_1^{*}K^{*}_{11}\left(\alpha_1^{*(k+1)}-\alpha_1^{*(k)}\right)-y_2^{*}K^{*}_{21}\left(\alpha_2^{*(k+1)}-\alpha_2^{*(k)}\right)+b^{(k)} \\ if\ \ 0<\alpha_2^{*(k+1)}<C \ \Rightarrow\ b^{(k+1)}=b_2^{(k+1)} \\ \qquad\qquad =-E_2^{*(k)}-y_1^{*}K^{*}_{12}\left(\alpha_1^{*(k+1)}-\alpha_1^{*(k)}\right)-y_2^{*}K^{*}_{22}\left(\alpha_2^{*(k+1)}-\alpha_2^{*(k)}\right)+b^{(k)} \\ if\ \ 0<\alpha_1^{*(k+1)},\alpha_2^{*(k+1)}<C \ \Rightarrow\ b^{(k+1)}=b_1^{(k+1)}=b_2^{(k+1)} \\ if\ \ \alpha_1^{*(k+1)},\alpha_2^{*(k+1)}=0 \text{ or } C \ \Rightarrow\ b^{(k+1)}=\dfrac{b_1^{(k+1)}+b_2^{(k+1)}}{2} \end{cases}$$

6. 更新$E^{(k)}$为$E^{(k+1)}$，计算：

$$\begin{cases} g\left(x_i\right)^{(k+1)}=\sum_{j=1}^{N}\alpha_j^{(k+1)}y_jK\left(x_i,x_j\right)+b^{(k+1)} \\ E^{(k+1)}=\left(E_1^{(k+1)},E_2^{(k+1)},...,E_N^{(k+1)}\right) \\ E_i^{(k+1)}=g\left(x_i\right)^{(k+1)}-y_i,\ i=1,2,...,N. \end{cases}$$

（3）对$\alpha=\alpha^{(k+1)}=\left(\alpha_1^{(k+1)},\alpha_2^{(k+1)},...,\alpha_N^{(k+1)}\right)$及每一$\left(x_i,y_i\right)$检验是否满足以下停止条件：（在精度$\varepsilon$范围内）

$$\sum_{i=1}^{N}\alpha_i^{(k+1)}y_i=0$$

$$\forall i,\ 0\le\alpha_i^{(k+1)}\le C,\ \ i=1,2,...,N.$$

$$\begin{cases} y_ig\left(x_i\right)^{(k+1)}\ge 1, & \alpha_i^{(k+1)}=0 \\ y_ig\left(x_i\right)^{(k+1)}=1, & 0<\alpha_i^{(k+1)}<C \\ y_ig\left(x_i\right)^{(k+1)}\le 1, & \alpha_i^{(k+1)}=C \end{cases}$$

$$g\left(x_i\right)^{(k+1)}=E_i^{(k+1)}+y_i,\ \ i=1,2,...,N.$$

若满足，则停止，置$\hat{\alpha}=\alpha^{(k+1)}$；否则，置$k=k+1$，转（2）.

（4）获得结果$\hat{\alpha}=\alpha^{(k+1)}$后计算：（线性分类器情形）

$$\hat{w}=\sum_{i=1}^{N}\hat{\alpha}_iy_ix_i$$

$$\hat{b}=y_j-\sum_{i=1}^{N}y_i\hat{\alpha}_i\left(x_i\cdot x_j\right),\ 0<\hat{\alpha}_j<C.$$

（5）工作集选择算法及停机准则的补充：

由于《统计学习方法》上写的关于工作集的选取方法并非十分详细，上面（2）中的[1]和[2]中描述的工作变量的选择方法是我根据书本的思路“想象”出来的。我在程序中也实现了这种方法，但效果不太稳定，时好时坏。为了弥补这个缺陷，我查阅了邓乃扬老师写的《支持向量机》一书，并根据书中P166 页介绍的工作集选择算法及 P159 页的停机准则，实现了基于 SMO 算法的支持向量机。算法及停机准则介绍如下：

①设当前问题的近似解为$\alpha^{(k)}$，计算目标函数

$$d(\alpha)=\frac{1}{2}\alpha^T H\alpha-e^T\alpha$$

在$\alpha^{(k)}$处的梯度：

$$\nabla d\left(\alpha^{(k)}\right)=H\alpha^{(k)}-e,\ H=\left(y_i y_j K\left(x_i,x_j\right)\right)_{N\times N},\ e=(1,1,...,1)^T.$$

②计算：

$$\nabla d\left(\alpha^{(k)}\right)=H\alpha^{(k)}-e,\ H=\left(y_i y_j K\left(x_i,x_j\right)\right)_{N\times N},\ e=(1,1,...,1)^T.$$

$$i=\arg\max_t\left\{-y_t\left[\nabla d\left(\alpha^{(k)}\right)\right]_t\middle|t\in I_{up}\left(\alpha^{(k)}\right)\right\},\ I_{up}(\alpha)=\left\{t\middle|\alpha_t<C,y_t=1\text{或}\alpha_t>0,y_t=-1\right\}.$$

$$j=\arg\max_t\left\{-y_t\left[\nabla d\left(\alpha^{(k)}\right)\right]_t\middle|t\in I_{low}\left(\alpha^{(k)}\right)\right\},I_{low}(\alpha)=\left\{t\middle|\alpha_t<C,y_t=-1\text{或}\alpha_t>0,y_t=1\right\}.$$

$$m(\alpha^*)=\max\left\{-y_t\left[\nabla d\left(\alpha^{(k)}\right)\right]_t\middle|t\in I_{up}\left(\alpha^{(k)}\right)\right\},M(\alpha^*)=\left\{-y_t\left[\nabla d\left(\alpha^{(k)}\right)\right]_t\middle|t\in I_{low}\left(\alpha^{(k)}\right)\right\}.$$

$\left[\nabla d\left(\alpha^{(k)}\right)\right]_t$指$\nabla d\left(\alpha^{(k)}\right)$的第$t$个分量.

所选取的工作变量集为：$B=\{i,\mathrm{j}\}$

停机准则为：$m(\alpha^*)-M(\alpha^*)\le\varepsilon$

好了，讲完算法了，渣君接下来就写一写算法的 R 语言实现吧。这次也是按照简单 S3 类的方法来编写函数的，模型输出结果的 class 属性设置为“smoSVM”.这次，本渣针对基于 SMO 算法的支持向量机所编写的函数包括了 lineKernel()、polyKernel()、gaussiKernel()、gfunc()、findAlpha()、SelectFunc()、smoSVM()、print.smoSVM()以及 predict.smoSVM()函数。函数 lineKernel()、polyKernel()、gaussiKernel()是辅助函数，分别负责计算线性核/多项式核/高斯核.gfunc()是辅助函数是辅助函数，负责计算$g(x_i)^{(k)}$；findAlpha()是辅助函数，负责按照（2）的方法搜索工作变量；SelectFunc()也是辅助函数，负责按照（5）的补充方法搜索工作变量。以上所有辅助函数均在 smoSVM()内部被调用。print.smoSVM()函数指定类“smoSVM”的打印方式；predict.smoSVM()函数负责回测及预测。（详细代码参见附录 7）

smoSVM(cls,atr,data,Kernel=c("line","poly","gaussi"),scale=T,C=10,ept=1e-2,alpha0=rep(0,nrow(data)),b0=0,p=2,Lp=2,lmada=sqrt(ncol(data))/sqrt(2),maxiter=10000,Change=T)函数中 cls 指定二分类变量，分类取值必须属于$\{-1,+1\}$，atr 指定训练特征；data 为至少包括 cls、atr 指定的变量数据的数据框；Kernel 指定核函数的类型，这里提供三种类型，即线性核/多项式核/高斯核；alpha0,b0 均为初始值，默认为 0;p=2，当核函数为多项式核”poly”时，p 用于指定多项式的次幂；lmada 用于指定高斯径向基核函数“gaussi”的宽度；Lp 指定高斯径向基核函数所使用的 Lp 范数，默

认为 2 范；scale 指定是否自变量归一化；Change=T,默认选用（5）所介绍的工作变量选择方法及停机准则，Change=F 时，则使用（2）所描述的方法及停机准则。

predict.smoSVM(SVMobj,cls,atr,atr_value,scale=T)函数中各参数的含义与 smoSVM()中基本相同；区别是，atr_value 仅包括 atr 中指定特征的数据，而不包括 cls 及其他特征的数据。scale=T，默认对特征数据进行归一化，与 smoSVM()中的 scale 对应，如果模型训练中设置了 scale=T，则在预测时也要设置 scale=T，反之亦然。

与以往一样，我们先用 R 的内置数据集 mtcars 来测试一下函数 smoSVM().代码如下。

数据准备：

```
####  函数测试  ####
####  mtcars 数据集测试  ####
datasvm_mtcars<-mtcars[,c(1,3,4,5,6,7,9)]#训练集
rownames(datasvm_mtcars)<-NULL
datasvm_mtcars$am<-ifelse(datasvm_mtcars$am==0,-1,1)
datasvm_mtcars_pred<-datasvm_mtcars[,c("mpg","disp","hp","qsec","drat")]#回测特征集
```

线性核模型代码：

```
#### mtcars 模型 1 线性核函数 ####
svm1<-smoSVM(cls="am",atr=c("mpg","disp","hp","qsec","drat"),data=datasvm_mtcars,
             Kernel = "line",maxiter = 100000,ept=1e-2,C=10,
             scale = T,Change = T);svm1#训练模型
pmt1<-predict(svm1,atr=c("mpg","disp","hp","qsec","drat"),cls="am",
        atr_value =datasvm_mtcars_pred,scale = TRUE );pmt1#模型回测
tblmt1<-table(datasvm_mtcars$am,pmt1);tblmt1
sum(diag(tblmt1))/sum(tblmt1)#回测正确率
```

线性核模型结果如下：

```
> svm1
The stoprule is :  (malp-Malp)<=ept
iteration :  136
formula :  am ~ mpg+disp+hp+qsec+drat
$nonZeroAlpha
          3           8          10          29          31          32
 3.6485064   0.3303682 10.0000000   0.5089418   0.8263335   5.3465864

$bMean
[1] 0.9521375

$support
[1]  3  8 10 29 31 32

$w
       mpg        disp          hp        qsec        drat
 2.982968 -2.066541   1.207484 -1.577009   1.262658
```

上述结果中，nonZeroAlpha 组件是指非零的$\alpha^{(k)}$值，bMean 指通过所有支持向量所计算出来的 b 的均值；support 组件指训练模型的支持向量及支持超平面内部的点；w 组件只有在线性核函数情形才会计算，是指决策函数的权重向量。我们现在来看看结果，算法在第 136 次收敛，速度较快；而且计算出来的$\alpha^{(k)}$各个值的大小也在要求范围内。接下来，我们来看一下回测的效果如何。

```
> tblmt1<-table(datasvm_mtcars$am,pmt1);tblmt1
    pmt1
     -1  1
  -1 15  4
  1   0 13
> sum(diag(tblmt1))/sum(tblmt1)#回测正确率
[1] 0.875
```

可见，这次有 4 个样本点被误分，回测正确率为 0.875,对训练集的训练效果尚算可以。渣君猜测 mtcars 是非线性可分的，接下来尝

试一下使用高斯径向基核函数时的分类效果。代码如下。

高斯核模型代码：

```
#### mtcars 模型 2 高斯径向基核函数 ####
svm2<-smoSVM(cls="am",atr=c("mpg","disp","hp","qsec","drat"),
                    data=datasvm_mtcars,Kernel = "gaussi",maxiter = 100000,
                    ept=1e-2,C=10,scale = T,Change = T);svm2#训练模型
pmt2<-predict(svm2,atr=c("mpg","disp","hp","qsec","drat"),cls="am",
                  atr_value =datasvm_mtcars_pred,scale = TRUE );pmt2#模型回测
tblmt2<-table(datasvm_mtcars$am,pmt2);tblmt2
sum(diag(tblmt2))/sum(tblmt2)#回测正确率
```

高斯核模型训练结果：

```
> svm2
The stoprule is :  (malp-Malp)<=ept
iteration :  58
formula :  am ~ mpg+disp+hp+qsec+drat
$nonZeroAlpha
             2              3              4              8             10
 5.633438e-02   1.000000e+01  -2.220446e-16   5.754790e+00   1.000000e+01
            21             24             29             30             31
 2.820065e+00   6.183411e+00   4.258212e+00   1.571536e+00   2.162442e+00
            32
 6.709741e+00

$bMean
[1] 1.254318

$support
 [1]  2  3  4  8 10 21 24 29 30 31 32
```

恩，很明显，这次使用高斯核函数后，算法收敛的速度更快了。

那么模型的分类效果如何呢，我们来看一下。

```
> tblmt2<-table(datasvm_mtcars$am,pmt2);tblmt2
    pmt2
     -1  1
  -1 18  1
  1   0 13
> sum(diag(tblmt2))/sum(tblmt2)#回测正确率
[1] 0.96875
```

不错不错，看来使用高斯核函数后，分类的正确率得到了有效的提高，这次回测正确率有 0.96875,只有一例样本点被误分。那么，我们接下来再试试使用多项式核函数时，smoSVM()函数分类效率会如何吧。代码如下。

多项式核模型代码：

```
#### mtcars 模型 3  多项式核函数  ####
svm3<-smoSVM(cls="am",atr=c("mpg","disp","hp","qsec","drat"),
                    data=datasvm_mtcars,Kernel = "poly",maxiter = 100000,
                    ept=1e-2,C=10,p=3,scale = T,Change = T);svm3#训练模型
pmt3<-predict(svm3,atr=c("mpg","disp","hp","qsec","drat"),cls="am",
                  atr_value =datasvm_mtcars_pred,scale = TRUE );pmt3#模型回测
tblmt3<-table(datasvm_mtcars$am,pmt3);tblmt3
sum(diag(tblmt3))/sum(tblmt3)#回测正确率
```

多项式核模型训练结果：

```
> svm3
The stoprule is :  (malp-Malp)<=ept
iteration :  128
formula :  am ~ mpg+disp+hp+qsec+drat
$nonZeroAlpha
             2              3              8             10             24
0.2552971325 0.0234885910 0.0203803604 0.4262119173 0.0056350613
            29             30             31             32
0.0019355586 0.0291209993 0.0001642405 0.1422208171

$bMean
[1] 0.01755172

$support
[1]  2  3  8 10 24 29 30 31 32
```

多项式核模型回测结果：

```
> tblmt3<-table(datasvm_mtcars$am,pmt3);tblmt3
    pmt3
     -1  1
  -1 16  3
  1   0 13
> sum(diag(tblmt3))/sum(tblmt3)#回测正确率
[1] 0.90625
```

嗯，回测正确率为 0.90625，也还可以。

最后，我们来看一下利用 smoSVM()函数对 R 内置数据集 iris 来做预测时的效果。代码如下。（我们选用前两类的 70 个样本作为训练集，余下 30 个样本作为测试集）

数据准备：

```
####  iris 数据集测试  ####
datasvmiris<-iris[1:100,]
datasvmiris$Species<-ifelse(datasvmiris$Species=="setosa",1,-1)
trainlab<-sample(1:100,70)
datasvmiris_train<-datasvmiris[trainlab,]#训练集
datasvmiris_test<-datasvmiris[-trainlab,]#测试集
datasvmiris_test_atr<-datasvmiris_test[,-5]#测试特征集
```

线性核模型代码：

```
#### iris 模型 1  线性核函数  ####
svmiris1<-smoSVM(cls="Species",atr=c("Sepal.Length","Sepal.Width"),
                 data=datasvmiris,Kernel = "line",maxiter = 10000,
                 ept=1e-2,C=5,scale = T,Change = TRUE);svmiris1#训练模型
piris1<-predict(svmiris1,atr=c("Sepal.Length","Sepal.Width"),cls="Species",
                  atr_value =datasvmiris_test_atr[,1:2],scale = TRUE );piris1#模型预测

tbliris1<-table(datasvmiris_test$Species,piris1);tbliris1
sum(diag(tbliris1))/sum(tbliris1)#预测正确率
```

线性核模型的训练结果：

```
> svmiris1
The stoprule is :  (malp-Malp)<=ept
iteration :  69
formula :  Species ~ Sepal.Length+Sepal.Width
$nonZeroAlpha
        37         42         58         85
4.120313 5.000000 4.120313 5.000000

$bMean
[1] -3.636189

$support
[1] 37 42 58 85

$w
Sepal.Length  Sepal.Width
   -3.160071     2.156382
```

线性核模型的预测结果：

```
> tbliris1<-table(datasvmiris_test$Species,piris1);tbliris1
     piris1
      -1  1
  -1 20  0
  1   2  8
> sum(diag(tbliris1))/sum(tbliris1)#预测正确率
[1] 0.9333333
```

可见，只有两例预测错误，预测效果比较理想。

如果换成高斯核函数，预测效果又会怎样呢？代码如下。

```
#### iris 模型 2 高斯径向基核函数 ####
svmiris2<-smoSVM(cls="Species",atr=c("Sepal.Length","Sepal.Width"),
                 data=datasvmiris,Kernel = "gaussi",maxiter = 10000,
                 ept=1e-2,C=5,scale = T,Change = TRUE,Lp=3);svmiris2
piris2<-predict(svmiris2,atr=c("Sepal.Length","Sepal.Width"),cls="Species",
                atr_value =datasvmiris_test_atr[,1:2],scale = TRUE );piris2#模型预测
tbliris2<-table(datasvmiris_test$Species,piris2);tbliris2
sum(diag(tbliris2))/sum(tbliris2)#预测正确率
```

高斯核模型训练结果：

```
> svmiris2
The stoprule is :  (malp-Malp)<=ept
iteration :  41
formula :  Species ~ Sepal.Length+Sepal.Width
$nonZeroAlpha
        21         26         37         42         58         60         61
0.4197886 5.0000000 5.0000000 5.0000000 5.0000000 0.8976318 0.7577815
        85         86
5.0000000 3.7643754

$bMean
[1] -1.685199

$support
[1] 21 26 37 42 58 60 61 85 86
```

高斯核模型预测结果：

```
> tbliris2<-table(datasvmiris_test$Species,piris2);tbliris2
     piris2
      -1  1
  -1 20  0
  1   4  6
> sum(diag(tbliris2))/sum(tbliris2)#预测正确率
[1] 0.8666667
```

这一次有 4 例样本被预测错误，预测效果比使用线性核时要差一点。这可能是由于初始参数的设置问题，也可能由于所使用的数据集是近似线性可分的，因此使用线性核的效果要更好。

渣君的支持向量机笔记就到此为止啦，对 SVM 十分感兴趣的小伙伴可以看看邓乃扬老师写的《支持向量机》一书，真心推荐，个人认为是一本好书。

■

8.1.2 Adaboost 算法（P139）

关键词 1：　　理解 $\alpha_m=\frac{1}{2}\log\frac{1-e_m}{e_m}$

说明 1：

有：　　$\frac{\partial\alpha_m}{\partial e_m}=-\frac{1}{2e_m(1-e_m)}<0$

当 $0<e_m\le\frac{1}{2}$ 时，$\alpha_m\ge 0$，且随着 $e_m\uparrow$，$\alpha_m\downarrow$.

当 $\frac{1}{2}<e_m\le 1$ 时，$\alpha_m<0$，且随着 $e_m\uparrow$，$\alpha_m\downarrow,|\alpha_m|\uparrow$.

分类器的线性组合为：$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x)$

最终分类器为：$G(x)=sign(f(x))=sign\left(\sum_{m=1}^{M}\alpha_m G_m(x)\right)$

当 $\frac{1}{2}<e_m\le 1$ 时，$\alpha_m<0$，令 $a_m=-\alpha_m>0$，$\alpha_m G_m(x)=-a_m G_m(x)$. 对于一个二分类问题而言，这相当于将原来 e_m 过高的模型 $G_m(x)$ 反过来用，即使用 $-G_m(x)$. $G_m(x)$ 的误分率 $e_m>\frac{1}{2}$，这意味着 $-G_m(x)$ 的误分率小于 $\frac{1}{2}$，并且当 $e_m>\frac{1}{2}$ 越大时，$-G_m(x)$ 的误分率 $1-e_m$ 就越小，模型 $-G_m(x)$ 的权重 $a_m=-\alpha_m=|\alpha_m|$ 就越大。这时，$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x)$ 依然符合设计的要求。

极端情况下，$e_m=0$，则 $G_m(x)$ 本身就是一个强分类器，Adaboost 算法可以考虑输出 $G_m(x)$ 作为最终的模型。若 $e_m=1$，则 $G_m(x)$ 是完全无效率的分类器，但 $-G_m(x)$ 本身是一个强分类器，因此，这时 Adaboost 算法可以考虑输出 $-G_m(x)$ 作为最终的模型。

■

8.2 AdaBoost 算法的训练误差分析（P142-P145）

请证明 1： $$\prod_{m=1}^{M}\sqrt{\left(1-4\gamma_m{}^2\right)}\le\exp\left(-2\sum_{m=1}^{M}\gamma_m{}^2\right)$$

证明 1：

对函数 $f(x)$ 在 $x=0$ 处二阶泰勒展开，有：

$$f(x)=f(0)+f'(0)x+\frac{f''(0)}{2}x^2+o\left(x^2\right)$$

因此有：

$$\begin{cases}e^x=1+x+\dfrac{1}{2}x^2+o\left(x^2\right)\\ \sqrt{1-x}=1-\dfrac{1}{2}x-\dfrac{1}{8}x^2+o\left(x^2\right)\end{cases}$$

则：
$$\begin{cases}e^{-2\gamma_m{}^2}=1-2\gamma_m{}^2+\dfrac{1}{2}\left(-2\gamma_m{}^2\right)+o\left(\gamma_m{}^4\right)\\ \sqrt{1-4\gamma_m{}^2}=1-2\gamma_m{}^2+\left(-\dfrac{1}{8}\right)\left(4\gamma_m{}^2\right)^2+o\left(\gamma_m{}^4\right)\end{cases}$$
$$\Rightarrow e^{-2\gamma_m{}^2}-\sqrt{1-4\gamma_m{}^2}=\gamma_m{}^4+o\left(\gamma_m{}^4\right)$$

因此：$$\sqrt{1-4\gamma_m{}^2}\le e^{-2\gamma_m{}^2}\Rightarrow\prod_{m=1}^{M}\sqrt{\left(1-4\gamma_m{}^2\right)}\le\exp\left(-2\sum_{m=1}^{M}\gamma_m{}^2\right)$$

小结：（AdaBoost 的训练误差界）

$$\frac{1}{N}\sum_{i=1}^{N}I\left(G(x_i)\ne y_i\right)\le\frac{1}{N}\sum_{i=1}^{N}\exp\left(-y_i f(x_i)\right)=\prod_{m=1}^{M}Z_m$$

$$G(x)=sign\left(f(x)\right)=sign\left(\sum_{m=1}^{M}\alpha_m G_m(x)\right)$$

这一定理说明，可以在每一轮选取适当的 G_m 使得 Z_m 最小，从而使得组合分类器的训练误差下降最快。

■

8.3.2 前向分步算法与 AdaBoost（P145-P146）

请证明 2：

$$G^*_m(x)=\arg\min_G\sum_{i=1}^N\bar{w}_{mi}I\left(y_i\neq G(x_i)\right),\ e_m=\frac{\sum_{i=1}^N\bar{w}_{mi}I\left(y_i\neq G_m(x_i)\right)}{\sum_{i=1}^N\bar{w}_{mi}}=\sum_{i=1}^N w_{mi}I\left(y_i\neq G(x_i)\right)$$

证明 2：

首先来证明 $G^*_m(x)=\arg\min_G\sum_{i=1}^N\bar{w}_{mi}I\left(y_i\neq G(x_i)\right)$，我们知道对任意的 $\alpha>0$，有：$G^*_m(x)=\arg\min_G\sum_{i=1}^N\bar{w}_{mi}\exp\left[-\alpha\cdot y_iG(x_i)\right]$，其中 $\bar{w}_{mi}=\exp\left[-y_if_{m-1}(x_i)\right]$ 与第 m 次最优化无关，与前 $m-1$ 最优化的结果有关。而且

$$\bar{w}_{mi}\exp\left[-\alpha\cdot y_iG(x_i)\right]=\begin{cases}\bar{w}_{mi}e^{-\alpha},\ y_i=G(x_i)\\ \bar{w}_{mi}e^{\alpha},\ y_i\neq G(x_i)\end{cases}$$

令 $\bar{w}_{l0}\in\left\{\bar{w}_{mi}\middle|y_i=G(x_i)\right\},\bar{w}_{l1}\in\left\{\bar{w}_{mi}\middle|y_i\neq G(x_i)\right\}$

$$l=N-\sum_{i=1}^N I\left(y_i\neq G(x_i)\right),n=\sum_{i=1}^N I\left(y_i\neq G(x_i)\right)$$

$$\Rightarrow\sum_{i=1}^N\bar{w}_{mi}\exp\left[-\alpha\cdot y_iG(x_i)\right]=\sum_{y_i=G(x_i)}\bar{w}_{mi}e^{-\alpha}+\sum_{y_i\neq G(x_i)}\bar{w}_{mi}e^{\alpha}$$

$$=\bar{w}_{l0}e^{-\alpha}+\ldots+\bar{w}_{l0}e^{-\alpha}+\bar{w}_{l1}e^{\alpha}+\ldots+\bar{w}_{m1}e^{\alpha}$$

$$\text{其中}\bar{w}_{mi}>0,0<e^{-\alpha}<e^{\alpha},\alpha>0.$$

$$\Rightarrow\arg\min_G\sum_{i=1}^N\bar{w}_{mi}\exp\left[-\alpha\cdot y_iG(x_i)\right]=\arg\min_G\sum_{i=1}^N\bar{w}_{mi}I\left(y_i\neq G(x_i)\right)$$

这主要是因为 $n=\sum_{i=1}^N I\left(y_i\neq G(x_i)\right)$ 越小，$\sum_{i=1}^N\bar{w}_{mi}\exp\left[-\alpha\cdot y_iG(x_i)\right]$ 也会越小.

接下来证明第二个等式，这里主要是要证明：

$$\frac{\bar{w}_{mi}}{\sum_{i=1}^N\bar{w}_{mi}}=w_{mi}$$

先列出已知条件：

$$\begin{cases}(1)\,\bar{w}_{mi}=\exp\left[-y_i f_{m-1}(x_i)\right]\\(2)\,w_{mi}=\dfrac{w_{m-1i}}{Z_{m-1}}\exp\left(-\alpha_{m-1}y_i G_{m-1}(x_i)\right)\\(3)\sum\limits_{i=1}^{N}w_{mi}=1\\(4)\,f_{m-1}(x_i)=\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\end{cases}$$

由（1）、（4）易得：

$$\frac{\bar{w}_{mi}}{\sum\limits_{i=1}^{N}\bar{w}_{mi}}=\frac{\exp\left[-y_i\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\right]}{\sum\limits_{i=1}^{N}\exp\left[-y_i\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\right]}$$

由（2）易得：

$$\begin{aligned}w_{mi}&=\frac{w_{m-1i}}{Z_{m-1}}\exp\left(-\alpha_{m-1}y_i G_{m-1}(x_i)\right)\\&=\frac{1}{Z_{m-1}}\exp\left(-\alpha_{m-1}y_i G_{m-1}(x_i)\right)\cdot\frac{w_{m-2i}}{Z_{m-2}}\exp\left(-\alpha_{m-2}y_i G_{m-2}(x_i)\right)\\&=\ldots=\frac{1}{\prod\limits_{j=1}^{m-1}Z_j}\exp\left[-y_i\sum_{j=1}^{m-1}\alpha_j G_j(x_i)\right]\end{aligned}$$

由（3）易得：

$$\sum_{i=1}^{N}w_{mi}=1=\frac{\sum\limits_{i=1}^{N}\exp\left[-y_i\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\right]}{\prod\limits_{j=1}^{m-1}Z_j}\Rightarrow\prod_{j=1}^{m-1}Z_j=\sum_{i=1}^{N}\exp\left[-y_i\sum_{j=1}^{m-1}\alpha_j G_j(x_i)\right]$$

于是有：

$$w_{mi}=\frac{1}{\prod\limits_{j=1}^{m-1}Z_j}\exp\left[-y_i\sum_{j=1}^{m-1}\alpha_j G_j(x_i)\right]=\frac{\exp\left[-y_i\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\right]}{\sum\limits_{i=1}^{N}\exp\left[-y_i\sum\limits_{j=1}^{m-1}\alpha_j G_j(x_i)\right]}=\frac{\bar{w}_{mi}}{\sum\limits_{i=1}^{N}\bar{w}_{mi}}$$

■

8.4.3 梯度提升（P151）

关键词 2： $$\hat{r}_{m-1i}=-\left[\frac{\partial L\left(y,f_m(x)\right)}{\partial f\left(x_i\right)}\right]_{f(x)=f_{m-1}(x)}$$

说明 2：

建立提升回归树时，对于一般损失函数而言，在每一步的优化并不容易。针对这一问题，有人提出可以利用损失函数的负梯度在当前模型的值，即等式右边的值，作为回归树提升算法中残差的近似值。那么，对于一般损失函数，而不仅仅是平方损失函数，为什么负梯度会近似等于当前模型的残差呢？渣君来写写自己的理解，如下。

令 $\nabla_i=-\left[\frac{\partial L\left(y,f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x)=f_{m-1}(x)}$，则 ∇_i 是 L 关于 $f\left(x_i\right)$ 的负梯度方向的单位增量。若当前模型为 $f_{m-1}\left(x_i\right)$，假设拟合 $\nabla=\left(\nabla_1,\nabla_2...,\nabla_N\right)$ 后得到一棵树 T_∇（$T_{\nabla i}\approx\nabla_i,i=1,2,...,N.$），则新的模型为：$f_m\left(x_i\right)=f_{m-1}\left(x_i\right)+T_{\nabla i}$，由于 $T_{\nabla i}\approx\nabla_i$ 表示负梯度方向的增量，则对于新的模型 $f_m(x)$，$L\left(y,f_m(x)\right)$ 以较快的速度向 0 下降，即学习到的模型 $f_m\left(x_i\right)=f_{m-1}\left(x_i\right)+T_{\nabla i}$ 更接近真实值 y_i. 因而有：

$$\nabla_i\approx T_{\nabla i}=f_m\left(x_i\right)-f_{m-1}\left(x_i\right)\approx y_i-f_{m-1}\left(x_i\right)=r_{m-1i}$$
$$\Rightarrow\hat{r}_{m-1i}=\nabla_i\approx r_{m-1i},i=1,2,...,N.$$

其中 $f_m\left(x_i\right)=f_{m-1}\left(x_i\right)+T_{\nabla i}$ 成立的原因是回归提升树采用的前向分步加法的学习方式。

■

8.1.3 AdaBoost 的例子（P140）

关键词 3： 例 8.1 的 R 实现

说明 3：

AdaBoost 算法在面对不同的机器学习模型时会有所不同，因此本渣猜测，应该是要根据实际情况（不同的机器学习模型）编写不同的 AdaBoost 算法程序，而可能没有所谓的统一的 AdaBoost 函数。毕竟 AdaBoost 算法主要是提供了一种将弱学习器转化为强学习器的方法及思路，而不会限制于某种特定的“弱学习器”。但是，编写一些简单的函数来求解一下书本的例题还是比较容易的。

为了方便解题，我编写了几个小函数，包括 AdaboostAlpha()、AdaboostWeight()、AdaboostError()、SearchOneTree()以及函数 AdaboostTreeStool()（TreeStool 中文意义为“树桩”）。前三个函数为辅助函数，分别用于求分类树提升的模型系数，样本权值分布以及当前模型在当前样本权值分布条件下的分类误差率。（详细代码参见附录 8）

SearchOneTree(atr,cls,weightm,data,sep=0.5)函数用于求解一次迭代的模型结果。atr 指特征，cls 指分类变量，data 是指包括 cls，atr 指定变量的数据的数据框（这里 atr 只有一个）；weightm 指样本的权值分布向量；sep=0.5 设定间距（这是根据例题的需要指定的），若果 sep=0，就是使用 atr 原始数据作为划分点。

AdaboostTreeStool(atr,cls,data,weight0=rep(1/length(clsvec),length(clsvec)),ept=0,maxiter=10000,sep=.5)函数可以一次

性求解所有的迭代结果，并输出每次迭代对应的最优模型及模型参数。atr,cls,data,sep 参数意义与 SearchOneTree()中的一样；weight0 指定初始权值分布，默认是均匀的权值分布；ept 指定阈值，当组合模型 $G(x)=sign(f(x))=sign\left(\sum_{m=1}^{M}\alpha_m G_m(x)\right)$ 的分类误差率小于 ept 时退出迭代（这里分类误差率采用的是均匀权值分布），默认为 0，指全部样本分类正确时退出迭代.

我们先不用 AdaboostTreeStool()函数一步到位，我们如例 8.1 那样，先来一步步求解结果。书本数据。

表 8.1　训练数据表

序号	1	2	3	4	5	6	7	8	9	10
x	0	1	2	3	4	5	6	7	8	9
y	1	1	1	−1	−1	−1	1	1	1	−1

代码如下：

```
x<-0:9
y<-c(1,1,1,-1,-1,-1,1,1,1,-1)
dataxy<-data.frame(x=x,y=y);dataxy#数据准备
##   求解第一个模型
D1<-rep(1/10,10)#初始权值分布
G1<-SearchOneTree(atr="x",cls = "y",data=dataxy,weightm =D1);G1
##   求模型 G1 的系数 alpha1 及 f1
alpha1<-AdaboostAlpha(G1$error);alpha1
f1<-alpha1*G1$preclsvec;f1
## 更新训练数据集的权值分布 D1 为 D2
D2<-AdaboostWeight(weightm = D1,alpham = alpha1,clsvec = y,
                                        preclsvec = G1$preclsvec);D2
```

结果如下：

```
> G1
$error
[1] 0.3

$ModelFinal
[1] "Model:: x<=2.5 is 1 else -1."

$preclsvec
 [1]  1  1  1 -1 -1 -1 -1 -1 -1 -1

> alpha1<-AdaboostAlpha(G1$error);alpha1
[1] 0.4236489

> D2
 [1] 0.07142857 0.07142857 0.07142857 0.07142857 0.07142857
 [6] 0.07142857 0.16666667 0.16666667 0.16666667 0.07142857

> f1<-alpha1*G1$preclsvec;f1
 [1]  0.4236489  0.4236489  0.4236489 -0.4236489 -0.4236489
 [6] -0.4236489 -0.4236489 -0.4236489 -0.4236489 -0.4236489
```

我们看到上述结果与书本模型 1 的求解结果是一致的，只是数值在精度上稍有区别。样本的权值分布的更新也是正确的。我们接下来看看例题 8.1 剩下的迭代结果。

模型 2 代码：

```
## 求解第二个模型
G2<-SearchOneTree(atr="x",cls = "y",data=dataxy,weightm = D2);G2
##  求模型 G2 的系数 alpha2 及 f2
alpha2<-AdaboostAlpha(G2$error);alpha2
f2<-alpha2*G2$preclsvec+f1;f2
## 更新训练数据集的权值分布 D2 为 D3
D3<-AdaboostWeight(weightm = D2,alpham = alpha2,clsvec = y,
                                        preclsvec = G2$preclsvec);D3
```

模型 2 结果：

```
> G2
$error
[1] 0.2142857

$ModelFinal
[1] "Model:: x<=8.5 is 1 else -1."

$preclsvec
 [1]  1  1  1  1  1  1  1  1  1 -1

> alpha2<-AdaboostAlpha(G2$error);alpha2
[1] 0.6496415

> D3
 [1] 0.04545455 0.04545455 0.04545455 0.16666667 0.16666667
 [6] 0.16666667 0.10606061 0.10606061 0.10606061 0.04545455

> f2<-alpha2*G2$preclsvec+f1;f2
 [1]  1.0732904  1.0732904  1.0732904  0.2259926  0.2259926
 [6]  0.2259926  0.2259926  0.2259926  0.2259926 -1.0732904
```

结果与书本一致。

模型 3 代码：

```
## 求解第三个模型
G3<-SearchOneTree(atr="x",cls = "y",data=dataxy,weightm = D3);G3
##  求模型 G3 的系数 alpha3 及 f3
alpha3<-AdaboostAlpha(G3$error);alpha3
f3<-alpha3*G3$preclsvec+f2;f3
## 更新训练数据集的权值分布 D3 为 D4
D4<-AdaboostWeight(weightm = D3,alpham = alpha3,clsvec = y,
                                    preclsvec = G3$preclsvec);D4
```

模型 3 结果：

```
> G3
$error
[1] 0.1818182

$ModelFinal
[1] "Model:: x>5.5 is 1 else -1."

$preclsvec
 [1] -1 -1 -1 -1 -1 -1  1  1  1  1

> alpha3<-AdaboostAlpha(G3$error);alpha3
[1] 0.7520387

> D4
 [1] 0.12500000 0.12500000 0.12500000 0.10185185 0.10185185
 [6] 0.10185185 0.06481481 0.06481481 0.06481481 0.12500000

> f3<-alpha3*G3$preclsvec+f2;f3
 [1]  0.3212517  0.3212517  0.3212517 -0.5260461 -0.5260461
 [6] -0.5260461  0.9780313  0.9780313  0.9780313 -0.3212517
```

模型 3 计算的 alpha3 与书本的结果稍有差别，这是由于这里使用了更精确的误差率 0.1818182 来计算 alpha3 的缘故，实际上与书本的是一样的。接下来我们看看组合模型的误分率。

```
> tblf3<-table(sign(f3),y);tblf3
     y
      -1 1
  -1   4 0
  1    0 6
> 1-sum(diag(tblf3))/sum(tblf3)#误分率
[1] 0
```

可见组合模型的误分率为 0，它完全正确划分的训练集，比单个树桩模型的分类正确率高。

现在，我们来看看使用函数 AdaboostTreeStool()来求解这道题的结果如何。代码及结果如下。

```
> AdaboostTreeStool(atr="x",cls = "y",data=dataxy,ept=0,sep=.5)
$stoprule
[1] "errf<=ept"

$errf
[1] 0

$iteration
[1] 3

$AdaboostModel
[1] "Model:: x<=2.5 is 1 else -1."
[2] "Model:: x<=8.5 is 1 else -1."
[3] "Model:: x>5.5 is 1 else -1."

$AdaboostAlpha
[1] 0.4236489 0.6496415 0.7520387

$AdaboostError
[1] 0.3000000 0.2142857 0.1818182

$AdaboostPredict
 [1]  1  1  1 -1 -1 -1  1  1  1 -1
```

输出的结果与之前展示的一致，其中组件 errf 是指最终组合模型的误分率;AdaboostModel 组件记录了每一次迭代生成的树桩模型；AdaboostAlpha 组件记录了每个树桩的参数系数值；AdaboostError 记录了每个模型基于对应权值分布的误分率；AdaboostPredict 组件是指利用组合模型回测的分类结果。

现在我们用 iris 数据集和 mtcars 数据集来测试一下这个函数。

mtcars 数据集模型 1 代码及结果：

```
> datamtcars1<-mtcars[,c("mpg","am")]
> datamtcars1$am<-ifelse(datamtcars1$am==1,1,-1)
> AdaboostTreeStool(atr="mpg",cls = "am",data=datamtcars1,ept=.1,sep=0)
$stoprule
[1] "errf<=ept"

$errf
[1] 0.09375

$iteration
[1] 22

$AdaboostModel
 [1] "Model:: mpg>19.2 is 1 else -1."  "Model:: mpg>24.4 is 1 else -1."
 [3] "Model:: mpg<=21 is 1 else -1."   "Model:: mpg>14.7 is 1 else -1."
 [5] "Model:: mpg>24.4 is 1 else -1."  "Model:: mpg>14.7 is 1 else -1."
 [7] "Model:: mpg<=15.8 is 1 else -1." "Model:: mpg>19.2 is 1 else -1."
 [9] "Model:: mpg<=15.8 is 1 else -1." "Model:: mpg>15.5 is 1 else -1."
[11] "Model:: mpg<=15 is 1 else -1."   "Model:: mpg>14.7 is 1 else -1."
[13] "Model:: mpg>24.4 is 1 else -1."  "Model:: mpg>14.7 is 1 else -1."
[15] "Model:: mpg<=15 is 1 else -1."   "Model:: mpg>15.5 is 1 else -1."
[17] "Model:: mpg<=15.8 is 1 else -1." "Model:: mpg>19.2 is 1 else -1."
[19] "Model:: mpg<=21 is 1 else -1."   "Model:: mpg>19.2 is 1 else -1."
[21] "Model:: mpg<=15.8 is 1 else -1." "Model:: mpg>15.5 is 1 else -1."

$AdaboostAlpha
 [1] 0.7331685 0.5156800 0.3159413 0.2854325 0.3355280 0.2494651 0.3043118
 [8] 0.2890585 0.2231551 0.2014789 0.1869559 0.2125642 0.2403248 0.1931768
[15] 0.1926273 0.1673026 0.1983713 0.2048165 0.2298826 0.1864138 0.1854130
[22] 0.1751776

$AdaboostError
 [1] 0.1875000 0.2628205 0.3470838 0.3610373 0.3382604 0.3777921 0.3523732
 [8] 0.3593660 0.3902384 0.4006019 0.4075962 0.3952902 0.3820988 0.4045954
[15] 0.4048602 0.4171205 0.4020952 0.3990001 0.3870415 0.4078580 0.4083415
[22] 0.4132963
```

提升 22 次后收敛，由结果可见，组合模型的误分率远远小于任意单个模型的误分率。

mtcars 模型 2 代码及结果：

```
> datamtcars2<-mtcars[,c("mpg","vs")]
> datamtcars2$vs<-ifelse(datamtcars2$vs==1,1,-1)
> AdaboostTreeStool(atr="mpg",cls = "vs",data=datamtcars2,ept=.1,sep=0)
$stoprule
[1] "errf<=ept"

$errf
[1] 0.0625

$iteration
[1] 7

$AdaboostModel
[1] "Model:: mpg>21 is 1 else -1."      "Model:: mpg>17.3 is 1 else -1."
[3] "Model:: mpg>26 is 1 else -1."      "Model:: mpg<=24.4 is 1 else -1."
[5] "Model:: mpg>26 is 1 else -1."      "Model:: mpg>17.3 is 1 else -1."
[7] "Model:: mpg<=18.1 is 1 else -1."

$AdaboostAlpha
[1] 0.9729551 0.6496415 0.4075185 0.4486213 0.3052984 0.3898139 0.3978986

$AdaboostError
[1] 0.1250000 0.2142857 0.3068182 0.2896175 0.3519231 0.3144001 0.3109253
```

同样，组合模型的误分率远远小于任意单个模型的误分率。

Iris 数据集模型代码及结果：

```
> datairis<-iris[1:100,c(3,5)]
> datairis$Species<-ifelse(datairis$Species=="setosa",1,-1)
> AdaboostTreeStool(atr="Petal.Length",cls = "Species",data=datairis)
$stoprule
[1] "err==0||err==1"

$Model
[1] "Model:: Petal.Length<=1.9 is 1 else -1."

$error
[1] 0
```

训练中找到能正确完全划分训练样本的模型，直接输出，没有必

要继续寻找组合模型了。这就是在“关键词 1”里面所讲的特殊情况。

最后，我们将思路放回到例 8.1，细心观察，我们就能发现，其实这个例子充分彰显了 AdaBoost 算法的神奇之处。就例 8.1 的样本数据而言，任何一个单独的树桩模型都无法将样本完全正确划分。但是通过 AdaBoost 算法找到组合模型后，竟然突破了这个限制，使得训练样本的误分率为 0，完全正确划分了样本集!!!

提升方法的笔记就写到这里了，书本后面还有个提升回归树的例子，用 R 重现应该也不难，在这里就不打算写了，有兴趣的小伙伴可以写写。恩，写完，可以的话，代码顺便发我一份哈 !^_^!

■

9.2 EM 算法的收敛性（P161）

请证明 1： $\log P\left(Y|\theta\right)=Q\left(\theta,\theta^{(i)}\right)-H\left(\theta,\theta^{(i)}\right)$

证明 1：

$$Q\left(\theta,\theta^{(i)}\right)=\sum_{z}\log P\left(Y,Z|\theta\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)$$

$$H\left(\theta,\theta^{(i)}\right)=\sum_{z}\log P\left(Z|Y,\theta\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)$$

$$Q\left(\theta,\theta^{(i)}\right)-H\left(\theta,\theta^{(i)}\right)=\sum_{z}\log P\left(Y,Z|\theta\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)-\sum_{z}\log P\left(Z|Y,\theta\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)$$

$$=\sum_{z}\left(\log\frac{P\left(Y,Z|\theta\right)}{P\left(Z|Y,\theta\right)}\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)=\sum_{z}\left(\log P\left(Y|\theta\right)\right)P\left(Z\left|Y,\theta^{(i)}\right.\right)$$

$$=\log P\left(Y|\theta\right)\sum_{z}P\left(Z\left|Y,\theta^{(i)}\right.\right)=\log P\left(Y|\theta\right),\sum_{z}P\left(Z\left|Y,\theta^{(i)}\right.\right)=1.$$

■

9.3.1 高斯混合模型（P163）

请证明 2： $P\left(y_j,\gamma_{j1},\gamma_{j2},...,\gamma_{jK}\right)=\prod_{k=1}^{K}\left[\alpha_k\phi\left(y_j|\theta_k\right)\right]^{\gamma_{jk}}$

证明 2：

如果 y_j 来自第 k 个分模型，则明确地有：$\gamma_{ji}=0,i\neq k,\gamma_{jk}=1.$ 于是有：

$$P\left(y_j,\gamma_{j1},\gamma_{j2},...,\gamma_{jK}\right)=\left[\alpha_k\phi\left(y_j|\theta_k\right)\right]^{\gamma_{jk}}=\left[\alpha_k\phi\left(y_j|\theta_k\right)\right]^{1}\cdot1\cdot1...\cdot1$$

$$=\left[\alpha_k\phi\left(y_j|\theta_k\right)\right]^{\gamma_{jk}}\cdot\left[\alpha_1\phi\left(y_j|\theta_1\right)\right]^{\gamma_{j1}}\cdot...\cdot\left[\alpha_{k-1}\phi\left(y_j|\theta_{k-1}\right)\right]^{\gamma_{jk-1}}$$

$$\cdot\left[\alpha_{k+1}\phi\left(y_j|\theta_{k+1}\right)\right]^{\gamma_{jk+1}}\cdot...\cdot\left[\alpha_K\phi\left(y_j|\theta_K\right)\right]^{\gamma_{jK}}$$

$$=\prod_{k=1}^{K}\left[\alpha_k\phi\left(y_j|\theta_k\right)\right]^{\gamma_{jk}}$$

■

9.4 EM 算法的推广（P167）

请证明 3： 若 $\tilde{P}_\theta(Z)=P(Z|Y,\theta)$，则 $F(\tilde{P},\theta)=\log P(Y|\theta)$.

证明 3：

$$
\begin{aligned}
F(\tilde{P},\theta) &= \int \log P(Y,Z|\theta)\tilde{P}_\theta(Z)\,dZ - \int \log \tilde{P}_\theta(Z)\cdot\tilde{P}_\theta(Z) \\
&= \int \log \frac{P(Y,Z|\theta)}{\tilde{P}_\theta(Z)}\cdot\tilde{P}_\theta(Z)\,dZ \xrightarrow{\tilde{P}\theta(Z)=P(Z|Y,\theta)} \int \log \frac{P(Y,Z|\theta)}{P(Z|Y,\theta)}\cdot P(Z|Y,\theta)\,dZ \\
&= \int \log P(Y|\theta)\cdot P(Z|Y,\theta)\,dZ = \log P(Y|\theta)\cdot\int P(Z|Y,\theta)\,dZ \\
&= \log P(Y|\theta), \int P(Z|Y,\theta)\,dZ = 1.
\end{aligned}
$$

■

9.3.1 高斯混合模型的 EM 算法（165）

关键词 1： 高斯混合模型 EM 算法的 R 实现

说明 1：

输入：观测数据 $y_1, y_2, ..., y_N.$，高斯混合模型

输出：高斯混合模型参数.

（1）取参数的初始值开始迭代。

（2）E 步：依当前模型参数，计算分模型 k 对观测数据 y_j 的响应度：

$$\hat{\gamma}_{jk} = \frac{\alpha_k \phi\left(y_j | \theta_k\right)}{\sum_{k=1}^{K} \alpha_k \phi\left(y_j | \theta_k\right)}, \ j = 1,2,...,N; \ k = 1,2,...,K.$$

（3）M 步：计算新一轮迭代的模型参数

$$\hat{\mu}_k = \frac{\sum_{k=1}^{K} \hat{\gamma}_{jk} y_j}{\sum_{k=1}^{K} \hat{\gamma}_{jk}}; \ \hat{\sigma}_k^2 = \frac{\sum_{k=1}^{K} \hat{\gamma}_{jk}\left(y_j - \hat{\mu}_k\right)}{\sum_{k=1}^{K} \hat{\gamma}_{jk}}; \ \hat{\alpha}_k = \frac{\sum_{k=1}^{K} \hat{\gamma}_{jk}}{N}, k = 1,2,...,K.$$

（4）重复（2）和（3）直至收敛。

（5）收敛准则：

$$\left\|\theta^{(i+1)} - \theta^{(i)}\right\| \le \varepsilon \quad \text{or} \quad \left\|Q\left(\theta^{(i+1)}, \theta^{(i)}\right) - Q\left(\theta^{(i)}, \theta^{(i)}\right)\right\| \le \varepsilon.$$

渣君根据这个算法编写了几个小函数来实现高斯混合分布的 EM 估计。这些函数分别是 gaussiResponse()、gaussiParameter()、gaussiEM() 以及 print.gaussiEM(). 其中 gaussiResponse() 和 gaussiParameter()是辅助函数，gaussiResponse()负责计算每次迭代的响应度，gaussiParameter()计算每次迭代更新的参数，它们均在 gaussiEM()内部被调用。print.gaussiEM()函数指定类“gaussiEM”

的打印方式。（详细代码参加附录 9）

gaussiEM(clsvec,K=2,mean0=rep(0,K),var0=rep(1,K),alpha0=rep(1/K,K),ept=1e-1,maxiter=10000,Lp=2)函数中 clsvec 输入我们需要用于估计的观测数据向量;K 指定高斯混合模型分模型的个数；mean0 指定初始均值；var0 指定初始方差；alpha0 指定初始的分模型权重；Lp 指定停止规则中使用的范数类型，默认为 2.

我们先来生成一组高斯混合分布的随机数，测试一下这几个函数，然后将其用于求解书本习题 9.3. 代码如下。

```
## 测试 1 K=2 高斯混合模型随机数测试 ##
comp <- sample(c(0, 1), size = 10000, prob = c(0.7, 0.3), replace = T)
sim1<-rnorm(10000, mean = ifelse(comp == 0, 0, 1), sd = ifelse(comp == 0, 1, 2))
g1<-gaussiEM(clsvec = sim1,ept=1e-3,mean0 = c(0,.5),K=2);g1
```

结果如下：

```
> gaussiEM(clsvec = sim1,ept=1e-3,mean0 = c(0,.5),K=2)
Stoprule :  Lpvalue<=ept
iteration :  102
the number of gaussi is :  2
$Mean
[1] -0.01956984  0.89069452

$Var
[1] 0.9746784 4.0441223

$Alpha
[1] 0.6689286 0.3310714
```

由以上结果可见，迭代收敛后的结果，均值 Mean/方差 Var/分模型权重均比较接近真实模型，效果尚算可以。我们来看另一个测试。

代码如下：

```
##   测试 2 ##
comp2 <- sample(c(0,1,2), size = 100000, prob = c(0.5,0.3,0.2), replace = T)
sim2<-rnorm(100000, mean =comp2, sd = ifelse(comp == 0, 1,ifelse(comp2==1,4,2)))
g2<-gaussiEM(clsvec = sim2,ept=1e-3,mean0 = c(0,.5,.8),K=3);g2
```

结果如下：

```
> g2<-gaussiEM(clsvec = sim2,ept=1e-3,mean0 = c(0,.5,.8),K=3);g2
Stoprule :   Lpvalue<=ept
iteration :   726
the number of gaussi is :   3
$Mean
[1] 0.6360976 0.2045092 1.5459884

$Var
[1] 13.017896   1.384495   1.626617

$Alpha
[1] 0.1540195 0.5311336 0.3148469
```

结果表明，Alpha 的估计较为接近真实模型，其余参数的估计均是有所偏差。EM 算依据初值的不同，效果也会有所不同。比如：

```
##   改变初始值
> g2<-gaussiEM(clsvec = sim2,ept=1e-3,mean0 = c(0,.8,.9),var0 = c(.6,7.8,2.3),K=3);g2
Stoprule :   Lpvalue<=ept
iteration :   225
the number of gaussi is :   3
$Mean
[1] 0.02613851 0.63258226 1.19422080

$Var
[1]   1.242649 12.839283   1.764226

$Alpha
[1] 0.3529045 0.1578533 0.4892422
```

最后，我们来用 EM 算法求解习题 9.3.代码及结果如下：

```
> y<-c(-67,-48,6,8,14,16,23,24,28,29,41,49,56,60,75)
> gaussiEM(y)
Stoprule :   Lpvalue<=ept
iteration :   25
the number of gaussi is :   2
$Mean
[1] -57.51107   32.98490

$Var
[1]   90.24988 429.45754

$Alpha
[1] 0.1331724 0.8668276
```

OK，EM 算法的笔记就到此为止了。

■

10.2.2 前向算法（P175-P176）

请证明 1：

$$\alpha_{t+1}(i)=\left[\sum_{j=1}^{N}\alpha_t(j)a_{ji}\right]b_i(o_{t+1}),i=1,2,...,N.$$

$$P(O|\lambda)=\sum_{i=1}^{N}\alpha_T(i),\ \alpha_T(i)=P(o_1,o_2,...,o_T,i_T=q_i|\lambda).$$

证明 1：

已知两个假设如下：

（1）齐次马尔可夫性：$P(i_t|i_{t-1},o_{t-1},...,i_1,o_1)=P(i_t|i_{t-1})$

（2）观测独立性：$P(o_t|i_T,o_T,i_{T-1},o_{T-1},...,i_{t+1},o_{t+1},i_t,i_{t-1},o_{t-1},...,i_1,o_1)=P(o_t|i_t)$

根据以上两个假设，可得：

$$\alpha_{t+1}(i)=P(o_1,o_2,...,o_t,o_{t+1},i_{t+1}=q_i|\lambda)=\sum_{j=1}^{N}P(o_1,o_2,...,o_t,o_{t+1},i_t=q_j,i_{t+1}=q_i|\lambda)$$

$$=\sum_{j=1}^{N}P(o_{t+1},i_{t+1}=q_i|o_1,o_2,...,o_t,i_t=q_j,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_j|\lambda)$$

$$=\sum_{j=1}^{N}\Big[P(o_{t+1}|i_{t+1}=q_i,o_1,o_2,...,o_t,i_t=q_j,\lambda)\cdot P(i_{t+1}=q_i|o_1,o_2,...,o_t,i_t=q_j,\lambda)$$

$$\cdot P(o_1,o_2,...,o_t,i_t=q_j|\lambda)\Big]$$

$$=\sum_{j=1}^{N}P(o_{t+1}|i_{t+1}=q_i,\lambda)\cdot P(i_{t+1}=q_i|i_t=q_j,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_j|\lambda)$$

$$=\sum_{j=1}^{N}b_i(o_{t+1})\cdot a_{ji}\cdot\alpha_t(j)=\left[\sum_{j=1}^{N}\alpha_t(j)a_{ji}\right]b_i(o_{t+1})$$

另外有：

$$P(O|\lambda)=P(o_1,o_2,...,o_T|\lambda)=\sum_{i=1}^{N}P(o_1,o_2,...,o_T,i_T=q_i|\lambda)=\sum_{i=1}^{N}\alpha_T(i)$$

■

10.2.3 后向算法（P178-P179）

请证明 2：

$$\beta_t(i)=\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})\beta_{t+1}(i),\ \ i=1,2,...,N.$$

$$P(O|\lambda)=\sum_{i=1}^{N}\pi_i b_i(o_1)\beta_1(i)$$

$$P(O|\lambda)=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(i)=\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)$$

证明 2：

先证明：$\beta_t(i)=\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})\beta_{t+1}(i),\ \ i=1,2,...,N.$

已知两个假设如下：

（1）齐次马尔可夫性：$P(i_t|i_{t-1},o_{t-1},...,i_1,o_1)=P(i_t|i_{t-1})$

（2）观测独立性：$P(o_t|i_T,o_T,i_{T-1},o_{T-1},...,i_{t+1},o_{t+1},i_t,i_{t-1},o_{t-1},...,i_1,o_1)=P(o_t|i_t)$

根据以上两个假设，可得：

$$\beta_t(i)=P(o_{t+1},o_{t+2},...,o_T|i_t=q_i,\lambda)=\sum_{j=1}^{N}P(o_{t+1},o_{t+2},...,o_T,i_{t+1}=q_j|i_t=q_i,\lambda)$$

$$=\sum_{j=1}^{N}\Big[P(o_{t+2},...,o_T|o_{t+1},i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\Big]$$

$$=\sum_{j=1}^{N}\Big[P(o_{t+2},...,o_T|i_{t+1}=q_j,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\Big]$$

$$=\sum_{j=1}^{N}\beta_{t+1}(j)\cdot b_j(o_{t+1})\cdot a_{ij}$$

上述推导中，由（2）知，o_{t+2}只依赖于i_{t+2}，而由（1）知i_{t+2}又只依赖于i_{t+1}，于是有：

$$P(o_{t+2},...,o_T|o_{t+1},i_{t+1}=q_j,i_t=q_i,\lambda)=P(o_{t+2},...,o_T|i_{t+1}=q_j,\lambda)=\beta_{t+1}(j)$$

同理，由（2）知：$P(o_{t+1}|i_{t+1}=q_j,i_t=q_i,\lambda)=P(o_{t+1}|i_{t+1}=q_j,\lambda)=b_j(o_{t+1})$

显然，有：$P(i_{t+1}=q_j|i_t=q_i,\lambda)=a_{ij}$

接下来证明： $P(O|\lambda)=\sum_{i=1}^{N}\pi_i b_i(o_1)\beta_1(i)$

$$
\begin{aligned}
&P(O|\lambda)=P(o_1,o_2,...,o_T|\lambda)=\sum_{i=1}^{N}P(o_1,o_2,...,o_T,i_1=q_i|\lambda)\\
&=\sum_{i=1}^{N}P(o_1,o_2,...,o_T|i_1=q_i,\lambda)P(i_1=q_i|\lambda)\\
&=\sum_{i=1}^{N}P(o_2,...,o_T|i_1=q_i,o_1,\lambda)P(o_1|i_1=q_i,\lambda)P(i_1=q_i|\lambda)\\
&=\sum_{i=1}^{N}P(o_2,...,o_T|i_1=q_i,\lambda)P(o_1|i_1=q_i,\lambda)P(i_1=q_i|\lambda)\\
&=\sum_{i=1}^{N}\beta_1(i)b_i(o_1)\pi_i
\end{aligned}
$$

最后证明： $P(O|\lambda)=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(i)=\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)$

$$
\begin{aligned}
&P(O|\lambda)=P(o_1,o_2,...,o_T|\lambda)=\sum_{i=1}^{N}\sum_{j=1}^{N}P(o_1,o_2,...,o_T,i_t=q_i,i_{t+1}=q_j|\lambda)\\
&=\sum_{i=1}^{N}\sum_{j=1}^{N}P(o_{t+1},o_{t+2},...,o_T,i_{t+1}=q_j|o_1,o_2,...,o_t,i_t=q_i,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_i|\lambda)\\
&=\sum_{i=1}^{N}\sum_{j=1}^{N}P(o_{t+1},o_{t+2},...,o_T,i_{t+1}=q_j|i_t=q_i,\lambda)\cdot\alpha_t(i)\\
&=\sum_{i=1}^{N}\alpha_t(i)\sum_{j=1}^{N}P(o_{t+2},...,o_T|o_{t+1},i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\\
&=\sum_{i=1}^{N}\alpha_t(i)\sum_{j=1}^{N}P(o_{t+2},...,o_T|i_{t+1}=q_j,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\\
&=\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)\cdot\beta_{t+1}(j)\cdot b_j(o_{t+1})\cdot a_{ij}=\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)
\end{aligned}
$$

■

10.4.2 维特比算法（P185）

请证明 3：

$$\begin{aligned}\delta_{t+1}(i) &= \max_{i_1,i_2,\ldots,i_t} P\left(i_{t+1}=i,i_t,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\\ &= \max_{1\le j\le N}\left[\delta_t(j)a_{ji}\right]b_i(o_{t+1}),\quad i=1,2,\ldots,N;t=1,2,\ldots,T-1.\end{aligned}$$

证明 3：

$$\begin{aligned}\delta_{t+1}(i) &= \max_{i_1,i_2,\ldots,i_t} P\left(i_{t+1}=i,i_t,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\\ &= \max_{i_1,i_2,\ldots,i_{t-1}}\left\{\max_{1\le i_t\le N} P\left(i_{t+1}=i,i_t,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\right\}\\ &= \max_{1\le j\le N}\left\{\max_{i_1,i_2,\ldots,i_{t-1}} P\left(i_{t+1}=i,i_t=j,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\right\}\end{aligned}$$

其中：

$$\begin{aligned}&P\left(i_{t+1}=i,i_t=j,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\\ &= P\left(i_{t+1}=i,o_{t+1}\middle|i_t=j,\ldots,i_1,o_t,\ldots,o_1,\lambda\right)\cdot P\left(i_t=j,\ldots,i_1,o_t,\ldots,o_1\middle|\lambda\right)\\ &= P\left(i_{t+1}=i,o_{t+1}\middle|i_t=j,\lambda\right)\cdot P\left(i_t=j,\ldots,i_1,o_t,\ldots,o_1\middle|\lambda\right)\\ &= P\left(o_{t+1}\middle|i_t=j,i_{t+1}=i,\lambda\right)\cdot P\left(i_{t+1}=i\middle|i_t=j,\lambda\right)\cdot P\left(i_t=j,\ldots,i_1,o_t,\ldots,o_1\middle|\lambda\right)\\ &= P\left(o_{t+1}\middle|i_{t+1}=i,\lambda\right)\cdot P\left(i_{t+1}=i\middle|i_t=j,\lambda\right)\cdot P\left(i_t=j,\ldots,i_1,o_t,\ldots,o_1\middle|\lambda\right)\\ &= b_i(o_{t+1})\cdot a_{ji}\cdot P\left(i_t=j,\ldots,i_1,o_t,\ldots,o_1\middle|\lambda\right)\end{aligned}$$

于是：

$$\begin{aligned}\delta_{t+1}(i) &= \max_{1\le j\le N}\left\{\max_{i_1,i_2,\ldots,i_{t-1}} P\left(i_{t+1}=i,i_t=j,\ldots,i_1,o_{t+1},\ldots,o_1\middle|\lambda\right)\right\}\\ &= \max_{1\le j\le N}\left[\delta_t(j)a_{ji}\right]b_i(o_{t+1}),\quad i=1,2,\ldots,N;t=1,2,\ldots,T-1.\end{aligned}$$

■

10.2.4 一些概率与期望值的计算（P179-P180）

请证明 4：

$$\alpha_t(i)\beta_t(i)=P(i_t=q_i,O|\lambda)$$
$$\xi_t(i,j)=P(i_t=q_i,i_{t+1}=q_j,O|\lambda)=\alpha_t(i)\cdot a_{ij}\cdot b_j(o_{t+1})\cdot\beta_{t+1}(j)$$

证明 4：

首先有：

$$\begin{aligned}&P(o_1,o_2,...,o_t,o_{t+1},...,o_T,i_t=q_i|\lambda)\\&=P(o_{t+1},...,o_T|i_t=q_i,o_1,o_2,...,o_t,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_i|\lambda)\\&=P(o_{t+1},...,o_T|i_t=q_i,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_i|\lambda)\\&=\beta_t(i)\alpha_t(i)\end{aligned}$$

然后：

$$\begin{aligned}&P(o_1,o_2,...,o_T,i_t=q_i,i_{t+1}=q_j|\lambda)\\&=P(o_{t+1},o_{t+2},...,o_T,i_{t+1}=q_j|o_1,o_2,...,o_t,i_t=q_i,\lambda)\cdot P(o_1,o_2,...,o_t,i_t=q_i|\lambda)\\&=P(o_{t+1},o_{t+2},...,o_T,i_{t+1}=q_j|i_t=q_i,\lambda)\cdot\alpha_t(i)\\&=\alpha_t(i)P(o_{t+2},...,o_T|o_{t+1},i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,i_t=q_i,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\\&=\alpha_t(i)P(o_{t+2},...,o_T|i_{t+1}=q_j,\lambda)\cdot P(o_{t+1}|i_{t+1}=q_j,\lambda)\cdot P(i_{t+1}=q_j|i_t=q_i,\lambda)\\&=\alpha_t(i)\cdot\beta_{t+1}(j)\cdot b_j(o_{t+1})\cdot a_{ij}\end{aligned}$$

于是：$\xi_t(i,j)=P(i_t=q_i,i_{t+1}=q_j,O|\lambda)=\alpha_t(i)\cdot a_{ij}\cdot b_j(o_{t+1})\cdot\beta_{t+1}(j).$

■

关键词 1： 理解在观测O下由状态i转移的期望值为$\sum_{t=1}^{T-1}\gamma_t(i)$

说明 1：在观测O下由状态i转移到j的期望值为：$\sum_{t=1}^{T-1}\xi_t(i,j)$，于是：

$$\sum_{j=1}^{N}\sum_{t=1}^{T-1}\xi_t(i,j)=\sum_{t=1}^{T-1}\sum_{j=1}^{N}\xi_t(i,j)=\sum_{t=1}^{T-1}\sum_{j=1}^{N}P(i_t=q_i,i_{t+1}=q_j,O|\lambda)=\sum_{t=1}^{T-1}P(i_t=q_i,O|\lambda)=\sum_{t=1}^{T-1}\gamma_t(i)$$

■

10.2.2 前向算法（P175-P177）

关键词 2：观测序列生成及前向算法的 R 实现/例 10.2 的程序求解

说明 2：

观测序列的生成算法（参考书本 P174）

前向算法，如下。

输入：隐马尔可夫模型λ，观测序列O；

输出：观测序列概率$P(O|\lambda)$.

（1）初值：$\alpha_1(i)=\pi_i b_i(o_1)$.

（2）递推：对$t=1,2,...,T-1$. 计算

$$\alpha_{t+1}(i)=\left[\sum_{j=1}^{N}\alpha_t(j)a_{ji}\right]b_i(o_{t+1}),i=1,2,...,N.$$

（3）停止：$P(O|\lambda)=\sum_{i=1}^{N}\alpha_T(i),\ \alpha_T(i)=P(o_1,o_2,...,o_T,i_T=q_i|\lambda)$.

OK，接下来就是码农时间了。HMM 模型不太好实现，渣君觉得还是一步步慢慢来吧。这次我用 R 编写了两个比较简单的函数，分别为 ObjectHMM()及 forwardHMM().函数 ObjectHMM()实现了隐马尔可夫模型的观测序列生成算法；forwardHMM()实现了 HMM 模型的前向算法。（详细代码参见附录 10）

ObjectHMM(size=1,Lth=5,A,B,PI,StateLabel=1:nrow(A), ObjectLabel=1:ncol(B),seed=NULL)函数中 size 参数指定随机生成多少组给定 HMM 模型的观测序列；Lth 指定每一组观测序列的长度，是一个长度等于 size 的向量；StateLabel 指定状态的表示方式，默认用 1,2，…等表示；ObjectLabel 指定观测值的表示方式，通常不

使用默认值，而是依据实际情况指定；seed=NULL，是否设定“种子”，即是否在函数 ObjectHMM()内部调用 set.seed()函数，以确保每次在同一个 seed（数值）下生成的随机观测序列是相同的；当 seed=NULL 时，每次运行同一代码生成的观测序列是不一样的。A 为转移概率矩阵；B 为观测概率矩阵；PI 为初始状态概率向量。

forwardHMM(obs,A,B,PI,StateLabel=as.character(1:nrow(A)),ObjectLabel=as.character(1:ncol(B)))函数中 A,B,PI,StateLabel，ObjectLabel 意义与 ObjectHMM()中的一致；obs 指定需要计算的概率的观测序列，obs 在这里必须是一个向量。

我们先来看一下函数 ObjectHMM()的用法。

数据准备：

```
## 数据准备： 利用例 10.1 的数据
a10.1<-matrix(c(0,1,0,0,
                .4,0,.6,0,
                0,.4,0,.6,
                0,0,.5,.5),nrow = 4,byrow = T);a10.1
pi10.1<-rep(.25,4);pi10.1
b10.1<-matrix(c(.5,.5,
                .3,.7,
                .6,.4,
                .8,.2),nrow = 4,byrow = T);b10.1
```

R 代码：

```
ObjectHMM(size=2,Lth = 5,A=a10.1,B=b10.1,PI=pi10.1,
          ObjectLabel = c("红","白"))#生成 2 个长度一样的观测序列
ObjectHMM(size=2,Lth = 5,A=a10.1,B=b10.1,PI=pi10.1,
          ObjectLabel = c("红","白"),seed=66)#这句运行两次，结果一样
ObjectHMM(size=3,Lth = c(3,4,5),A=a10.1,B=b10.1,
          PI=pi10.1,ObjectLabel = c("红","白"))#生成 3 个长度不一样的观测序列
```

结果 1 如下：

```
> ObjectHMM(size=2,Lth = 5,A=a10.1,B=b10.1,PI=pi10.1,
+                 ObjectLabel = c("红","白"))#生成 2 个长度一样的观测序列
$obs
$obs[[1]]
[1] "白" "红" "红" "红" "白"

$obs[[2]]
[1] "红" "红" "白" "白" "白"

$state
$state[[1]]
[1] 3 4 4 4 3

$state[[2]]
[1] 4 3 2 1 2
```

上述结果中，obs 组件就是最终生成的观测序列；state 组件装载了 obs 中对应观测序列的生成过程，即每一个具体的观测值是在哪个状态下的。因为没有设定 seed，所以这句代码每次运行结果不同。

结果 2 如下：

```
> ObjectHMM(size=2,Lth = 5,A=a10.1,B=b10.1,PI=pi10.1,
+                 ObjectLabel = c("红","白"),seed=66)#第一次运行
$obs
$obs[[1]]
[1] "红" "白" "白" "红" "白"

$obs[[2]]
[1] "红" "白" "红" "红" "红"

$state
$state[[1]]
[1] 1 2 1 2 3

$state[[2]]
[1] 3 4 3 4 3
```

```
> ObjectHMM(size=2,Lth = 5,A=a10.1,B=b10.1,PI=pi10.1,
+                 ObjectLabel = c("红","白"),seed=66)#第二次运行
$obs
$obs[[1]]
[1] "红" "白" "白" "红" "白"

$obs[[2]]
[1] "红" "白" "红" "红" "红"

$state
$state[[1]]
[1] 1 2 1 2 3

$state[[2]]
[1] 3 4 3 4 3
```

由结果 2 可见 seed 的作用。

结果 3 如下：

```
> ObjectHMM(size=3,Lth = c(3,4,5),A=a10.1,B=b10.1,
+                 PI=pi10.1,ObjectLabel = c("红","白"))#生成 3 个长度不一样的观测序列
$obs
$obs[[1]]
[1] "白" "红" "红"

$obs[[2]]
[1] "红" "红" "红" "红"

$obs[[3]]
[1] "红" "白" "白" "红" "白"


$state
$state[[1]]
[1] 2 3 4

$state[[2]]
[1] 2 3 4 4

$state[[3]]
[1] 4 3 2 3 4
```

接下来，我们利用函数 forwardHMM()来求解例 10.2.

代码如下：

```
## 例 10.2 的程序求解
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2

forwardHMM(obs=c("红","白","红"),A=A10.2,
           B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
```

结果如下：

```
> forwardHMM(obs=c("红","白","红"),A=A10.2,
+            B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
$FinalProb
[1] 0.130218

$AlpMat
       st:1     st:2     st:3
T:1 0.10000 0.160000 0.280000
T:2 0.07700 0.110400 0.060600
T:3 0.04187 0.035512 0.052836
```

恩，看来我们运气很好嘛，程序求解的结果与书本的结果相同。上述结果中，FinalProb 组件就是我们所求的观测序列在指定 HMM 模型下的出现概率；AlpMat 组件装载了每次迭代计算出来的前向概率值，一行为一次结果，稍作对比，与书本计算结果完全一致。

最后，我们来结合例 10.1 的数据，使用 forwardHMM()函数和 ObjectHMM()函数再做一个例子。

代码如下：

```
hmm1<-ObjectHMM(size=3,Lth = c(3,4,5),A=a10.1,B=b10.1,
                PI=pi10.1,ObjectLabel = c("红","白"));hmm1
hmm1_obs<-hmm1$obs;hmm1_obs
lapply(hmm1_obs,forwardHMM,A=a10.1,B=b10.1,
                PI=pi10.1,ObjectLabel=c("红","白"))
```

结果如下：（省略部分输出）

```
> lapply(hmm1_obs,forwardHMM,A=a10.1,B=b10.1,
+             PI=pi10.1,ObjectLabel=c("红","白"))
[[1]]
[[1]]$FinalProb
[1] 0.19418

[[1]]$AlpMat
        st:1    st:2    st:3    st:4
T:1 0.1250 0.07500 0.15000 0.20000
T:2 0.0150 0.05550 0.08700 0.15200
T:3 0.0111 0.01494 0.06558 0.10256


[[2]]
[[2]]$FinalProb
[1] 0.0517356

[[2]]$AlpMat
           st:1         st:2         st:3         st:4
T:1 0.125000 0.1750000 0.1000000 0.0500000
T:2 0.035000 0.1155000 0.0520000 0.0170000
T:3 0.023100 0.0167400 0.0466800 0.0317600
T:4 0.003348 0.0292404 0.0103696 0.0087776


[[3]]
[[3]]$FinalProb
[1] 0.0269663

[[3]]$AlpMat
              st:1           st:2           st:3           st:4
T:1 0.12500000 0.075000000 0.15000000 0.200000000
```

10.2.3 后向算法（P178）

关键词 3： HMM 模型后向算法的 R 实现/例 10.2 的后向算法求解

说明 3：

后向算法，如下。

输入：隐马尔可夫模型λ，观测序列O；

输出：观测序列概率$P(O|\lambda)$.

（1）初值：$\beta_T(i)=1,i=1,2,...,N.$

（2）递推：对$t=T-1,T-2,...,1.$计算

$$\beta_t(i)=\sum_{j=1}^{N}a_{ij}b_j(o_{t+1})\beta_{t+1}(j),i=1,2,...,N.$$

（3）停止：$P(O|\lambda)=\sum_{i=1}^{N}\pi_i b_i(o_1)\beta_1(i).$

我利用 R 编写了函数 backwardHMM()来实现后向算法。

（详细代码参见附录 10）

backwardHMM(obs,A,B,PI,StateLabel=as.character(1:nrow(A)),ObjectLabel=as.character(1:ncol(B)))函数中，所有参数的含义与 forwardHMM()函数中的参数相同，不再一一赘述。那么，怎么去检验这个函数计算结果的正确性呢？书本并没有单独列举后向算法计算的例子。但是，由于前向算法、后向算法仅仅是计算同一个观测序列在指定 HMM 模型下出现概率的不同方法而已，所以，理论上，两种不同的方法的计算出来的概率应该相同。按照这种想法，我们可以利用例 10.2 来检验函数 backwardHMM()的计算是否正确。

代码如下：

```
## 例 10.2 的程序求解 后向算法
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2
backwardHMM(obs=c("红","白","红"),A=A10.2,
            B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
```

结果如下：

```
> backwardHMM(obs=c("红","白","红"),A=A10.2,
+             B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
$FinalProb
[1] 0.130218

$BtMat
       st:1   st:2   st:3
T:3 1.0000 1.0000 1.0000
T:2 0.5400 0.4900 0.5700
T:1 0.2451 0.2622 0.2277
```

和 forwardHMM()函数的结果对比，发现 backwardHMM()函数与 forwardHMM()函数的概率计算结果是相同的，只是计算过程以及迭代顺序不同。backwardHMM()函数中的迭代是从最后一个观测开始的，而 forwardHMM()函数中的迭代是从第一个观测开始的。上述结果中，组件 BtMat 记录了从后往前每次迭代的结果，一行为一次结果。

我们利用 ObjectHMM()函数再多做一个例子来检验。

代码如下：

```
hmm<-ObjectHMM(size=2,Lth = c(3,4),A=A10.2,B=B10.2,
                PI=pi10.2,ObjectLabel = c("红","白"))#生成隐马尔可夫观测序列
hmm_obs<-hmm$obs;hmm_obs
lapply(hmm_obs,forwardHMM,A=A10.2,B=B10.2,PI=pi10.2,ObjectLabel=c("红","白"))
lapply(hmm_obs,backwardHMM,A=A10.2,B=B10.2, PI=pi10.2,ObjectLabel=c("红","白"))
```

结果如下：

```
>
lapply(hmm_obs,forwardHMM,A=A10.2,B=B
10.2,
+               PI=pi10.2,ObjectLabel=c("红","白
"))
[[1]]
[[1]]$FinalProb
[1] 0.128978

[[1]]$AlpMat
          st:1        st:2        st:3
T:1 0.10000 0.240000 0.120000
T:2 0.07300 0.070400 0.096600
T:3 0.03847 0.031512 0.058996


[[2]]
[[2]]$FinalProb
[1] 0.0530488

[[2]]$AlpMat
             st:1          st:2          st:3
T:1 0.1000000 0.24000000 0.12000000
T:2 0.0730000 0.10560000 0.04140000
T:3 0.0382300 0.03192800 0.04460400
T:4 0.0188071 0.02219472 0.01204698
```

```
>
lapply(hmm_obs,backwardHMM,A=A10.2,B
=B10.2,
+               PI=pi10.2,ObjectLabel=c("红","白
"))
[[1]]
[[1]]$FinalProb
[1] 0.128978

[[1]]$BtMat
          st:1      st:2      st:3
T:3 1.0000 1.0000 1.0000
T:2 0.5400 0.4900 0.5700
T:1 0.2939 0.2588 0.3123


[[2]]
[[2]]$FinalProb
[1] 0.0530488

[[2]]$BtMat
             st:1        st:2        st:3
T:4 1.000000 1.000000 1.000000
T:3 0.460000 0.510000 0.430000
T:2 0.246100 0.231200 0.257700
T:1 0.112462 0.121737 0.104881
```

这么看来，后向算法的函数应该是没有编错。

■

10.2.4 一些概率与期望值的计算（P179-P178）

关键词 3： $\gamma_t(i)$和$\xi_t(i,j)$计算的 R 实现

说明 3：

$\gamma_t(i)$为给定模型λ和观测O，在时刻t处于状态q_i（或i）的概率.

$\xi_t(i,j)$为给定模型λ和观测O，在时刻t处于状态q_i（或i）且在时刻$t+1$处于状态q_j（或j）的概率（转移概率）。根据书本公式有：

$$(1)\ \ \gamma_t(i)=P\left(i_t=q_i|O,\lambda\right)=\frac{\alpha_t(i)\beta_t(i)}{P\left(O|\lambda\right)}=\frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}$$

$$(2)\ \ \xi_t(i,j)=P\left(i_t=q_i,i_t=q_i|O,\lambda\right)$$

$$=\frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P\left(O|\lambda\right)}=\frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}$$

我利用 R 编写了函数 stijHMM()来计算以上这两个概率，在函数 stijHMM()内部调用了 forwardHMM()和 backwardHMM()函数。我在编写函数的过程中都尽量避免使用循环，大部分计算过程是通过 R 的向量化运算实现的，这样会牺牲一些代码可读性。因此小伙们看代码时得有点耐心哈。现在先来简单介绍一下这个函数的用法。（详细代码参见附录 10）

stijHMM(obs,sti=NULL,stij=NULL,time=1,A,B,PI,StateLabel=as.character(1:nrow(A)),ObjectLabel=as.character(1:ncol(B)),if.sti=F,if.stij=F)函数中参数 obs, A,B,PI,StateLabel 以及 ObjectLabel 与 forwardHMM()和 backwardHMM()中同名参数的含义相同，不再赘述；sti 输入需要计算概率的第“i”个状态，sti 是一个

字符或一个字符串向量；stij 输入需要转移概率的 t 时刻的状态“i”,t+1 时刻的状态“j”，stij 是一个二维字符向量，第一维表示 t 时刻的状态“i”，第二维表示 t+1 时刻的状态“j”；time 输入所谓的“t”时刻，是数值或数值向量，当 time=NULL 时，函数结果只输出 sti，stij 对应的所有的时刻“t”的结果;if.sti=F 指定是否只计算 sti;if.stij=F 指定是否只计算转移概率 stij；if.sti,if.stij 不能同时为 TRUE.

函数 stijHMM()编写完成后，问题就来了。思考一下，我们究竟要怎样去检验 stijHMM()算结果的正确性呢？书本没有提到这一点。但是,作为一名勤勤恳恳的学渣,渣君总不能因为书本没有提就放弃吧。如果 stijHMM()结果有误,那么接下来的 HMM 模型编写就没有意义了，真是细思极恐。其实，我们可以从$\gamma_t(i)$和$\xi_t(i,j)$的某些简单的性质来确定一些较为粗糙的检验方法。思路如下：

首先，针对$\gamma_t(i)$，我们知道必有：

$$[1]\ \sum_{j=1}^{N}\gamma_t(j)=1,\ t=1,2,...,T.$$

另外，令$\delta(j)=\sum_{t=1}^{T}\gamma_t(j)$，则一般情况下应有：

$$[2]\ \ \delta(j)\neq C, j=1,2,...,N.$$

否则，若$\delta(j)=C, j=1,2,...,N.\ \sum_{t=1}^{T}\gamma_t(j)$就失去了计算对比的意义。

最后，针对$\xi_t(i,j)$，利用本章笔记“关键词 1”的结论有：

$$[3]\ \sum_{j=1}^{N}\sum_{t=1}^{T-1}\xi_t(i,j)=\sum_{t=1}^{T-1}\gamma_t(i)$$

于是，可以通过以下的步骤检验函数 stijHMM()计算结果：

步骤一：通过[1]、[2]检验$\gamma_t(i)$的正确性

步骤二：若$\gamma_t(i)$正确，通过[3]检验$\xi_t(i,j)$的正确性

我们就函数就上述方法进行检验，代码如下。

```
##   检查函数的正确性
stij1<-stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","1"),time=1,A=A10.2,
                    B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
stij2<-stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","2"),time=1,A=A10.2,
                    B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
stij3<-stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","3"),time=1,A=A10.2,
                    B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"))
apply(stij1$PstiMat,1,sum)#检验[1]
apply(stij1$PstiMat,2,sum)#检验[2]
sum(c(stij1$Pstijvec,stij2$Pstijvec,stij3$Pstijvec))#检验[3]
sum(stij1$PstiMat[1:2,1]) #检验[3]
```

结果如下：

```
> apply(stij1$PstiMat,1,sum) #检验[1]
T:1 T:2 T:3
  1   1   1
> apply(stij1$PstiMat,2,sum) #检验[2]
      st:1       st:2       st:3
0.8290712 1.0103058 1.1606230
> sum(c(stij1$Pstijvec,stij2$Pstijvec,stij3$Pstijvec)) #检验[3]
[1] 0.5075335
> sum(stij1$PstiMat[1:2,1]) #检验[3]
[1] 0.5075335
```

可见，三个检验均认为 stijHMM()计算的结果是正确的。要理解上述结果，就得先了解一下 stijHMM()输出结果中各个组件的含义。结果中我保存了较多的组件，这主要是为了之后编写 HMM 模型时避免重复计算。用 names()查看 stijHMM()输出结果的组件。代码如下：

```
> names(stij1)
 [1] "Probs"     "psti"      "pstij"     "PstiMat"   "Pstijvec"
 [6] "AlpMat"    "BtMat"     "sti"       "stij"      "time"
```

sti,stij,time 这几个组件仅仅负责将参数输入保存下来，以备后用。其他组件的含义，归纳如下：

$$\text{Probs} = P\left(O\middle|\lambda\right)$$
$$\text{psti} = \gamma_t\left(i\right),\ t = \text{time},\ i = \text{sti}.$$
$$\text{pstij} = \xi_t\left(i,j\right),\ t = \text{time},\ i = \text{stij}[1],\ j = \text{stij}[2].$$
$$\text{PstiMat} = \left[\gamma_t\left(i\right)\right]_{T\times N}, i = 1,2,...,N;\ t = 1,2,...,T.$$
$$\text{Pstijvec} = \left(\xi_t\left(i,j\right)\right),\ t = 1,2,...,T-1;\ i = \text{stij}[1],\ j = \text{stij}[2].$$
$$\text{AlpMat} = \left[\alpha_t\left(i\right)\right]_{T\times N}, i = 1,2,...,N;\ t = 1,2,...,T.$$
$$\text{BtMat} = \left[\beta_t\left(i\right)\right]_{T\times N}, i = 1,2,...,N;\ t = 1,2,...,T.$$

上面的代码就是将 stij[1]固定为“1”，stij[2]遍历各个状态（“1”，“2”，“3”）后计算的结果。对照上述组件的含义，小伙伴们应该能很容易明白各行代码的作用，本渣就不在此赘述了。稍微注意一下就是，stijHMM()结果中的组件 BtMat 与 backwardHMM()结果中的组件 BtMat 有些不同，stijHMM()的 BtMat 的行是按时间顺序排的，而函数 backwardHMM()中的 BtMat 的行是按时间逆序排的。

再看一下 if.sti 以及 if.stij 参数的作用，这两个参数的目的主要是为了在编写 HMM 的非监督算法时减少不必要的运算。

代码如下：

```
stijHMM(obs=c("红","白","红"),sti="1",stij=c("2","3"),time=NULL,A=A10.2,
                B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"),if.sti = TRUE)

stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","1"),time=NULL,A=A10.2,
        B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"),if.stij=TRUE)

stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","1"),time=NULL,A=A10.2,
        B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"), if.sti = TRUE ,if.stij=TRUE)
```

结果如下：

```
> stijHMM(obs=c("红","白","红"),sti="1",
+    stij=c("2","3"),time=NULL,A=A10.2,
+       B=B10.2,PI=pi10.2,ObjectLabel=c("
红","白")
+    ,if.sti = TRUE)
$Probs
[1] 0.130218

$psti
NULL

$PstiMat
                st:1          st:2          st:3
T:1 0.1882228 0.3221674 0.4896097
T:2 0.3193107 0.4154264 0.2652629
T:3 0.3215377 0.2727119 0.4057504

$AlpMat
            st:1         st:2         st:3
T:1 0.10000 0.160000 0.280000
T:2 0.07700 0.110400 0.060600
T:3 0.04187 0.035512 0.052836

$BtMat
          st:1     st:2     st:3
T:1 0.2451 0.2622 0.2277
T:2 0.5400 0.4900 0.5700
T:3 1.0000 1.0000 1.0000
```

```
> stijHMM(obs=c("红","白","红"),sti="1",
+      stij=c("1","1"),time=NULL,A=A10.2,
+       B=B10.2,PI=pi10.2,ObjectLabel=c("红
","白"),
+      if.stij=TRUE)
$Probs
[1] 0.130218

$pstij
NULL

$Pstijvec
          T:1           T:2
0.1036723 0.1478290

$AlpMat
            st:1         st:2         st:3
T:1 0.10000 0.160000 0.280000
T:2 0.07700 0.110400 0.060600
T:3 0.04187 0.035512 0.052836

$BtMat
          st:1     st:2     st:3
T:1 0.2451 0.2622 0.2277
T:2 0.5400 0.4900 0.5700
T:3 1.0000 1.0000 1.0000
```

```
> stijHMM(obs=c("红","白","红"),sti="1",stij=c("1","1"),time=NULL,A=A10.2,
+                B=B10.2,PI=pi10.2,ObjectLabel = c("红","白"), if.sti = TRUE ,if.stij=TRUE)
Error in stijHMM(obs = c("红", "白", "红"), sti = "1", stij = c("1", "1"),    :
  if.sti and if.stij can not both TRUE.
```

10.3.1 监督学习方法（P180）

关键词 4： 监督学习方法的 R 实现

说明 4：

监督学习方法是指在观测序列和对应的状态序列均为已知的情况下，采用极大似然估计法来估计隐马尔可夫模型的参数。如下：

输入：S 个同长度的观测/状态序列$\{(O_1,I_1),(O_2,I_2),...,(O_S,I_S)\}$

输出：

（1）转移概率a_{ij}的估计：

设样本中时刻t处于状态i时刻$t+1$转移到状态 j 的频数为A_{ij}，则转移概率a_{ij}的估计为：

$$\hat{a}_{ij}=\frac{A_{ij}}{\sum_{j=1}^{N}A_{ij}},\ i=1,2,...,N;\ j=1,2,...,N.$$

（2）观测概率$b_j(k)$的估计：

设样本中状态为 j 并观测为k 的频数是B_{jk}，那么状态为 j 观测为k 的概率$b_j(k)$的估计是：

$$\widehat{b_j}(k)=\frac{B_{jk}}{\sum_{k=1}^{M}B_{jk}},\ j=1,2,...,N;\ k=1,2,...,M.$$

（3）初始状态概率π_i的估计$\hat{\pi}_i$为S 个样本中初始状态为i的频率。

书本在介绍这个算法时，有一个小点讲得不是很清楚。我来写写自己的思路。在（1）中，我们知道时刻t是可变的，对于转移概率，时间长度范围为：$t=1,2,...,T-1.$但是书本并没有明确提到频数A_{ij}的计算方法是针对某个特定的时刻t还是说需要遍历$t=1,2,...,T-1.$渣君在

网上查了一下，也还没找到明确的说法。不过我认为后一种比较合理。因为，按照前一种思路，在不同的时刻t会估计出不同的转移概率$\hat{a}_{ij}$，这显然不合理。举个例子：

$$\begin{pmatrix} i_{11} & \cdots & i_{1S} \\ \vdots & \ddots & \vdots \\ i_{T1} & \cdots & i_{TS} \end{pmatrix}_{T\times S} \Rightarrow e.g: \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 3 & 3 \end{pmatrix}_{T\times S}, T=3, S=3, I=\{1,2,3\}.$$

$$(1)\begin{cases} t=1\text{时}, i=1, j=2 \Rightarrow A_{12}=1 \\ t=1\text{时}, i=1, j=1 \Rightarrow A_{11}=0 \\ t=1\text{时}, i=1, j=3 \Rightarrow A_{13}=0 \\ \Rightarrow \hat{a}_{12}=A_{12}/\sum_{j=1}^{3}A_{1j}=1 \ \ ??? \end{cases} \quad \begin{cases} t=2\text{时}, i=1, j=2 \Rightarrow A_{12}=0 \\ t=2\text{时}, i=1, j=1 \Rightarrow A_{11}=0 \\ t=2\text{时}, i=1, j=3 \Rightarrow A_{13}=2 \\ \Rightarrow \hat{a}_{12}=A_{12}/\sum_{j=1}^{3}A_{1j}=0 \ \ ??? \end{cases}$$

$$(2)\hat{a}_{12}=(1+0)/\left[(0+0)+(1+0)+(0+2)\right]=\frac{1}{3}$$

可见，按照前一种方法估计转移概率会容易得出极端估计及不确定估计；而采用后一种方法计算的结果是确定的，结果也更为合理。后一种方法中，对于转移概率，要遍历的时间为：$t=1,2,...,T-1;T=3$.我在实现 HMM 模型的监督学习算法时，估计转移概率采用的是后一种方法。

我用 R 编写了函数 superviseHMM()和 print.superviseHMM()函数来实现 HMM 模型的监督学习算法。print.superviseHMM()函数指定训练结果的打印方式。（详细代码参见附录 10）

superviseHMM(obsMat,stMat,StateLabel=NULL,ObjectLabel=NULL)函数中，obsMat 是指观测序列组成的矩阵，行数表示序列的时间长度，列数表示样本量，obsMat 必须按照这种方式存储；stMat 是指状态序列组成的矩阵，行数表示序列的时间长度，列数表示样本量，

stMat 必须按照这种方式存储。StateLabel、ObjectLabel 参数的含义用法均与之前的函数中的一样。

那么，我们要怎样来测试这个函数呢？到哪里去找适用于隐马尔可夫模型的数据？不用担心这个问题，还记得函数 ObjectHMM()吗，它可用于生成指定模型参数的随机观测序列及状态序列。于是我们可以利用 ObjectHMM()函数及例 10.2 给定的模型参数生成一批随机数据，并用 superviseHMM()函数对这批数据进行拟合，以此来检查函数 superviseHMM()计算结果的正确性及拟合效果。

数据准备代码：

```
## 测试   利用 ObjectHMM()函数生成数据
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2

## 生成 2000 组隐马尔可夫观测序列及状态序列
test1<-ObjectHMM(size=2000,Lth = 5,A=A10.2,B=B10.2,PI=pi10.2,
                 StateLabel = as.character(1:3),ObjectLabel = c("红","白"))
```

模型训练代码：

```
obs1<-test1$obs
st1<-test1$state
obsmat<-do.call(cbind,obs1)#观测矩阵
stmat<-do.call(cbind,st1)#状态矩阵

shmm1<-superviseHMM(obsMat = obsmat,stMat = stmat,
          StateLabel =as.character(1:3),ObjectLabel =c("红","白") )#训练模型
shmm1
```

模型训练结果：

```
> shmm1#训练结果
State:: 1 2 3 ;   Observation:: 红 白
$pi
[1] 0.1875 0.4020 0.4105

$aijMat
          1         2         3
1 0.5184382 0.1817787 0.2997831
2 0.2933057 0.5165631 0.1901311
3 0.2052199 0.2971040 0.4976761

$bjkMat
          红         白
1 0.5056780 0.4943220
2 0.4058012 0.5941988
3 0.7062229 0.2937771
```

```
#真实参数
> pi10.2
[1] 0.2 0.4 0.4
> A10.2
     [,1] [,2] [,3]
[1,]  0.5  0.2  0.3
[2,]  0.3  0.5  0.2
[3,]  0.2  0.3  0.5
> B10.2
     [,1] [,2]
[1,]  0.5  0.5
[2,]  0.4  0.6
[3,]  0.7  0.3
```

由以上结果可以发现，当样本量较大时（这里是 2000），监督学习方法估计出来的参数与真实参数非常接近，具有非常好的拟合效果。我们还可以尝试一下生成其他的数据集来测试 superviseHMM()，应该可以得到类似的结论。

代码如下：

```
#更改模拟的真实参数
A10<-matrix(c(0.3,0.3,0.4,0.4,0.3,0.3,0.1,0.7,0.2),nrow=3,byrow = T);A10.2
B10<-matrix(c(0.8,0.2,0.3,0.7,0.4,0.6),nrow = 3,byrow = T);B10.2
pi10<-c(0.3,0.6,0.1);pi10
## 生成隐马尔可夫观测序列及状态序列
test2<-ObjectHMM(size=2000,Lth = 5,A=A10,B=B10,PI=pi10,
                 StateLabel = as.character(1:3),ObjectLabel = c("红","白"))
obs2<-test2$obs
st2<-test2$state
obsmat2<-do.call(cbind,obs2)#观测矩阵
stmat2<-do.call(cbind,st2)#状态矩阵
shmm2<-superviseHMM(obsMat = obsmat2,stMat = stmat2,
                    StateLabel =as.character(1:3),ObjectLabel =c("红","白") )#训练模型
shmm2
```

结果如下：

```
> shmm2#训练结果
State:: 1 2 3 ;   Observation:: 红 白
$pi
[1] 0.2970 0.6125 0.0905

$aijMat
          1         2         3
1 0.3108723 0.3009508 0.3881769
2 0.4060742 0.2871204 0.3068054
3 0.1170370 0.6874074 0.1955556

$bjkMat
         红         白
1 0.8001321 0.1998679
2 0.3135593 0.6864407
3 0.4027618 0.5972382
```

```
#真实参数
> pi10
[1] 0.3 0.6 0.1
> A10
     [,1] [,2] [,3]
[1,]  0.3  0.3  0.4
[2,]  0.4  0.3  0.3
[3,]  0.1  0.7  0.2
> B10
     [,1] [,2]
[1,]  0.8  0.2
[2,]  0.3  0.7
[3,]  0.4  0.6
```

易见，估计出来的参数和真实参数很接近。

■

10.3.2 Baum-Welch 算法（P181-P184）

关键词 5： Baum-Welch 无监督学习算法的 R 实现

说明 5：

假设给定的训练数据只包含 S 个长度为 T 的观测序列 $\{O_1,O_2,...,O_S\}$ 而没有对应状态序列。

输入：观测数据 $O=(o_1,o_2,...,o_T)$

输出：隐马尔可夫模型参数

（1）初始化

对 $n=0$，选取 $a_{ij}^{(0)},b_j(k)^{(0)},\pi_i^{(0)}$，得到模型 $\lambda^{(0)}=\left(A^{(0)},B^{(0)},\pi^{(0)}\right)$

（2）递推. 对 $n=1,2,...,$

$$a_{ij}^{(n+1)}=\frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)},\quad b_j(k)^{(n+1)}=\frac{\sum_{t=1,o_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)},\quad \pi_i^{(n+1)}=\gamma_t(i)$$

右端各值按观测 $O=(o_1,o_2,...,o_T)$ 和模型 $\lambda^{(n)}=\left(A^{(n)},B^{(n)},\pi^{(n)}\right)$ 计算.

（3）终止. 得到模型参数 $\lambda^{(n+1)}=\left(A^{(n+1)},B^{(n+1)},\pi^{(n+1)}\right)$

对于上面这个算法，我有一些疑问。算法的输入只涉及一个观测序列，但事实上，我们收集了 S 个观测序列。那么剩下的 S-1 个观测序列要怎么才能用上呢？书本没有讲清楚这一点。按照这个算法的思路，每一个观测序列就能训练出一个 HMM 模型，S 个观测序列就会训练出 S 个 HMM 模型，那么那个比较准确呢？如何选择最优的模型呢？嗯，渣君想了想，也稍微查了一下。我的答案是，不知道。如果书本的算法没有错的话，它的意思应该就是一个观测序列训练出一个 HMM

模型，根据 EM 算法的特点，训练出来的这个模型仅仅是局部最优的，而且初始值的不同会影响迭代的结果。我用 R 实现了基于一个观测的 BuamWelch 算法，对于多个有用的观测，我采取的简单粗暴的取平均值的方法。就是对于 S 个观测序列出来的 S 个 HMM 模型，我们对其每个参数对应位置的 S 个估计取平均值，作为这个参数的最终估计。我对这样的方法进行了测试，发现估计参数与真实参数的差距较大，效果没有使用监督学习方法时理想。算法停止的准则书本也没详细讲，我参考 EM 算法的停止准则，当估计出的前后两个参数序列足够接近时停止迭代。

我编写了函数 BuamWelchHMM()、MoreBuamWelchHMM()以及函数 Print.BuamWelchHMM()来实现上述的想法。BuamWelchHMM()函数只能对一个观测序列进行训练，得到一个 HMM 模型；MoreBuamWelchHMM()对多个观测进行训练，取其平均估计作为最终的参数估计。函数 Print.BuamWelchHMM()指定打印方式。（详细代码参见附录 10）

MoreBuamWelchHMM(obsMat,A0,B0,PI0,StateLabel=as.character(1:nrow(A0)),ObjectLabel=as.character(1:ncol(B0)),ept=1e-2,maxiter=10000,Lp=2)函数中 obsMat 是观测矩阵，行数表示序列的时间长度，列数表示样本量，obsMat 必须按照这种方式存储. A0,B0,PI0 是初始值；Lp 是指定停止条件使用的范数；ept 指定精度。其余参数与之前使用过的函数中相同参数的含义一样。

BuamWelchHMM(obs,…)用法和 MoreBuamWelchHMM()一样，区别仅仅是 obs 为一个观测序列向量。

我们接下来，用一个例子来测试一下这两个函数。

数据准备：

```
## 训练时使用的初始化参数
A1<-matrix(c(0.5,0.3,0.2,
             0.4,0.4,0.2,
             0.1,0.2,0.7),nrow=3,byrow = T);A10.2
B1<-matrix(c(0.6,0.4,
             0.3,0.7,
             0.4,0.6),nrow = 3,byrow = T);B10.2
pi1<-c(0.3,0.3,0.4);pi1
## 生成隐马尔可夫观测序列及状态序列的参数
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2
## 生成 2000 组隐马尔可夫观测序列及状态序列
test1<-ObjectHMM(size=2000,Lth = 5,A=A10.2,B=B10.2,PI=pi10.2,
                 StateLabel = as.character(1:3),ObjectLabel = c("红","白"))
obs1<-test1$obs
st1<-test1$state
obsmat<-do.call(cbind,obs1)#观测矩阵
```

如以上代码所示，我们利用 ObjectHMM()函数和例 10.2 给定的参数随机生成 2000 组观测序列，并将 2000 组观测序列储存在指定的矩阵里。我们将利用这 2000 组观测序列来做测试，并以 A1，B1，pi1 作为初始化参数，训练隐马尔可夫模型。

模型训练代码：

```
BuamWelchHMM(obs=obsmat[,1],A0=A1,B0=B1,PI0=pi1,
             ObjectLabel =  c("红","白"),ept = 1e-2)#仅使用一个观测序列
BuamWelchHMM(obs=obsmat[,10],A0=A1,B0=B1,PI0=pi1,
             ObjectLabel =  c("红","白"),ept = 1e-2)#使用另一个观测序列
MoreBuamWelchHMM(obsMat = obsmat,A0=A1,B0=B1,PI0=pi1,
                 ObjectLabel =  c("红","白"),ept = 1e-2)#使用所有观测序列
```

模型 1 结果如下：

```
> BuamWelchHMM(obs=obsmat[,1],A0=A1,B0=B1,Pl0=pi1,
+                     ObjectLabel =    c("红","白"),ept = 1e-2)
State:: 1 2 3 ;   Observation:: 红 白
$pi
       st:1       st:2       st:3
0.3890536 0.2063683 0.4045781

$aijMat
                [,1]       [,2]       [,3]
[1,] 0.41626161 0.3963571 0.2090570
[2,] 0.38127254 0.4206943 0.2006476
[3,] 0.08847808 0.2299958 0.6744256

$bjkMat
              [,1]      [,2]
[1,] 0.5895796 0.4118752
[2,] 0.2335789 0.7989355
[3,] 0.4097492 0.5886133
```

模型 2 结果如下：

```
> BuamWelchHMM(obs=obsmat[,10],A0=A1,B0=B1,Pl0=pi1,
+                     ObjectLabel =    c("红","白"),ept = 1e-2)
State:: 1 2 3 ;   Observation:: 红 白
$pi
       st:1       st:2       st:3
0.4733334 0.1320404 0.3946262

$aijMat
              [,1]      [,2]      [,3]
[1,] 0.4830922 0.3402503 0.1817940
[2,] 0.4607611 0.3940093 0.1381377
[3,] 0.1165423 0.2428466 0.6229280

$bjkMat
              [,1]      [,2]
[1,] 0.7815387 0.2156843
[2,] 0.3406534 0.6704480
[3,] 0.5963210 0.3923584
```

模型 3 结果：

```
> MoreBuamWelchHMM(obsMat = obsmat,A0=A1,B0=B1,PI0=pi1,
+                          ObjectLabel =   c("红","白"),ept = 1e-2)
State:: 1 2 3 ;   Observation:: 红 白
$pi
       st:1         st:2         st:3
0.3870926 0.2743239 0.3385834

$aijMat
             [,1]         [,2]         [,3]
[1,] 0.4846068 0.2840203 0.2354732
[2,] 0.5585095 0.3116410 0.1288155
[3,] 0.1280099 0.4483946 0.4172317

$bjkMat
             [,1]         [,2]
[1,] 0.7889034 0.2132785
[2,] 0.2039376 0.8046028
[3,] 0.5827148 0.4179189
```

我们可以看到，对于同样的初始值，使用不同的观测序列训练出来的结果是不一样的。当我们使用所有的观测序列进行训练时，训练出的模型与真实参数有一定的差距，反而与初始化参数较为接近。所以在这个测试中 EM 算法的作用是对初始化参数进行局部修改，之后每一步对上一步的结果进行逐步修改，使得更新的参数更接近真实参数，但这仅仅是局部最优的，不是全局最优的。

我检查了两遍程序，认为程序应该没有编错。而训练结果与真实参数差距较大，我认为可能有两个方面的原因。首先，可能是我对这个算法理解有偏误（我暂时没花时间找更详细的资料），比如说，我认为 BuamWelch 算法只利用了一个观测，但完整的详细的算法是否真如我所想，是否可能是书本笔误，有待查证；另一点就是，迭代停止

条件的设置，我并没有去找 BuamWelch 算法明确的停止条件，书本也没提到，我仅仅参考 EM 算法的停止思路，“构造”了一个停止条件，而这个停止条件是否合适，也尚未可知。

总的来说，非监督学习方法不如监督学习方法准确；又根据，函数 MoreBuamWelchHMM()训练结果的特点（即训练结果与初始参数有一定程度的接近），我们在实际应用时，应该根据已有经验，先大致确定出一组初始参数，而不是盲目使用初始参数，然后再使用函数 MoreBuamWelchHMM()对初始参数进行局部修正，最后，得到更合理的 HMM 模型。对 BuamWelch 算法更多的尝试就留给小伙伴们了，时间有限，本渣决定弃疗了。

■

10.4.1 近似算法（P184）

关键词 6： 近似算法的 R 实现

说明 6：

近似算法的想法是，在每个时刻t选择在该时刻最有可能出现的状态i_t^*，从而得到一个状态序列$I^*=(i_1^*,i_2^*,...,i_T^*)$，将它作为预测的结果。

给定隐马尔可夫模型λ和观测序列O，计算在t时刻处于状态q_i的概率：

$$\gamma_t(i)=P(i_t=q_i|O,\lambda)=\frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

在每一时刻t最有可能的状态i_t^*是:

$$i_t^*=\arg\max_{1\le i\le N}\left[\gamma_t(i)\right],\quad i=1,2,...,T.$$

从而得到状态序列$I^*=(i_1^*,i_2^*,...,i_T^*)$.

近似算法的优点是计算简单，其缺点是不能保证预测的状态序列整体是最有可能的状态序列，但近似算法还是有用的。曾记否，我们在“关键词 3”中，已经编写了函数 stijHMM()实现了$\gamma_t(i)$的计算，因此要进一步实现近似算法用于预测，就变得非常简单了。我用 R 编写了函数 approxHMM()实现了这个预测算法。（详细代码参见附录 10）

approxHMM(obsMat,A,B,PI,StateLabel=as.character(1:nrow(A)),ObjectLabel=as.character(1:ncol(B)))函数中所有参数的意义与之前使用过的函数中同名参数的意义一致，不再赘述。

我们使用 ObejctHMM()函数来随机生成几批不同的数据，并用这些数据来测试一下近似算法的预测效率。代码如下。

测试 1 代码：

```
#### 测试 1 数据准备
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2
## 生成隐马尔可夫观测序列及状态序列
test1<-ObjectHMM(size=100,Lth = 100,A=A10.2,B=B10.2,PI=pi10.2,
                 StateLabel = as.character(1:3),ObjectLabel = c("红","白"),seed=100)
obs1<-test1$obs
st1<-test1$state
obsmat<-do.call(cbind,obs1)#观测矩阵
stmat<-do.call(cbind,st1)#状态矩阵

t1<-approxHMM(obsMat = obsmat,A=A10.2,B=B10.2,PI=pi10.2,
                          ObjectLabel = c("红","白"))#近似算法预测
sum(t1==stmat)/10000#预测准确率
```

测试 1 结果：

```
> t1<-approxHMM(obsMat = obsmat,A=A10.2,B=B10.2,
+                       PI=pi10.2,ObjectLabel = c("红","白"))#近似算法预测
> sum(t1==stmat)/10000#预测准确率
[1] 0.4403
```

OK，我现在来解释一下这个预测准确率是怎么回事。这里的预测准确率是按照单个时刻的状态来核对计算的，不是整个序列的状态一起核对的。经过多次测试，发现在这个指定的隐马尔可夫模型下，预测的准确率稳定在 0.43~0.51 的范围。小伙伴们这下子应该要提出疑问了，准确率最高就 0.5，不是和随机乱蒙差不多嘛，这近似算法哪来的“有用”之说？哎，真的是这样吗？其实不然，我们知道，测试 1 指定的模型是一个 3 状态模型，对于任意一个时刻的一个观测值，因此，我们随机乱蒙一个状态能够蒙对的概率应该为 1/3（0.33），而

不是 0.5，所以，近似算法还是有用的，它的将该模型的预测效率比随机猜测提高了（0.4403-1/3）/(1/3)= 32.1%.那么，是否对于不同的模型，近似算法都会有类似稳定的效果呢？我们来再做 2 个测试。

测试 2 代码如下：

```
#### 测试 2 数据准备  增加了状态数 5 个状态
A10<-matrix(c(0.3,0.2,0.2,0,0.3,
              0.1,0.2,0.3,0.3,0.1,
              0.2,0.2,0.3,0.15,0.15,
              0.2,0.1,0.1,0.3,0.4,
              0.1,0.2,0.3,0.3,0.1),nrow=5,byrow = T);A10
B10<-matrix(c(0.5,0.5,
              0.4,0.6,
              0.7,0.3,
              0.4,0.6,
              0.45,0.55),nrow = 5,byrow = T);B10
pi10<-c(0.2,0.1,0.25,0.2,0.25);pi10
## 生成隐马尔可夫观测序列及状态序列
test2<-ObjectHMM(size=100,Lth = 100,A=A10,B=B10,PI=pi10,
                 StateLabel = as.character(1:5),ObjectLabel = c("红","白"),seed=888)
obs2<-test2$obs
st2<-test2$state
obsmat2<-do.call(cbind,obs2)#观测矩阵
stmat2<-do.call(cbind,st2)#状态矩阵

t2<-approxHMM(obsMat = obsmat2,A=A10,B=B10,PI=pi10,ObjectLabel = c("红","白"))
sum(t2==stmat2)/10000#预测准确率
```

测试 2 结果如下：

```
> t2<-approxHMM(obsMat = obsmat2,A=A10,B=B10,
+               PI=pi10,ObjectLabel = c("红","白")))#近似算法预测
> sum(t2==stmat2)/10000#预测准确率
[1] 0.2901
```

可见，状态数增加后，预测准确率明显降低了，我做了多次测试，预测准确了基本稳定在 0.28~0.31 的范围。由于状态取值个数为 5，因此随机猜测的准确率为 0.2，近似算法将预测效率比随机猜测时提高了约（0.2901-0.2）/0.2=45.05%.

测试 3 代码：

```
#### 测试 3 减少状态个数 状态个数为 2
a10<-matrix(c(0.4,0.6,
              0.7,0.3),nrow = 2,byrow = T);a10
b10<-matrix(c(0.5,0.5,
              0.3,0.7),nrow = 2,byrow = T);b10
p10<-c(0.3,0.7);p10
## 生成隐马尔可夫观测序列及状态序列
test3<-ObjectHMM(size=100,Lth = 100,A=a10,B=a10,PI=p10,
                 StateLabel = as.character(1:2),ObjectLabel = c("红","白"),seed=666)
obs3<-test3$obs
st3<-test3$state
obsmat3<-do.call(cbind,obs3)#观测矩阵
stmat3<-do.call(cbind,st3)#状态矩阵

t3<-approxHMM(obsMat = obsmat3,A=a10,B=a10,PI=p10,ObjectLabel = c("红","白"))
sum(t3==stmat3)/10000#预测准确率
```

测试 3 结果：

```
> t3<-approxHMM(obsMat = obsmat3,A=a10,B=a10,
+               PI=p10,ObjectLabel = c("红","白"))#近似算法预测
> sum(t3==stmat3)/10000#预测准确率
[1] 0.6434
```

这次预测准确率在 0.6 以上!!! 不过，先别那么激动，近似算法在这里对预测效率的提升效果为(0.6434-0.5)/0.5=28.68%，预测效率的提升不如前两个测试明显。把上述结果列表如下：（样本为 100 组长度为 100 的观测序列）

	测试 3	测试 1	测试 2
状态数	2	3	5
随机猜测准确率	0.5	0.333	0.2
近似算法准确率	0.63~0.67	0.43~0.51	0.28~0.31
预测效率提升	26%~34%	33.3%~54.54%	40%~55%

■

10.4.2 维特比算法（P185-P186）

关键词 7： 维特比算法的 R 实现

说明 7：

定义以下 3 个变量：

$$\delta_{t+1}(i)=\max_{i_1,i_2,...,i_t} P\left(i_{t+1}=i,i_t,...,i_1,o_{t+1},...,o_1\middle|\lambda\right)$$
$$=\max_{1\le j\le N}\left[\delta_t(j)a_{ji}\right]b_i(o_{t+1}),\quad i=1,2,...,N;t=1,2,...,T-1.$$
$$\delta_t(i)=\max_{i_1,i_2,...,i_{t-1}} P\left(i_t=j,...,i_1,o_t,...,o_1\middle|\lambda\right),\ i=1,2,...,N.$$
$$\psi_t(i)=\arg\max_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right],\ i=1,2,...,N.$$

维特比算法如下

输入：模型 $\lambda=(A,B,\pi)$ 和观测 $O=(o_1,o_2,...,o_T)$；

输出：最优路径 $I^*=(i_1^*,i_2^*,...,i_T^*)$

（1）初始化

$$\delta_1(i)=\pi_i b_i(o_1),\quad i=1,2,...,N.$$
$$\psi_1(i)=0,\quad i=1,2,...,N.$$

（2）递推. 对 $t=2,3,...,T$

$$\delta_t(i)=\max_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right]b_i(o_t),\quad i=1,2,...,N.$$
$$\psi_t(i)=\arg\max_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right],\ i=1,2,...,N.$$

（3）终止

$$P^*=\max_{1\le i\le N}\delta_T(i)$$
$$i_T^*=\arg\max_{1\le j\le N}\left[\delta_T(i)\right]$$

（4）最优路径回溯.对$t=T-1,T-2,...,1.$

$$i_t^*=\psi_{t+1}\left(i_{t+1}^*\right)$$

维特比算法实现起来也不会太复杂。我利用 R 语言编写了两个函数来实现这个预测算法，分别是 ViterbiHMM()和 MoreViterbiHMM(). ViterbiHMM()函数只能实现对一个观测序列的各个状态进行预测，而函数 MoreViterbiHMM()可以实现对多个观测序列的状态序列进行预测。ViterbiHMM()在 MoreViterbiHMM()的内部被调用。（详细代码参见附录 10）

ViterbiHMM(obs,A,B,PI,StateLabel=as.character(1:nrow(A)), ObjectLabel=as.character(1:ncol(B)),if.show=TRUE)函数中参数 if.show=TRUE 表示是否输出所有的计算结果,默认输出所有的结果；当 if.show=FALSE 时，只输出最终预测的状态序列。其余所有参数的含义及用法与本章所用函数中的同名参数的含义一致，不再赘述。

MoreViterbiHMM(obsMat,…)函数中没有 if.show 参数，obsMat 是观测矩阵，行数表示序列的时间长度，列数表示样本量，obsMat 必须按照这种方式存储。其余参数的含义及用法与 ViterbiHMM()函数一样。

OK，接下来，我们分别用 ViterbiHMM()函数和 approxHMM()函数来求解书本例题 10.3.然后，再使用随机数据对 ViterbiHMM()函数做几个测试。

例题 10.3 求解代码：

```
####   求解例 10.3
A10.3<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.3
B10.3<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.3
pi10.3<-c(0.2,0.4,0.4);pi10.3

ViterbiHMM(obs=c("红","白","红"),A=A10.3,B=B10.3,PI=pi10.3,
           ObjectLabel =c("红","白") )
approxHMM(obs=c("红","白","红"),A=A10.3,B=B10.3,PI=pi10.3,
           ObjectLabel =c("红","白") )
```

例题 10.3 维特比算法求解结果：

```
> ViterbiHMM(obs=c("红","白","红"),A=A10.3,B=B10.3,PI=pi10.3,
+            ObjectLabel =c("红","白") )
$FinalState
T:1 T:2 T:3
"3" "3" "3"

$deltaMat
          st:1       st:2     st:3
T:1 0.10000 0.16000 0.2800
T:2 0.02800 0.05040 0.0420
T:3 0.00756 0.01008 0.0147

$pasiMat
      st:1 st:2 st:3
T:1      0    0    0
T:2      3    3    3
T:3      2    2    3
```

我们可以看到，上述结果与书本的求解过程及预测结果完全一致。上述结果中，组件 FinalState 表示最终预测出的状态序列；deltaMat 组件表示递推过程中的$\delta_t(i)$，一行表示一次递推结果，行数按时间顺序排；pasiMat 组件表示递推过程中的$\psi_t(i)$，一行表示一次递推结果，

行数按时间顺序排。接下来看看近似算法的预测结果。

例题 10.3 近似算法求解结果：

```
> approxHMM(obs=c("红","白","红"),A=A10.3,B=B10.3,PI=pi10.3,
+                 ObjectLabel =c("红","白") )
      [,1]
[1,] "3"
[2,] "2"
[3,] "3"
```

在本例，近似算法与维特比算法的预测结果并非完全一样。

最后，我们利用 ObjectHMM()随机生成几批观测数据，并用这几批数据测试一下 MoreViterbiHMM()函数的预测效率。

测试 1 代码如下：

```
#### 测试 1 状态个数为 3
A10.2<-matrix(c(0.5,0.2,0.3,
                0.3,0.5,0.2,
                0.2,0.3,0.5),nrow=3,byrow = T);A10.2
B10.2<-matrix(c(0.5,0.5,
                0.4,0.6,
                0.7,0.3),nrow = 3,byrow = T);B10.2
pi10.2<-c(0.2,0.4,0.4);pi10.2
## 生成隐马尔可夫观测序列及状态序列
test1<-ObjectHMM(size=100,Lth = 5,A=A10.2,B=B10.2,PI=pi10.2,
                 StateLabel = as.character(1:3),ObjectLabel = c("红","白"),seed=666)
obs1<-test1$obs
st1<-test1$state
obsmat<-do.call(cbind,obs1)#观测矩阵
stmat<-do.call(cbind,st1)#状态矩阵

t1<-MoreViterbiHMM(obsMat = obsmat,A=A10.2,B=B10.2,
                   PI=pi10.2,ObjectLabel = c("红","白"))
sum(t1==stmat)/500#预测准确率
```

测试 1 结果如下：

```
> t1<-MoreViterbiHMM(obsMat = obsmat,A=A10.2,B=B10.2,
+                    PI=pi10.2,ObjectLabel = c("红","白"))
> sum(t1==stmat)/500#预测准确率
[1] 0.436
```

这次的预测准确率（预测准确率的含义参考“关键词 6”）为 0.436，比随机猜测提高了 (0.436-1/3)/(1/3)=30.8%的预测效率。

测试 2 代码如下：

```
#### 测试 2 增加状态个数 5 个状态
A10<-matrix(c(0.3,0.2,0.2,0,0.3,
              0.1,0.2,0.3,0.3,0.1,
              0.2,0.2,0.3,0.15,0.15,
              0.2,0.1,0.1,0.3,0.4,
              0.1,0.2,0.3,0.3,0.1),nrow=5,byrow = T);A10
B10<-matrix(c(0.5,0.5,
              0.4,0.6,
              0.7,0.3,
              0.4,0.6,
              0.45,0.55),nrow = 5,byrow = T);B10
pi10<-c(0.2,0.1,0.25,0.2,0.25);pi10
## 生成隐马尔可夫观测序列及状态序列
test2<-ObjectHMM(size=100,Lth = 5,A=A10,B=B10,PI=pi10,
                 StateLabel = as.character(1:5),ObjectLabel = c("红","白"),seed=888)
obs2<-test2$obs
st2<-test2$state
obsmat2<-do.call(cbind,obs2)#观测矩阵
stmat2<-do.call(cbind,st2)#状态矩阵

t2<-MoreViterbiHMM(obsMat = obsmat2,A=A10,B=B10,
              PI=pi10,ObjectLabel = c("红","白"))
sum(t2==stmat2)/500#预测准确率
```

测试 2 结果：

```
> t2<-MoreViterbiHMM(obsMat = obsmat2,A=A10,B=B10,
+                    PI=pi10,ObjectLabel = c("红","白"))
> sum(t2==stmat2)/500#预测准确率
[1] 0.268
```

这次预测效率提高了 (0.268-0.2)/0.2=34%.

测试 3 代码如下：

```
#### 测试 3 减少状态个数  2 个
a10<-matrix(c(0.4,0.6,
              0.7,0.3),nrow = 2,byrow = T);a10
b10<-matrix(c(0.5,0.5,
              0.3,0.7),nrow = 2,byrow = T);b10
p10<-c(0.3,0.7);p10
## 生成隐马尔可夫观测序列及状态序列
test3<-ObjectHMM(size=100,Lth = 5,A=a10,B=a10,PI=p10,
                 StateLabel = as.character(1:2),ObjectLabel = c("红","白"),seed=666)
obs3<-test3$obs
st3<-test3$state
obsmat3<-do.call(cbind,obs3)#观测矩阵
stmat3<-do.call(cbind,st3)#状态矩阵

t3<-MoreViterbiHMM(obsMat = obsmat3,A=a10,B=a10,
                   PI=p10,ObjectLabel = c("红","白"))
sum(t3==stmat3)/500#预测准确率
```

测试 3 结果：

```
> t3<-MoreViterbiHMM(obsMat = obsmat3,A=a10,B=a10,
+                    PI=p10,ObjectLabel = c("红","白"))
> sum(t3==stmat3)/500#预测准确率
[1] 0.614
```

这次预测效率提高了(0.614-0.5)/0.5=22.8%.我们将上述测试结果列表如下：（样本为 100 组长度为 5 的观测序列）

	测试 3	测试 1	测试 2
状态数	2	3	5
随机猜测准确率	0.5	0.333	0.2
维特比算法准确率	0.614	0.436	0.268
预测效率提升	22.8%	30.8%	34%

■

小结：整理这一章的笔记真是相当消耗精力，不过幸好坚持下来了，欲起高楼，得先会搬砖，然后再砌墙……最终收获还不错，待本渣把这章所有的算法都实现了一遍后，对 HMM 模型的理解也是更清晰了一些。相信利用这章编写好的函数，小伙伴们应该也可以轻而易举地消灭掉这章的课后习题了，在此，我就不铺张篇幅了。

十一章　条件随机场

关键词 1：　　没看懂

说明 1：

老实说，这章真没怎么看懂，更别说实现了。看到那些恐龙级别的数学推导，本渣就头疼；还有那恐龙级别抽象的概率图模型，本渣也尚未理解。遂，弃疗！

还是等什么时候看懂了，再来补充一下这份笔记吧，现在就先不纠结了。

参考文献

[1]《统计学习方法》第二版，李航，2017 年 3 月

[2]《支持向量机-理论算法与扩展》，邓乃扬、田英杰等

[3]《概率论基础》第三版，李贤平，复旦大学出版社

[4]《高等数学下册》第六版，同济大学出版社

[5]《R 语言编程艺术》，Norman Matloff，机械工业出版社

[6]《R 语言核心技术手册》第二版，Josepb Adler

后记

今天是 9 月 10 号，下午 2 点刚刚和狐朋狗友下完馆子，挺着肚子回到宿舍午觉都没睡就接着开始编维特比算法，这是这份笔记实现的最后一个小算法。看看日期，说来也是巧，从我刚开始准备这份笔记时是 8 月 10 号，中间稍有断续，到今天整理完，刚好一个月。整理这份笔记的初衷有几点。

首先是好奇心，因为刚刚开始接触机器学习，我对其中的算法十分感兴趣，基于优化的思路而非基于假设检验的思路实在是刷新了我以往对统计的看法。这些基于优化思维的算法让我觉得有趣且实用。我在网上查过有不少人都实现了书本里面的算法，基本实用 python 实现的，没怎么发现有用 R 来实现的，奈何我只会 R，这真是个悲剧。要是我也会 python，估计我就不会自己编了。于是，我也想着，既然找不到用 R 实现的代码，那就尝试自己编一下吧。既然人家能编出来，为什么我就不能编出来呢？试一下也未尝不可嘛。另一点就是，我所查到的大部分的笔记是以博文+代码的形式出现的，没有整理成册方便阅读的，而且博文中的笔记大部分是将书本的思路重述了一遍，并没有涉及多少我想要知道及需要解决的问题。求人不如求己，当时想法是，既然找不到相应的学习笔记，那就只能自己干了，除了多看几遍书，多去理解，别无它法，在这过程中顺便把自己的思考记录下来。

当时，编的第一个函数就是二维 kd 树的实现，因为自己的思路与书本的解题思路有出入，于是就想把例题以自己的思路重解一遍。但是有懒得自己手动去画图标线，嗯，那就编一个函数，让电脑自己画

吧。结果这个小函数就像一根导火线，直接点燃了我想去实现书本里的算法的欲望。到现在，有些算法实现成功了，有些效果则不那么理想。但是，整个过程还是相当有趣的。

最后，如我在这本笔记开始所言，希望小伙伴们看完这本笔记后，并没有后悔为此所耗之时间。（2017/9/10）

附录 1 例 1.1 的 R 实现/训练误差与预测误差的对比

Function_Code: polyfit()

```
polyfit<-function(y,x,maxdeg){
   pwrs<-powers(x,maxdeg)#生成不同阶数的 x
   lmout<-list()
   class(lmout)<-"polyreg"#创建一个新类
   for(i in 1:maxdeg){
      lmo<-lm(y~pwrs[,1:i])
      lmo$fitted.cvvalues<-lvoneout(y,pwrs[,1:i,drop=FALSE])
      lmout[[i]]<-lmo
   }
   lmout$x<-x
   lmout$y<-y
   return(lmout)
}
```

Function_Code: print.polyreg()

```
print.polyreg<-function(fits){
   maxdeg<-length(fits)-2#计算拟合的模型数
   n<-length(fits$y)
   tbl<-matrix(nrow = maxdeg,ncol=2)
   cat("mean squared prediction errors,by degree\n")
   colnames(tbl)<-c("MSPE","TRAIN")
   for(i in 1:maxdeg){
      fi<-fits[[i]]
      errs<-fits$y-fi$fitted.cvvalues
      spe<-sum(errs^2)
      tbl[i,1]<-spe/n#计算 MSPE
      tbl[i,2]<-sum(fi$residuals^2)/n#计算训练误差 TRAIN
   }
   print(tbl)
   return(tbl)
}
```

Function_Code: plot.polyreg()

```
plot.polyreg<-function(fits){
   plot(fits$x,fits$y,xlab="X",ylab="Y")
   maxdg<-length(fits)-2
   cols<-c("red","green","blue")
   dg<-curvecount<-1
   while(dg<maxdg){
      prompt<-paste("RETURN for CV fit for degree",dg,"or type degree",
                    "or q for quit:")
      rl<-readline(prompt)
      dg<-if(rl=="") dg else if(rl!="q") as.integer(rl) else break
      lines(fits$x,fits[[dg]]$fitted.values,col=cols[curvecount%%3+1])
      dg<-dg+1
      curvecount<-curvecount+1
   }
}
```

Function_Code: powers()

```
powers<-function(x,dg){
   pw<-matrix(x,nrow = length(x))
   prod<-x
   for(i in 2:dg){
      prod<-prod*x
      pw<-cbind(pw,prod)
   }
   return(pw)
}
```

Function_Code: lvoneout()

```
lvoneout<-function(y,xmat){
   n<-length(y)
   predy<-vector(length = n)
   for(i in 1:n){
      lmo<-lm(y[-i]~xmat[-i,])
      betahat<-as.vector(lmo$coef)
      predy[i]<-betahat%*%c(1,xmat[i,])#交叉验证中 y 的预测值
   }
   return(predy)
}
```

附录 2 线性可分/不可分感知机的 R 实现

Function_Code:linePercept()

```
linePercept<-function(cls="y",atr=c("x1","x2"),data=NULL,aita=1,
                              endure=0,maxiter=1000,w0=rep(0,length(atr)),b0=0){
  datause<-data;datause$xb<-1#建立扩充数据框，主要是为了将 b 归入扩充权重向量
  wmat<-matrix(c(w0,b0),nrow=length(atr)+1,ncol=1)#先用矩阵按列储存初始扩充权重
  iterk<-0
  misssample<-vector()
  while(iterk>=0){
    sign_mat<-as.matrix(datause[,c(atr,"xb"),drop=F])%*%wmat[,iterk+1,drop=F]%*%
      t(as.matrix(datause[,cls,drop=F]))#计算时注意将 data.frame 转换为 matrix
    sign_vec<-diag(sign_mat)
    minlab<-which.min(sign_vec)#误分情况最严重的点
    if(endure==0){
      if(sign_vec[minlab]>endure){
        cat("The Final sign_min is : ",sign_vec[minlab],"\n")
        break
      }
    } else if(endure>0){#abs(sign_vec[minlab])表示最大误差距离
      if(all(w0==0)&&b0==0) stop("w0 and b0 must not all be 0 when endure>0.")
      if(all(sign_vec>0)) break
      if(abs(sign_vec[minlab])<endure){
        cat("The Final sign_min is : ",abs(sign_vec[minlab]),"\n")
        break
      }
    } else stop("The endure must not be smaller than 0. ")
    if(iterk>maxiter) break #当迭代次数大于 maxiter 时停止
    wchange<-wmat[,iterk+1,drop=F]+
      aita*datause[,cls][minlab]*t(as.matrix(datause[minlab,c(atr,"xb"),drop=F]))
    wmat<-cbind(wmat,wchange)
    misssample[iterk+1]<-minlab
    iterk<-iterk+1
  }
  rownames(wmat)<-c(atr,"b");colnames(wmat)<-paste0("iter",0:iterk)
  Percept<-list(Finalweight=t(wmat[,ncol(wmat)]),weight=wmat,
                iteration=iterk,miss=misssample,origindata=data,
                atrdata=data[,atr,drop=F],clsdata=data[,cls,drop=F],
                endure=endure,aita=aita,w0=w0,b0=b0)
  class(Percept)<-"linePercept"
  return(Percept)
}
```

Function_Code:plot.linePercept()

```
plot.linePercept<-function(obj){#只对二维数据有用
   plot(obj$atrdata[,1],obj$atrdata[,2],
        xlim=c(min(obj$atrdata[,1])-+abs(max(obj$atrdata[,1]))/3,
               max(obj$atrdata[,1])+abs(max(obj$atrdata[,1]))/3),
        ylim=c(min(obj$atrdata[,2])-abs(max(obj$atrdata[,2]))/3,
               max(obj$atrdata[,2])+abs(max(obj$atrdata[,2]))/3),
        col=2*abs(obj$clsdata[,1])+obj$clsdata[,1],pch=19,
        xlab=colnames(obj$atrdata)[1],ylab=colnames(obj$atrdata)[2])
   abline(b=-obj$Finalweight[1,1]/obj$Finalweight[1,2],
          a=-obj$Finalweight[1,3]/obj$Finalweight[1,2],
          col="red",lwd=1.25)
   text(obj$atrdata[,1],obj$atrdata[,2],obj$clsdata[,1])
}
```

Function_Code:print.linePercept()

```
print.linePercept<-function(obj){
   print.default(obj[c(1,3)])
}
```

Function_Code: preClinePercept()

```
preClinePercept<-function(lPobj,cls="y",atr=c("x1","x2"),atr_value=c(0,1)){
   latr<-length(atr)#特征个数
   levelcls<-unique(lPobj$clsdata[,1])
   numcls<-as.numeric(levelcls)
   atrmat<-matrix(c(atr_value,1),nrow=latr+1,ncol=1)
   sgn<-ifelse(sign(lPobj$Finalweight%*%atrmat)>0,max(numcls),min(numcls))
   return(as.vector(sgn))
}
```

Function_Code: predict.linePercept()

```
predict.linePercept<-function(lPobj,cls="y",atr=c("x1","x2"),atr_value=NULL){
   predvalue<-apply(atr_value,1,preClinePercept,lPobj=lPobj,atr=atr,cls=cls)
   out_pre<-atr_value
   out_pre[,cls]<-predvalue
   if(length(atr)==2){
      plot(lPobj);points(out_pre[,atr[1]],out_pre[,atr[2]],pch=23,col="red",cex=2.5,lwd=2)
      text(out_pre[,atr[1]],out_pre[,atr[2]],predvalue,col="red")
   }
   return(out_pre)
}
```

Function_Code: DualPercept()

```
#### 线性感知机对偶算法的实现 ####
####   DualPercept()函数只能用于线性可分集 ####
DualPercept<-function(cls="y",atr=c("x1","x2"),data=NULL,aita=1,
                      maxiter=1000,alpha0=rep(0,nrow(data)),b0=0){
  datause<-as.matrix(data)#转换成矩阵，方便运算
  sample_num<-nrow(datause)#样本个数
  clsdata<-datause[,cls,drop=F];atrdata<-datause[,atr,drop=F]
  Gram<-atrdata%*%t(atrdata)#先计算 Gram 矩阵
  alphaMat<-matrix(c(alpha0,b0),nrow=sample_num+1,ncol=1)#先建立参数扩充矩阵
  iterk<-0;misssample<-vector()
  while(iterk>=0){
    alpha_vec<-alphaMat[1:sample_num,iterk+1]#vector
    b<-alphaMat[sample_num+1,iterk+1]#一个数
    alpha_cls<-matrix(alpha_vec*clsdata[,1],nrow = sample_num,ncol=1)
    signMat<-(Gram%*%alpha_cls+b)%*%t(clsdata)#计算判断矩阵
    sign_vec<-diag(signMat)#得到不同点的计算结果
    minlab<-which.min(sign_vec)#挑出误分最严重的点
    if(sign_vec[minlab]>0) break
    alphaChange<-alpha_vec
    alphaChange[minlab]<-alphaChange[minlab]+aita
    bChange<-b+aita*clsdata[,1][minlab]
    AllChange<-matrix(c(alphaChange,bChange),sample_num+1,1)
    alphaMat<-cbind(alphaMat,AllChange)
    misssample[iterk+1]<-minlab
    iterk<-iterk+1
  }
  rownames(alphaMat)<-c(paste0("alpha",1:sample_num),"b")
  colnames(alphaMat)<-paste0("iter",0:iterk)
  Finalalpha<-t(alphaMat[,ncol(alphaMat)])#vector
  Finalweight<-rep(0,length(atr))
  for(i in 1:sample_num){#计算 weight
    weight<-clsdata[,1][i]*Finalalpha[i]*atrdata[i,]
    Finalweight<-Finalweight+weight
  }
  Finalweight<-c(Finalweight,Finalalpha[sample_num+1])
  Finalweight<-matrix(Finalweight,nrow=1);colnames(Finalweight)<-c(atr,"b")
  PerceptDual<-list(Finalweight=Finalweight,Finalalpha=Finalalpha,
                    iteration=iterk,Alpha=alphaMat,miss=misssample,
                    atrdata=atrdata,clsdata=clsdata,aita=aita,
                    alpha0=alpha0,b0=b0)
  class(PerceptDual)<-c("DualPercept","linePercept")
  return(PerceptDual)
}
```

附录 3 离散特征的 2 维平衡 kd 树 R 代码

Function_Code: kd_tie(x1,x2)

```
kd_tie<-function(x1,x2){
  x_1<-list(list(x1));x_2<-list(list(x2));l<-length(x1)
  timelab<-1;tielist<-list()
  while(timelab<=l){
    if(!timelab%%2==0){#奇数为第一维的数据及结点，偶数对应的是第二维的数据及结点
      x<-x_1[[timelab]]#获取一个 list
    } else {x<-x_2[[timelab]]}
    tie<-sapply(x,median)
    tie<-round(tie);tielist[[timelab]]<-tie
    x_1_new<-list(); x_2_new<-list()
    lstx<-length(x)
    for(j in 1:lstx){
      xj<-x[[j]]
      x_left<-which(xj<round(median(xj)))
      x_right<-which(xj>round(median(xj)))
      x_1_new[[2*j-1]]<-x_1[[timelab]][[j]][x_left]
      x_1_new[[2*j]]<-x_1[[timelab]][[j]][x_right]
      x_2_new[[2*j-1]]<-x_2[[timelab]][[j]][x_left]
      x_2_new[[2*j]]<-x_2[[timelab]][[j]][x_right]
    }
    x_1[[timelab+1]]<-x_1_new
    x_2[[timelab+1]]<-x_2_new
    lbreak<-sapply(x_1_new,length)
    if(any(lbreak<=1)){
      end_timetab<-timelab+1
      if(!end_timetab%%2==0){
        end_tie<-sapply(x_1_new,median)
        tielist[[end_timetab]]<-round(end_tie)
      } else {
        end_tie<-sapply(x_2_new,median)
        tielist[[end_timetab]]<-round(end_tie)
      }
      break
    }
    timelab<-timelab+1
  }
  list(tielist=tielist,x_1=x_1,x_2=x_2)
}
```

Function_Code: kd_plot(x1,x2)

```
kd_plot<-function(x1,x2){
   m1<-max(x1);m2<-max(x2);s1<-min(x1);s2<-min(x2);l<-length(x1)
   kd2_out<-kd_tie(x1=x1,x2=x2)
   tie_kd2<-kd2_out$tielist#提取包含结点信息的 list
   ltie<-length(tie_kd2)
   plot(x1,x2,xlim = c(s1,m1),ylim = c(s2,m2),type = "n",xlab ="x(1)",ylab="x(2)",
        main="Balance kd2 tree plot")
   points(x1,x2,pch=19)
   xkd<-tie_kd2[[1]][1]
   abline(v=xkd,col="red",lty=3)
   for(i in 2:ltie){
      plt<-tie_kd2[[i]];lplt<-length(plt)
      lsep<-seq(1,lplt,by=2)
      if(i%%2==0){
         for(j in lsep){
            lines(c(s1-1,xkd[(j+1)/2]),c(plt[j],plt[j]),col="red",lty=3)
            lines(c(xkd[(j+1)/2],m1+1),c(plt[j+1],plt[j+1]),col="red",lty=3)
         }
      }else{
         for(j in lsep){
            lines(c(plt[j],plt[j]),c(s2-1,xkd[(j+1)/2]),col="red",lty=3)
            lines(c(plt[j+1],plt[j+1]),c(xkd[(j+1)/2],m2+1),col="red",lty=3)
         }
      }
      xkd<-tie_kd2[[i]]
   }
   return(tiekd2=kd2_out)
}
```

Function_Code: LpSim.Plot()

```
####   Lp 范数递减性模拟  ####
LpSim.Plot<-function(number,maxp=1){
   if(any(number<0)) stop("The number must not smaller than 0.")
   max_num<-max(number)
   LpVec<-vector(length = maxp)
   for(i in 1:maxp) LpVec[i]<-(sum(number^i))^(1/i)
   tye<-ifelse(maxp<=20,"b","l")
   cols<-ifelse(maxp<=20,"blue","red")
   plot(1:maxp,LpVec,type = tye,ylab="LpValue",xlab="p",col=cols,
        main="Simulate Plot of Lp")
   list(maxnumber=max_num,minLp=min(LpVec),LpValue=LpVec)
}
```

Function_Code: lineKnn()

```
lineKnn<-function(cls="y",atr=c("x1","x2","x3"),dataTrain=NULL,
                              dataTest=NULL,k=3,p=2){
   data_use<-dataTrain#注意数据框中有字符时所有被转成字符
   atrdata<-data_use[,atr,drop=F];clsdata<-data_use[,cls]#vector
   dataTest<-dataTest[,atr]
   LpMat<-t(apply(dataTest,1,LpCalculate,atr=atr,dataTrain=atrdata,p=p))
   options(warn=-1)
   library(dprep)
   if(k==1){
      clsMatk<-apply(LpMat,1,function(x) clsdata[order(x)[1:k]])
      kPredict<-clsMatk#vector
   } else{
      clsMatk<-t(apply(LpMat,1,function(x) clsdata[order(x)[1:k]]))#只需要知道最近 k 个的
类别
      kPredict<-apply(clsMatk,1,function(x) sample(moda(x),1))
   }#moda 适用于字符向量
   detach("package:dprep")
   outPredict<-dataTest;outPredict[,cls]<-kPredict
   df<-list(FinalPredict=kPredict,PredictMat=outPredict,clsMatk=clsMatk,
            LpMat=LpMat,dataTrain=dataTrain,dataTest=dataTest,
            atr=atr,cls=cls,k=k,p=p)
   class(df)<-"lineKnn"#指定输出的类为"lineKnn"
   return(df)
}
```

Function_Code: LpCalculate()

```
LpCalculate<-function(dataTest,atr=c("x1","x2","x3"),dataTrain=NULL,p=2){
   datause<-as.matrix(dataTrain);n<-nrow(datause)
   LpVec<-vector(length = n)
   for(i in 1:n) LpVec[i]<-(sum(abs(dataTest-datause[i,atr])^p))^(1/p)
   return(LpVec)
}
```

Function_Code: print.lineKnn ()

```
print.lineKnn<-function(Knnobj){
   print(Knnobj[1])
}
```

Function_Code: plot.lineKnn ()

```
plot.lineKnn<-function(Knnobj){
   Train<-Knnobj$dataTrain
   Test<-Knnobj$dataTest
   atr<-Knnobj$atr;cls<-Knnobj$cls
   latr<-length(atr)
   if(latr==2){
      plot(Train[,atr[1]],Train[,atr[2]],xlab=atr[1],ylab=atr[2],

col=as.numeric(as.factor(Train[,cls])),pch=abs(as.numeric(as.factor(Train[,cls]))),
            main="Predict Plot of Knn")
      points(Test[,atr[1]],Test[,atr[2]],col="blue",
               pch=abs(as.numeric(as.factor(Knnobj$FinalPredict))),cex=2)
   }
}
```

附录 4 离散特征的朴素贝叶斯法 R 代码

Function_Code :navieBayes()

```
navieBayes<-function(cls="Y",atr=c("X1","X2"),data=NULL,lmada=0){
  if(!is.data.frame(data)) stop("Please enter a data.frame.")
  if(lmada<0) stop("lmada must be greater than or equal to ZERO.")
  d<-as.data.frame(apply(data,2,as.factor))
  n<-nrow(d)
  prodvar_lst<-list()#用来装计算出来的概率
  prec_var<-d[cls][,1];levelprec<-levels(prec_var);lprec<-length(levelprec)
  prec_p<-data.frame(level=levelprec,prob=NA)
  for(i in 1:lprec){
    prec_p[i,2]<-(sum(prec_var==levelprec[i])+lmada)/(n+lprec*lmada)#类 Y 的先验概率
  }
  prodvar_lst[[cls]]<-prec_p
  lvar=length(atr)#特征个数
  for(i in 1:lvar){#特征的条件先验概率
    xvar<-d[atr[i]][,1]
    txy<-table(xvar,prec_var)+lmada
    ptxy<-prop.table(txy,2)
    prodvar_lst[[atr[i]]]<-ptxy
  }
  prodvar_lst$lmada<-lmada
  prodvar_lst$response<-cls
  prodvar_lst$variables<-atr
  class(prodvar_lst)<-"navieBayes"   #指定输出对象的类为"navieBayes"，以便编写 S3 类泛函
  return(prodvar_lst)
}
```

Function_Code :print.navieBayes()

```
print.navieBayes<-function(obj){
  cat("response = prec_var:   ",obj$response,";","lmada = ",obj$lmada,"\n","\n")
  cat("The variables are : ",obj$variables,"\n","\n")
  lobj<-length(c(obj$response,obj$variables))
  print.default(obj[1:lobj])
}
```

Function_Code : preCnavieBayes()

```
#### preCnavieBayes() 只能进行一个实例的预测 ####
preCnavieBayes<-function(NBobj,cls=NULL,atr=NULL,atr_value=NULL){
  level<-NBobj[[cls]][,1];ncls<-length(level)
  latr<-length(NBobj)-4#特征的个数
  start_atr<-2
  end_atr<-latr+1
  predict_df<-data.frame(matrix(NA,ncls,latr+2))#先建立一个数据框储存结果
  colnames(predict_df)<-c(atr,"level","post_p")
  for(l in 1:latr){
    predict_df[1:ncls,l]<-atr_value[l]
  }
  predict_df$level<-level
  for(i in 1:ncls){
    xvec<-NULL
    for(j in start_atr:end_atr){
      xwhich<-which(rownames(NBobj[[atr[j-1]]])==as.character(atr_value[j-1]))
      ywhich<-which(colnames(NBobj[[atr[j-1]]])==as.character(predict_df$level[i]))
      px<-NBobj[[atr[j-1]]][xwhich,ywhich]
      xvec<-c(xvec,px)
    }
    ypre<-NBobj[[1]][,2][i]
    predict_df[i,4]<-ypre*prod(xvec)
  }
  return(predict_df)
}
```

Function_Code : predict.navieBayes()

```
#### 泛函 predict.navieBayes()针对类“navieBayes”，可一次进行多个样本实例的预测 ####
#### predict.navieBayes()在内部调用了 preCnavieBayes()函数 ####
predict.navieBayes<-function(NBobj,cls=NULL,atr=NULL,atr_value=NULL){
  if(!is.data.frame(atr_value)) stop("atr_value must be a data.frame!")
  post_lst<-apply(atr_value,1,preCnavieBayes,NBobj=NBobj,atr=atr,cls=cls)
  lst<-length(post_lst)
  post_df<-NULL
  for(i in 1:lst){
    prc_df<-post_lst[[i]]
    post_df<-rbind(post_df,prc_df)
  }
  cat("The response : ",cls,"\n")
  return(post_df)
}
```

附录 5 决策树的实现的 R 代码

Function_Code : InfoGain ()

```
#### 编写函数计算信息增益及信息增益比 InfoGain() ####
InfoGain<-function(cls=NULL,atr=NULL,method=c("info","inforate"),data=NULL){
  HDfunc<-function(atrcls){#atrcls 为向量
    l<-length(atrcls)
    tatrcls<-table(atrcls)
    atrclspvec<-as.vector(tatrcls)/l
    logatrclspvec<-ifelse(atrclspvec==0,0,log(atrclspvec,2))
    HD<--as.vector(atrclspvec%*%logatrclspvec)
    return(HD)
  }
  HDcls<-HDfunc(atrcls = data[,cls])
  HDatr<-apply(data[,atr],2,HDfunc)
  HatrVec<-apply(data[,atr],2,Hatr,clsvec=data[,cls])
  if(method=="info"){
    infogain<-HDcls-HatrVec
  } else if(method=="inforate"){
    infogain<-(HDcls-HatrVec)/HDatr
  } else stop("Please choose a useable method.")
  names(infogain)<-atr
  list(infogain=infogain,HDcls=HDcls,HatrVec=HatrVec,HDatr=HDatr)
}
```

Function_Code : Hatr ()

```
#### 计算经验条件熵 ####
Hatr<-function(atrvec=NULL,clsvec=NULL){#输入为特征向量及类别向量
  n<-length(atrvec)
  tatr<-table(atrvec)
  atrpvec<-as.vector(tatr)/n
  txy<-table(atrvec,clsvec)
  ptxy<-prop.table(txy,1)
  logptxy<-ifelse(ptxy==0,0,log(ptxy,2))
  loctab<-ptxy*logptxy#对应位置元素相乘,table
  atr_clspvec<-apply(loctab,1,sum)#vector
  hatr<--as.vector(atrpvec%*%atr_clspvec)
  return(hatr)
}
```

Function_Code : subtree()

```
#### 编写求决策树一步迭代的函数  需加载 dprep 包 ####
library(dprep)
subTree<-function(cls="Y",atr=c("A1","A2","A3","A4"),method=c("info","inforate"),
                          data=NULL,ept=0.1){
  atrcl<-atr;clscl<-cls;datacl<-data
  clsclvalue<-unique(datacl[,clscl])
  infoCalcul<-InfoGain(cls=clscl,atr=atrcl,data=datacl,method = method)#首次迭代结果
  subtree<-list()
  if(length(clsclvalue)==1){
    subtree[["origindata"]]<-datacl
    subtree[["single"]]<-clsclvalue
    subtree[["infoatr"]]<-"None"
    return(subtree)
  } else if(length(atrcl)==0||max(infoCalcul$infogain)<ept){
    lab<-moda(datacl[,clscl])
    subtree[["origindata"]]<-datacl
    if(length(lab)==1) subtree[["single"]]<-lab
    if(length(lab)>=2) subtree[["single"]]<-sample(lab,1)
    subtree[["infoatr"]]<-"None"
    return(subtree)
  }
  atrlab<-which.max(infoCalcul$infogain);
  atrchs<-datacl[,atrcl[atrlab]]#挑选信息增益最大的特征
  unqatrchs<-unique(atrchs);lunq<-length(unqatrchs)
  for(i in 1:lunq){
    subtree[[i]]<-datacl[which(atrchs==unqatrchs[i]),-atrlab]
    #每一个组件都是 data.frame
  }
  names(subtree)<-paste0(atrcl[atrlab],"=",unqatrchs)
  subtree[["newatr"]]<-atrcl[-atrlab]
  subtree[["infoatr"]]<-atrcl[atrlab]
  return(subtree)
}
```

Function_Code: Extree()

```
####  提取数据  ####
Extree<-function(obj){#用于提取叶节点的子集,obj 必须是两层的 list
  lobj<-length(obj)
  lvec<-sapply(obj,length)-2
  newlst<-list()
  st<-0
  for(i in 1:lobj){
    for(j in 1:lvec[i]){
      newlst[[st+j]]<-obj[[i]][[j]]
    }
    st<-st+lvec[i]
  }
  return(newlst)
}
```

Function_Code: HDfunc()

```
####  编写函数计算经验熵 ####
HDfunc<-function(atrcls){#atrcls 为向量
  l<-length(atrcls)
  tatrcls<-table(atrcls)
  atrclspvec<-as.vector(tatrcls)/l
  logatrclspvec<-ifelse(atrclspvec==0,0,log(atrclspvec,2))
  HD<--as.vector(atrclspvec%*%logatrclspvec)
  return(HD)
}
```

Function_Code: cutTree()

```
#### 计算损失函数 ####
cutTree<-function(cls="Y",data=NULL,alpha=1){#data 为 Extree()的输出结果
  ldata<-length(data)
  clslst<-list()
  for(i in 1:ldata){
    clslst[[i]]<-data[[i]][,cls]
  }
  hdvec<-sapply(clslst,HDfunc)#每个叶节点的经验熵,vector
  ldvec<-sapply(clslst,length)#每个叶结点的样本量,vector
  Cfunc<-hdvec%*%ldvec+alpha*ldata
  return(Cfunc)
}
```

Function_Code: Gpsim()

```
GpSim<-function(p){#基尼系数
  p2<-1-p
  gp<-p%*%p2
  return(gp)
}
```

Function_Code: GiniSingle()

```
GiniSingle<-function(atrvec=NULL,clsvec=NULL){#输入的是特征向量与属性向量
  D<-length(clsvec)
  txy<-table(atrvec,clsvec)
  nam<-rownames(txy)
  unqatr<-unique(atrvec)
  lunq<-length(unqatr)
  giniatr<-vector(length = lunq)
  for(i in 1:lunq){
    t1<-txy[i,];st1<-sum(t1)
    t2<-txy[-i,,drop=F];st2<-sum(t2)
    p1<-t1/st1;p2<-apply(t2,2,sum)/st2
    giniatr[i]<-(st1/D)*GpSim(p1)+(st2/D)*GpSim(p2)
  }
  names(giniatr)<-nam
  return(giniatr)
}
```

Function_Code: GiniCART()

```
GiniCART<-function(cls=NULL,atr=NULL,data=NULL){
  if(length(unique(data[,cls]))==1) return(list(Finalabel="None",D=data))
  ginilst<-apply(data[,atr],2,GiniSingle,clsvec=data[,cls])#list
  nlst<-names(ginilst)
  outgini<-sapply(ginilst,function(x) rbind(which.min(x),min(x)))
  nvec<-vector(length = length(atr))
  for(i in 1:length(atr)){
    ns<-names(ginilst[[i]])
    nvec[i]<-ns[outgini[1,i]]
  }
  minlab<-which.min(outgini[2,])
  atrlab<-outgini[1,minlab];atrchs<-names(ginilst[[minlab]])[atrlab]
  lab<-which(data[,nlst[minlab]]==atrchs)
  list(Finalabel=c(nlst[minlab],atrchs),FinalGini=outgini[2,minlab],
       GiniMat=outgini,Ginilst=ginilst,data[lab,-minlab],data[-lab,-minlab])
}
```

附录 6 逻辑斯蒂回归及最大熵模型的 R 实现

Function_Code: gradLogistic ()

```
gradLogistic<-function(cls=NULL,atr=NULL,data=NULL,scale=TRUE,
                                w0=rep(0,length(atr)+1),aita=1,ept=1e-5,maxiter=100000){
  if(!is.data.frame(data)) stop("data must be a data.frame.")
  datause<-data;datause$xb<-1#扩充矩阵
  atrdata<-datause[,c(atr,"xb"),drop=F];atrdata<-as.matrix(atrdata)
  if(scale){#自变量数据标准化
    for(i in 1:length(atr)){
      atrdata[,i]<-scale(atrdata[,i])#0,1 标准化
    }
  }
  clsdata<-datause[,cls,drop=F];clsdata<-as.matrix(clsdata)
  N<-nrow(datause)
  MinusLog<-function(wuse,y=clsdata[,1],x=atrdata){
    n<-nrow(atrdata)
    MLog<-vector(length = n)
    for(i in 1:n){
      ep<-as.vector(wuse%*%x[i,])
      epe<-exp(ep)
      if(is.infinite(epe)){
        MLog[i]<-ep-y[i]*ep
      } else{
        MLog[i]<-log(1+epe)-y[i]*ep
      }
    }
    return(sum(MLog))
  }
  calpi<-function(x){
    ex<-exp(w%*%x)
    if(is.infinite(ex)){
      px<-1
    } else{
      px<-ex/(1+ex)
    }
    return(px)
  }
  w<-w0#指定 w0,vector
  iterk<-1
```

（接下页）

```
  while(iterk>=1){
    pi<-apply(atrdata,1,calpi)#指定 pi(k),vector
    piMinusy<-matrix(pi-clsdata[,1],nrow = N,ncol=1)#N*1 矩阵
    gradf<-t(atrdata)%*%piMinusy#利用矩阵乘法,N*1 矩阵
    gradfvec<-gradf[,1]
    #print(sqrt(sum(gradfvec^2)))
    if(sqrt(sum(gradfvec^2))<=ept){
      stoprule<-'sqrt(sum(gradfvec^2))<=ept'
      break
    }
    wbefore<-w
    #print(w)
    w<-w-aita*gradfvec
    MinusLogBtw<-MinusLog(wuse=w)-MinusLog(wuse=wbefore)
    wBtw<-w-wbefore
    if(abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept){
      stoprule<-'abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept'
      break
    }
    if(iterk>=maxiter){
      stoprule<-'iterk>=maxiter'
      break
    }
    iterk<-iterk+1
    #print(iterk)
  }
  names(w)<-c(atr,"b")
  outlst<-list(weight=w,minusLogkplus1=MinusLog(wuse=w),
               minusLogk=MinusLog(wuse=wbefore),variable=atr,
               response=cls,origindata=data,iteration=iterk,
               formula=paste(cls,"~",paste(atr,collapse = "+")),
               stoprule=stoprule)
  class(outlst)<-"gradLogistic"
  return(outlst)
}
```

Function_Code: predict.gradLogistic ()

```
predict.gradLogistic<-function(obj,atr=NULL,atr_value=NULL){
   weight<-obj$weight
   atr_value$b<-1
   for(i in 1:length(atr)){
      atr_value[,i]<-scale(atr_value[,i])
   }
   predone<-function(x,w){#x,w 均是向量
      ep1<-exp(w%*%x)
      if(is.infinite(ep1)){
         p1<-1-0.001
      } else{
         p1<-ep1/(1+ep1)
      }
      return(p1)
   }
   P1<-apply(atr_value,1,predone,w=weight)
   P0<-1-P1
   predvec<-ifelse(P1>=0.5,1,0)
   pMatdf<-data.frame(P1=P1,P0=P0,predict=predvec)
   list(FinalPredict=predvec,PredictMat=pMatdf)
}
```

Function_Code: print.gradLogistic ()

```
print.gradLogistic<-function(obj){
   cat("The stoprule is : ",obj$stoprule,"\n")
   cat("iteration : ",obj$iteration,"\n")
   cat("formula : ",obj$formula,"\n")
   oldlst<-options()
   options(digits = 9)
   print(obj[1:3])
   options(oldlst)
}
```

Function_Code: DFPLogistic ()

```
#### DFP 算法的实现 ####
DFPLogistic<-function(cls=NULL,atr=NULL,data=NULL,scale=TRUE,ept=1e-5,
                      G0=diag(rep(1,length(atr)+1)),MoreStopRule=FALSE,
                      w0=rep(0,length(atr)+1),aita=.1,maxiter=100000,
                      SearchAita=F,maxsearch=1000){
  if(!is.data.frame(data)) stop("data must be a data.frame.")
  datause<-data;datause$xb<-1#扩充矩阵
  atrdata<-datause[,c(atr,"xb"),drop=F];atrdata<-as.matrix(atrdata)
  if(scale){#自变量数据标准化
    for(i in 1:length(atr)){
      atrdata[,i]<-scale(atrdata[,i])#0,1 标准化
    }
  }
  clsdata<-datause[,cls,drop=F];clsdata<-as.matrix(clsdata)
  N<-nrow(datause)
  MinusLog<-function(wuse,y=clsdata[,1],x=atrdata){#计算个 g(w)
    n<-nrow(atrdata)
    MLog<-vector(length = n)
    for(i in 1:n){
      ep<-as.vector(wuse%*%x[i,])
      epe<-exp(ep)
      if(is.infinite(epe)){
        MLog[i]<-ep-y[i]*ep
      } else{
        MLog[i]<-log(1+epe)-y[i]*ep
      }
    }
    return(sum(MLog))
  }
  calpi<-function(x,wx){#计算 pi
    oldex<-wx%*%x
    ex<-exp(oldex)
    if(is.infinite(ex)){
      px<-1+sample(c(-ept*10/3,-ept*6,-ept*5/4,-ept/2),1)
    } else if(ex==0){
      px<-sample(c(ept*10/3,ept*15,ept*9/4,ept*17/2),1)
    } else{
      px<-ex/(1+ex)
    }
    return(px)
  }
  calgrad<-function(dfatr,dfcls,Nl,wc){#计算梯度          (接下页)
```

```
    pi<-apply(dfatr,1,calpi,wx=wc)#指定 pi(k),vector
    piMinusy<-matrix(pi-dfcls[,1],nrow = Nl,ncol=1)#N*1 矩阵
    gradfCal<-t(dfatr)%*%piMinusy#利用矩阵乘法，（n+1）*1 矩阵，计算梯度
    return(gradfCal)
}
findAita<-function(dataatr,datacls,wkk,dtakk,ata_ept=1e-1,#一维搜索函数
                   ata0=1,maxatak=maxsearch){#wk,dtak 为 vector
    expata1<-function(wk,dtak,ati,x){
    exaita1<-as.vector((wk-ati*dtak)%*%x)
    expcal1<-exp(exaita1)
    if(is.infinite(expcal1)){
       pi1<-1+sample(c(-ata_ept/3,-ata_ept,-ata_ept/4,-ata_ept/2),1)
    } else if(expcal1==0){
       pi1<-sample(c(ata_ept/3,ata_ept,ata_ept/4,ata_ept/2),1)
    } else{
       pi1<-expcal1/(1+expcal1)
    }
    pi1
}
    expata2<-function(wk,dtak,ati,x){
    exaita2<-as.vector((wk-ati*dtak)%*%x)
    expcal2<-exp(exaita2)
    if(is.infinite(expcal2)){
       pi2<-sample(c(ata_ept/3,ata_ept,ata_ept/4,ata_ept/2),1)
    } else if(expcal2==0){
       pi2<-sample(c(ata_ept/3,ata_ept,ata_ept/4,ata_ept/2),1)
    } else{
       pi2<-expcal2/(1+expcal2)^2
    }
    pi2
}
    ata<-ata0
    iteratak<-1
    while(iteratak>=1){
       p1<-apply(dataatr,1,expata1,wk=wkk,dtak=dtakk,ati=ata)
       p2<-apply(dataatr,1,expata2,wk=wkk,dtak=dtakk,ati=ata)
       ppi<-p1-datacls[,1]
       dtkM<-matrix(dtakk,nrow=length(dtakk),ncol=1)
       dtkx<-as.vector(dataatr%*%dtkM)
       H1<-as.vector(ppi%*%dtkx)
       H2<-as.vector(p2%*%(dtkx^2))
       ataold<-ata                                          （接下页）
```

```
        atanew<-ata-H1/H2
        ata<-atanew
        if(abs(atanew-ataold)<ata_ept) break
        if(iteratak>=maxatak) break
        iteratak<-iteratak+1
        }
return(ata)
}
w<-w0#指定 w0,vector
G<-G0#指定初始正定矩阵
changeG<-0
changeW<-0
changeAita<-0
iterk<-1
while(iterk>=1){
   #if(iterk>=4) browser()
   if(any(is.infinite(w))||any(is.nan(w))){
        w<-sample(seq(-2,2,length=200),length(w0))
        changeW<-changeW+1
   }
   #如果计算出 Inf 或-Inf 或 NaN，则重新选择 w 继续迭代
   gradf<-calgrad(dfatr=atrdata,dfcls=clsdata,Nl=N,wc=w)#（n+1）*1 矩阵，计算梯度
   gradfvec<-gradf[,1]#获得梯度向量
   #print(sqrt(sum(gradfvec^2)))
   if(sqrt(sum(gradfvec^2))<=ept){
      stoprule<-'sqrt(sum(gradfvec^2))<=ept'
      break
   }
   wbefore<-w
   #print(w)
   if(any(is.infinite(G))||any(is.nan(G))){
     G<-G0*sample(seq(-1,1,length=1000),1)
     w<-sample(seq(-10,10,length=200),length(w0))
     aita<-sample(seq(0.01,2,length=100),1)
     changeW<-changeW+1
     changeG<-changeG+1
     changeAita<-changeAita+1
   }
   #如果计算出 Inf 或-Inf 或 NaN，则重新选择正定的 G/w/aita，重新迭代
   ## 进入一维搜索，寻找最优步长 ##
   olddelta<-G%*%gradf
   if(SearchAita){
      aita<-findAita(dataatr = atrdata,datacls = clsdata,wkk = wbefore,          （接下页）
```

```
                        dtakk = as.vector(olddelta),ata0 = aita)
          aita<-max(0.1,aita)
      }
      deltak<--aita*olddelta#(n+1)*1 矩阵
      wnew<-wbefore+as.vector(deltak)#更新 w
      w<-wnew
      gradfnew<-calgrad(dfatr=atrdata,dfcls=clsdata,Nl=N,wc=wnew)#w 已经改变
      yk<-gradfnew-gradf#(n+1)*1 矩阵
      G<-G+(deltak%*%t(deltak))/as.vector(t(deltak)%*%yk)-
        (G%*%yk%*%t(yk)%*%G)/as.vector(t(yk)%*%G%*%yk)#更新 G
      if(MoreStopRule){
        MinusLogBtw<-MinusLog(wuse=w)-MinusLog(wuse=wbefore)
        wBtw<-w-wbefore
        if(abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept){
        stoprule<-'abs(MinusLogBtw)<ept||sqrt(sum(wBtw^2))<ept'
          break
        }
      }
      if(iterk>=maxiter){
        stoprule<-'iterk>=maxiter'
        break
      }
      iterk<-iterk+1
      #print(iterk)
    }
    names(w)<-c(atr,"b")
    outlst<-list(weight=w,minusLogkplus1=MinusLog(wuse=w),

minusLogk=MinusLog(wuse=wbefore),LpGradf=sqrt(sum(gradfvec^2)),
                        changW=changeW,changeG=changeG, changeAita=changeAita,
                        variable=atr,response=cls,
                        origindata=data,iteration=iterk,
                        formula=paste(cls,"~",paste(atr,collapse = "+")),
                        stoprule=stoprule)
    class(outlst)<-"DFPLogistic"
    return(outlst)
}
```

Function_Code: predict.DFPLogistic ()

```
predict.DFPLogistic<-function(obj,atr=NULL,atr_value=NULL){
   weight<-obj$weight
   atr_value$b<-1
   for(i in 1:length(atr)){
      atr_value[,i]<-scale(atr_value[,i])
   }
   predone<-function(x,w){#x,w 均是向量
      ep1<-exp(w%*%x)
      if(is.infinite(ep1)){
         p1<-1-0.001
      } else{
         p1<-ep1/(1+ep1)
      }
      return(p1)
   }
   P1<-apply(atr_value,1,predone,w=weight)
   P0<-1-P1
   predvec<-ifelse(P1>=0.5,1,0)
   pMatdf<-data.frame(P1=P1,P0=P0,predict=predvec)
   list(FinalPredict=predvec,PredictMat=pMatdf)
}
```

Function_Code: print.DFPLogistic ()

```
print.DFPLogistic<-function(obj){
   cat("The stoprule is : ",obj$stoprule,"\n")
   cat("iteration : ",obj$iteration,"\n")
   cat("formula : ",obj$formula,"\n")
   oldlst<-options()
   options(digits = 9)
   print(obj[1:7])
   options(oldlst)
}
```

附录 7 基于 SMO 算法的支持向量机的 R 实现

Function_Code: smoSVM ()

```
smoSVM<-function(cls,atr,data,Kernel=c("line","poly","gaussi"),scale=T,
                 C=10,ept=1e-2,alpha0=rep(0,nrow(data)),b0=0,p=2,Lp=2,
                 lmada=sqrt(ncol(data))/sqrt(2),maxiter=10000,Change=T){
  if(!is.data.frame(data)) stop("data must be a data.frame.")
  datause<-data
  N<-nrow(datause)
  IN<-1:N
  atrdata<-datause[,atr,drop=F];atrdata<-as.matrix(atrdata)
  clsdata<-datause[,cls,drop=F];clsdata<-as.matrix(clsdata)
  clsVec<-clsdata[,1]
  if(scale){#自变量数据标准化
    for(i in 1:length(atr)){
      atrdata[,i]<-scale(atrdata[,i])#0,1 标准化
    }
  }
  ## 计算 Gram 矩阵
  if(Kernel=="line") Gram<-lineKernel(data=atrdata)#matrix
  if(Kernel=="poly") Gram<-polyKernel(data=atrdata,p=p)#matrix
  if(Kernel=="gaussi") Gram<-gaussiKernel(data=atrdata,lmada = lmada,Lp=Lp)#matrix
  alpha<-alpha0;b<-b0
  iterk<-1
  while(iterk>=1){
    #if(iterk>=9) browser()
    gk<-gfunc(clsvec = clsVec,gram=Gram,alphak = alpha,bk=b)#vector
    Ek<-gk-clsVec#vector
    #获取 alphastar1 的标签，对应的位置是第几
    if(Change){
      Clst<-SelectFunc(gram=Gram,clsdata=clsdata,C=C,alphak = alpha)
      alp1<-Clst$chslup;Ekalp1<-Ek[alp1]
      alp2<-Clst$chsllow;Ekalp2<-Ek[alp2]
      malp<-Clst$chsm;Malp<-Clst$chsM
      y1k<-clsVec[alp1];y2k<-clsVec[alp2]
      #停止条件
      if((malp-Malp)<=ept){
        stoprule<-"(malp-Malp)<=ept"
        break
      }
    } else{
      lst<-findAlpha(alphak = alpha,gveck = gk,ept=ept,C=C,clsvec = clsVec)
```

（接下页）

```
        alp1<-lst$labmax;alllab<-lst$alllab;outlab<-lst$outlab
        ## 停止条件
        if(is.null(alp1)||(length(alllab)/N)<ept||(length(outlab)/N)<ept){
          stoprule<-"is.null(alp1)||(length(alllab)/N)<ept||(length(outlab)/N)<ept"
          break
        }
        Ekalp1<-Ek[alp1]
        y1k<-clsVec[alp1]
        chooseIN<-IN[-alp1];chooseEk<-Ek[-alp1]
        alp2<-ifelse(Ekalp1>0,chooseIN[which.min(chooseEk)],
                             chooseIN[which.max(chooseEk)])
        y2k<-clsVec[alp2]
        Ekalp2<-Ek[alp2]
      }
      alp2old<-alpha[alp2];alp1old<-alpha[alp1]
      #计算上下界，nk
      Hk<-ifelse(y1k==y2k,min(C,alp2old+alp1old),min(C,C+alp2old-alp1old))
      Lk<-ifelse(y1k==y2k,max(0,alp2old+alp1old-C),max(0,alp2old-alp1old))
      k11<-Gram[alp1,alp1];k22<-Gram[alp2,alp2];k12<-Gram[alp1,alp2];k21<-k12
      nk<-k11+k22-2*k12
      #更新选出的 alpha
      alp2kplus1_unc<-alp2old+y2k*(Ekalp1-Ekalp2)/nk
      if(alp2kplus1_unc>Hk){
        alpha[alp2]<-Hk
      } else if(alp2kplus1_unc<Lk){
        if(Lk!=0){
          alpha[alp2]<-Lk
        } else if(Lk==0&&alp2old!=0){
          alpha[alp2]<-Lk
        } else{#只有在
          alpha[alp2]<-sample(c(.15,.1,ept/6,ept*10,ept),1)
        }
      } else{
        alpha[alp2]<-alp2kplus1_unc
      }
      alpha[alp1]<-alp1old+y1k*y2k*(alp2old-alpha[alp2])
      ## alpha 已经更新
      alp1new<-alpha[alp1];alp2new<-alpha[alp2]
##检查 alp2new 与 alp2old 是否不一样，特殊情况：alp2kplus1_unc<Lk，Lk=0 时
#alp2new==alp2old，这是 alpha 就没有完成更新，那么接下来的迭代也不会更新；极端
#情况下 Hk 也会==0
      ## 接着更新阈值 bk 为 bkplus1
      b_old<-b                                                    （接下页）
```

```
      b1kplus1<--Ekalp1-y1k*k11*(alp1new-alp1old)-y2k*k21*(alp2new-alp2old)+b_old
      b2kplus1<--Ekalp2-y1k*k12*(alp1new-alp1old)-y2k*k22*(alp2new-alp2old)+b_old
      if(alp1new>0&&alp1new<C){
        b<-b1kplus1
      } else if(alp2new>0&&alp2new<C){
        b<-b2kplus1
      } else if((alp1new==0||alp1new==C)&&(alp2new==0||alp2new==C)){
        b<-(b2kplus1+b1kplus1)/2
      } else if((alp1new>0&&alp1new<C)&&(alp2new>0&&alp2new<C)){
        b<-b1kplus1
      }
      #b 已经更新
      if(iterk>=maxiter){
        stoprule<-"iterk>=maxiter"
        break
      }
      iterk<-iterk+1
    }
    nonzero<-which(alpha!=0);lnz<-length(nonzero)
    nonZeroAlpha<-alpha[nonzero]
    names(nonZeroAlpha)<-nonzero
    alpy<-alpha[nonzero]*clsVec[nonzero]
    if(Kernel=="line"){
      w<-rep(0,length(atr))
      for(i in 1:lnz){
        w<-w+alpy[i]*atrdata[nonzero[i],]
      }
    } else{
      w<-NULL
    }
    bvec<-vector(length = lnz)
    for(j in 1:lnz){
      gramvec<-as.vector(Gram[nonzero,j])
      bvec[j]<-clsVec[nonzero[j]]-as.vector(alpy%*%gramvec)
    }
    outlst<-list(nonZeroAlpha=nonZeroAlpha,bMean=mean(bvec),support=nonzero,w=w,
                 stoprule=stoprule,formula=paste(cls,"~",paste0(atr,collapse = "+")),
                 variables=atr,response=cls,iteration=iterk,clsvec=clsVec,
     ScaleAtr=atrdata,Gram=Gram,Kernel=Kernel,data=data,p=p,Lp=Lp,lmada=lmada)
    class(outlst)<-"smoSVM"
    return(outlst)
}
```

Function_Code: lineKernel ()

```
lineKernel<-function(data){#data 为数据框
  dMat<-as.matrix(data)
  return(dMat%*%t(dMat))#返回 Gram 矩阵
}
```

Function_Code: polyKernel ()

```
polyKernel<-function(data,p=2){
  dMat<-as.matrix(data)
  lineGram<-dMat%*%t(dMat)
  polyGram<-(lineGram+1)^p
  return(polyGram)#返回 Gram 矩阵
}
```

Function_Code: gaussiKernel ()

```
gaussiKernel<-function(data,lmada=sqrt(ncol(data))/sqrt(2),Lp=2){
  #lmada 指定高斯核函数的标准差，默认为特征个数的平方根，基于 Lp 范
  dMat<-as.matrix(data)
  ln<-nrow(data)
  Gram1<-matrix(NA,nrow=ln,ncol=ln)
  for(i in 1:ln){#i 行
    for(j in 1:ln){#j 列
      dij<-dMat[i,]-dMat[j,]
      absdij<-abs(dij)
      Lpdij<-(sum(absdij^Lp))^(1/Lp)
      Gram1[i,j]<-Lpdij^2
    }
  }
  Gram2<-Gram1/(-2*(lmada^2))
  gaussiGram<-exp(Gram2)
  return(gaussiGram)
}
```

Function_Code: gfunc ()

```
gfunc<-function(clsvec,gram,alphak,bk){
#clsvec 为向量,Gram 为矩阵,alphak 为 alpha 向量
   ncls<-length(clsvec)
   gvec<-vector(length = ncls)
   for(i in 1:ncls){
      gonevec<-vector(length = ncls)
      for(j in 1:ncls){
         gonevec[j]<-alphak[j]*clsvec[j]*gram[i,j]
      }
      gvec[i]<-sum(gonevec)+bk
   }
   return(gvec)#返回一个 vector
}
```

Function_Code: SelectFunc ()

```
####  (5)：工作集选取算法 ####

SelectFunc<-function(gram,clsdata,C,alphak){
   clsv<-clsdata[,1]
   la<-length(alphak)
   e<-rep(1,la)
   clsMat<-clsdata%*%t(clsdata)
   hMat<-clsMat*gram
   gradk<-as.vector(hMat%*%matrix(alphak,nrow=la,ncol=1))-e
   chsvec<--clsv*gradk
   lup<-which((alphak<C&clsv==1)|(alphak>0&clsv==-1))
   llow<-which((alphak<C&clsv==-1)|(alphak>0&clsv==1))
   chslup<-lup[which.max(chsvec[lup])];chsm<-chsvec[chslup]
   chsllow<-llow[which.min(chsvec[llow])];chsM<-chsvec[chsllow]
   list(chslup=chslup,chsllow=chsllow,chsm=chsm,chsM=chsM)
}
```

Function_Code: findAlpha ()

```
####    （2）：工作工作集的方法 ####
findAlpha<-function(alphak,gveck,ept,C,clsvec){#alphak,gveck 为向量
   ygk<-clsvec*gveck
   lab1<-which(alphak==0)
   lab2<-which(alphak==C)
   lab3<-which(alphak>0&alphak<C)
   alllab<-NULL
   if(length(lab1)>=1){
      ygkright<-ygk[lab1]
      yuselab1<-which(ygkright<(1-ept))#违反 KKT
      alllab<-c(alllab,lab1[yuselab1])
   }
   if(length(lab2)>=1){
      ygkerror<-ygk[lab2]
      yuselab2<-which(ygkerror>(1+ept))#违反 KKT
      alllab<-c(alllab,lab2[yuselab2])
   }
   if(length(lab3)>=1){
      ygksupport<-ygk[lab3]
      ygkuse<-abs(ygksupport-1)
      yuselab3<-which(ygkuse>ept)#违反 KKT
      alllab<-c(alllab,lab3[yuselab3])
   }
   ##先检查支持向量
   if(exists("yuselab3")&&length(yuselab3)>=1){
      outlab<-lab3[yuselab3]
   } else{#再检查其他的样本
      outlab<-alllab
   }
   if(!is.null(outlab)){
      ygkMinus1<-abs(ygk-1)[outlab]
      maxerrlab<-outlab[which.max(ygkMinus1)]
      labmax<-maxerrlab
   } else{
      labmax<-NULL
   }
   outlst<-list(alllab=alllab,outlab=outlab,labmax=labmax)
   return(outlst)
}
```

附录 8 提升算法的 R 代码

Function_Code: AdaboostAlpha ()

```
## 计算模型 Gm 的系数 alpham
AdaboostAlpha<-function(em){#em 是错误率
  if(em==0) alpham<-Inf
  if(em==1) alpham<--Inf
  if(em>0&em<1){
    alp<-(1-em)/em
    alplog<-(1/2)*log(alp)
    alpham<-alplog
  }
  alpham
}
```

Function_Code: AdaboostWeight ()

```
## 计算样本权重 m+1 次迭代，Dm+1
AdaboostWeight<-function(weightm,alpham,clsvec,preclsvec){
  #alpham 不能是 Inf,-Inf,NaN
  calog<--alpham*clsvec*preclsvec
  expwm<-weightm*exp(calog)
  wmplus1<-expwm/sum(expwm)
  wmplus1
}
```

Function_Code: AdaboostError ()

```
## 计算带权重的 em
AdaboostError<-function(clsvec,preclsvec,weightm){
    #输入都是向量
    sum(weightm[which(clsvec!=preclsvec)])
}
```

Function_Code: SearchOneTree ()

```
## 编写简单的树桩决策树：一次搜索
SearchOneTree<-function(atr,cls,weightm,data){
  atrvec<-data[,atr]
  clsvec<-data[,cls]
  latr<-length(atrvec)
  searchx<-atrvec+.5
  searchx<-searchx[-latr]
  emveclow<-vector(length = latr-1)
  emvecup<-vector(length = latr-1)
  for(i in 1:(latr-1)){
    sch<-searchx[i]
    clslow<-ifelse(atrvec<=sch,1,-1)
    clsup<-ifelse(atrvec<=sch,-1,1)
    emveclow[i]<-AdaboostError(weightm = weightm,clsvec=clsvec,preclsvec = clslow)
    emvecup[i]<-AdaboostError(weightm = weightm,clsvec=clsvec,preclsvec = clsup)
  }
  lowmin<-which.min(emveclow)
  upmin<-which.min(emvecup)
  if(emveclow[lowmin]!=emvecup[upmin]){
    error<-min(emveclow[lowmin],emvecup[upmin])
    finalab<-ifelse(emveclow[lowmin]<emvecup[upmin],lowmin,upmin)
  } else{
    error<-emveclow[lowmin]
    finalab<-lowmin
  }
  if(finalab==lowmin){
    ModelFinal<-paste("Model:: ",atr,"<",searchx[lowmin]," is ","1"," else"," -1.",sep = "")
    preclsvec<-ifelse(atrvec<=searchx[lowmin],1,-1)
  } else{
    ModelFinal<-paste("Model:: ",atr,">",searchx[upmin]," is ","1"," else"," -1.",sep = "")
    preclsvec<-ifelse(atrvec<=searchx[upmin],-1,1)
  }
  list(error=error,ModelFinal=ModelFinal,preclsvec=preclsvec)
}
```

Function_Code: AdaboostTreeStool ()

```
####    例 8.1 一步到位 ####
AdaboostTreeStool<-function(atr,cls,data,weight0=rep(1/length(clsvec),length(clsvec)),
                                    ept=0,maxiter=10000){
  atrvec<-data[,atr];clsvec<-data[,cls]
  weight<-weight0
  f<-rep(0,length(clsvec))
  Gmodel<-NULL
  Galpha<-NULL
  iterk<-1
  while(iterk>=1){
    G<-SearchOneTree(atr=atr,cls=cls,data=data,weightm = weight)
    err<-G$error;pcls<-G$preclsvec
    if(err==0||err==1){
      stoprule<-"err==0||err==1"
      outlst<-list(stoprule=stoprule,Model=G$ModelFinal,error=err)
      break
    }
    ModelG<-G$ModelFinal
    Gmodel<-c(Gmodel,ModelG)
    alpha<-AdaboostAlpha(err)
    Galpha<-c(Galpha,alpha)
    D<-AdaboostWeight(weightm = weight,alpham = alpha,clsvec = clsvec,preclsvec =
pcls)
    weight<-D
    f<-f+alpha*pcls;sgnf<-sign(f);sgnf<-ifelse(sgnf==1,1,-1)
    errf<-1-sum(sgnf==clsvec)/length(clsvec)#f 的误分率
    if(errf<=ept){
      stoprule<-"errf<=ept"
      outlst<-list(stoprule=stoprule,iteration=iterk,AdaboostModel=Gmodel,
                   AdaboostAlpha=Galpha,AdaboostPredict=sgnf)
      break
    }
    if(iterk>=maxiter){
      stoprule<-"iterk>=maxiter"
      outlst<-list(stoprule=stoprule,iteration=iterk,AdaboostModel=Gmodel,
                   AdaboostAlpha=Galpha,AdaboostPredict=sgnf)
      break
    }
    iterk<-iterk+1
  }
  return(outlst)
}
```

附录 9 EM 算法的 R 实现

Function_Code: gaussiEM ()

```
####   高斯混合模型参数估计的 EM 算法  ####
gaussiEM<-function(clsvec,K=2,mean0=rep(0,K),var0=rep(1,K),alpha0=rep(1/K,K),
                           ept=1e-1,maxiter=10000,Lp=2){
   lN<-length(clsvec)
   mean<-mean0
   var<-var0
   alpha<-alpha0
   iterk<-1
   while(iterk>=1){
      parameterold<-c(mean,var)
      rjkMat<-gaussiResponse(clsvec = clsvec,K=K,Ml=mean,Vl=var,Alpl=alpha)
      paralst<-gaussiParameter(rMat = rjkMat,clsvec = clsvec,K=K)
      mean<-paralst$M
      var<-paralst$V
      alpha<-paralst$A
      parameternew<-c(mean,var)
      pnewMinusoldLp<-(parameternew-parameterold)^Lp
      Lpvalue<-sum(pnewMinusoldLp)^(1/Lp)
      if(Lpvalue<=ept){
         stoprule<-"Lpvalue<=ept"
         break
      }
      if(iterk>=maxiter){
         stoprule<-"iterk>=maxiter"
         break
      }
      #print(mean);print(var);print(alpha)
      iterk<-iterk+1
   }
   outlst<-list(stoprule=stoprule,iteration=iterk,Mean=mean,
                        Var=var,Alpha=alpha,K=K)
   class(outlst)<-"gaussiEM"
   return(outlst)
}
```

Function_Code: gaussiResponse ()

```
gaussiResponse<-function(clsvec,K=2,Ml=rep(0,K),Vl=rep(1,K),Alpl=rep(1/K,K)){
   lj<-length(clsvec);lk<-K
   rjkMat<-matrix(0,nrow=lj,ncol=lk)
   for(j in 1:lj){
      rvec<-vector(length = lk)
      for(k in 1:lk){
         rjk<-Alpl[k]*dnorm(clsvec[j],mean=Ml[k],sd=sqrt(Vl[k]))
         if(rjk==0) rjk<-sample(c(1e-20,1e-30,1e-40,1e-100,1e-65),1)
         rvec[k]<-rjk
      }
      rjkMat[j,]<-rvec/sum(rvec)
   }
   return(rjkMat)
}
```

Function_Code: gaussiResponse ()

```
##   更新计算各个参数
gaussiParameter<-function(rMat,clsvec,K=2){
   N<-length(clsvec)
   Mplus<-vector(length = K)
   Vplus<-vector(length = K)
   Alplus<-vector(length = K)
   for(i in 1:K){
      rk<-rMat[,i];srk<-sum(rk)
      Mk<-as.vector(rk%*%clsvec)/srk
      ymvec<-(clsvec-Mk)^2
      Vk<-as.vector(rk%*%ymvec)/srk
      Alpk<-srk/N
      Mplus[i]<-Mk;Vplus[i]<-Vk;Alplus[i]<-Alpk
   }
   list(M=Mplus,V=Vplus,A=Alplus)
}
```

Function_Code: print.gaussiEM()

```
print.gaussiEM<-function(obj){
   cat("Stoprule : ",obj$stoprule,"\n")
   cat("iteration : ",obj$iteration,"\n")
   cat("the number of gaussi is : ",obj$K,"\n")
   print(obj[3:5])
}
```

附录 10 HMM 模型的 R 实现

Function_Code: ObjectHMM ()

```
####  隐马尔可夫模型观测序列的生成 ####
#A 是转移概率矩阵，B 是观测概率矩阵，PI 是初始概率向量，Lth 是输出的观测序列长度
#size 为想要生成的观测序列的组数，size，Lth 可以是 vector
ObjectHMM<-function(size=1,Lth=5,A,B,PI,StateLabel=1:nrow(A),
                              ObjectLabel=1:ncol(B),seed=NULL){
  stlab<-StateLabel#各个状态值采用的标记
  objlab<-ObjectLabel#各个观测值采用的标记
  lsi<-size
  if(length(Lth)==1) Lth<-rep(Lth,size)
  stlst<-list()
  objlst<-list()
  if(!is.null(seed)) set.seed(seed=seed)
  for(i in 1:lsi){
    lt<-Lth[i]
    stvec<-vector(length = lt)
    objvec<-vector(length = lt)
    stvec[1]<-sample(stlab,1,prob = PI)#确定初始状态
    st1<-which(stlab==stvec[1])#在 B 中对应的行数，即状态
    objvec[1]<-sample(objlab,1,prob = B[st1,])#确定初始观测
    for(j in 2:lt){
      st<-which(stlab==stvec[j-1])#确定当前状态
      stvec[j]<-sample(stlab,1,prob =A[st,])#确定下一个状态
      stnew<-which(stlab==stvec[j])#下一个状态在 B 中对应的行数
      objvec[j]<-sample(objlab,1,prob = B[stnew,])#确定下一个观测
    }
    stlst[[i]]<-stvec
    objlst[[i]]<-objvec
  }
  outlst<-list(obs=objlst,state=stlst)
  return(outlst)
}
```

Function_Code: forwardHMM ()

```
#### HMM 前向算法实现 ####
forwardHMM<-function(obs,A,B,PI,StateLabel=as.character(1:nrow(A)),
                                ObjectLabel=as.character(1:ncol(B))){
  #obs 为观测值序列
  lT<-length(obs)#观测序列的长度
  Bmat<-B;colnames(Bmat)<-ObjectLabel;rownames(Bmat)<-StateLabel
  Amat<-A;colnames(Amat)<-StateLabel;rownames(Amat)<-StateLabel
  fA<-function(alpcal,Acol) alpcal%*%Acol
  #计算前向概率初值
  AlpMat<-matrix(nrow=lT,ncol = nrow(Amat))
  colnames(AlpMat)<-paste0("st:",StateLabel)
  rownames(AlpMat)<-paste0("T:",1:lT)
  alpha<-PI*Bmat[,which(ObjectLabel==obs[1])]#vector
  AlpMat[1,]<-alpha
  iterk<-2
  while(iterk<=lT){
    #更新迭代结果
    alp<-apply(Amat,2,fA,alpcal=alpha)
    alpha<-alp*Bmat[,which(ObjectLabel==obs[iterk])]
    AlpMat[iterk,]<-alpha
    iterk<-iterk+1
  }
  list(FinalProb=sum(alpha),AlpMat=AlpMat)
}
```

Function_Code: backwardHMM ()

```
#### HMM 模型后向算法的实现 ####
backwardHMM<-function(obs,A,B,PI,StateLabel=as.character(1:nrow(A)),
                                ObjectLabel=as.character(1:ncol(B))){
  #obs 为观测值序列
  IT<-length(obs)#观测序列的长度
  lst<-nrow(A)#状态的取值个数
  Bmat<-B;colnames(Bmat)<-ObjectLabel;rownames(Bmat)<-StateLabel
  Amat<-A;colnames(Amat)<-StateLabel;rownames(Amat)<-StateLabel
  fB<-function(Arow,Bcol,btcal) sum(Arow*Bcol*btcal)
  beta<-rep(1,lst)
  BtMat<-matrix(nrow=IT,ncol = lst)
  colnames(BtMat)<-paste0("st:",StateLabel)
  rownames(BtMat)<-paste0("T:",IT:1)
  BtMat[1,]<-beta
  iterk<-1
  while(iterk<=(IT-1)){#迭代是从最后一个观测开始的
    bcol<-Bmat[,which(ObjectLabel==obs[IT-iterk+1])]
    beta<-apply(Amat,1,fB,Bcol=bcol,btcal=beta)
    BtMat[iterk+1,]<-beta
    iterk<-iterk+1
  }
  bo1<-Bmat[,which(ObjectLabel==obs[1])]
  finalprob<-sum(PI*bo1*beta)
  list(FinalProb=finalprob,BtMat=BtMat)
}
```

Function_Code: stijHMM ()

```
stijHMM<-function(obs,sti=NULL,stij=NULL,time=1,A,B,PI,
                  StateLabel=as.character(1:nrow(A)),
                  ObjectLabel=as.character(1:ncol(B)),
                  if.sti=F,if.stij=F){
  #sti 输入指定的状态，obs 为当前观测序列,stijz 指定转移状态,time 指定时刻
  #sti,time 均可以是长度相等的向量;stij 只能是一个二维向量
  lT<-length(obs)
  AlpIst<-forwardHMM(obs=obs,A=A,B=B,PI=PI,StateLabel = StateLabel,
                     ObjectLabel = ObjectLabel)
  BtIst<-backwardHMM(obs=obs,A=A,B=B,PI=PI,StateLabel = StateLabel,
                     ObjectLabel = ObjectLabel)
  AlpMat<-AlpIst$AlpMat;BtMat<-BtIst$BtMat
  btmat<-BtMat[lT:1,,drop=F]
  Probs<-AlpIst$FinalProb
  if(!if.stij){
    PstiMat<-AlpMat*btmat/Probs#t 时刻状态为 i 的概率矩阵
    rownames(PstiMat)<-rownames(AlpMat);colnames(PstiMat)<-colnames(AlpMat)
    si<-which(StateLabel==sti)
    if(!is.null(sti)&&!is.null(time)) psti<-PstiMat[time,si] else psti<-NULL
  }
  if(!if.sti){
    fbj<-function(j,x,BM) BM[j,which(ObjectLabel==x)]
    wj<-which(StateLabel==stij[2]);wi<-which(StateLabel==stij[1])
    bjDf<-data.frame(jobs=obs[-1])
    bjvec<-apply(bjDf,1,fbj,j=wj,BM=B)
    Pstijvec<-A[wi,wj]*AlpMat[1:(lT-1),wi]*bjvec*btmat[-1,wj]/Probs#长度为 lT-1
    if(!is.null(time)) pstij<-Pstijvec[time] else pstij<-NULL
  }
  if(!if.sti&&!if.stij){
    outlst<-list(Probs=Probs,psti=psti,pstij=pstij,PstiMat=PstiMat,Pstijvec=Pstijvec,
                 AlpMat=AlpMat,BtMat=btmat,sti=sti,stij=stij,time=time)
  } else if(!if.stij&&if.sti){
    outlst<-list(Probs=Probs,psti=psti,PstiMat=PstiMat,AlpMat=AlpMat,
                 BtMat=btmat,sti=sti,stij=stij,time=time)
  } else if(if.stij&&!if.sti){
    outlst<-list(Probs=Probs,pstij=pstij,Pstijvec=Pstijvec,AlpMat=AlpMat,
                 BtMat=btmat,sti=sti,stij=stij,time=time)
  } else{
    stop("if.sti and if.stij can not both TRUE.")
  }
  return(outlst)
}
```

Function_Code: superviseHMM ()

```
####  HMM 监督学习方法的 R 实现 ####
superviseHMM<-function(obsMat,stMat,StateLabel=NULL,ObjectLabel=NULL){
  #obsdf 为观测序列组成的矩阵 T*S，每一列为一次观测序列，行数代表时刻 t=1,2,...,T.
  #stdf 为状态序列组成的矩阵 T*S，每一列为一次状态序列，行数代表时刻 t=1,2,...,T.
  #StateLabel 输入状态的字符标签，ObjectLabel 输入观测的字符标签
  lT<-nrow(obsMat)#时间长度
  S<-ncol(obsMat)#样本量
  stMatchs<-stMat[-lT,]
  lst<-length(StateLabel)#状态集合长度
  lobs<-length(ObjectLabel)#观测集合长度
  obsvec<-as.vector(obsMat);stvec<-as.vector(stMat)
  aijMat<-matrix(nrow=lst,ncol=lst)##转移概率矩阵
  colnames(aijMat)<-StateLabel;rownames(aijMat)<-StateLabel
  bjkMat<-matrix(nrow = lst,ncol=lobs)##观测概率矩阵
  colnames(bjkMat)<-ObjectLabel;rownames(bjkMat)<-StateLabel
  pivec<-vector(length=lst)##初始概率向量
  findaij<-function(ichr,jchr){#计算一个转移概率
    #在 stdf 进行搜索，从时间 t=1 开始至 t=lT-1
    SAij<-length(which(stMatchs==ichr))
    Aij<-0
    for(t in 1:(lT-1)){
      tvec<-stMat[t,];tplus1vec<-stMat[t+1,]
      tlab<-which(tvec==ichr);tplus1st<-tplus1vec[tlab]
      sj<-sum(tplus1st==jchr)
      Aij<-Aij+sj
    }
    return(Aij/SAij)
  }
  findbjk<-function(jchr,kchr){#计算一个观测概率
    jlab<-which(stvec==jchr)
    kvec<-obsvec[jlab]
    sum(kvec==kchr)/length(jlab)
  }
  #计算转移概率矩阵
  for(i in 1:lst){
    for(j in 1:lst){
      aijMat[i,j]<-findaij(ichr=StateLabel[i],jchr = StateLabel[j])
    }
  }
  #计算观测概率矩阵
```

（接下页）

```
#计算观测概率矩阵
   for(j in 1:lst){
      for(k in 1:lobs){
         bjkMat[j,k]<-findbjk(jchr=StateLabel[j],kchr = ObjectLabel[k])
      }
   }
   #计算初始概率向量
   first<-stMat[1,]#初始状态
   for(i in 1:lst){
      pi[i]<-length(which(first==StateLabel[i]))/S
   }
   outlst<-list(pi=pi,aijMat=aijMat,bjkMat=bjkMat,
                        StateLabel=StateLabel,ObjectLabel=ObjectLabel)
   class(outlst)<-c("superviseHMM","HMM")
   return(outlst)
}
```

Function_Code: print.superviseHMM ()

```
print.superviseHMM<-function(obj){
   cat("State::",obj$StateLabel,"; ","Observation::",obj$ObjectLabel,"\n")
   print(obj[1:3])
}
```

Function_Code: print.superviseHMM ()

```
#### HMM 模型 Buam-Welch 无监督学习算法的 R 实现 ####
BuamWelchHMM<-function(obs,A0,B0,PI0,StateLabel=as.character(1:nrow(A0)),
                       ObjectLabel=as.character(1:ncol(B0)),ept=1e-2,
                       maxiter=10000,Lp=2){
  #obs 输入一个观测序列，vector
  lT<-length(obs);lst<-length(StateLabel);lobs<-length(ObjectLabel)
  Abw<-A0;Bbw<-B0;PIbw<-PI0
  iterk<-1
  while(iterk>=1){
    #保存旧参数
    Abwold<-Abw;Bbwold<-Bbw;PIbwold<-PIbw
    #更新初始概率向量
    pilst<-stijHMM(obs=obs,A=Abw,B=Bbw,PI=PIbw,if.sti = TRUE,
                   StateLabel = StateLabel,ObjectLabel = ObjectLabel)
    PIbw<-pilst$PstiMat[1,]
    #更新转移概率矩阵、观测概率矩阵
    for(i in 1:lst){#i 行
      ir<-StateLabel[i]
      #更新转移概率矩阵
      for(j in 1:lst){
        jr<-StateLabel[j]#获取状态标签
        calij<-stijHMM(obs=obs,stij=c(ir,jr),A=Abw,B=Bbw,PI=PIbw,
                       StateLabel = StateLabel,ObjectLabel=ObjectLabel)
        pstiMat<-calij$PstiMat;pstijvec<-calij$Pstijvec
        Abw[i,j]<-sum(pstijvec)/sum(pstiMat[1:(lT-1),i])
      }
      #更新观测概率矩阵
      for(k in 1:lobs){
        klab<-which(obs==ObjectLabel[k])
        cali<-stijHMM(obs=obs,A=Abw,B=Bbw,PI=PIbw,StateLabel = StateLabel,
                      ObjectLabel=ObjectLabel,if.sti=TRUE)
        piMat<-cali$PstiMat
        pstivec<-piMat[,i];pkvec<-pstivec[klab]
        Bbw[i,k]<-sum(pkvec)/sum(pstivec)
      }
    }
    # 停止条件
    Abwoldvec<-as.vector(Abwold);Abwvec<-as.vector(Abw)
    Bbwoldvec<-as.vector(Bbwold);Bbwvec<-as.vector(Bbw)
```

（接下页）

```
      abw<-Abwvec-Abwoldvec;Lpabw<-sum(abs(abw)^Lp)*(1/Lp)
      bbw<-Bbwvec-Bbwoldvec;Lpbbw<-sum(abs(bbw)^Lp)*(1/Lp)
      pibw<-PIbw-PIbwold;Lppibw<-sum(abs(pibw)^Lp)*(1/Lp)
      allbw<-c(abw,bbw,pibw);Lpallbw<-sum(abs(allbw)^Lp)*(1/Lp)
      if(Lpabw<=ept&&Lpbbw<=ept&&Lppibw<=ept){
         stoprule<-"Lpabw<=ept&&Lpbbw<=ept&&Lppibw<=ept"
         break
      }
      if(Lpallbw<=ept){
         stoprule<-"Lpallbw<=ept"
         break
      }
      if(iterk>=maxiter){
         stoprule<-"iterk>=maxiter"
         break
      }
      iterk<-iterk+1
   }
   outlst<-list(pi=PIbw,aijMat=Abw,bjkMat=Bbw,iteration=iterk,
                        stoprule=stoprule,StateLabel=StateLabel,ObjectLabel=ObjectLabel)
   class(outlst)<-c("BuamWelchHMM","HMM")
   return(outlst)
}
```

Function_Code: print. BuamWelchHMM ()

```
print. BuamWelchHMM <-function(obj){
   cat("State::",obj$StateLabel,"; ","Observation::",obj$ObjectLabel,"\n")
   print(obj[1:3])
}
```

Function_Code: MoreBuamWelchHMM ()

```
MoreBuamWelchHMM<-function(obsMat,A0,B0,PI0,StateLabel=as.character(1:nrow(A0)),
                                    ObjectLabel=as.character(1:ncol(B0)),ept=1e-2,
                                    maxiter=10000,Lp=2){
  lMat<-ncol(obsMat)
  lst<-length(StateLabel)
  lobs<-length(ObjectLabel)
  lT<-nrow(obsMat)
  A<-matrix(0,nrow=lst,ncol=lst)
  B<-matrix(0,nrow = lst,ncol=lobs)
  PI<-rep(0,lst)
  for(i in 1:lMat){
    obsuse<-obsMat[,i]
    calst<-BuamWelchHMM(obs=obsuse,A0=A0,B0=B0,PI0=PI0,StateLabel = StateLabel,
                                ObjectLabel = ObjectLabel,maxiter = maxiter,ept = ept,Lp=Lp)
    Arev<-calst$aijMat;Brev<-calst$bjkMat;pirev<-calst$pi
    A<-A+Arev
    B<-B+Brev
    PI<-PI+pirev
  }
  A<-A/lMat;B<-B/lMat;PI<-PI/lMat
  outlst<-list(pi=PI,aijMat=A,bjkMat=B,
                    StateLabel=StateLabel,ObjectLabel=ObjectLabel)
  class(outlst)<-c("BuamWelchHMM","HMM")
  return(outlst)
}
```

Function_Code: approxHMM ()

```
####   预测算法：近似算法的 R 实现  ####
approxHMM<-function(obsMat,A,B,PI,StateLabel=as.character(1:nrow(A)),
                             ObjectLabel=as.character(1:ncol(B))){
  approxone<-function(obs){
    calst<-stijHMM(obs = obs,A=A,B=B,PI=PI,StateLabel = StateLabel,
                          ObjectLabel = ObjectLabel,if.sti = TRUE)
    #只计算状态概率矩阵，不计算转移概率
    pstiMat<-calst$PstiMat#状态概率矩阵
    #对每一行搜索概率最大的状态
    stlab<-apply(pstiMat,1,which.max)
    StateLabel[stlab]
  }
  if(is.vector(obsMat)) obsMat<-matrix(obsMat,nrow=length(obsMat))
  apply(obsMat,2,approxone)
}
```

Function_Code: ViterbiHMM ()

```
#### 预测算法 维特比算法的 R 实现 ####
ViterbiHMM<-function(obs,A,B,PI,StateLabel=as.character(1:nrow(A)),
                                ObjectLabel=as.character(1:ncol(B)),if.show=TRUE){
  lst<-length(StateLabel)
  lT<-length(obs)
  lobs<-length(ObjectLabel)
  obs1<-which(ObjectLabel==obs[1])
  delta<-PI*B[,obs1]#初始化 delta
  pasi<-rep(0,lst)#初始化 pasi
  deltaMat<-matrix(nrow=lT,ncol = lst)#delta 矩阵，行为时刻，列为状态
  pasiMat<-matrix(nrow=lT,ncol = lst)#pasi 矩阵，行为时刻，列为状态
  deltaMat[1,]<-delta;pasiMat[1,]<-pasi
  #进入递推
  iterk<-2
  while(iterk<=lT){
    obslab<-which(ObjectLabel==obs[iterk])
    bitvec<-B[,obslab]
    #利用 R 矩阵乘法特点
    delm<-matrix(rep(delta,lst),nrow = lst,byrow = TRUE)
    useMat<-delm*t(A)#对应位置相乘
    pasi<-apply(useMat,1,which.max)#每行取最大,vector
    caldelta<-vector(length = lst)
    for(i in 1:lst){
      caldelta[i]<-useMat[i,pasi[i]]
    }
    delta<-caldelta*bitvec
    deltaMat[iterk,]<-delta
    pasiMat[iterk,]<-pasi
    iterk<-iterk+1
  }
  #先找到最优路径的概率及终点
  statelab<-vector(length = lT)
  finalstatelab<-which.max(delta)#获取位置
  finalprob<-delta[finalstatelab]
  statelab[lT]<-finalstatelab
  #回溯
  for(j in (lT-1):1){
    statelab[j]<-pasiMat[j+1,statelab[j+1]]
  }
  rownames(deltaMat)<-paste0("T:",1:lT)
```

（接下页）

```
colnames(deltaMat)<-paste0("st:",StateLabel)
   rownames(pasiMat)<-paste0("T:",1:lT)
   colnames(pasiMat)<-paste0("st:",StateLabel)
   predvec<-StateLabel[statelab];names(predvec)<-paste0("T:",1:lT)
   if(if.show){
      out<-list(FinalState=predvec,deltaMat=deltaMat,pasiMat=pasiMat)
   } else{
      out<-predvec
   }
   return(out)
}
```

Function_Code: MoreViterbiHMM ()

```
####   批量预测
MoreViterbiHMM<-function(obsMat,A,B,PI,StateLabel=as.character(1:nrow(A)),
                                    ObjectLabel=as.character(1:ncol(B))){
   if(is.vector(obsMat)) obsMat<-matrix(obsMat,nrow=length(obsMat))
   apply(obsMat,2,ViterbiHMM,A=A,B=B,PI=PI,
            StateLabel=StateLabel,ObjectLabel=ObjectLabel,if.show=FALSE)
}
```