第8章 基于树的方法

（一）决策树基本原理

基于树的方法

- 本章我们学习**基于树(tree-based)**的回归和分类方法。
- 这些方法主要根据**分层(stratifying)**和**分割(segmenting)**的方式将预测变量空间划分为一系列简单区域。
- 由于划分预测变量空间的分裂规则可以被概括为一棵树，所以这类方法被称为**决策树(decision-tree)**方法。

优缺点

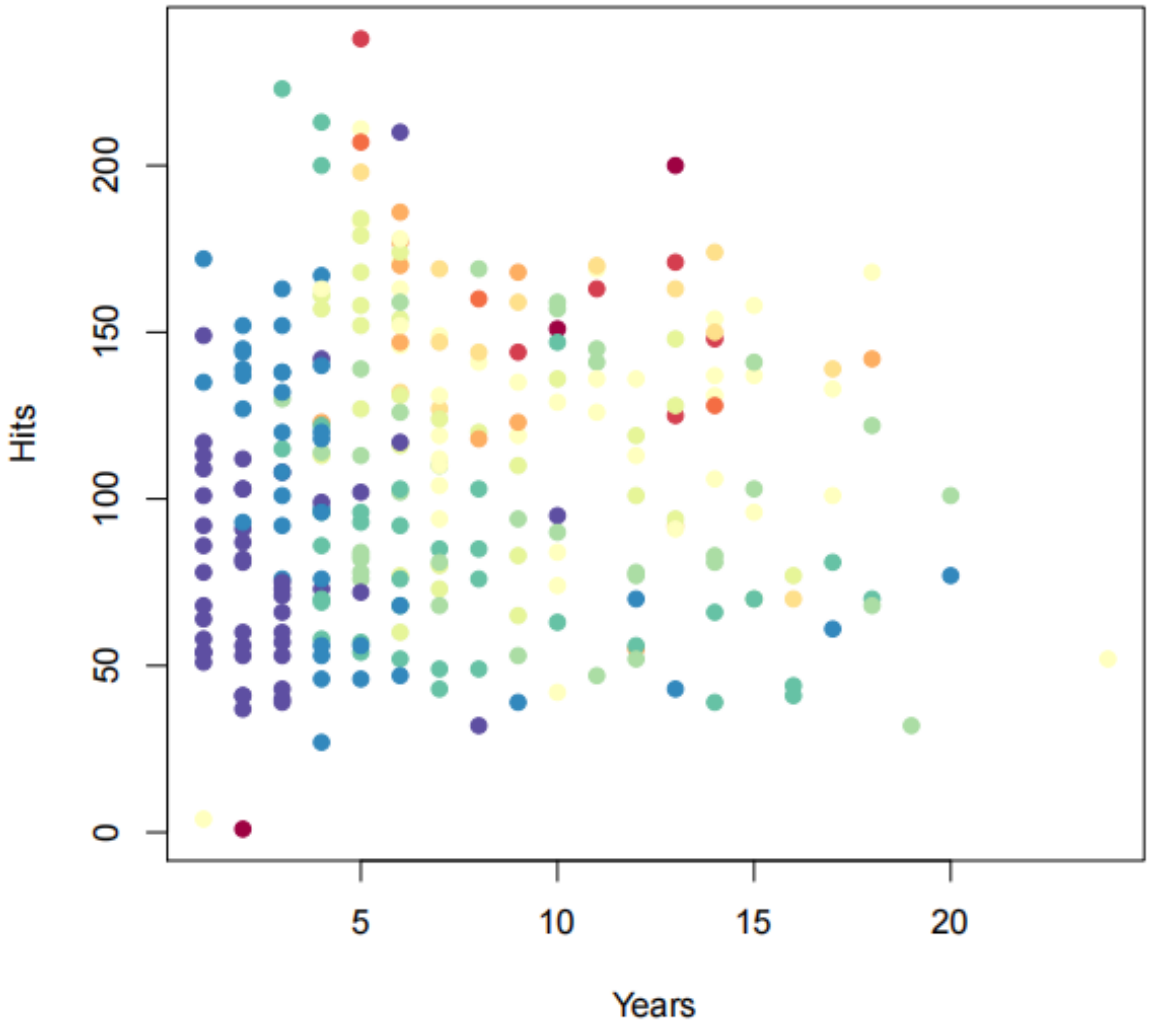
- 基于树的方法简便且易于解释。
- 然而，就预测准确性而言，基于树的方法通常达不到最佳监督学习方法的水平。
- 因此，我们还介绍了**装袋法(bagging)**、**随机森林(random forests)**和**提升法(boosting)**。这些方法都是先建立多棵树，然后将其组合并根据表决产生预测。
- 将大量的树组合后通常可以显著提高预测准确性，但会损失一些解释性。

决策树基本原理

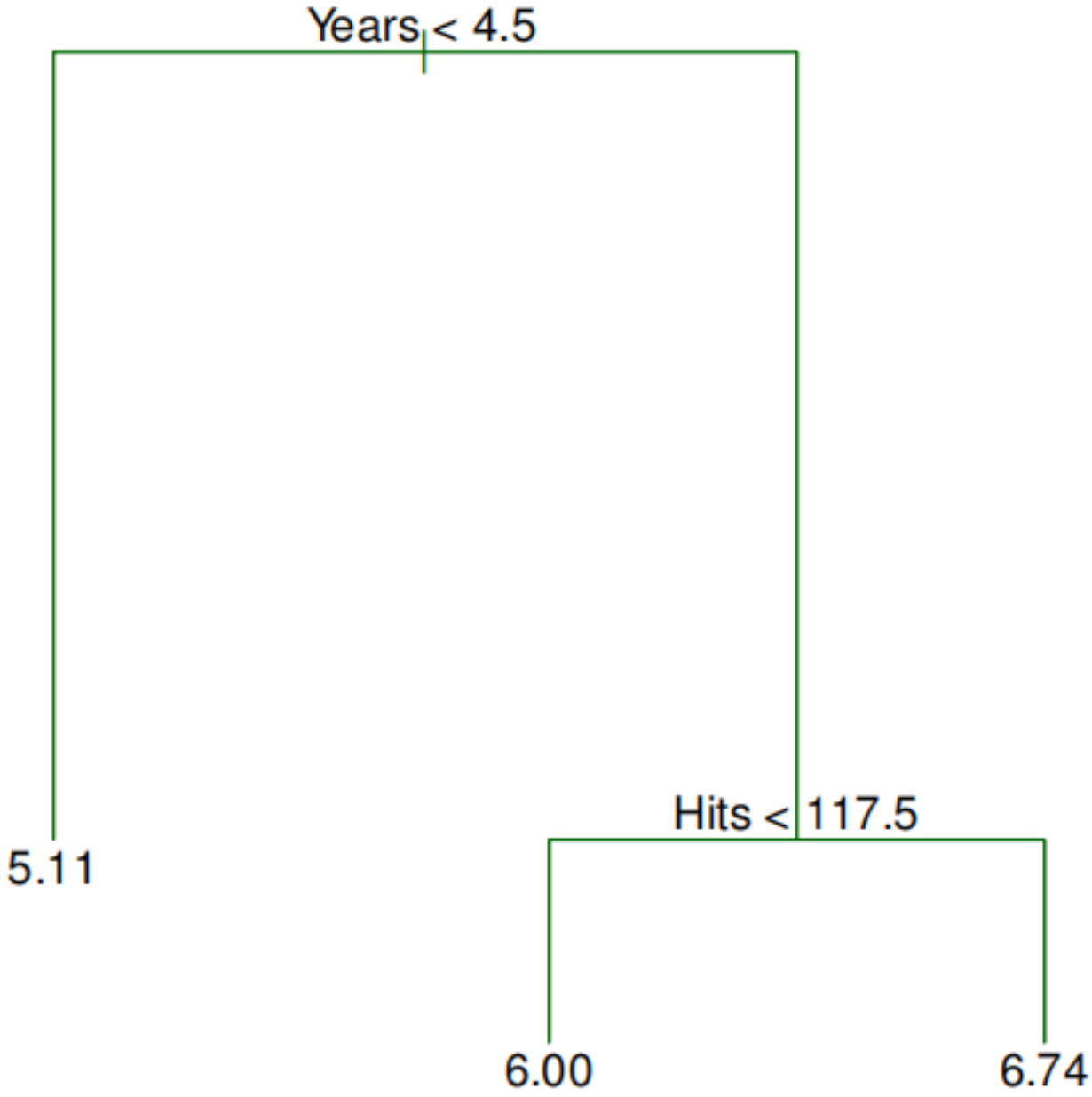
- 决策树可以应用于回归问题和分类问题。
- 下面先介绍**回归树(regression tree)**，再介绍**分类树(classification tree)**。

棒球运动员薪水数据：你会如何分层?

- 薪水用颜色编码，从低(蓝色、绿色)到高(黄色、红色)。



基于Hitters数据集的回归树

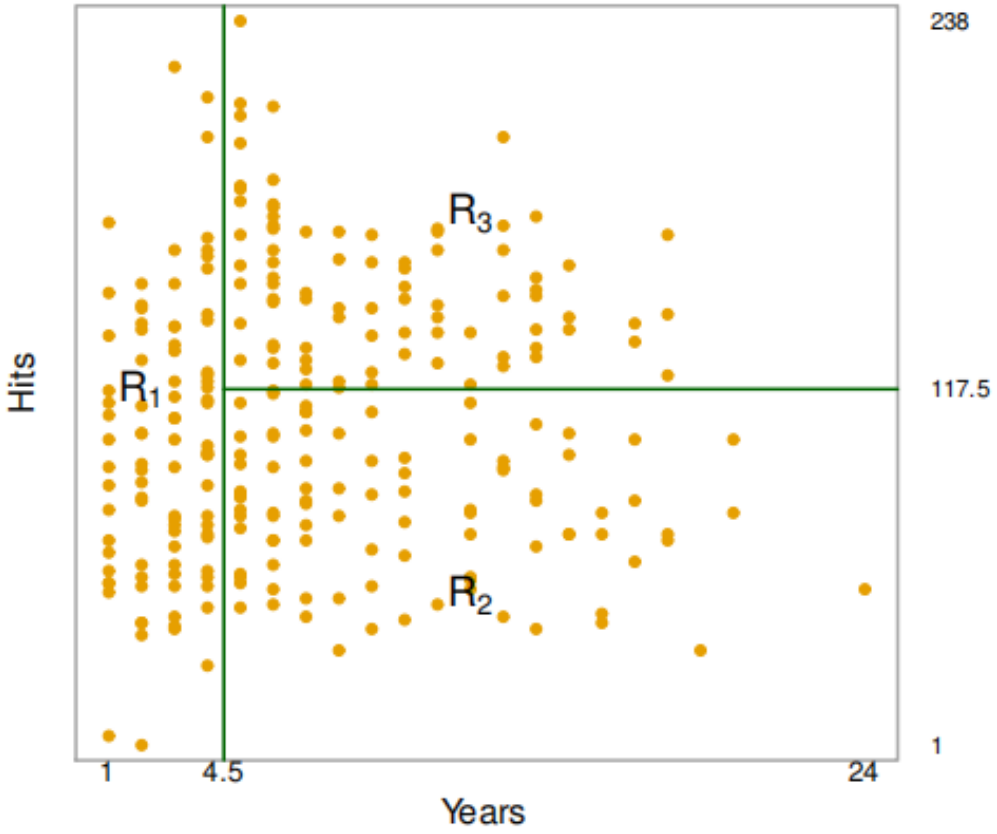


基于Hitters数据集的回归树

- 这是根据Hitters数据建立的一个回归树，根据运动员在职业棒球大联盟效力的**年数(Years)**和上一年的**安打次数(Hits)**预测运动员的对数薪水。
- 内部结点上的标记(形式为$X_j < t_k$)表示的是从这个分裂点发散出的左分支，右侧的分支对应于$X_j \geq t_k$。以这棵树为例，树顶部的分裂点产生出两个大分支。左侧分支对应效力于职业棒球大联盟的年头数**Years**<4.5，右侧分支对应效力于职业棒球大联盟的年头数**Years**>=4.5。
- 这棵树有两个内部结点和三个终端结点(树叶)。每个树叶上的数字表示落在这个树叶处观测值的平均响应值。

基于Hitters数据集的回归树

- 总的来说，回归树将Hitters数据划分成预测空间的三个区域：$R_1 = \{X \mid Years < 4.5\}$，$R_2 = \{X \mid Years >= 4.5, Hits$



基于Hitters数据集的回归树

树的术语：

- 与自然界中的树类似，区域R_1、R_2和R_3称为树的**终端结点(terminal nodes)**或**树叶(leaf)**。
- 决策树通常是从上到下绘制的，树叶位于树的底部。
- 沿树将预测变量空间分开的点称为**内部结点(internal nodes)**。
- 在Hitters树中，文字**Years**< 4.5和**Hits**< 117.5标示出了两个内部结点。

基于Hitters数据集的回归树

结果解释：

- 在Hitters树中，文字**Years**< 4.5和**Hits**< 117.5标示出了两个内部结点。
- **Years**是决定薪水的最重要因素，和有经验的运动员相比，经验少的运动员薪水更低。
- 如果一名运动员效力年数少，那么他在前一年的**Hits**次数对薪水几乎没有影响。
- 但在联盟效力5年及以上的运动员中，前一年的**Hits**次数对薪水是有影响的，去年完成更多Hits的运动员往往会得到更高的薪水。
- 回归树或许是对变量**Years**，**Hits**和**Salary**之间真实关系的一种过度简化，但与其他形式的回归模型相比，它的优势在于解释起来更简便，且能很好的用图形来表示。

建立回归树的过程

- 将预测变量空间(即$X_1, X_2,\ldots,X_p$)分割成J个互不重叠的区域$R_1, R_2, \ldots, R_j$。
- 对于落入区域R_j的每个观测值做同样的预测，预测值等于R_j上训练集的响应值的简单算术平均。

建立回归树的过程

- 理论上，这些区域的形状是任意的。但出于模型简化和增强可解释性的考虑，我们选择将预测变量空间划分为高维矩形，或称**盒子(box)**。
- 划分区域的目标是找到使模型的残差平方和RSS最小的矩形区域$R_1, R_2, \ldots, R_j$。RSS的定义为：

$$\sum_{j=1}^{J}\sum_{i\in R_j}\left(y_i-\hat{y}_{R_j}\right)^2$$

上式中的$\hat{y}_{R_j}$是第j个矩形区域中训练集的平均响应值。

建立回归树的过程

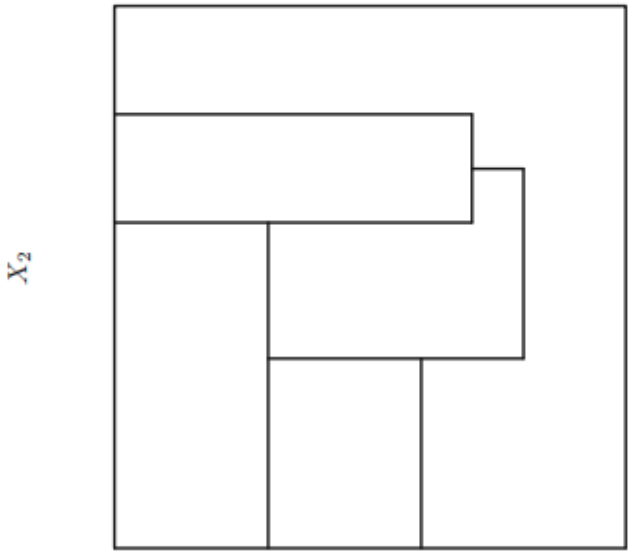
- 遗憾的是，要想考虑将特征空间划分为J个矩形区域的所有可能性，在计算上是不可行的。
- 因此，我们一般采用一种**自上而下(top-down)**、**贪婪(greedy)**方法：**递归二叉分裂(recursive binary splitting)**。
- “自上而下”指的是它从树顶端开始依次分裂预测变量空间，每个分裂点都产生两个新的分支。
- “贪婪”意指在建立树的每一步中，**最优(best)**分裂确定仅限于某一步进程，而不是针对全局去选择那些能够在未来进程中构建出更好的树的分裂点。

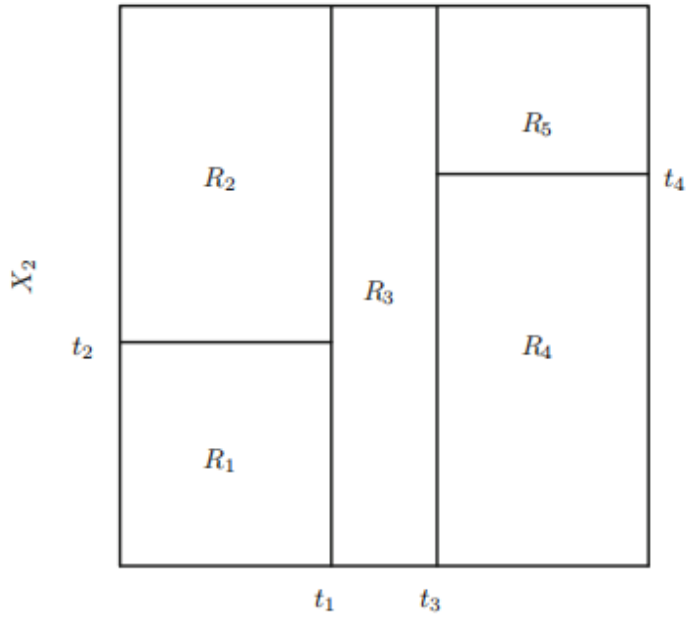
建立回归树的过程

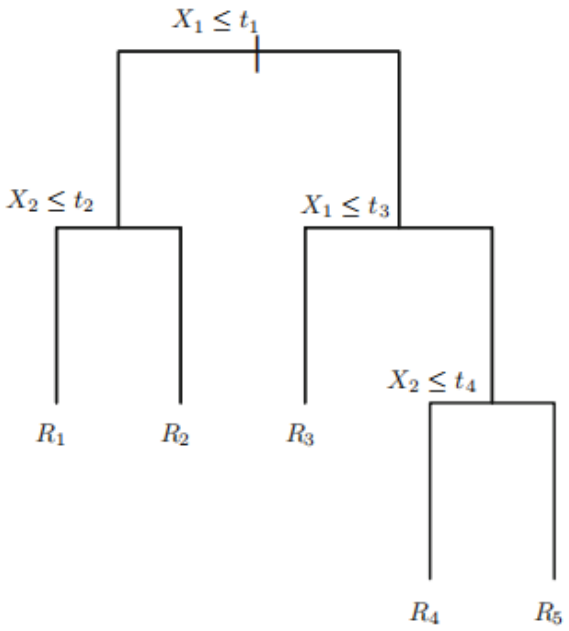
- 我们首先选择预测变量X_j和分割点s，将预测变量空间分为两个区域$\{X \mid X_j < s\}$和$\{X \mid X_j \geq s\}$，使RSS尽可能地减小。
- 重复上述步骤，寻找继续分割数据集的最优预测变量和最优分割点，使随之产生的区域中的RSS达到最小。
- 此时被分割的不再是整个预测变量空间，而是之前确定的两个区域之一。如此一来就能得到3个区域。
- 接着进一步分割3个区域之一以最小化RSS。这一过程不断持续，直到符合某个停止准则:譬如，当所有区域包含的观测值个数都不大于5时，分裂停止。

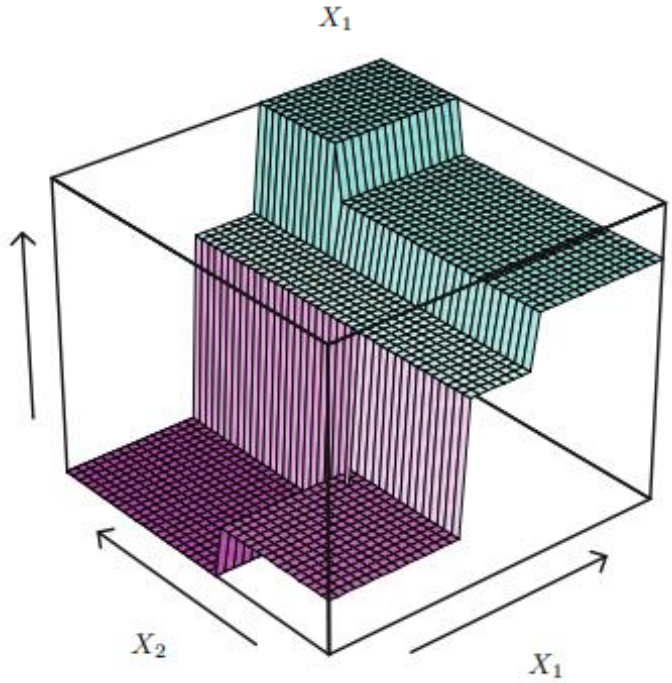
预测

- 区域$R_1, R_2, \ldots, R_j$产生后，就可以确定某一给定测试数据所属的区域，并用这一区域的训练集平均响应值对其进行预测。
- 下一个幻灯片显示了将预测变量空间划分为五个区域的示例。


X_2


R_1
R_2
R_3
R_4
R_5
t_1
t_2
t_3
t_4
X_1
X_2


$X_1 \le t_1$
$X_2 \le t_2$
$X_1 \le t_3$
$X_2 \le t_4$
R_1
R_2
R_3
R_4
R_5


X_1
X_2
X_1

上图详细说明：

- 左上：一种不能由递归二叉分裂得到的二维特征空间划分。
- 右上：二维例子中的递归二叉分裂结果。
- 左下：右上图对应的回归树。
- 右下：回归树所对应的预测平面的透视图。

过拟合、欠拟合问题怎么解决？

树的剪枝

- 上述方法会在训练集中取得良好的预测效果，却很有可能造成数据的过拟合，导致在测试集上效果不佳。为什么？
- 一棵分裂点更少、规模更小(区域$R_1, R_2, \ldots, R_j$的个数更少)的树会有更小的方差和更好的可解释性(以增加微小偏差为代价)。
- 针对上述问题，一种可能的解决办法是：仅当分裂使残差平方和RSS的减小量超过某阈值时，才分裂树结点。
- 这种策略能生成较小的树，但可能产生过于短视的问题，一些起初看来不值得的分裂却可能之后产生非常好的分裂，也就是说在下一步中，RSS大幅减小。

树的剪枝

- 因此，更好的策略是生成一棵很大的树T_0，然后通过**剪枝(prune)**得到**子树(sub-tree)**。
- **代价复杂性剪枝(cost complexity pruning)**——也称**最弱联系剪枝(weakest link pruning)**——可以完成这一任务。
- 我们考虑以非负调整参数α标记的一系列子树。每一个α的取值对应一棵子树$T \subset T_0$，当α一定时，其对应的子树使下式最小:

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha\,|T|$$

这里的$|T|$表示树T的终端结点数，R_m是第m个终端结点对 应的矩形(预测向量空间的一个子集)，$\hat{y}_{R_m}$是与R_m对应的预测值，也就是R_m中训练集的平均值。

选择最佳子树

- 调整系数α在子树的复杂性和与训练数据的契合度之间控制权衡。
- 我们使用交叉验证选择一个最优值$\hat{\alpha}$。
- 然后在整个数据集中找到与$\hat{\alpha}$对应的子树。

建立回归树算法

1. 利用递归二叉分裂在训练集中生成一颗大树，只有当终端结点包含的观测值个数低于某个最小值时才停止。
2. 对大树进行代价复杂性剪枝，得到一系列最优子树，子树是α的函数。
3. 利用K折交叉验证选择α。具体做法是将训练集分为K折。对所有的$k = 1, \ldots, K$,有：
   a. 对训练集上所有不属于第k折的数据重复步骤1和2，得到与 α 一一对应的子树。
   b. 求出上述子树在第k折上的均方预测误差。

   上述操作结束后，每个α会有相应的K个均方预测误差，对这K个值求平均，选出使平均误差最小的α。
4. 找出选定的α值在步骤2中对应的子树即可。

Boston数据集建立回归树

首先，创建数据集并根据训练数据生成树。

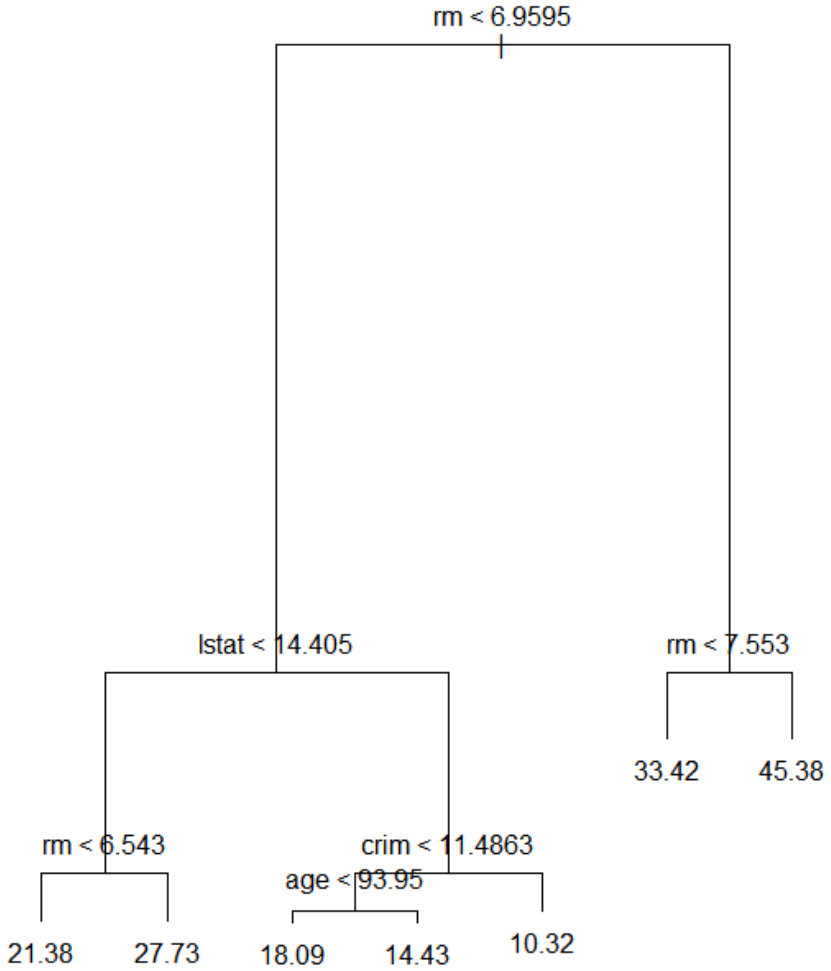
```
> library(tree)
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> tree.boston=tree(medv~.,Boston,subset=train)
> summary(tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"  "age"
Number of terminal nodes:  7
Residual mean deviance:  10.38 = 2555 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

注意到summary()的输出表明在构造树时只使用了三个变量。在回归树的情况下，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)
```

Boston数据集建立回归树

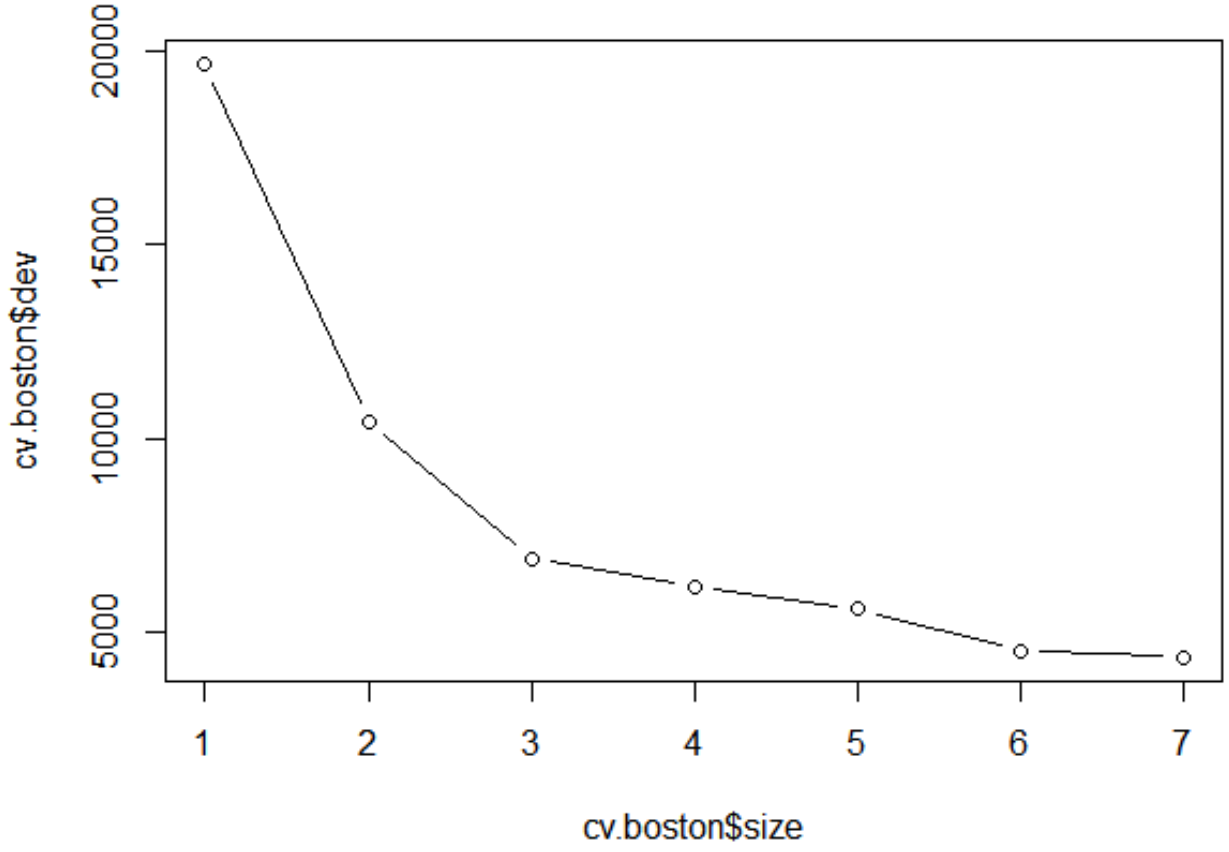
生成的树如下图所示，变量lstat衡量的是社会经济地位低的个体所占比例。该树表明较低的lstat值对应于较昂贵的房子。



Boston数据集建立回归树

用cv.tree()函数观察剪枝是否提升了树的预测效果。

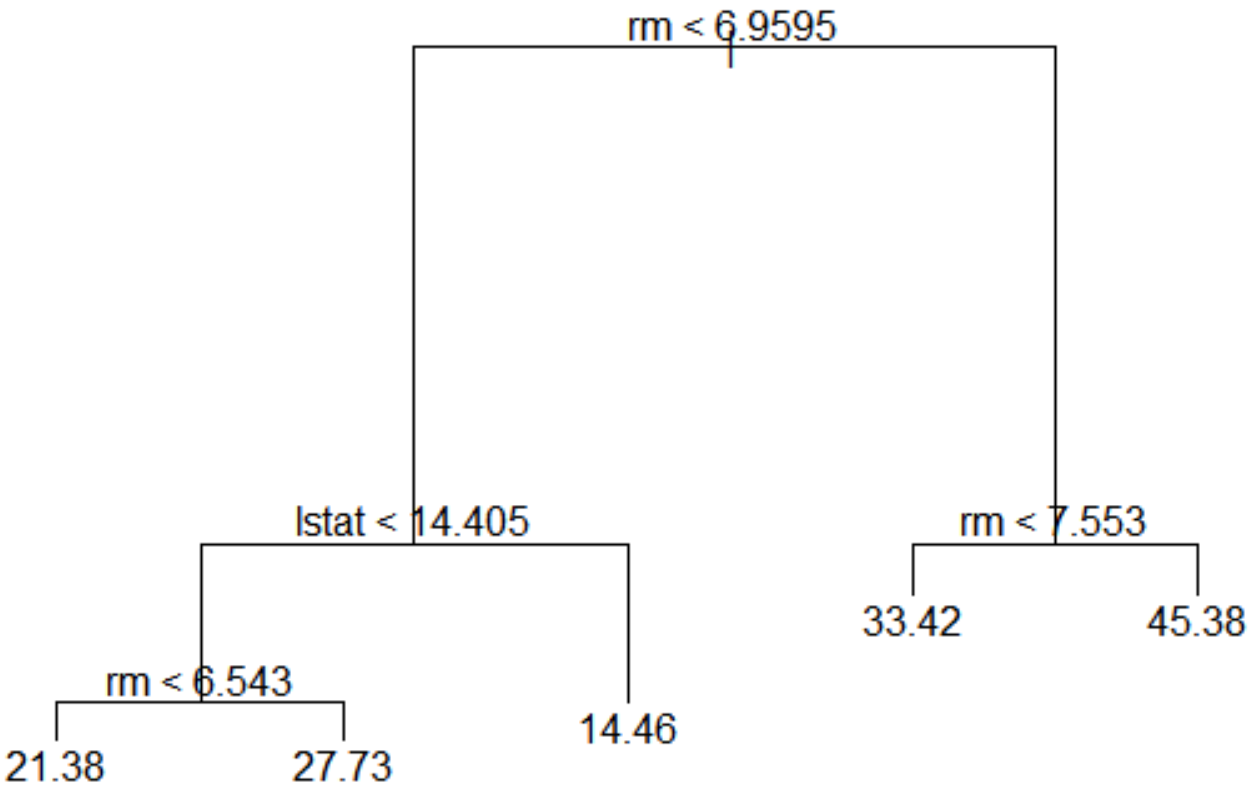
```
> cv.boston=cv.tree(tree.boston)
> plot(cv.boston$size,cv.boston$dev,type='b')
```



Boston数据集建立回归树

本例中，由交叉验证选出最复杂的树。但如果希望对树剪枝，可以用prune.tree()函数进行下列操作：

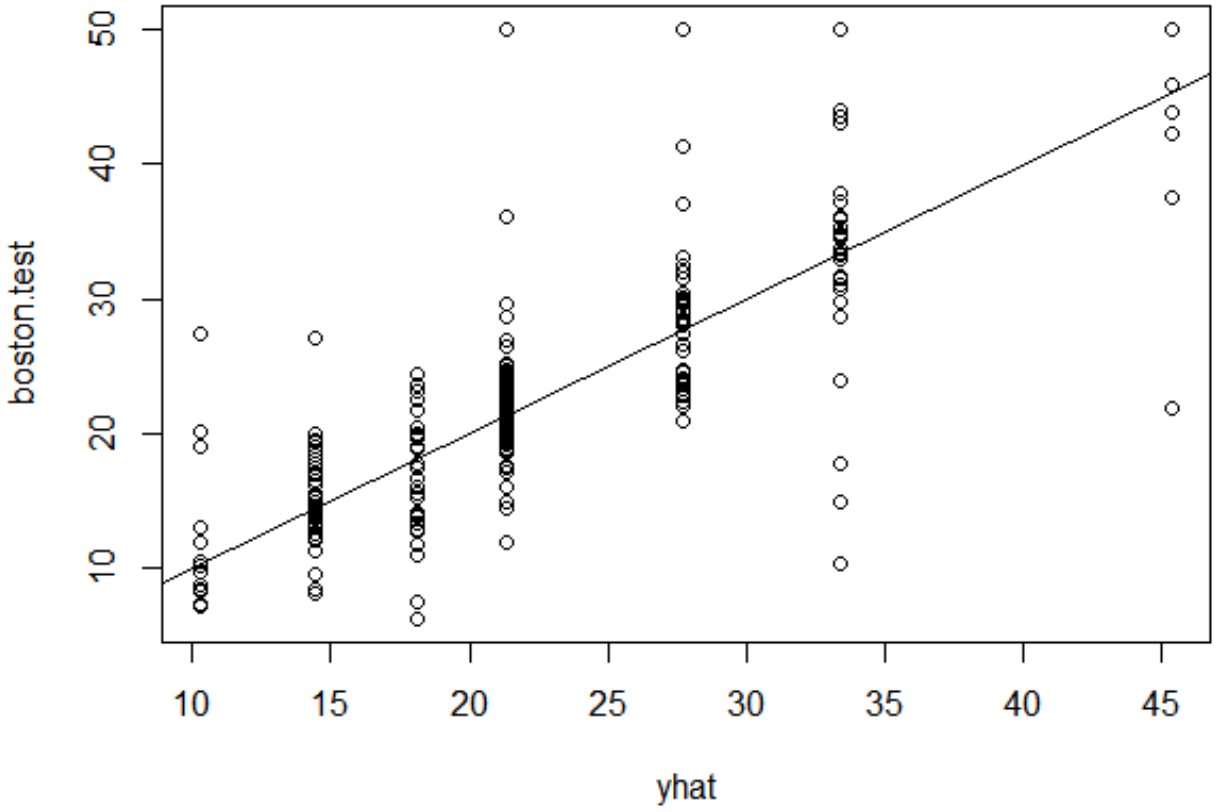
```
> prune.boston=prune.tree(tree.boston,best=5)
> plot(prune.boston)
> text(prune.boston,pretty=0)
```



Boston数据集建立回归树

为了与交叉验证的结果保持一致，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] 35.28688
```

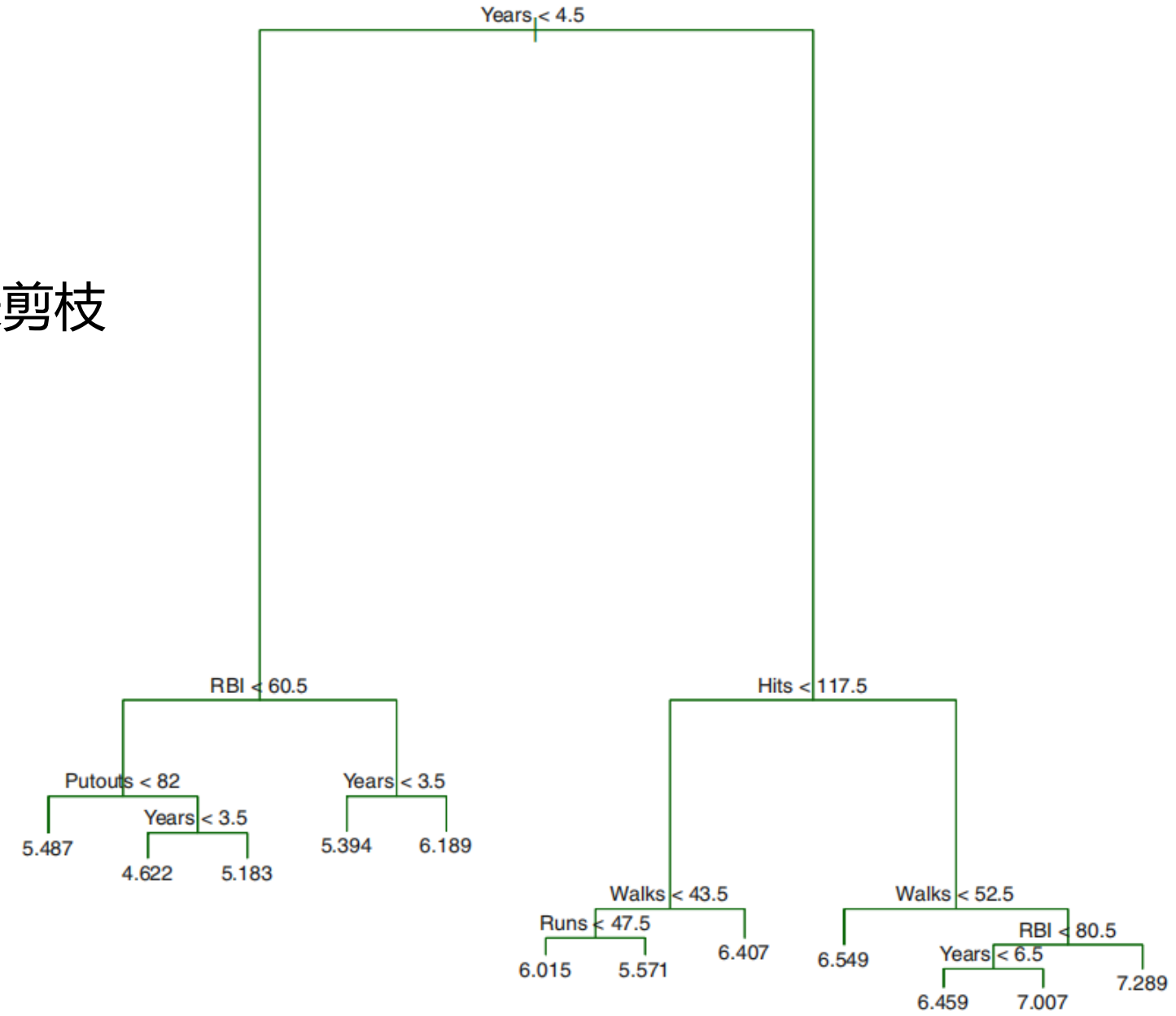


棒球例子续

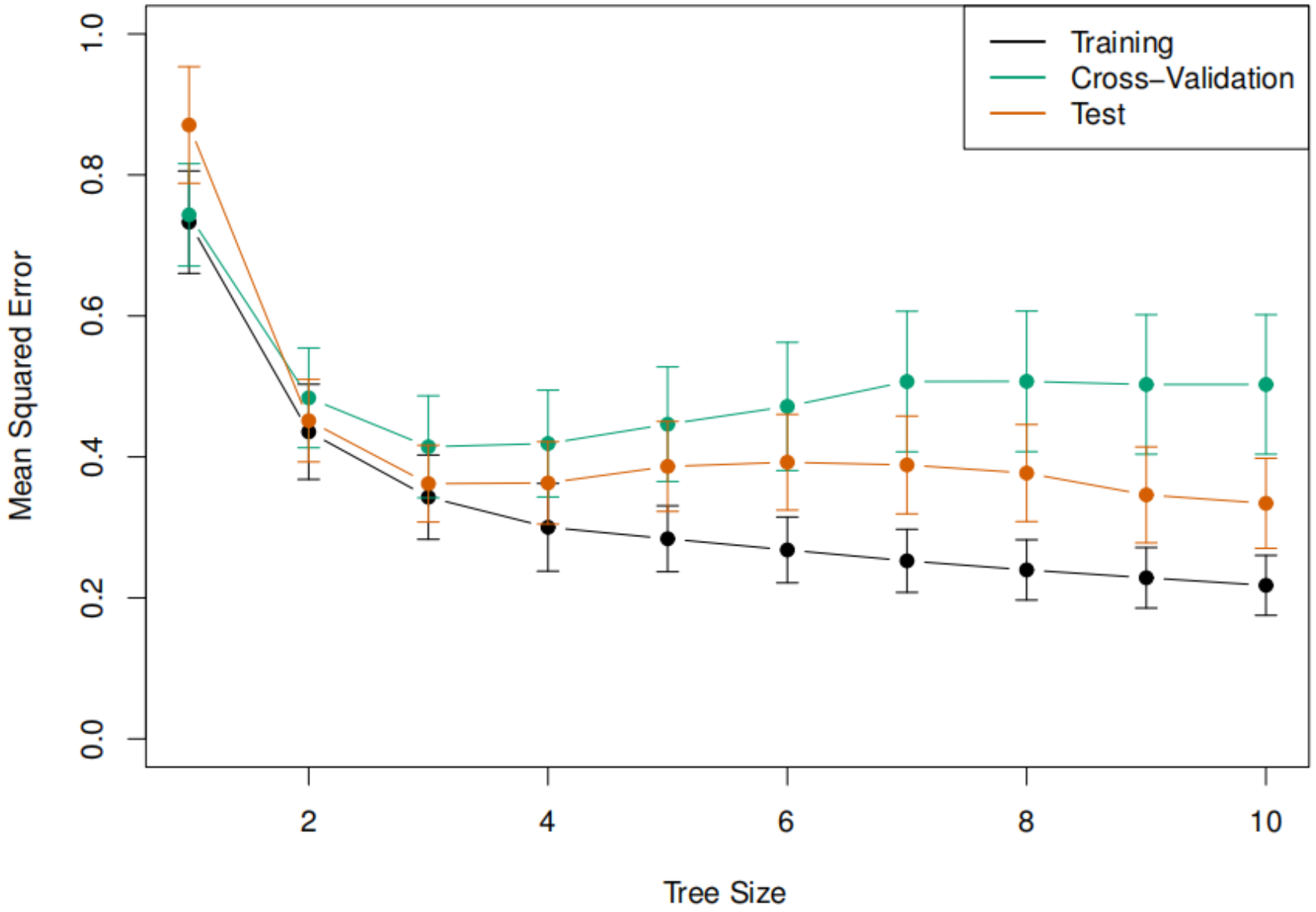
- 首先，将数据集随机分为两半，训练集中包含132个观测值，测试集中含131个观测值。
- 然后在训练数据和不同的α上建立一棵很大的回归树，再取不同的α值，得到终端结点数各不相同的子树。
- 最后用6折交叉验证法估计交叉验证的均方误差(MSE)，均方误差是α的函数。

棒球例子续

未剪枝

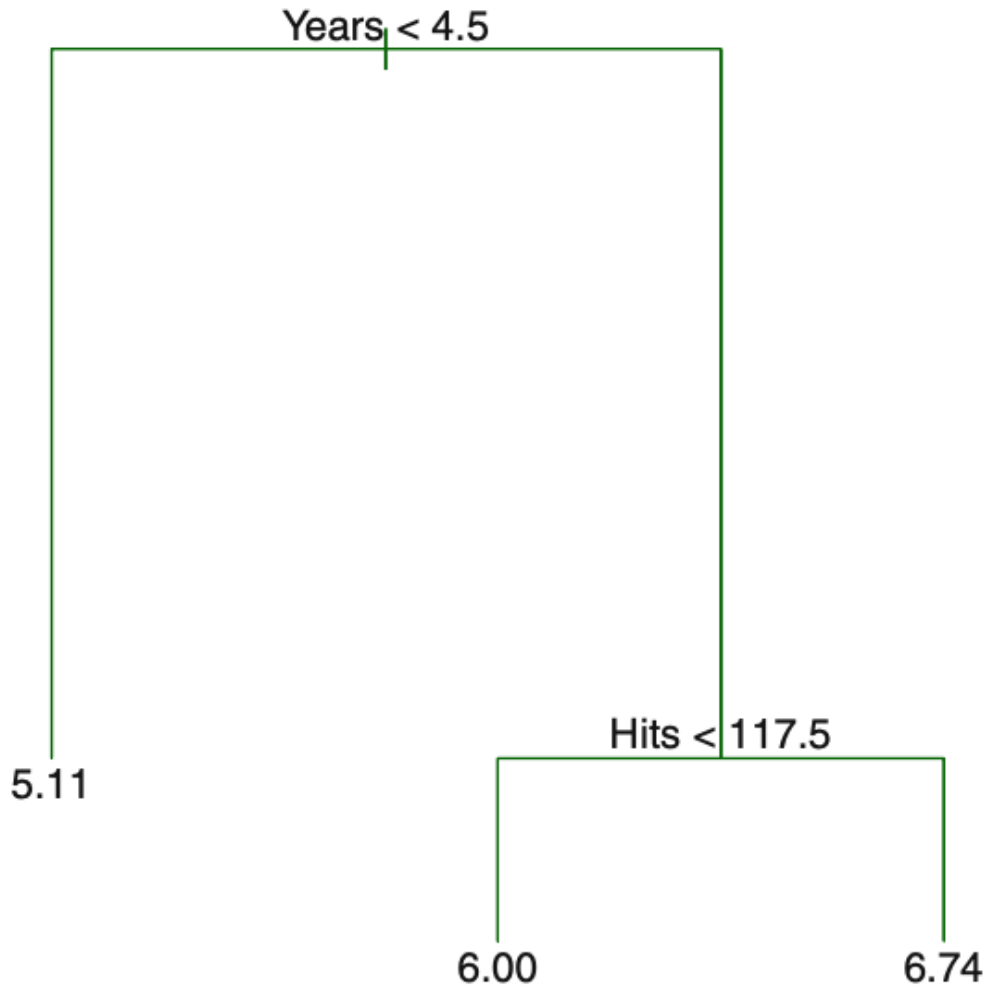

Years < 4.5
RBI < 60.5
Hits < 117.5
Putouts < 82
Years < 3.5
Years < 3.5
Walks < 43.5
Walks < 52.5
Runs < 47.5
RBI < 80.5
Years < 6.5
5.487
4.622
5.183
5.394
6.189
6.015
5.571
6.407
6.549
6.459
7.007
7.289

棒球例子续


Training
Cross–Validation
Test
Mean Squared Error
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Tree Size

棒球例子续

剪枝后



分类树

- **分类树(classification tree)**和回归树十分相似，它们的区别在于：分类树被用于预测定性变量而非定量变量。
- 对分类树来说，给定观测值被预测为它所属区域内的训练集中**最常出现的类(most commonly occurring class)**。

分类树的构造

- 与回归树一样，分类树采用递归二叉分裂。
- 但在分类树中，RSS无法作为确定二叉分裂点的准则。
- 一个很自然的替代指标是**分类错误率(classification error rate)**。分类错误率的定义为：此区域的训练集中非最常见类所占的比例。其表达式如下：

$$E = 1 - \max_k(\hat{p}_{mk})$$

上式中的$\hat{p}_{mk}$代表第m个区域的训练集中第k类所占比例。

- 但事实证明，分类错误率这一指标在构建树时不够敏感。在实践中，另外两个指标更有优势。

基尼系数和偏差

- **基尼系数(Gini index)**定义如下：

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

它衡量的是K个类别的总方差。不难发现，如果所有$\hat{p}_{mk}$的取值都接近0或1，基尼系数会很小。

- 因此基尼系数被视为衡量结点**纯度(purity)**的指标——如果它较小，就意味着某个结点包含的观测值几乎都来自同一类别。

基尼系数和偏差

- **基尼系数(Gini index)**定义如下：

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

它衡量的是K个类别的总方差。不难发现，如果所有$\hat{p}_{mk}$的取值都接近0或1，基尼系数会很小。

- 因此基尼系数被视为衡量结点**纯度(purity)**的指标——如果它较小，就意味着某个结点包含的观测值几乎都来自同一类别。
- 可以替代基尼系数的指标是**互熵(cross-entropy)**，定义如下：

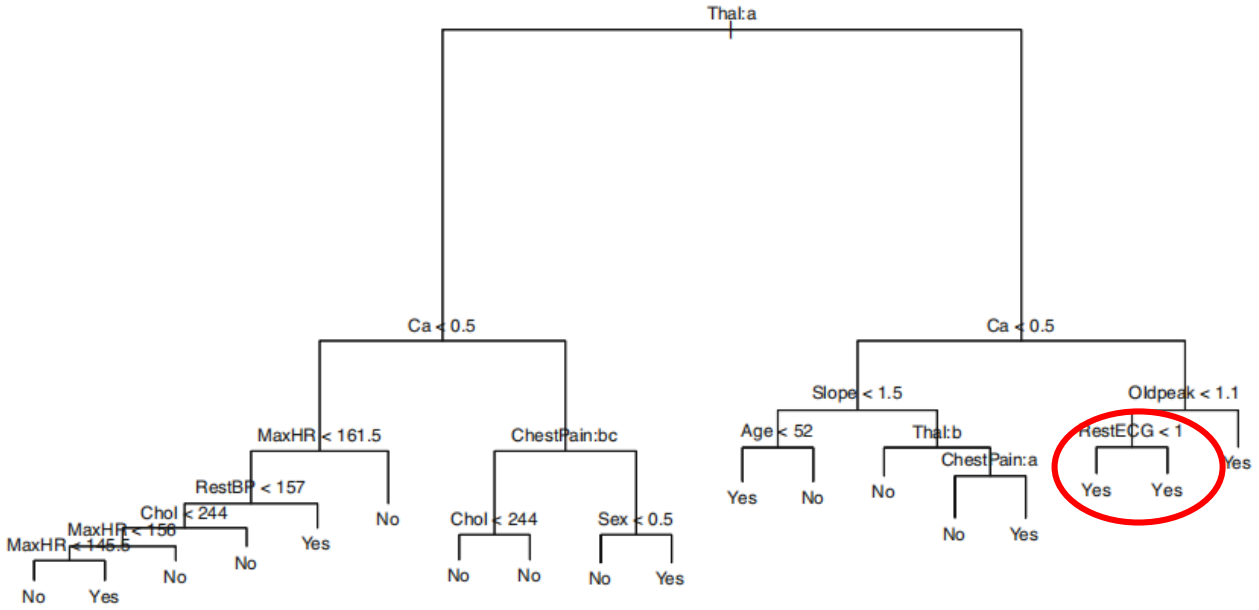
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

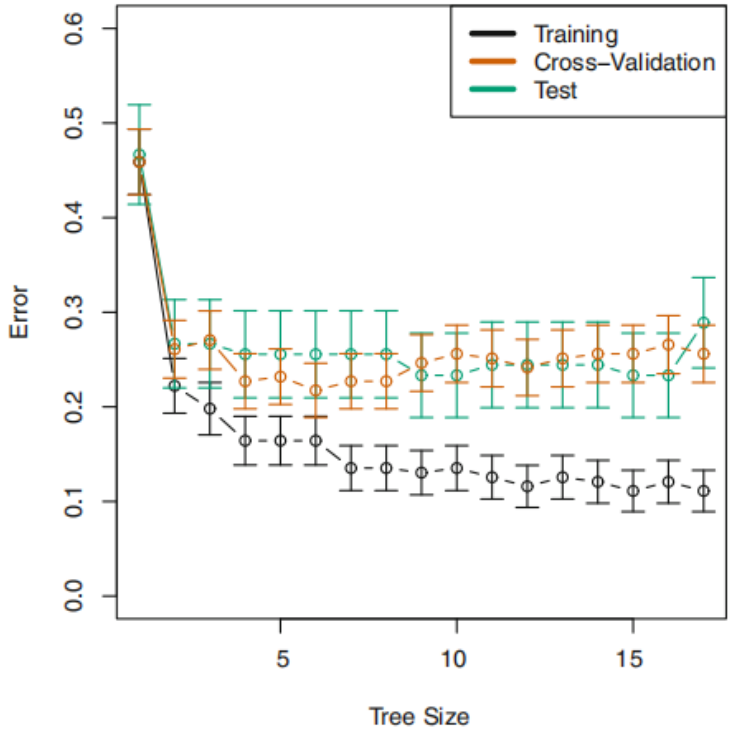
- 事实上，基尼系数和互熵在数值上是相当接近的。

基于心脏数据集的例子

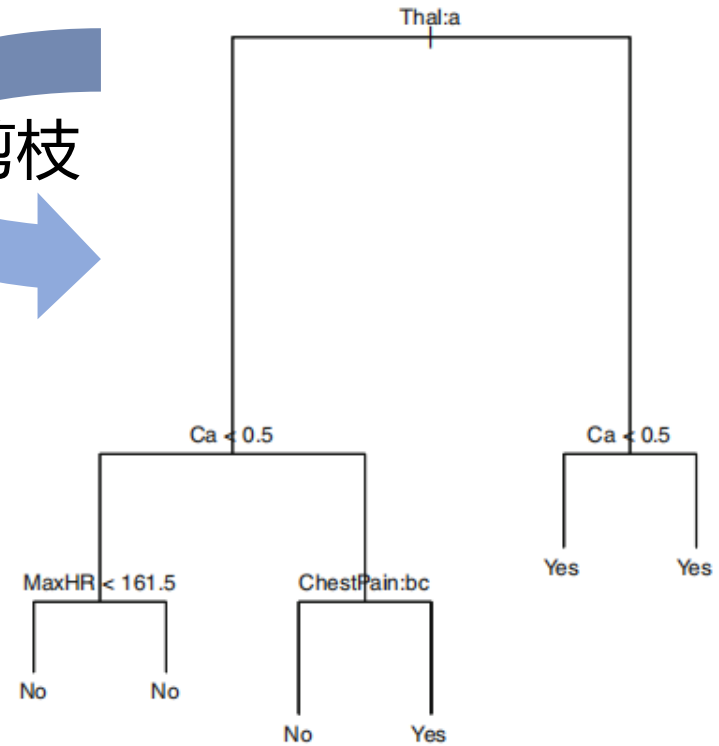
- 数据中包含一个关于303位有胸痛症状的病人的二元变量**HD**(是否患病)。
- **Yes**表示经血管造影诊断，病人患有心脏病，**No**表示未患心脏病。
- 预测变量有13个，包括 **Age**(年龄)、**Sex**(性别)、**Chol** (胆固醇指标)和其它一些心肺功能指标。
- 根据交叉验证，最终得到一棵有6个终端结点的树。请参见下图。

基于心脏数据集的例子

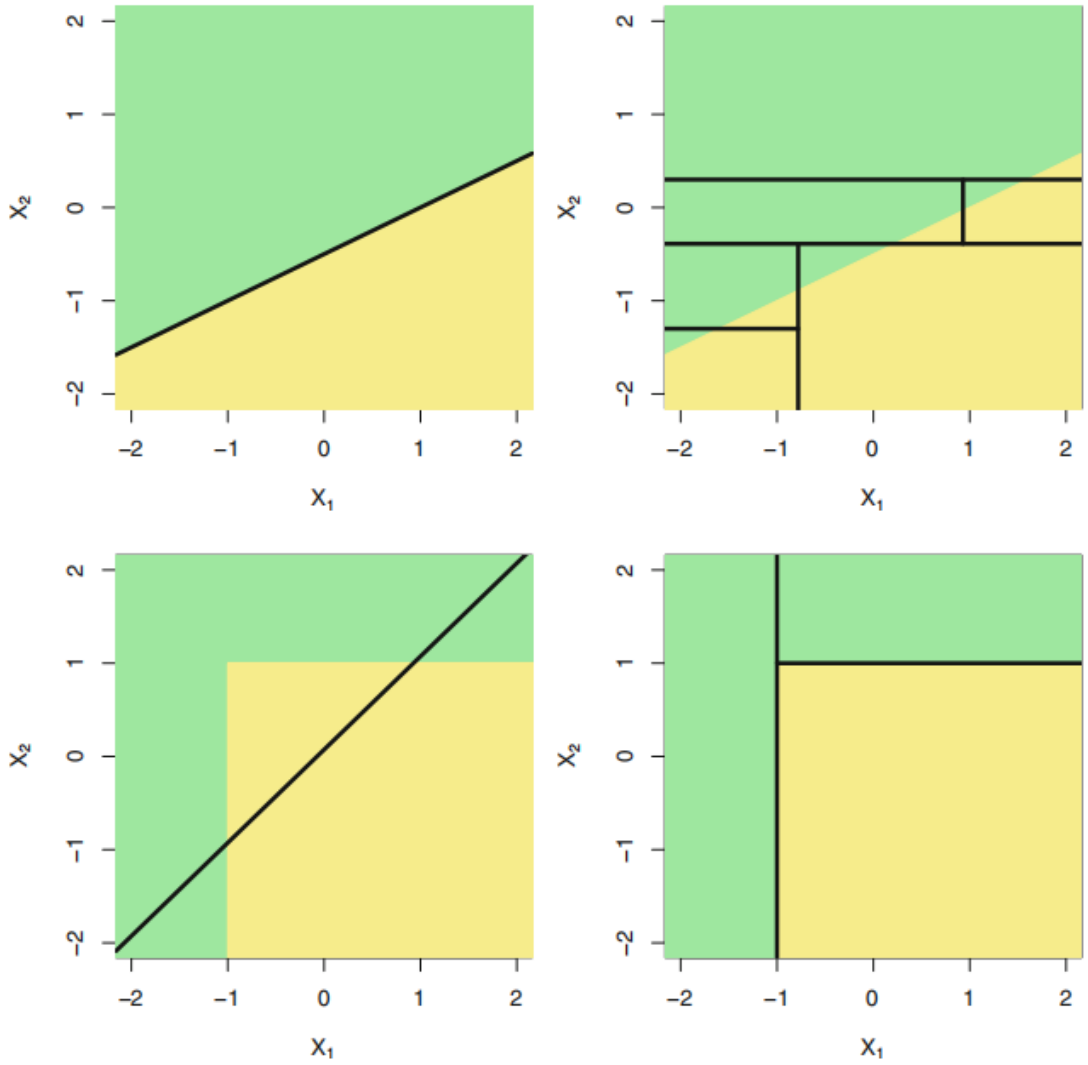




剪枝



树与线性模型



上方：真实决策边界是线性的，阴影区域标出；

下方：真实决策边界是非线性的。

左列：线性模型；右列：基于树的模型。

树的优缺点

哪些优缺点?

树的优缺点

- ▲ 决策树解释性强，在解释性方面甚至比线性回归更方便。
- ▲ 有人认为与传统的回归和分类方法相比，决策树更接近人的决策模式。
- ▲ 树可以用图形表示，非专业人士也可以轻松解释(尤其是当树规模较小时)。
- ▲ 树可以直接处理定性的预测变量而不需创建哑变量。
- ▼ 遗憾的是，树的预测准确性一般无法达到其他回归和分类方法的水平。

不过，通过用装袋法、随机森林和提升法等方法组合大量决策树，可以显著提升树的预测效果。下一节将介绍这些内容。

建立分类树

加载tree包用于构造分类树和回归树。

首先用分类树来分析Carseats（座椅）数据集。数据中的Sales（销量）是一个连续变量，因此将记为二元变量。用ifelse()函数创建一个High（高销量）变量，如果Sales变量大于8，变量High取Yes，否则取No。

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

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

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

使用tree()函数来建立分类树，用除Sales以外的所有变量预测High。

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

建立分类树

summary()函数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。

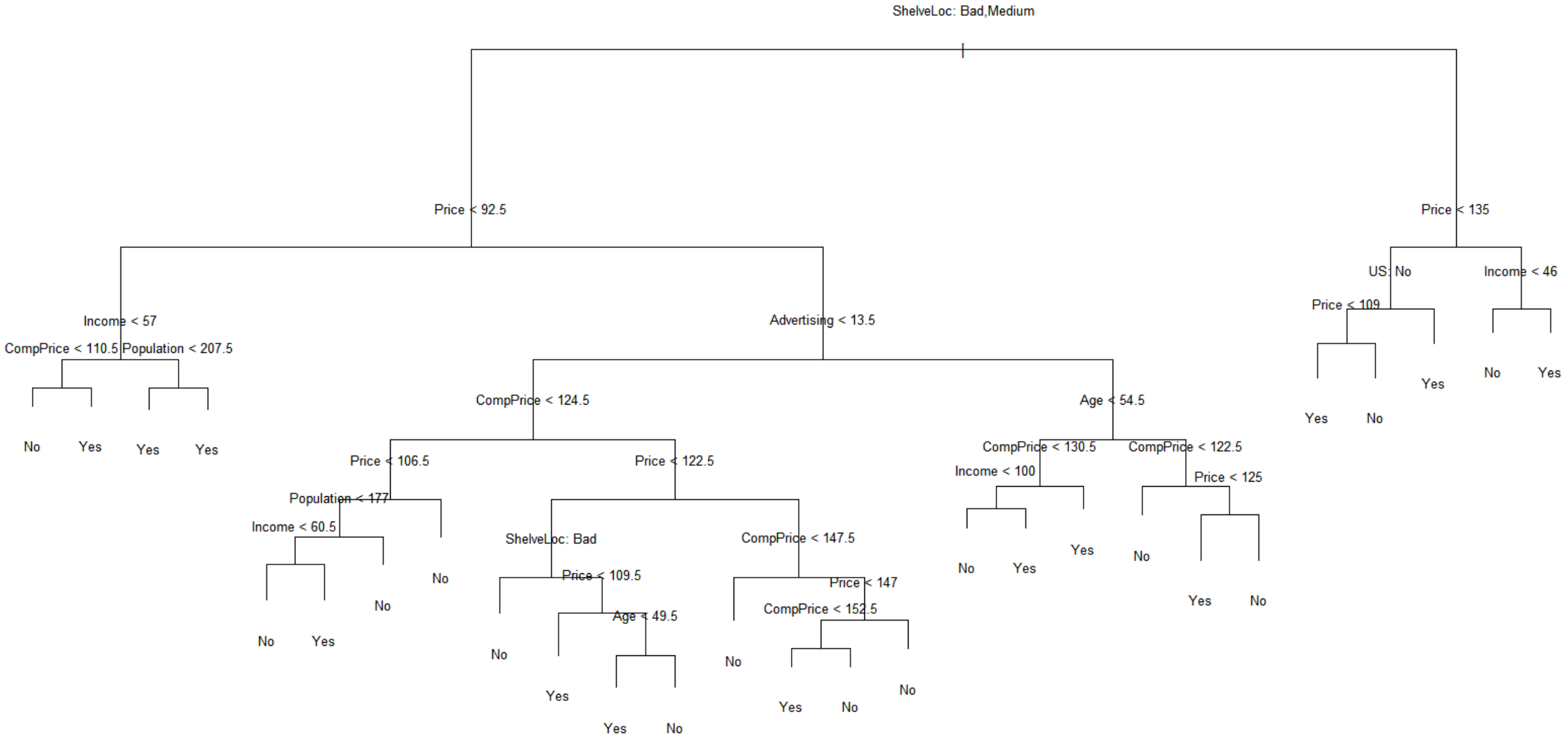
```
> summary(tree.carseats)

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

用plot()函数显示树的结构，用text()函数显示结点标记。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,cex=0.8)
```

生成的图如下：

建立分类树


ShelveLoc: Bad,Medium
Price < 92.5
Price < 135
Income < 57
Advertising < 13.5
US: No
Income < 46
CompPrice < 110.5
Population < 207.5
CompPrice < 124.5
Age < 54.5
Price < 109
No
Yes
Yes
Yes
Price < 106.5
Price < 122.5
CompPrice < 130.5
CompPrice < 122.5
Yes
No
No
Yes
Population < 177
Income < 60.5
ShelveLoc: Bad
CompPrice < 147.5
Income < 100
Price < 125
No
No
Price < 109.5
Price < 147
Yes
No
No
Yes
Age < 49.5
CompPrice < 152.5
No
Yes
No
No
Yes
No
Yes
Yes
No
Yes
No

建立分类树

如果只输入树对象的名称，R会输出树上每个分支的结果。R将会分裂规则(Price<92.5)、该分支中的观测值数量、偏差、该分支的总体预测（Yes或No），以及该分支中取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表示。

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

  1) root 400 541.500 No ( 0.59000 0.41000 )
    2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
      4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
        8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
```

将观察结果分成一个训练集和一个测试集，使用训练集构建分类树，在测试集上评估此树的预测效果。

```
> 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  104  33
      Yes  13  50
> (86+57)/200
[1] 0.715
```

建立分类树

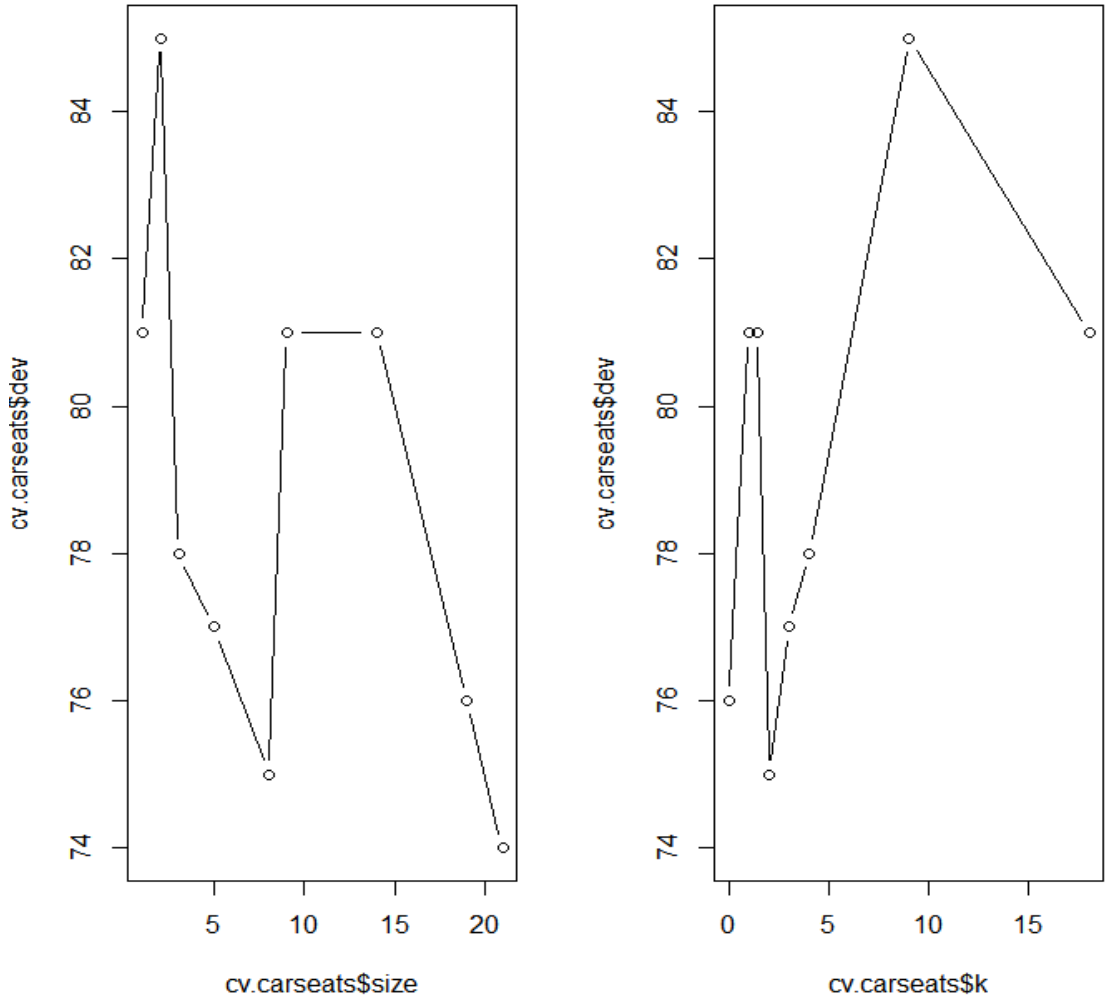
接下来，考虑剪枝树能否改进预测效果。用函数cv.tree()进行交叉验证确定最优的树复杂性，用成本复杂性剪枝选择要考虑的一系列树。

```
> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)#用分类错误率来剪枝
> names(cv.carseats)
[1] "size"   "dev"    "k"      "method"
> # "size"   "dev"    "k"      "method"，k为成本复杂性参数值
> cv.carseats
$size
[1] 21 19 14  9  8  5  3  2  1

$dev
[1] 74 76 81 81 75 77 78 85 81

$k
[1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0

$method
[1] "misclass"

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

建立分类树

画出错误率对size和k的函数。

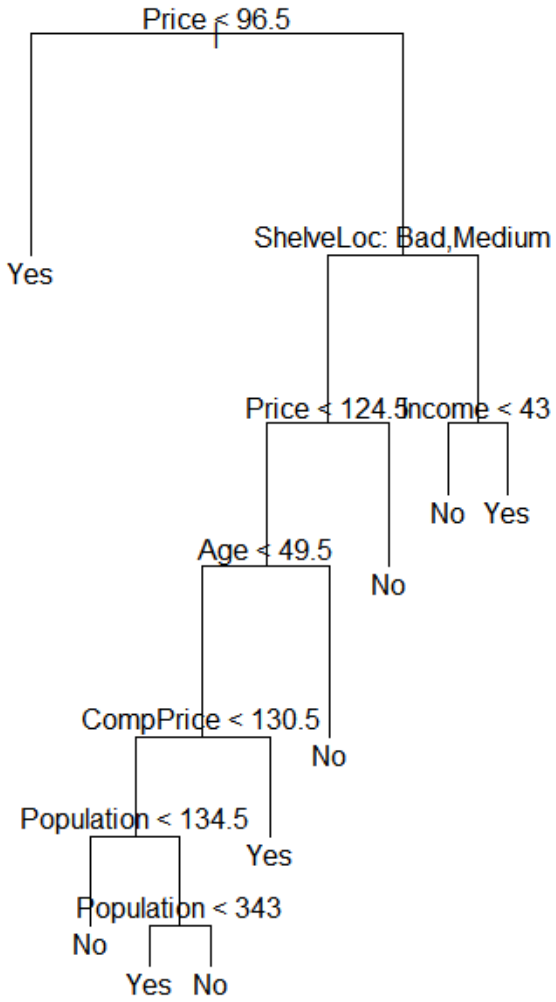
```
> par(mfrow=c(1,2))
> plot(cv.carseats$size,cv.carseats$dev,type="b")
> plot(cv.carseats$k,cv.carseats$dev,type="b")
```



建立分类树

用prune.misclass()函数剪枝以得到有9个结点的树。

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



建立分类树

剪枝后的树在测试集上的效果如何？再一次使用predict()函数。

```
> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
         High.test
tree.pred No Yes
      No  97  25
      Yes 20  58
> (94+60)/200
[1] 0.77
```

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  102  30
      Yes  15  53
> (86+62)/200
[1] 0.74
```

（二）装袋法、随机森林和提升法

装袋法

- 本节将介绍一种减小统计学习方法方差的通用做法，叫做**自助法聚集(bootstrap aggregation)**，或称**装袋法(bagging)**，它在决策树情境下效果很好，颇为常用。
- 回忆前文内容，给定n个独立观测值$Z_1, \ldots, Z_n$，每个观测值的方差都是σ^2，它们的平均值$\bar{Z}$的方差为σ^2/n。
- 换句话说，对一组观测值求平均可以减小方差。当然，这并不实际，因为我们通常不能访问多个训练集。

装袋法

- 自助法可以作为替代，即从某个单一的训练集中重复取样。
- 这样一来就能生成B个不同的自助抽样训练集。接下来，用第b个自助抽样训练集拟合模型并求得预测值$\hat{f}^{*b}(x)$，最后对所有预测值求平均得:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

这就是装袋法。

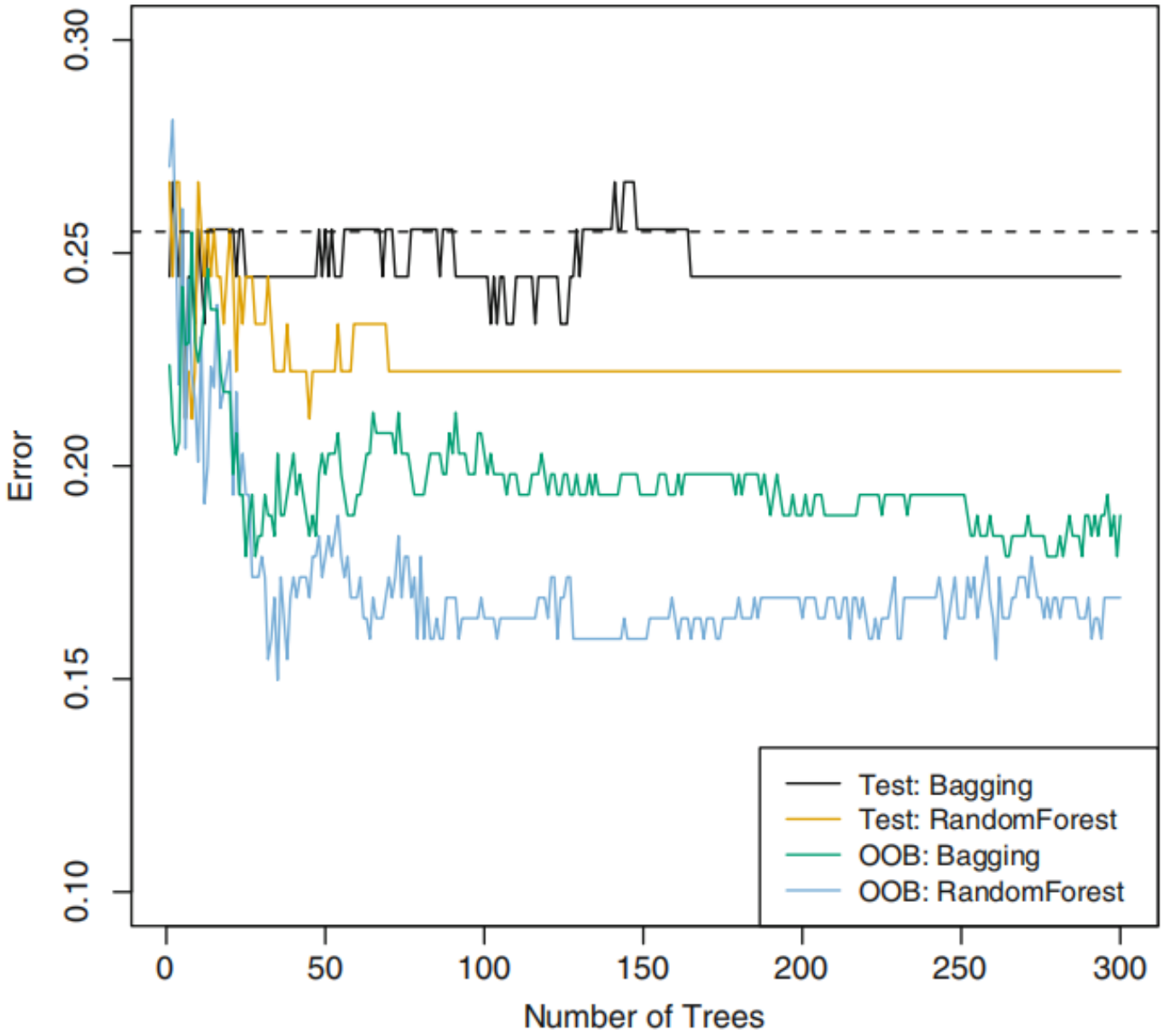
装袋法分类树

- 上述公式适用于回归树。
- 对于分类树：对一个给定的测试值，记录B棵树各自给出的预测类别，然后采取**多数投票(majority vote)**：将B个预测中出现频率最高的类作为总体预测。

基于心脏数据集的装袋法

基于心脏数据集的装袋法和随机森林结果

- 测试错误率(黑色和橙色线)被表示为B的函数，B是自助抽样训练集的个数。
- 随机森林的参数取$m = \sqrt{p}$。
- 虚线表示单个分类树的测试错误率。
- 绿色和蓝色曲线表示OOB错误率，在本例中它明显低于测试错误率。



袋外误差估计

- 事实上，有一种无需使用交叉验证就能直接估计装袋模型测试误差的方法。
- 回忆前文可知，装袋法的关键是用自助法得到观测值的若干子集，并对它们建立相应的树。可以证明，平均每棵树能利用约三分之二的观测值。
- 剩余三分之一没有使用的观测值被称为此树的**袋外(out-of-bag，OOB)**观测值。
- 可以用所有将第i个观测值作为OOB的树来预测第i个观测值的响应值。这样，便会生成约B / 3个对第i个观测值的预测。
- 当B足够大时，OOB误差实质上与留一法交叉验证误差是等价的。

随机森林

- **随机森林(Random forests)**通过对树做去除相关性处理，实现了对装袋法树的改进。
- 在随机森林中需对自助抽样训练集建立一系列决策树，这与装袋法类似。
- 不过，在建立这些决策树时，每考虑树上的一个分裂点，都要从全部的p个预测变量中选出一个包含m个预测变量的随机样本作为候选变量。这个分裂点所用的预测变量只能从这m个变量中选择。
- 在每个分裂点处都重新进行抽样，选出m个预测变量，通常$m \approx \sqrt{p}$——也就是说，每个分裂点所考虑的预测变量的个数约等于预测变量总数的平方根(例如Heart数据集共有 13个预测变量，则取$m = 4$)。

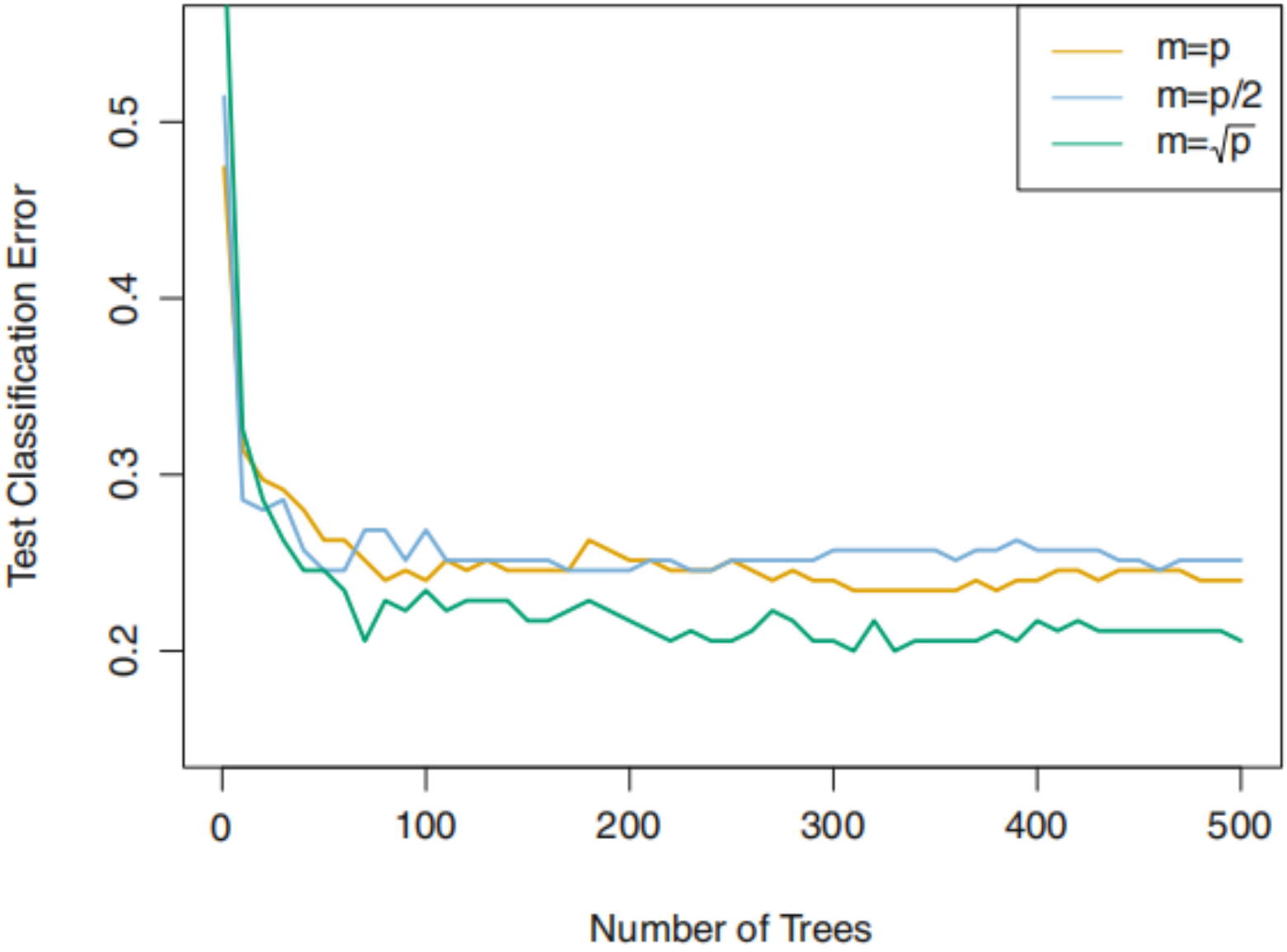
06

基因表达数据集随机森林例子

- 我们对一组包含4718个基因的表达量的高维生物学数据应用随机森林，这些基因来自349位病人的组织样本。
- 人类有大约20000个基因，每个基因在特定的细胞、组织和生理环境中有着不同程度的表达。
- 每个病人样本含有一个定性变量，此变量有15个不同的水平：正常或14种不同癌症中的一种。
- 用随机森林方法在训练集上根据500个方差最大的基因预测癌症种类。
- 将观测值随机分为训练集和测试集两部分，并用三个不同的m值在训练集上建立随机森林，m是可用于分裂的变量的个数。

06

基因表达数据集随机森林例子



基因表达数据集随机森林例子

上图详细说明：

- 15类基因表达数据集的随机森林结果，数据集含有$p = 500$个预测变量。
- 测试错误率被表示为树的个数的函数。每条彩线对应m的不同取值，m是每个内部分割点可用的预测变量数目。
- 随机森林$(m < p)$的预测效果略优于装袋法$(m = p)$。单个分类树的分类错误率是45.7%。

装袋法和随机森林

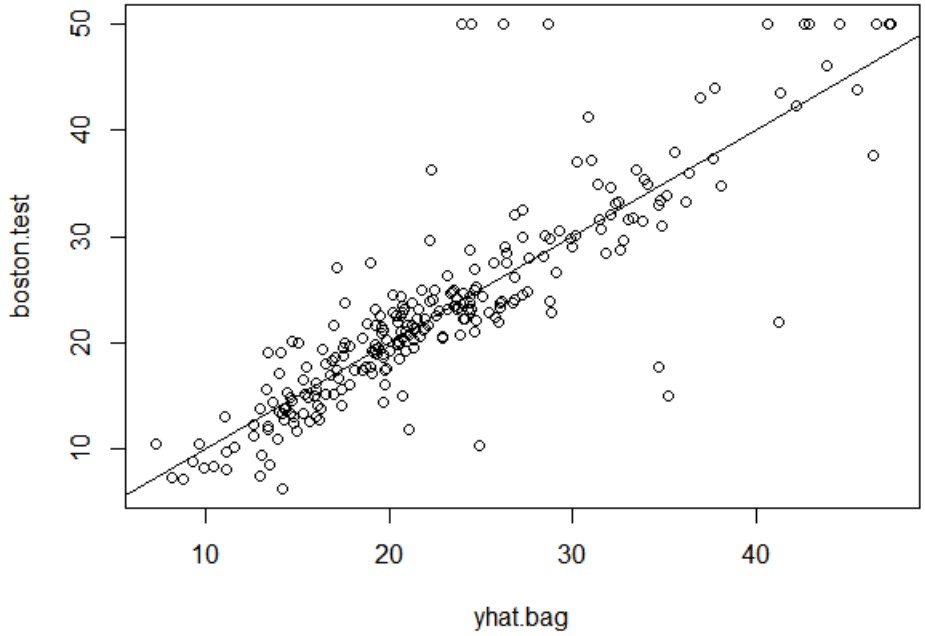
这里用R中的randomForest包在Boston数据集上实现装袋法和随机森林。回想一下，装袋法是随机森林在m = p时的特殊情况。因此randomForest()函数既可以用来做随机森林，也可以执行装袋法。执行装袋法如下：

```
> library(randomForest)
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,importance=TRUE)
> #参数mtry=13表示树上的每一个分裂点都考虑了所有的预测变量
> 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: 11.39601
                    % Var explained: 85.17
```

装袋法模型在测试集上的表现如何?

```
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> plot(yhat.bag, boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 23.59273
```

装袋法和随机森林



可以使用ntree参数来改变由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] 23.66716
```

装袋法和随机森林

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。

```
> 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] 19.62021
```

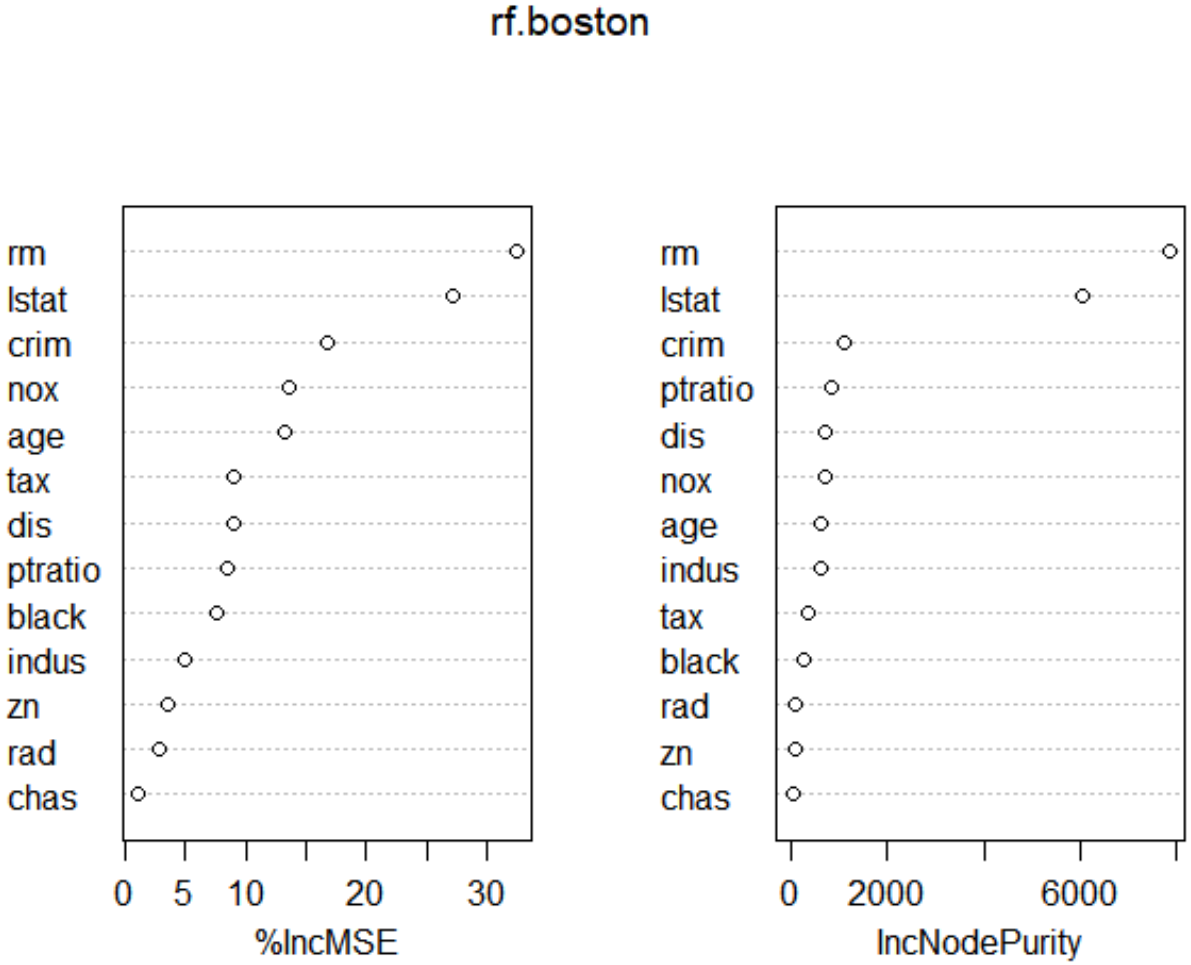
可以用importance()函数查看各变量的重要性

```
> importance(rf.boston)# %IncMSE IncNodePurity 前者衡量准确性，后者衡量纯度
          %IncMSE IncNodePurity
crim    16.697017    1076.08786
zn       3.625784      88.35342
indus    4.968621     609.53356
chas     1.061432      52.21793
nox     13.518179     709.87339
rm      32.343305    7857.65451
age     13.272498     612.21424
dis      9.032477     714.94674
rad      2.878434      95.80598
tax      9.118801     364.92479
ptratio  8.467062     823.93341
black    7.579482     275.62272
lstat   27.129817    6027.63740
```

装袋法和随机森林

反映变量重要性的图可以使用varImpPlot()函数生成。

```
> varImpPlot(rf.boston)#变量重要性图
```



07

提升法

- 与装袋法类似，**提升法(boosting)**也是一种通用方法，能改进许多用于解决回归或分类问题的统计学习方法。这里仅关注提升法在决策树中的应用。
- 回忆前文，装袋法利用自助抽样创建初始训练集的多个副本，再对每个副本建立决策树，最后将这些树结合起来建立一个预测模型。
- 值得注意的是，每一棵树都建立在一个自助抽样数据集之上，与其余树独立。
- 提升法也采用相似的方式，只是这里的树都是**顺序(sequentially)**生成的：每棵树的构建都需要用到之前生成的树中的信息。

对回归树应用提升法

1. 对训练集中的所有的i，令$\hat{f}(x)=0$，$r_i=y_i$。
2. 对$b=1,2,\ldots,B$重复以下过程：

   a. 对训练数据(X,r)建立一棵有d个分裂点($d+1$个终端结点)的树$\hat{f}^b$。

   b. 将压缩后的新树加入模型以更新$\hat{f}$：

$$\hat{f}(x)\leftarrow\hat{f}(x)+\lambda\hat{f}^b(x)$$

   c. 更新残差：

$$r_i\leftarrow r_i-\lambda\hat{f}^b(x_i)$$

3. 输出经过提升的模型：

$$\hat{f}(x)=\sum_{b=1}^{B}\lambda\hat{f}^b(x)$$

对回归树应用提升法

这一过程背后的理论依据是什么?

- 提升方法与生成一棵大规模的决策树不同，生成大树意味着对数据的**严格契合(fitting a data hard)**和可能的过拟合，而提升法则是一种**舒缓(learning slowly)**的训练模型的方法。
- 我们用现有模型的残差生成决策树。也就是说，将现有残差而不是结果Y作为响应值。然后再把这棵新生成的树加到相应的函数中以更新残差。
- 算法中的参数d控制着树的规模，通过对d的调整，所有这些树都可以变得更小，只有少数终端结点。
- 通过对残差生成小型的树，在$\hat{f}$效果不佳的地方缓慢地对它进行改进。压缩参数λ允许更多不同结构的树改变残差，它的存在让学习过程进一步变慢。

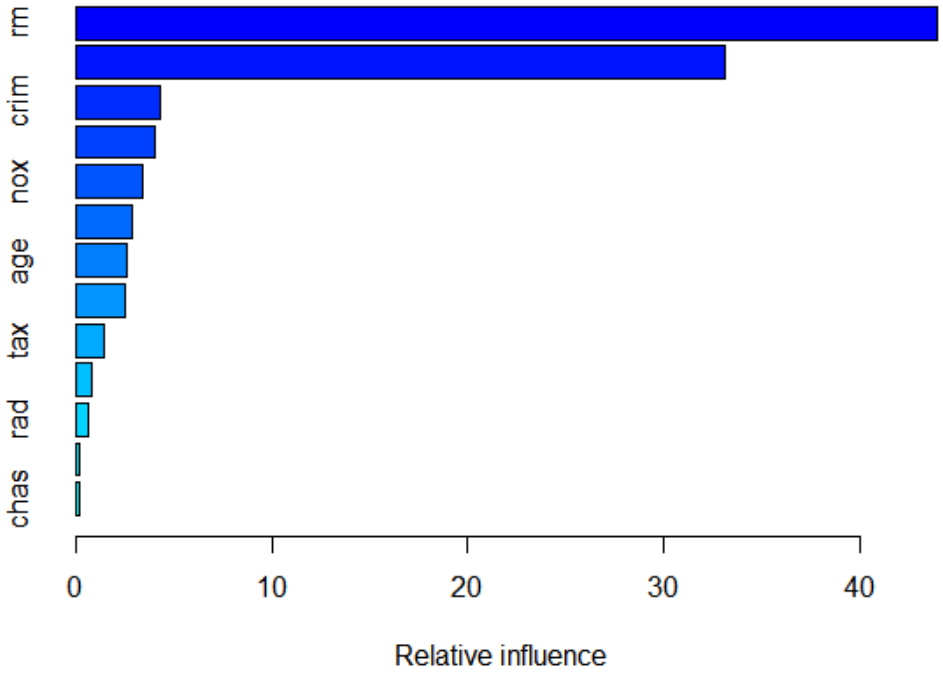
对回归树应用提升法

用gbm包和其中的gbm（）函数对Boston数据集建立回归树。

```
> library(gbm)
Loaded gbm 2.1.8.1
> set.seed(1)
> boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",n.trees=500
0,interaction.depth=4)
```

用summary()函数生成一张相对影响图，并输出相对影响统计数据。

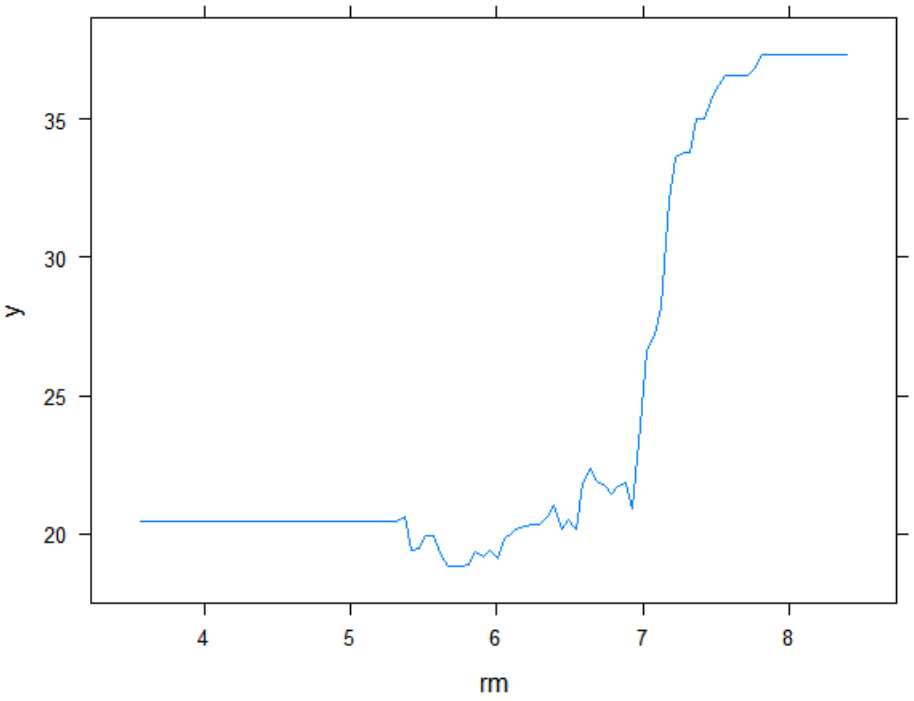
```
> summary(boost.boston)#生成了变量相对影响图
             var    rel.inf
rm           rm 43.9919329
lstat     lstat 33.1216941
crim       crim  4.2604167
dis         dis  4.0111090
nox         nox  3.4353017
black     black  2.8267554
age         age  2.6113938
ptratio ptratio  2.5403035
tax         tax  1.4565654
indus     indus  0.8008740
rad         rad  0.6546400
zn           zn  0.1446149
chas       chas  0.1443986
```

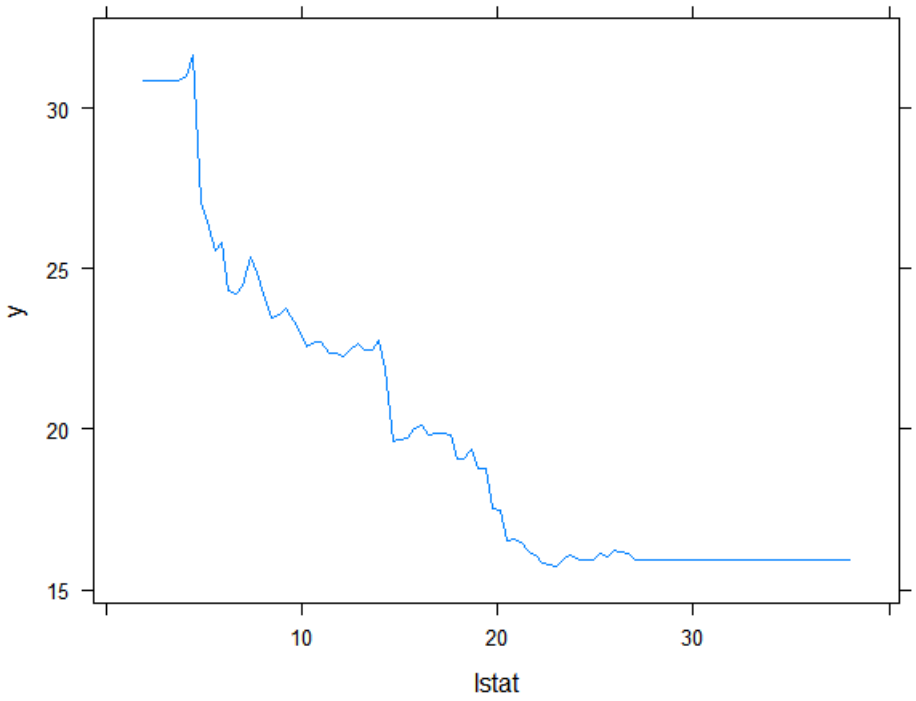


对回归树应用提升法

我们看到，lstat和rm是目前最重要的变量。我们也可以为这两个变量生成偏相关图（partial dependence plots）。这些图说明了排除其他变量后，选定变量对响应值的边际影响。

```
> par(mfrow=c(1,1))
> plot(boost.boston,i="rm")#画出两个变量的偏相关图
> plot(boost.boston,i="lstat")
```





对回归树应用提升法

使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] 18.84709
```

如果需要可以使用不同的压缩参数λ做提升法。λ默认值为0.001，这里我们用λ=0.2。

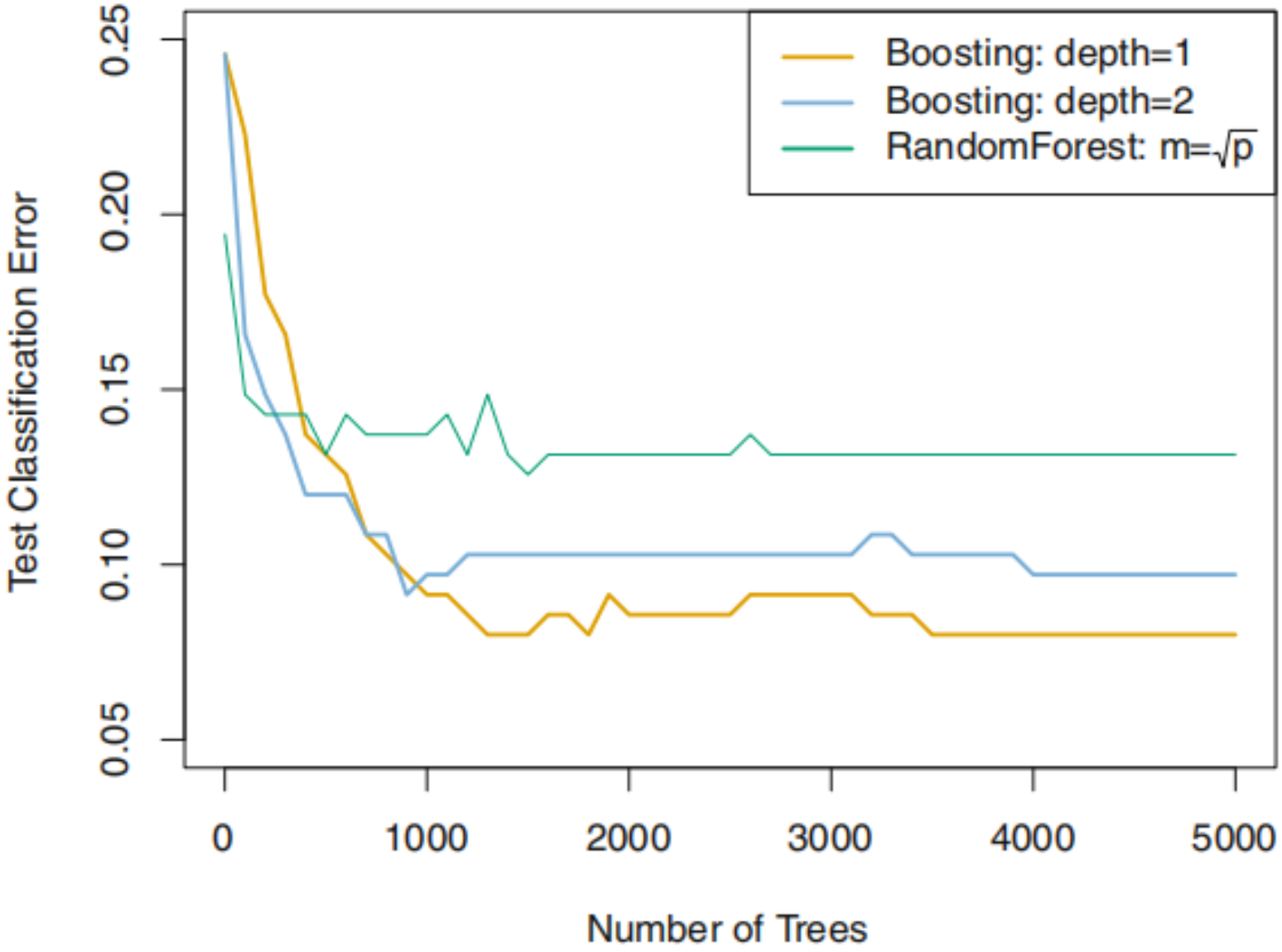
```
> boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",n.trees=500
0,interaction.depth=4,shrinkage=0.2,verbose=F)
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 18.33455
```

09

对分类树应用提升法

- 提升分类树的生成与生成提升回归树的过程相似，但更复杂一些，这里不作详述。
- R语言包**gbm**(梯度提升模型)处理各种回归和分类问题。

基因表达数据集续


Boosting: depth=1
Boosting: depth=2
RandomForest: m=√p
Test Classification Error
0.05
0.10
0.15
0.20
0.25
0
1000
2000
3000
4000
5000
Number of Trees

基因表达数据集续

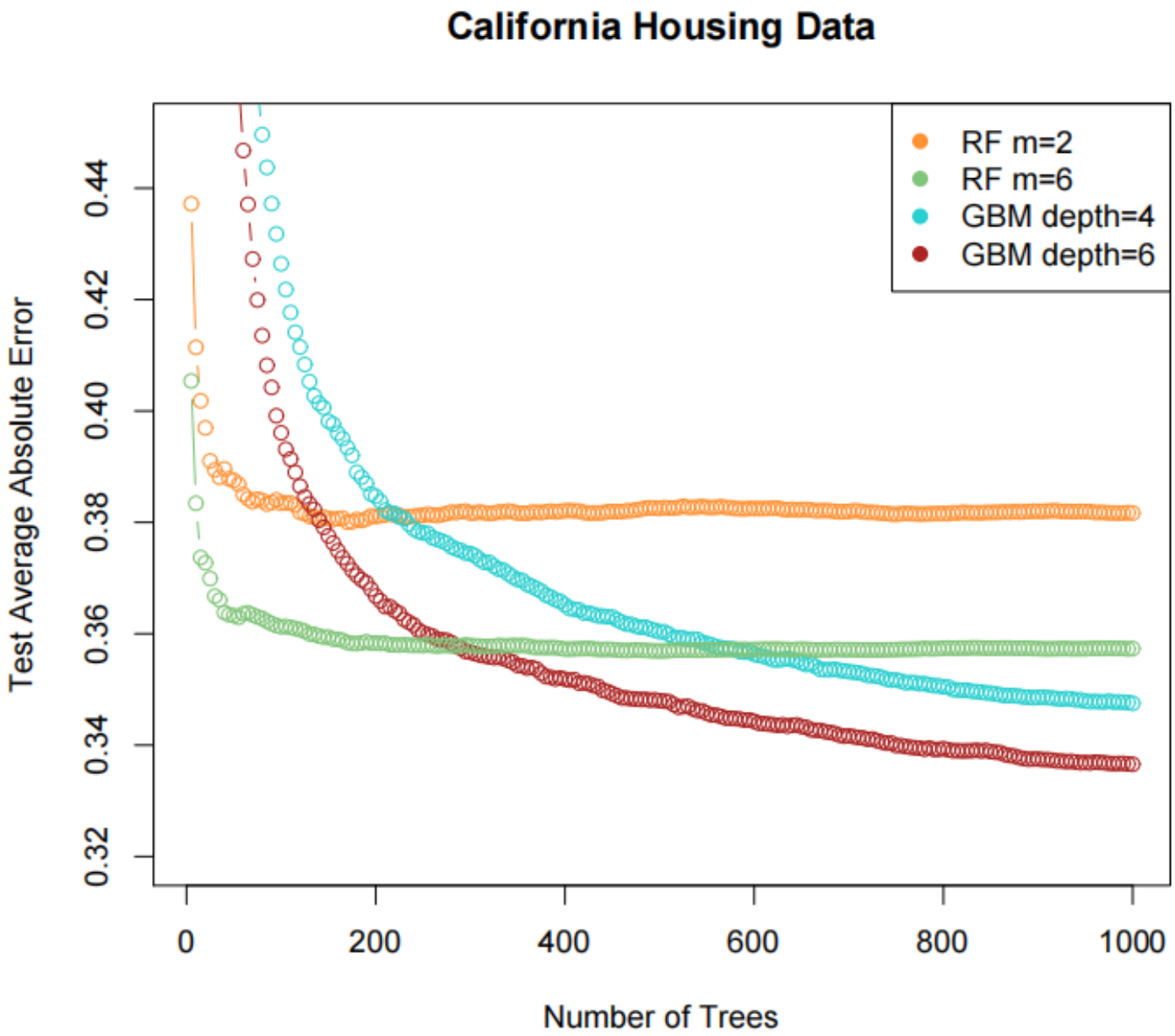
上图详细说明：

- 提升法和随机森林在15类基因表达数据集上的预测结果。
- 测试误差表示为树的总量的函数。比较两个提升模型(均取$\lambda = 0.01$)的预测效果，深度为1模型的略优于深度为2的模型，二者均优于随机森林，虽然低至0.02左右的标准误差使这些区别并不显著。
- 单棵树的测试错误率是24%。

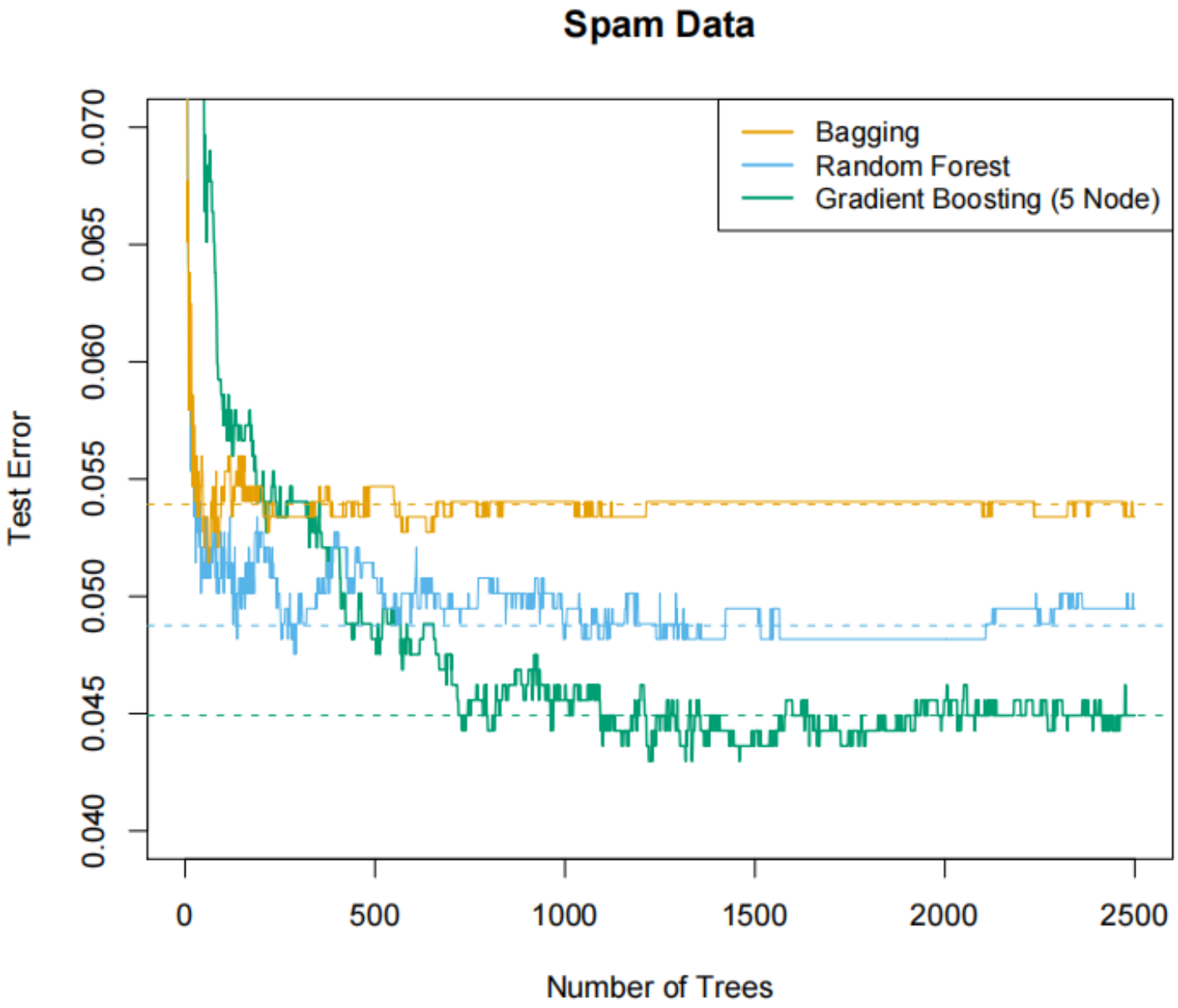
提升方法的三个调整参数

1. 树的总数B。与装袋法和随机森林不同，如果B值过大，提升法可能出现过拟合，不过即使出现过拟合，其发展也很缓慢。我们用交叉验证来选择B。
2. 取极小正值的压缩参数λ。它控制着提升法的学习速度。λ通常取 0.01或0.001 ，合适的取值视具体问题而定。若λ值很小，则需要很大的B才能获得良好的预测效果。
3. 每棵树的分裂点数d，它控制着整个提升模型的复杂性。用 $d = 1$ 构建模型通常能得到上佳效果，此时每棵树都是一个**树桩(stump)**，仅由一个分裂点构成。这种情况下的提升法整体与加法模型相符，因为每个树只包含一个变量。更多情况下，d表示**交互深度(interaction depth)**，它控制着提升模型的交互顺序，因为d个分裂点最多包含d个变量。

另一个回归例子

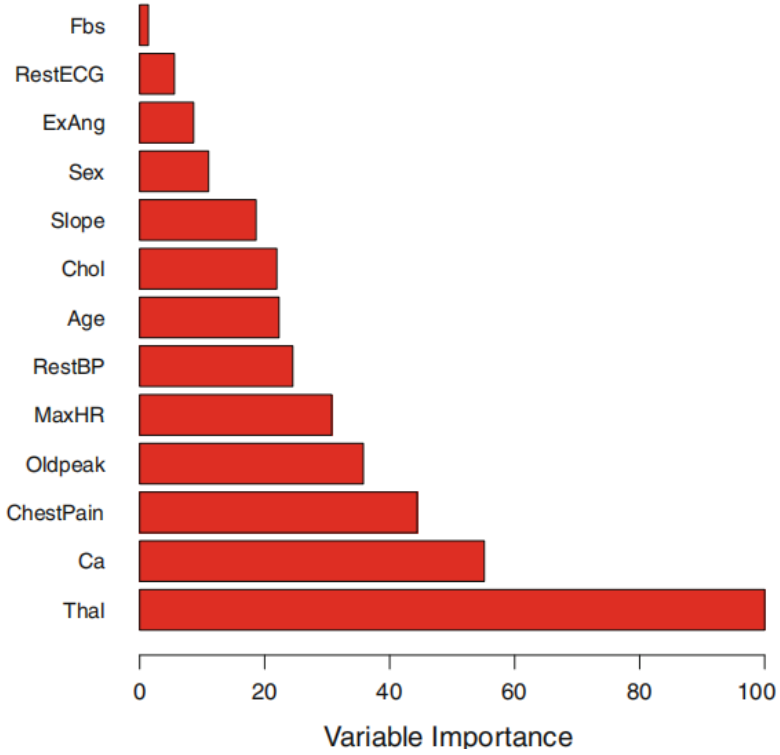

California Housing Data
RF m=2
RF m=6
GBM depth=4
GBM depth=6
Test Average Absolute Error
0.32
0.34
0.36
0.38
0.40
0.42
0.44
0
200
400
600
800
1000
Number of Trees

另一个回归例子


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees

变量重要性度量

- 在装袋回归树建模过程中，可以记录下任一给定预测变量引发的分裂而减小的RSS的总量，对每个减小总量在所有B棵树上取平均。结果值越大则说明预测变量越重要。
- 同样的道理，在装袋法分类树建模过程中，可以对某一给定的预测变量在一棵树上因分裂而使基尼系数的减小量加总，再取所有B棵树的平均。



Heart数据集变量重要性图

总结

- 决策树是用于回归和分类的简便且易于解释的模型。
- 然而，在预测精度方面，树一般无法达到其他回归和分类方法的水平。
- 装袋法、随机森林和提升法是提高树预测精度的有效方法。它们的工作原理是在训练数据上生成大量的树，然后将生成的树进行组合，从而得出预测结果。
- 后两种方法——随机森林和提升法——都是最先进的监督学习方法。然而，他们的结果可能很难解释。