ID3
C4.5 多叉
CART 二叉
RF.

8.3 实验：决策树

8.3.1 构建分类树

加载 tree 包以建立分类树和回归树。

```
> library(tree)
```

首先用分类树分析 Carseats（座椅）数据集。由于数据中的 Sales（销量）是一个连续变量，所以需要将它记为二元变量。用函数 ifelse()创建一个名为 High（高销量）的变量，若 Sales 的值大于 8，变量 High 就取 Yes，否则取 No。

```
> library(ISLR)
> attach(Carseats)
> High=ifelse(Sales<=8,"No","Yes")
```

最后用函数 data.frame()将变量 High 与 Carseats 数据集中的其他数据合并。

```
> Carseats=data.frame(Carseats,High)
```

现在用函数 tree()建立分类树，用除 Sales 之外的所有变量预测 High。函数 tree()的语法与函数 lm()类似。

```
> tree.carseats=tree(High~.-Sales,Carseats)
```

函数 summary()列出了用于生成终端结点的所有变量、终端结点个数和（训练）错误率。

```
> summary(tree.carseats)

Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"
[5] "Population"  "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

可知训练错误率是 9%。函数 summary()输出的分类树偏差由下式给出：

$$-2\sum_m\sum_k n_{mk}\log\hat{p}_{mk}$$

这里的 n_{mk} 是第 m 个终端结点处属于第 k 类的观测值的个数。偏差小说明一棵树很好地拟合了（训练）数据。输出的平均残差是用偏差除以 $n-|T_0|$ 得到的，在这里为 400 − 27 = 373。

树最吸引人的特点之一就是它可以用图形表示。用函数 plot()显示树的结构，用函数 text()显示结点标记。参数 pretty=0 使 R 输出所有定性预测变量的类别名，而不是仅仅展示各个类名的首字母。

```
> plot(tree.carseats)
> text(tree.carseats,pretty=0)
```

对 Sales 最重要的指标应该是架设位置，因为第一个分支就将 Good 位置与 Bad 和 Medium 位置进行了区分。

如果只输入树对象的名字，R 会输出树上每个分支的结果。R 会将分裂规则（例如 Price <92.5）、这一分支上的观测值的数量、偏差、这一分支的整体预测（Yes 或 No）和这一分

注意，与其变量名不同，dev 此时对应的是交叉验证错误率。当结
错误率最低，共有 50 个交叉验证误差。我们画出错误率对 size 和

```
> par(mfrow=c(1,2))
> plot(cv.carseats$size,cv.carseats$dev,type="b")
> plot(cv.carseats$k,cv.carseats$dev,type="b")
```

用函数 prune.misclass()剪枝以得到有 9 个结点的树。

```
> prune.carseats=prune.misclass(tree.carseats,best=9)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
```

剪枝后的树在测试集上的效果如何？再一次使用函数 predict(

```
> tree.pred=predict(prune.carseats,Carseats.test,type="c
> table(tree.pred,High.test)
          High.test
tree.pred No Yes
      No  94  24
      Yes 22  60
> (94+60)/200
[1] 0.77
```

现在 77% 的测试观测被分到正确的类别中，所以剪枝过程不仅生成了
而且提高了分类准确性。

如果增大 best 的值，剪枝后的树会更大，同时分类正确率也更低。

```
> prune.carseats=prune.misclass(tree.carseats,best=15)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
> tree.pred=predict(prune.carseats,Carseats.test,type="clas
> table(tree.pred,High.test)
          High.test
tree.pred No Yes
      No  86  22
      Yes 30  62
> (86+62)/200
[1] 0.74
```

8.3.2 构建回归树

这里用 Boston 数据集建立回归树。首先，创建训练集并根据训练集

```
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> tree.boston=tree(medv~.,Boston,subset=train)
> summary(tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.1000  -2.0420  -0.0536  0.0000   1.9800  12.6000
```

支中取 Yes 和 No 的观测值的比例都显示出来。引申出终端结点的分支用星号标出。

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
 1) root 400 541.5 No ( 0.590 0.410 )
   2) ShelveLoc: Bad,Medium 315 390.6 No ( 0.689 0.311 )
     4) Price < 92.5 46  56.53 Yes ( 0.304 0.696 )
       8) Income < 57 10  12.22 No ( 0.700 0.300 )
```

为合理评价分类树在这个数据集上的分类效果，必须估计测试误差，而不是仅仅计算训练误差。将所有观测值分为训练集和测试集两部分，用训练集建立分类树，在测试集上评估此树的预测效果。可以用函数 predict()完成这一任务。在分类树情况下，参数 type = " class" 使 R 返回真实的预测类别。这种方法能对测试集上约 71.5% 的数据作出正确预测。

```
> set.seed(2)
> train=sample(1:nrow(Carseats), 200)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]
> tree.carseats=tree(High~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  86  27
      Yes 30  57
> (86+57)/200
[1] 0.715
```

接下来，考虑剪枝能否改进预测结果。用函数 cv.tree()执行交叉验证以确定最优的树复杂性，用成本复杂性剪枝选择要考虑的一系列树。选择对象属性 FUN = prune.misclass 表明，用分类错误率而不是函数 cv.tree()的默认值偏差来控制交叉验证和剪枝过程。函数 cv.tree()给出了所考虑的每棵树的终端结点个数（size），相应的分类错误率，以及使用的成本复杂性参数值（k，对应图 8-4 中的 α）。

```
> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
> names(cv.carseats)
[1] "size"   "dev"    "k"      "method"
> cv.carseats
$size
[1] 19 17 14 13  9  7  3  2  1

$dev
[1] 55 55 53 52 50 56 69 65 80

$k
[1]       -Inf  0.0000000  0.6666667  1.0000000  1.7500000
    2.0000000  4.2500000
[8]  5.0000000 23.0000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

注意，与其变量名不同，dev 此时对应的是交叉验证错误率。当终端结点数为 9 时，交叉验证错误率最低，共有 50 个交叉验证误差。我们画出错误率对 size 和 k 的函数。

```
> par(mfrow=c(1,2))
> plot(cv.carseats$size,cv.carseats$dev,type="b")
> plot(cv.carseats$k,cv.carseats$dev,type="b")
```

用函数 prune.misclass()剪枝以得到有 9 个结点的树。

```
> prune.carseats=prune.misclass(tree.carseats,best=9)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
```

剪枝后的树在测试集上的效果如何？再一次使用函数 predict()。

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
          High.test
tree.pred  No Yes
      No   94  24
      Yes  22  60
> (94+60)/200
[1] 0.77
```

现在 77% 的测试观测被分到正确的类别中，所以剪枝过程不仅生成了一棵更易于解释的树，而且提高了分类准确性。

如果增大 best 的值，剪枝后的树会更大，同时分类正确率也更低。

```
> prune.carseats=prune.misclass(tree.carseats,best=15)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
          High.test
tree.pred  No Yes
      No   86  22
      Yes  30  62
> (86+62)/200
[1] 0.74
```

8.3.2 构建回归树

这里用 Boston 数据集建立回归树。首先，创建训练集并根据训练数据生成树。

```
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> tree.boston=tree(medv~.,Boston,subset=train)
> summary(tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.1000  -2.0420  -0.0536   0.0000   1.9600  12.6000
```

注意到 summary() 的输出结果意味着在创建树时只用了三个变量。在回归树情况下，偏差是树的预测值的平方误差的简单相加。现在画出这棵树：

```
> plot(tree.boston)
> text(tree.boston,pretty=0)
```

变量 lstat（社区财富水平）代表社会经济地位低的个体所占比例。这棵树表明 lstat 值越低对应的房价越贵。这棵树预测：在常住居民社会经济地位较高的郊区，大家庭的房子的房价中位数是 46 400 美元（房屋面积 rm >=7.437 和 lstat <9.715）。

用函数 cv.tree() 观察剪枝是否提升了树的预测效果。

```
> cv.boston=cv.tree(tree.boston)
> plot(cv.boston$size,cv.boston$dev,type='b')
```

本例中，最复杂的树是由交叉验证选出的。但如果希望对树剪枝，可以用函数 prune.tree() 进行如下操作：

```
> prune.boston=prune.tree(tree.boston,best=5)
> plot(prune.boston)
> text(prune.boston,pretty=0)
```

为了与交叉验证的结果相符，用未剪枝的树对测试集进行预测。

```
> yhat=predict(tree.boston,newdata=Boston[-train,])
> boston.test=Boston[-train,"medv"]
> plot(yhat,boston.test)
> abline(0,1)
> mean((yhat-boston.test)^2)
[1] 25.05
```

换句话说，回归树的测试均方误差是 25.05。因此均方误差的平方根是 5.005，这意味着这个模型的测试预测值与郊区真实房价的中位数之差在 5 005 美元之内。

8.3.3 装袋法和随机森林

这里用 R 中的 randomForest 包在 Boston 数据集上实现装袋法和随机森林。本书中结果的准确性取决于 randomForest 包和 R 的具体版本。回忆前文，装袋法是随机森林在 m=p 时的一种特殊情况。因此函数 randomForest() 既可以用来做随机森林，也可以执行装袋法。应用装袋法如下：

```
> library(randomForest)
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=13,importance=TRUE)
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13,
     importance = TRUE,       subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 10.77
                    % Var explained: 86.96
```

参数 mtry = 13 意味着树上的每一个分裂点都应该考虑全部 13 个预测变量——也就是说，应该执行装袋法。那么，装袋法模型在测试集上效果如何呢？

```
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> plot(yhat.bag, boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 13.16
```

装袋法回归树的测试均方误差是 13.16，几乎仅为经剪枝后最好的单棵树的测试均方误差的一半。可以用参数 ntree 改变由 randomForest() 生成的树的数目。

```
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=13,ntree=25)
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> mean((yhat.bag-boston.test)^2)
[1] 13.31
```

生成随机森林的过程和生成装袋法模型的过程一样，区别只是所取的 mtry 值更小。函数 randomForest() 默认在用回归树建立随机森林时取 $p/3$ 个变量，而用分类树建立随机森林时取$\sqrt{p}$个变量。这里取 mtry = 6。

```
> set.seed(1)
> rf.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=6,importance=TRUE)
> yhat.rf = predict(rf.boston,newdata=Boston[-train,])
> mean((yhat.rf-boston.test)^2)
[1] 11.31
```

测试均方误是 11.31，这意味着在这种情况下，随机森林会对装袋法有所提升。

可以用函数 importance() 浏览各变量的重要性。

```
> importance(rf.boston)
        %IncMSE IncNodePurity
crim     12.384       1051.54
zn        2.103         50.31
indus     8.390       1017.64
chas      2.294         56.32
nox      12.791       1107.31
rm       30.754       5917.26
age      10.334        552.27
dis      14.641       1223.93
rad       3.583         84.30
tax       8.139        435.71
ptratio  11.274        817.33
black     8.097        367.00
lstat    30.962       7713.63
```

上面列出了变量重要性的两个测度。前者基于当一个给定的变量被排除在模型之外时，预测袋外样本的准确性的平均减小值。后者衡量的是由此变量导致的分裂使结点不纯度减小的总量（见图 8-9）。在回归树中，结点不纯度是由训练集 RSS 衡量的，而分类树的结点纯度是由偏差衡量的。反映这些变量重要程度的图可由函数 varImpPlot() 画出。

```
> varImpPlot(rf.boston)
```

结果表明，在随机森林考虑的所有变量中，lstat 和 rm 是目前最重要的两个变量。

8.3.4 提升法

用 gbm 包和其中的 gbm() 函数对 Boston 数据集建立回归树。由于是回归问题，在执行 gbm() 时选择 distribution = " gaussian"；如果是二分类问题，应选择 distribution = " bernoulli"。对象 n.trees = 5 000 表示提升法模型共需要 5 000 棵树，选项 interaction.depth = 4 限制了每棵树的深度。

```
> library(gbm)
> set.seed(1)
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=
          "gaussian",n.trees=5000,interaction.depth=4)
```

用函数 summary() 生成一张相对影响图，并输出相对影响统计数据。

```
> summary(boost.boston)
        var    rel.inf
1     lstat   45.96
2        rm   31.22
3       dis    6.81
4      crim    4.07
5       nox    2.56
6   ptratio    2.27
7     black    1.80
8       age    1.64
9       tax    1.36
10    indus    1.27
11     chas    0.80
12      rad    0.20
13       zn    0.015
```

可见 lstat 和 rm 是目前最重要的变量。还可以画出这两个变量的**偏相关图**（partial dependence plot）。这些图反映的是排除其他变量后，所选变量对响应值的边际影响。在这个例子中，住宅价格中位数随 rm 的增大而增大，随 lstat 的增大而减小。

```
> par(mfrow=c(1,2))
> plot(boost.boston,i="rm")
> plot(boost.boston,i="lstat")
```

用提升法模型在测试集上预测 medv。

```
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],
          n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 11.8
```

所得的测试均方误差是 11.8，这个结果与随机森林类似，比装袋法略好。如果需要，可以在式 (8.10) 中取不同的压缩参数 λ 做提升法。λ 的默认值是 0.001，但很容易修改。这里取 $\lambda = 0.2$。

```
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=
    "gaussian",n.trees=5000,interaction.depth=4,shrinkage=0.2,
    verbose=F)
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],
          n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 11.5
```

在本例中，用 $\lambda=0.2$ 得到的测试均方误差比 $\lambda=0.001$ 略低。

8.4 习题

概念

1. 画出一个可以由递归二叉分裂得到的二维特征空间划分。例子中至少包含六个区域。画出与这一划分相对应的决策树。确保图中的所有要点都被标出，包括 R_1，R_2，…和分割点 t_1，t_2，…，等等。

   提示：结果应与图 8-1 和图 8-2 相似。

2. 8.2.3 节中提到，用深度为 1 的树（或树桩）构建提升法会得到加法模型，也就是如下形式的模型：

$$f(X)=\sum_{j=1}^{p}f_j(X_j)$$

   解释这种情况出现的原因。可以从算法 8.2 中的式（8.12）开始。

3. 在一个仅有两个类别的简单分类情况下考虑基尼系数、分类误差和互熵。创建一张图，将这几个量分别表示为 $\hat{p}_{m1}$ 的函数。$\hat{p}_{m1}$ 在 x 轴上，取值范围是从 0 到 1。y 轴应展示基尼系数、分类误差和互熵的值。

   提示：在仅有两个类别的情况下，$\hat{p}_{m1}=1-\hat{p}_{m2}$。可以手工作图，但在 R 中完成这一工作要简单得多。

4. 本题与图 8-12 相关。

   （a）简述与图 8-12（左）的预测变量空间划分相对应的树。矩形区域内的数字表示各个区域中 Y 的均值。

   （b）用图 8-12（右）中的树创建一个与图 8-12（左）类似的图。将预测变量空间正确地划分为多个区域，并标出每个区域的均值。

5. 假设我们从包含红绿两个类别的数据集中产生 10 个自助抽样样本。然后对每个自助抽样样本建立一棵分类树，并对特定的 X 值，产生 P（类别是 Red $|$ X 的概率）的 10 个估计：

   0.1，0.15，0.2，0.2，0.55，0.6，0.6，0.65，0.7 和 0.75

   有两种常用的方法可以将这些结果结合成一个预测类别。其一是本章讨论的多数投票。其二是根据平均概率进行分类。在本例中，两种方法的最终分类结果分别是什么？

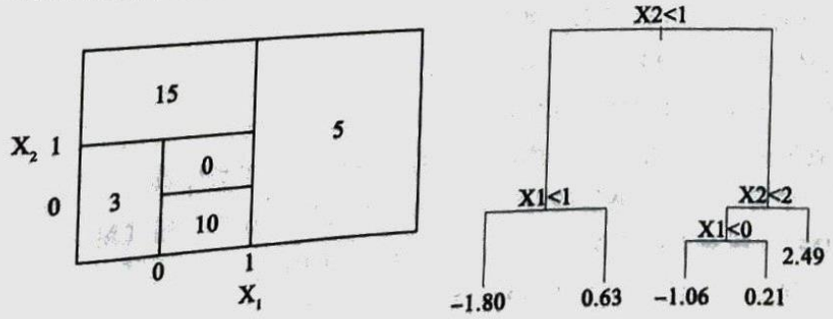

图 8-12 左：习题 4a 中的预测变量空间划分 右：习题 4b 中的树。

6. 对用于回归树的算法作详细说明。