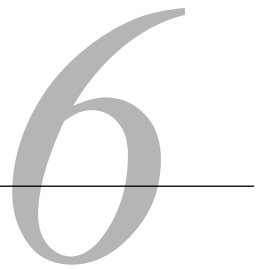
第 6 章 分类回归树



“最好的分类器可能是随机森林（RF），最好的版本（用R开发，通过caret调用）可以在84.3%的数据集上达到94.1%的最大正确率，超过了90%的其他分类器。”

——费尔南德斯–德尔加多 等（2014）

上文引自费尔南德斯–德尔加多等人在《机器学习研究期刊》上发表的一篇论文，从这段话可以看出，我们在这一章要讨论的技术非常强大，特别是用于分类问题时。当然，它们不是永远的最优解，但确实提供了一个非常好的起点。

我们之前研究的技术或者用于预测定量结果，或者用于预测分类问题。本章要讨论的技术对这两类问题都适用，而处理业务问题的方式和前几章不同。我们不引入新的问题，而是把这些技术用于前面已经解决的问题，着眼于这些技术能否提高预测能力。总而言之，本章案例的目的就是要看看是否能继续改善以前的模型。

第一个要讨论的项目是基础的决策树，模型建立和解释都非常简单。但单决策树模型的表现并不如我们研究过的模型那样好，比如支持向量机模型；也比不上我们将要学习的模型，比如神经网络模型。所以，我们要讨论的是建立多个（有时是上百个）不同的树，然后把每个树的结果组合在一起，最后形成一个综合的预测。

正像本章开头引用的论文所说，这些方法产生的效果不次于甚至超过书中涉及的所有其他技术。这些技术被称为**随机森林**和**梯度提升树**。此外，在讨论商业案例之余，我们还会介绍如何在数据集上应用随机森林方法，以帮助实现特征消除（特征选择）。

6.1 本章技术概述

我们要简单介绍本章涵盖的技术，包括回归树与分类树、随机森林和梯度提升，为使用这些技术分析实际商业案例打下基础。

决策树包括：
回归树：预测定量结果，基于残差平方和最小原则
分类树：预测定性结果，基于基尼系数最小准则

偏差与方差：https://liam0205.me/2017/03/25/bias-variance-tradeoff/
贪心算法：https://zh.wikipedia.org/wiki/%E8%B4%AA%E5%BF%83%E6%B3%95

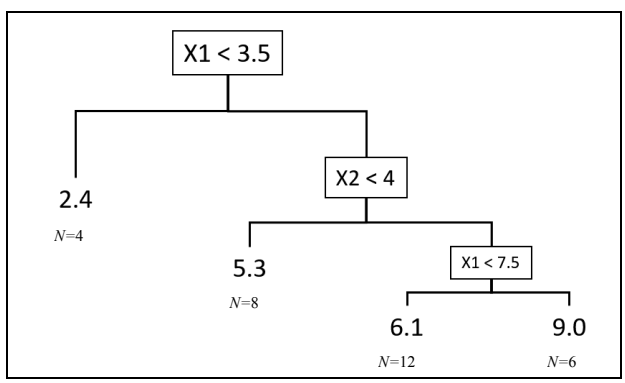
6.1.1　回归树

RSS:residual sum of squares,残差平方和

要想理解基于树的方法，比较容易的方式是从预测定量结果开始，然后学习它是如何解决分类问题的。树方法的精髓就是划分特征，从第一次分裂开始就要考虑如何最大程度改善RSS，然后持续进行二叉分裂，直到树结束。后面的划分并不作用于全体数据集，而仅作用于上次划分时落到这个分支之下的那部分数据。这个自顶向下的过程被称为“递归划分”。这个过程是**贪婪**的，这个名词在研究机器学习方法时会经常遇到。贪婪的含义是，算法在每次分裂中都追求最大程度减少RSS，而不管以后的划分中表现如何。这样做的结果是，你可能会生成一个带有无效分支的树，尽管偏差很小，但是方差很大。为了避免这个问题，生成完整的树之后，你要对树进行剪枝，得到最优的规模。

递归划分：recursive partitioning

下图给出了使用树进行分类的一个直观的实际例子。假设数据集中有30个观测，响应变量的范围是1 ~ 10，有两个预测特征X_1和X_2，范围都是0 ~ 10。整个树有3个分支和4个终端节点。每次都按照基本的`if-then`语句，或者使用R函数`ifelse()`进行分裂。第一次分裂的条件是X_1是否小于3.5，响应变量被分出4个观测，平均值为2.4，剩余26个观测。包含4个观测的左侧分支是一个终端节点，因为更深的分裂已经不能明显改善RSS了。树的这部分划分包含的4个观测的预测值就是平均值2.4。下一次分裂的条件是$X_2 < 4$，最后是$X_1 < 7.5$。



有3个分支和4个节点的回归树，节点下面标有均值和观测数量

这种方法的优点是可以处理高度非线性关系，但它还存在一些潜在的问题，你能发现吗？首要的问题就是，一个观测被赋予所属终端节点的平均值，这会损害整体预测效果（高偏差）。相反，如果你一直对数据进行划分，树的层次越来越深，这样可以达到低偏差的效果，但是高方差又成了问题。和其他方法一样，你也可以用交叉验证来选择合适的深度。

6.1.2　分类树

分类树与回归树的运行原理是一样的，区别在于决定分裂过程的不是RSS，而是误差率。这

里的误差率不是你想的那样，简单地由误分类的观测数除以总观测数算出。实际上，进行树分裂时，误分类率本身可能会导致这样一种情况：你可以从下次分裂中获得一些有用信息，但误分类率却没有改善。我们看一个例子。

假设有一个节点N_0，节点中有7个标号为No的观测和3个标号为Yes的观测，我们就可以说误分类率为30%。记住这个数，然后通过另一种误差测量方式进行计算，这种方式称为**基尼指数**。单个节点的基尼指数计算公式如下：

$$\text{基尼指数} = 1 - (\text{类别1的概率})^2 - (\text{类别2的概率})^2$$

所以，对于N_0，基尼指数为$1-(0.7)^2-(0.3)^2$，等于0.42，与之相对的误分类率为30%。

将这个例子讨论得更深入一些，假设将节点N_0分裂成两个节点N_1和N_2，N_1中有3个观测属于类别1，没有属于类别2的观测；N_2中有4个观测属于类别1，3个属于类别2。现在，树的这个分支的整体的误分类率还是30%，那么看看整体的基尼指数是如何改善的：

- 基尼指数$(N_1) = 1-(3/3)^2-(0/3)^2 = 0$
- 基尼指数$(N_2) = 1-(4/7)^2-(3/7)^2 = 0.49$
- 新基尼指数 $= (N_1\text{比例} \times \text{基尼指数}(N_1)) + (N_2\text{比例} \times \text{基尼指数}(N_2)) = (0.3 \times 0) + (0.7 \times 0.49) = 0.343$

通过一次基于替代误差率的分裂，我们确实改善了模型的不纯度，将其从原来的0.42减小到0.343，误分类率却没有变化。rpart()包就是使用Gini指数测量误差的，在本章中我们会使用这个程序包。

6.1.3 随机森林

为了显著提高模型的预测能力，我们可以生成多个树，然后将这些树的结果组合起来。随机森林技术在模型构建过程中使用两种奇妙的方法，以实现这个构想。第一个方法称为**自助聚集，或称装袋**。bagging

在装袋法中，使用数据集的一次随机抽样建立一个独立树，抽样的数量大概为全部观测的2/3（请记住，剩下的1/3被称为**袋外数据**，out-of-bag）。这个过程重复几十次或上百次，最后取平均结果。其中每个树都任其生长，不进行任何基于误差测量的剪枝，这意味着每个独立树的方差都很大。但是，通过对结果的平均化处理可以降低方差，同时又不增加偏差。

另一个在随机森林中使用的方法是，对数据进行随机抽样（装袋）的同时，独立树每次分裂时对输入特征也进行随机抽样。在randomForest包中，我们使用随机抽样数的默认值来对预测特征进行抽样。对于分类问题，默认值为所有预测特征数量的平方根；对于回归问题，默认值为所有预测特征数量除以3。在模型调优过程中，每次树分裂时，算法随机选择的预测特征数量是可变的。

通过每次分裂时对特征的随机抽样以及由此形成的一套方法，你可以减轻高度相关的预测特征的影响，这种预测特征在由装袋法生成的独立树中往往起主要作用。这种方法还使你不必思索如何减少装袋导致的高方差。独立树彼此之间的相关性减少之后，对结果的平均化可以使泛化效果更好，对于异常值的影响也更加不敏感，比仅进行装袋的效果要好。

6.1.4　梯度提升 Gradient boosting

损失函数（loss function）是用来估量模型的预测值f(x)与真实值Y的不一致程度，它是一个非负实值函数,通常使用L(Y, f(x))来表示，损失函数越小，模型的鲁棒性就越好

提升法的学习和理解可能会非常复杂，但你一定要对其中原理有基本的了解。它的主要思想是，先建立一个某种形式的初始模型（线性、样条、树或其他），称为基学习器；然后检查残差，在残差的基础上围绕**损失函数**拟合模型。损失函数测量模型和现实之间的差别，例如，在回归问题中可以用误差的平方，在分类问题中可以用逻辑斯蒂函数。一直继续这个过程，直到满足某个特定的结束条件。这与下面的情形有点相似：一个学生进行模拟考试，100道题中错了30道，然后只研究那30道错题；在下次模考中，30道题中又错了10道，然后只研究那10道题，以此类推。如果你想更加深入地研究背后的理论，可以参考Frontiers in Neurorobotics, *Gradient boosting machines, a tutorial*, Natekin A., Knoll A. (2013)（http://www.ncbi.nlm.nih.gov/pmc/atricles/PMC3885826/）。

刚才提到过，提升方法可以用于很多不同类型的基分类器，本章只集中讨论**基于树的学习**这个主题。 每个树迭代的次数都很少，我们要通过一个调优参数决定树的分裂次数，这个参数称为交互深度。事实上，有些树很小，只可以分裂一次，这样的树就被称为“树桩”。

这些树依次按照损失函数拟合残差，直至达到我们指定的树的数量（我们的结束条件）。

使用Xgboost包进行建模的过程中，有一些参数需要调整。Xgboost表示**eXtreme Gradient Boosting**。这个程序包表现优异，广泛应用于网上的数据科学竞赛。在下面这个网站中，你可以找到关于提升树和Xgboost的非常详尽的背景资料：

http://xgboost.readthedocs.io/en/latest/model.html

在后面的商业案例中，我们要看看如何找到最优的超参数，并使模型生成有意义的结果和预测。这些参数之间会互相作用，如果你只攻一点而不顾其他，模型的效果可能会越来越差。在调参过程中，caret包会助你一臂之力。

6.2　商业案例

在前几章中，我们通过努力建立了一些模型，本章的总体目标是看看我们能否提高这些模型的预测能力。对于回归问题，重新回到第4章中的前列腺癌数据集，以0.444的均方误差作为基准，看看能否改善。

对于分类问题，我们使用第3章中的乳腺癌数据集和第5章中的皮玛印第安人糖尿病数据集。

在乳腺癌数据集中，我们取得了97.6%的预测正确率。在糖尿病数据集中，正确率是79.6%，我们希望再提高一些。

随机森林和提升方法都会用于这3个数据集，简单树模型只用于乳腺癌数据集和前列腺癌数据集。

6.2.1 模型构建与模型评价

要进行模型构建，需要加载7个不同的R包。我们会实验每种技术，并与前面章节中介绍的方法进行对比，看看它们在数据分析中会有什么突出表现。

1. 回归树

我们直接跳到前列腺癌数据集，但首先要加载所需的R包。当然，要确定在加载之前你已经安装了它们：

```
  >   library(rpart) #classification and regression trees
> library(partykit) #treeplots
> library(MASS) #breast and pima indian data
> library(ElemStatLearn) #prostate data
> library(randomForest) #random forests
> library(xgboost) #gradient boosting
> library(caret) #tune hyper-parameters for easy machine learning workflow
```

先用prostate数据集做回归，像第4章中那样准备好数据，包括调出数据集。使用`ifelse()`函数将gleason评分编码为指标变量，划分训练数据集和测试数据集，训练数据集为pros.train，测试数据集为pros.test，如下所示：

```
  >   data(prostate)
> prostate$gleason <- ifelse(prostate$gleason == 6, 0, 1)
> pros.train <- subset(prostate, train == TRUE)[, 1:9]
> pros.test <- subset(prostate, train == FALSE)[, 1:9]
```

要在训练数据集上建立回归树，需要使用party包中的`rpart()`函数，语法与我们在其他建模技术中使用过的非常类似：

```
> tree.pros <- rpart(lpsa ~ ., data = pros.train)
```

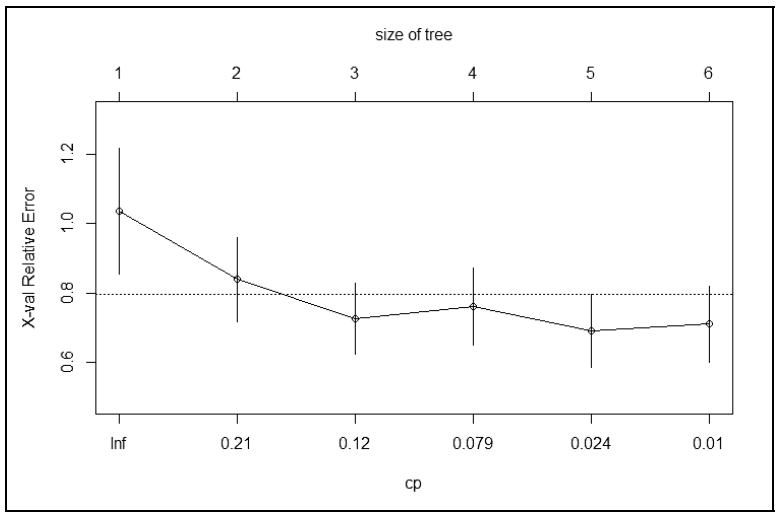
可以调出这个对象，然后检查每次分裂的误差，以决定最优的树分裂次数：

```
  > print(tree.pros$cptable)
   CP nsplit rel error  xerror    xstd
1 0.35852251    0 1.0000000 1.0364016 0.1822698
2 0.12295687    1 0.6414775 0.8395071 0.1214181
3 0.11639953    2 0.5185206 0.7255295 0.1015424
4 0.05350873    3 0.4021211 0.7608289 0.1109777
5 0.01032838    4 0.3486124 0.6911426 0.1061507
6 0.01000000    5 0.3382840 0.7102030 0.1093327
```

这个表对于我们的分析非常重要。标号为CP的第一列是成本复杂性参数，第二列nsplit是树分裂的次数，rel error列表示相对误差，即某次分裂的RSS除以不分裂的RSS（RSS(k)/RSS(0)），xerror和xstd都是基于10折交叉验证的，xerror是平均误差，xstd是交叉验证过程的标准差。可以看出，5次分裂在整个数据集上产生的误差最小，但使用交叉验证时，4次分裂产生的误差略微更小。可以使用`plotcp()`函数查看统计图：

```
> plotcp(tree.pros)
```

上述命令输出如下。



这幅图使用误差条表示树的规模和相对误差之间的关系，误差条和树规模是对应的。选择树的规模为5，也就是经过4次分裂可以建立一个新的树对象，这个树的xerror可以通过剪枝达到最小化。剪枝的方法是先建立一个cp对象，将这个对象和表中第5行相关联，然后使用`prune()`函数完成剩下的工作：

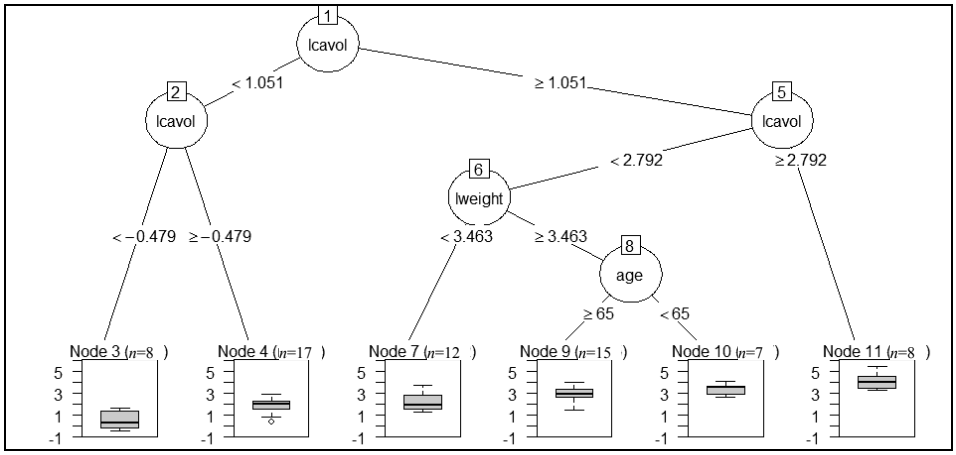
```
>   cp <- min(tree.pros$cptable[5, ])
> prune.tree.pros <- prune(tree.pros, cp = cp)
```

完成上面的代码后，可以用统计图比较完整树和剪枝树。由partykit包生成的树图明显优于party包生成的，在`plot()`函数中，你可以简单地使用`as.party()`函数作为包装器函数：

```
> plot(as.party(tree.pros))
```

上述命令输出如下页图。

完整树

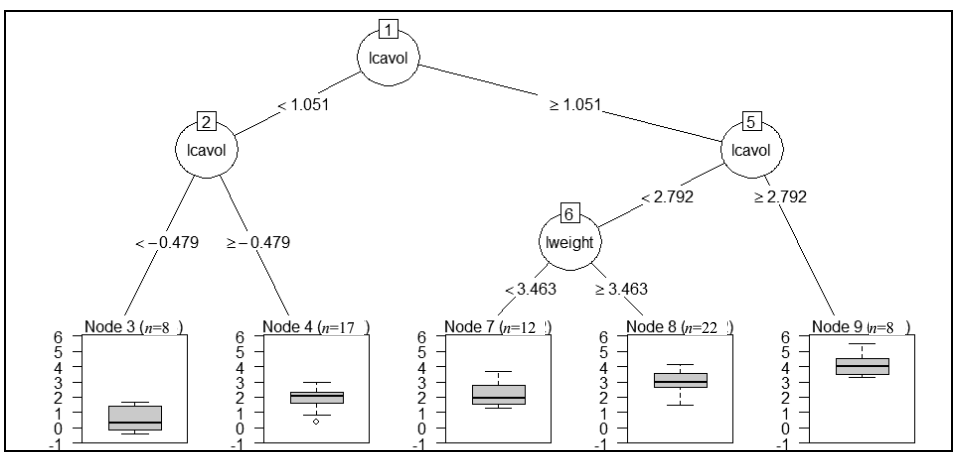


同样地，使用as.party()函数处理剪枝树：

```
> plot(as.party(prune.tree.pros))
```

上述命令输出如下。

剪枝树



请注意，除了最后一次分裂（完整树包含变量age），两个树是完全一样的。有趣的是，树的第一次分裂和第二次分裂都与肿瘤大小的对数（lcavol）有关。这两张图的信息量都很大，因为它们显示了树的分裂、节点、每节点观测数，以及预测结果的箱线图。

我们看看剪枝树在测试集上表现如何。在测试数据上使用predict()函数进行预测，并建立一个对象保存这些预测值。然后用预测值减去实际值，得到误差，最后算出误差平方的平均值：

```
  > party.pros.test <- predict(prune.tree.pros, newdata = pros.test)
> rpart.resid <- party.pros.test - pros.test$lpsa
> mean(rpart.resid^2) #caluclate MSE mean squared error (MSE)
[1] 0.5267748
```

第4章中的基准MSE为0.44，我们的努力并没有改善预测结果。但是，这种方法并非一无是处。从生成的树图中可以看出对响应变量影响最大的特征，并且很容易解释。就像引言中说过的，树模型很容易理解和解释，这往往比正确率重要得多。

2. 分类树

对于分类问题，我们先像第3章一样准备好乳腺癌数据。加载数据之后，你要删除患者ID，对特征进行重新命名，删除一些缺失值，然后建立训练数据集和测试数据集。如下所示：

```
  > data(biopsy)
> biopsy <- biopsy[, -1] #delete ID
> names(biopsy) <- c("thick", "u.size", "u.shape", "adhsn",
    "s.size", "nucl",
  "chrom", "n.nuc", "mit", "class") #change the feature names
> biopsy.v2 <- na.omit(biopsy) #delete the observations with
    missing values
> set.seed(123) #random number generator
> ind <- sample(2, nrow(biopsy.v2), replace = TRUE, prob = c(0.7,
    0.3))
> biop.train <- biopsy.v2[ind == 1, ] #the training data set
> biop.test <- biopsy.v2[ind == 2, ] #the test data set
```

准备好合适的数据之后，使用和前面回归问题一样的代码风格来解决分类问题。建立分类树之前，要确保结果变量是一个因子，可以用`str()`函数进行检查：

```
 > str(biop.test[, 10])
Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 2 1 1 ...
```

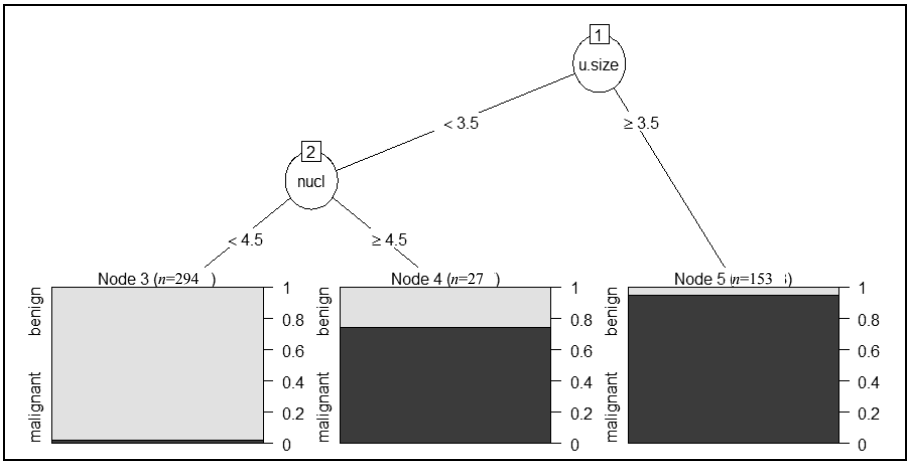
先生成树，然后检查输出中的表格，找到最优分裂次数：

```
    > set.seed(123)
> tree.biop <- rpart(class ~ ., data = biop.train)
> tree.biop$cptable
  CP nsplit rel error  xerror     xstd
1 0.79651163    0 1.0000000 1.0000000 0.06086254
2 0.07558140    1 0.2034884 0.2674419 0.03746996
3 0.01162791    2 0.1279070 0.1453488 0.02829278
4 0.01000000    3 0.1162791 0.1744186 0.03082013
```

交叉验证误差仅在两次分裂后就达到了最小值（见第3行）。现在可以对树进行剪枝，再在图中绘制剪枝树，看看它在测试集上的表现：

```
  > cp <- min(tree.biop$cptable[3, ])
> prune.tree.biop <- prune(tree.biop, cp = cp)
> plot(as.party(prune.tree.biop))
```

上述命令输出如下页图。



从对树图的检查中可以发现，细胞大小均匀度是第一个分裂点，第二个是nuclei。完整树还有一个分支是细胞浓度。使用`predict()`函数并指定`type="class"`，在测试集上进行预测。如下所示：

```
  > rparty.test <- predict(prune.tree.biop, newdata = biop.test, type
   ="class")
> table(rparty.test, biop.test$class)
rparty.test benign malignant
 benign     136      3
 malignant    6     64
> (136+64)/209
[1] 0.9569378
```

只有两个分支的基本树模型给出了差不多96%的正确率。这虽然比不上逻辑斯蒂回归的正确率97.6%，但足以激励我们相信，在随后的随机森林方法中会得到更好的改善效果。

3. 随机森林回归

在这一节中，我们还是先处理前列腺癌数据集，然后处理乳腺癌数据集和皮玛印第安人糖尿病数据集，使用的程序包是randomForest。建立一个随机森林对象的通用语法是使用`randomForest()`函数，指定模型公式和数据集这两个基本参数。回想一下每次树迭代默认的变量抽样数，对于回归问题，是$p/3$；对于分类问题，是p的平方根，p为数据集中预测变量的个数。对于大规模数据集，就p而言，你可以调整`mtry`参数，它可以确定每次迭代的变量抽样数值。如果p小于10，可以省略上面的调整过程。想在多特征数据集中优化`mtry`参数时，可以使用caret包，或使用randomForest包中的`tuneRF()`函数。了解这些情况之后，即可建立随机森林模型并检查模型结果。如下所示：

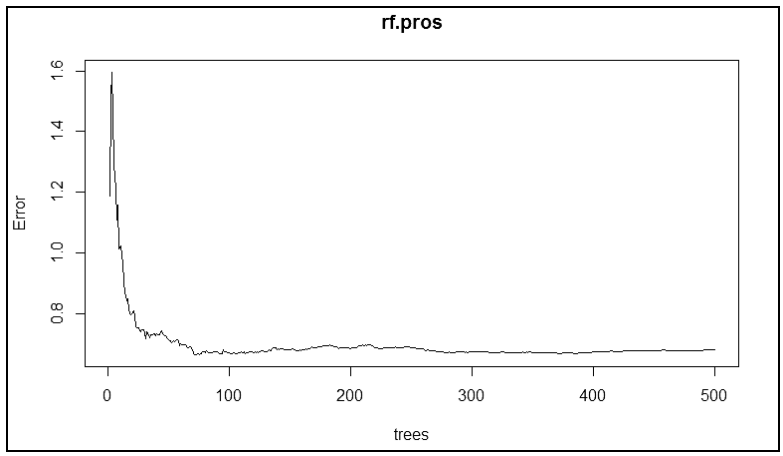
```
  > set.seed(123)
> rf.pros <- randomForest(lpsa ~ ., data = pros.train)
> rf.pros
```

```
Call:
  randomForest(formula = lpsa ~ ., data = pros.train)
  Type of random forest: regression
  Number of trees: 500
  No. of variables tried at each split: 2
  Mean of squared residuals: 0.6792314
  % Var explained: 52.73
```

调用rf.pros对象，可以知道随机森林生成了500个不同的树（默认设置），并且在每次树分裂时随机抽出两个变量。结果的MSE为0.68，差不多53%的方差得到了解释。下面看看能否对默认数量的树做一些改善。过多的树会导致过拟合，当然，“多大的数量是‘过多’”依赖于数据规模。两件事可以帮助我们达到目的，第一是做出rf.pros的统计图，另一件是求出最小的MSE：

```
> plot(rf.pros)
```

上述命令输出如下。



这个图表示MSE与模型中树的数量之间的关系。可以看出，树的数量增加时，一开始MSE会有显著改善，当森林中大约建立了100棵树之后，改善几乎停滞。

可以通过`which.min()`函数找出具体的最优树数量，如下所示：

```
  > which.min(rf.pros$mse)
[1] 75
```

可以试试有75棵树的随机森林，在模型语法中指定`ntree = 75`即可：

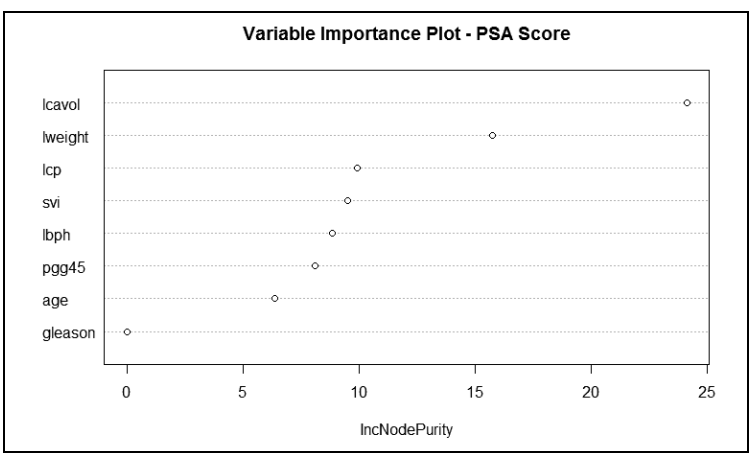
```
  > set.seed(123)
> rf.pros.2 <- randomForest(lpsa ~ ., data = pros.train, ntree
   =75)
> rf.pros.2
Call:
   randomForest(formula = lpsa ~ ., data = pros.train, ntree = 75)
   Type of random forest: regression
```

```
Number of trees: 75
No. of variables tried at each split: 2
Mean of squared residuals: 0.6632513
% Var explained: 53.85
```

可以看到，MSE和解释方差都有一点微小的改善。对模型进行检验之前，先看看另一张统计图。如果使用自助抽样和两个随机预测变量建立了75棵不同的树，要想将树的结果组合起来，需要一种方法确定哪些变量驱动着结果。独木不成林，你需要做出变量重要性统计图及相应的列表。Y轴是按重要性降序排列的变量列表，X轴是MSE改善百分比。请注意，在分类问题中，X轴应该是基尼指数的改善。使用`varImpPlot()`函数生成统计图：

```
> varImpPlot(rf.pros.2, scale = T,
 main = "Variable Importance Plot - PSA Score")
```

上述命令输出如下。



和单个树模型一致，`lcavol`是最重要的变量，`lweight`次之。如果想查看具体数据，可以使用`importance()`函数。如下所示：

```
  > importance(rf.pros.2)
    IncNodePurity
lcavol  24.108641
   lweight  15.721079
     age  6.363778
     lbph  8.842343
     svi  9.501436
     lcp  9.900339
   gleason  0.000000
    pgg45  8.088635
```

现在可以看看模型在测试数据上的表现：

```
  > rf.pros.test <- predict(rf.pros.2, newdata = pros.test)
> rf.resid = rf.pros.test - pros.test$lpsa #calculate residual
```

```
> mean(rf.resid^2)
[1] 0.5136894
```

MSE依然高于我们在第4章中使用LASSO得到的0.44，也不比单个树模型更好。

4. 随机森林分类

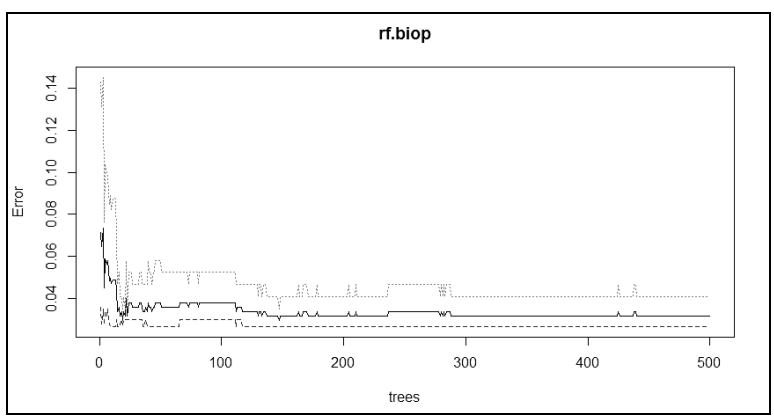
你可能会对随机森林回归模型的表现感到失望，但这种技术的真正威力在于解决分类问题。我们从乳腺癌诊断数据开始，这个过程和在回归问题中的做法几乎一样：

```
  > set.seed(123)
> rf.biop <- randomForest(class ~. , data = biop.train)
> rf.biop
Call:
 randomForest(formula = class ~ ., data = biop.train)
        Type of random forest: classification
              Number of trees: 500
No. of variables tried at each split: 3
    OOB estimate of error rate: 3.16%
Confusion matrix:
     benign malignant class.error
benign    294      8 0.02649007
malignant   7    165 0.04069767
```

OOB（袋外数据）误差率为3.16%。随机森林还是由默认的全部500棵树组成。画出表示误差和树的数量关系的统计图：

```
> plot(rf.biop)
```

上述命令输出如下。



图中显示，误差和标准误差的最小值是在树的数量很少的时候取得的，可以使用`which.min()`函数找出具体值。和前面不同的一点是，需要指定第一列来得到误差率，这是整体误差率。算法会为每个类标号的误差率生成一个附加列，本例中不需要它们。模型结果中也没有`mse`，而是代之以`err.rate`。如下页代码所示：

```
  > which.min(rf.biop$err.rate[, 1])
[1] 19
```

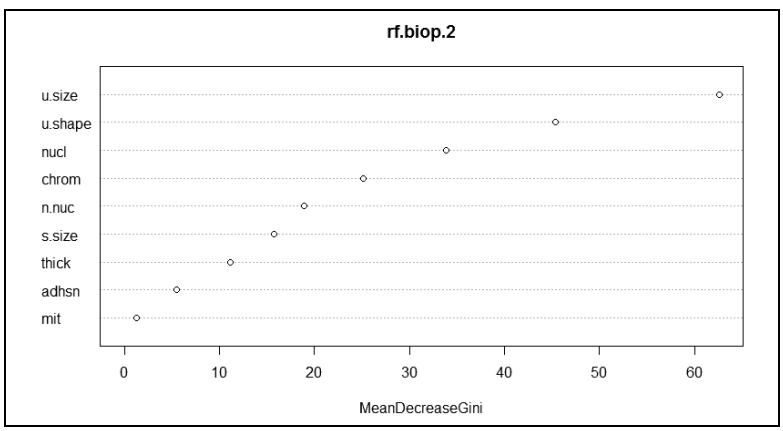
只需19棵树就可以使模型正确率达到最优。实验一下，看看模型的表现：

```
    > set.seed(123)
    > rf.biop.2 <- randomForest(class~ ., data = biop.train, ntree =
     19)
> print(rf.biop.2)
Call:
 randomForest(formula = class ~ ., data = biop.train, ntree = 19)
        Type of random forest: classification
              Number of trees: 19
No. of variables tried at each split: 3
    OOB estimate of error rate: 2.95%
Confusion matrix:
     benign malignant class.error
benign    294      8 0.02649007
malignant   6    166 0.03488372
> rf.biop.test <- predict(rf.biop.2, newdata = biop.test, type =
   "response")
> table(rf.biop.test, biop.test$class)
rf.biop.test benign malignant
  benign   139       0
  malignant  3      67
> (139 + 67) / 209
[1] 0.9856459
```

怎么样？训练集上的误差率还不到3%，在测试集上甚至表现得更好。测试集的209个观测中，只有3个观测被误分类，而且没有一个是误诊为“恶性”的。回忆一下，我们使用逻辑斯蒂回归得到的正确率最好结果是97.6%，所以随机森林模型就是至今为止在乳腺癌数据上的最佳模型。进行下面的内容之前，先看看变量重要性统计图。

```
> varImpPlot(rf.biop.2)
```

上述命令输出如下。



上页图中，变量重要性是指每个变量对基尼指数平均减少量的贡献，此处的变量重要性与单个树分裂时有很大区别。回忆一下，单个树是在细胞大小均匀度开始分裂的（与随机森林一致），然后是nuclei，接着是细胞密度。这揭示了随机森林技术具有非常大的潜力，不但可以提高模型预测能力，还可以改善特征选择的结果。

现在，开始面对皮玛印第安人糖尿病模型的艰巨挑战。我们先做好数据准备，如下所示：

```
  > data(Pima.tr)
> data(Pima.te)
> pima <- rbind(Pima.tr, Pima.te)
> set.seed(502)
> ind <- sample(2, nrow(pima), replace = TRUE, prob = c(0.7, 0.3))
> pima.train <- pima[ind == 1, ]
> pima.test <- pima[ind == 2, ]
```

做完数据准备工作，下一步建立模型。如下所示：

```
  > set.seed(321)
> rf.pima = randomForest(type~., data=pima.train)
> rf.pima
Call:
 randomForest(formula = type ~ ., data = pima.train)
        Type of random forest: classification
           Number of trees: 500
No. of variables tried at each split: 2
    OOB estimate of error rate: 20%
Confusion matrix:
   No Yes class.error
No 233 29  0.1106870
Yes 48 75  0.3902439
```

我们在训练数据集上得到了20%的误分类误差率，并不比以前的结果更好。下面对树的数目进行优化，看看能否显著改善模型结果：

```
  > which.min(rf.pima$err.rate[, 1])
[1] 80
> set.seed(321)
> rf.pima.2 = randomForest(type~., data=pima.train, ntree=80)
> print(rf.pima.2)
Call:
 randomForest(formula = type ~ ., data = pima.train, ntree = 80)
        Type of random forest: classification
           Number of trees: 80
No. of variables tried at each split: 2
    OOB estimate of error rate: 19.48%
Confusion matrix:
   No Yes class.error
No 230 32  0.1221374
Yes 43 80  0.3495935
```

当随机森林中有80棵树时，OOB误差有些许改善。随机森林能够在测试集上证明自己吗？我

们来看一下。如下所示：

```
  > rf.pima.test <- predict(rf.pima.2, newdata= pima.test,
   type = "response")
> table(rf.pima.test, pima.test$type)
rf.pima.test No Yes
     No 75 21
     Yes 18 33
> (75+33)/147
[1] 0.7346939
```

我们在测试集上只得到了73%的正确率，低于使用SVM的结果。

虽然随机森林在糖尿病数据集上令人失望，但它已经证明了，自己在乳腺癌诊断方面是迄今为止最好的分类器。最后，我们实验一下梯度提升方法。

5. 极限梯度提升——分类

我们提到过，在这一节要使用已经加载的xgboost包。因为这种方法的效果如雷贯耳，所以我们直接用它解决最有挑战性的皮玛印第安人糖尿病问题。

正如在提升算法简介中所说，我们要调整一些参数。

- nrounds：最大迭代次数（最终模型中树的数量）。
- colsample_bytree：建立树时随机抽取的特征数量，用一个比率表示，默认值为1（使用100%的特征）。
- min_child_weight：对树进行提升时使用的最小权重，默认为1。
- eta：学习率，每棵树在最终解中的贡献，默认为0.3。
- gamma：在树中新增一个叶子分区时所需的最小减损。
- subsample：子样本数据占整个观测的比例，默认值为1（100%）。
- max_depth：单个树的最大深度。

使用expand.grid()函数可以建立实验网格，以运行caret包的训练过程。对于前面列出的参数，如果没有设定具体值，那么即使有默认值，运行函数时也会收到出错信息。下面的参数取值是基于我以前的一些训练迭代而设定的。建议各位亲自实验参数调整过程。

使用以下命令建立网格：

```
> grid = expand.grid(
 nrounds = c(75, 100),
 colsample_bytree = 1,
 min_child_weight = 1,
 eta = c(0.01, 0.1, 0.3), #0.3 is default,
 gamma = c(0.5, 0.25),
 subsample = 0.5,
```

```
 max_depth = c(2, 3)
 )
```

以上命令会建立一个具有24个模型的网格，caret包会运行这些模型，以确定最好的调优参数。此处必须提示各位，对于我们现在所用的这种规模的数据集，代码运行过程只需几秒钟，但对于一些大数据集，这个运行过程可能需要几小时。所以，在时间非常宝贵的情况下，你必须应用自己的判断力，并使用小数据样本来找出合适的调优参数，否则硬盘空间可能不足。

使用car包的train()函数之前，我要创建一个名为cntrl的对象，来设定trainControl的参数。这个对象会保存我们要使用的方法，以训练调优参数。我们使用5折交叉验证，如下所示：

```
> cntrl = trainControl(
 method = "cv",
 number = 5,
 verboseIter = TRUE,
 returnData = FALSE,
 returnResamp = "final"
 )
```

要使用train.xgb()函数，只需和训练其他模型一样，设定好所需参数即可：训练数据集、标号、训练控制对象和实验网格。记得要设定随机数种子：

```
> set.seed(1)
> train.xgb = train(
 x = pima.train[, 1:7],
 y = ,pima.train[, 8],
 trControl = cntrl,
 tuneGrid = grid,
 method = "xgbTree"
 )
```

因为在trControl中设定了verboseIter为TURE，所以可以看到每折交叉验证中的每次训练迭代。

调用train.xgb这个对象可以得到最优的参数，以及每种参数设置的结果，如下所示（有所简省）：

```
> train.xgb
eXtreme Gradient Boosting
No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 308, 308, 309, 308, 307
Resampling results across tuning parameters:
eta max_depth gamma nrounds Accuracy Kappa
0.01 2     0.25  75    0.7924286 0.4857249
0.01 2     0.25  100    0.7898321 0.4837457
0.01 2     0.50  75    0.7976243 0.5005362
.................................................
0.30 3     0.50  75    0.7870664 0.4949317
0.30 3     0.50  100    0.7481703 0.3936924
```

```
Tuning parameter 'colsample_bytree' was held constant at a
value of 1
Tuning parameter 'min_child_weight' was held constant at a
value of 1
Tuning parameter 'subsample' was held constant at a value of 0.5
Accuracy was used to select the optimal model using the largest
 value.
The final values used for the model were nrounds = 75, max_depth =
 2,
eta = 0.1, gamma = 0.5, colsample_bytree = 1, min_child_weight = 1
and subsample = 0.5.
```

由此可以得到最优的参数组合来建立模型。模型在训练数据上的正确率是81%，Kappa值是0.55。下面要做的事情有点复杂，但我认为这才是最佳实践。首先创建一个参数列表，供Xgboost包的训练函数`xgb.train()`使用。然后将数据框转换为一个输入特征矩阵，以及一个带标号的数值型结果列表（其中的值是0和1）。接着，将特征矩阵和标号列表组合成符合要求的输入，即一个`xgb.Dmatrix`对象。代码如下：

```
> param <- list( objective = "binary:logistic",
 booster = "gbtree",
 eval_metric = "error",
 eta = 0.1,
 max_depth = 2,
 subsample = 0.5,
 colsample_bytree = 1,
 gamma = 0.5
 )
> x <- as.matrix(pima.train[, 1:7])
> y <- ifelse(pima.train$type == "Yes", 1, 0)
> train.mat <- xgb.DMatrix(data = x, label = y)
```

这些准备工作完成之后，即可创建模型：

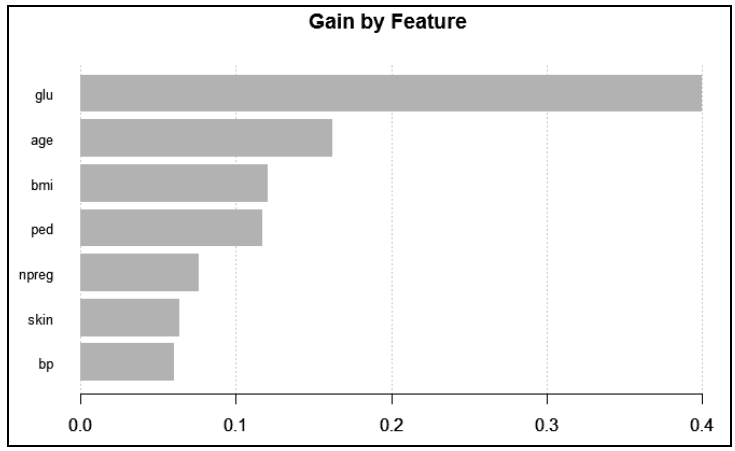
```
> set.seed(1)
> xgb.fit <- xgb.train(params = param, data = train.mat, nrounds =
 75)
```

在测试集上查看模型效果之前，先检查变量重要性，并绘制统计图。你可以检查3个项目：gain、cover和frequecy。gain是这个特征对其所在分支的正确率做出的改善，cover是与这个特征相关的全体观测的相对数量，frequency是这个特征在所有树中出现的次数百分比。以下代码会生成我们需要的输出：

```
> impMatrix <- xgb.importance(feature_names = dimnames(x)[[2]],
 model = xgb.fit)
> impMatrix
  Feature    Gain   Cover Frequency
1:   glu 0.40000548 0.31701688 0.24509804
2:   age 0.16177609 0.15685050 0.17156863
3:   bmi 0.12074049 0.14691325 0.14705882
4:   ped 0.11717238 0.15400331 0.16666667
```

```
5:  npreg 0.07642333 0.05920868 0.06862745
6:  skin 0.06389969 0.08682105 0.10294118
7:   bp 0.05998254 0.07918634 0.09803922
> xgb.plot.importance(impMatrix, main = "Gain by Feature")
```

上述命令输出如下。



与其他方法相比，这个特征重要性图有什么不同吗？

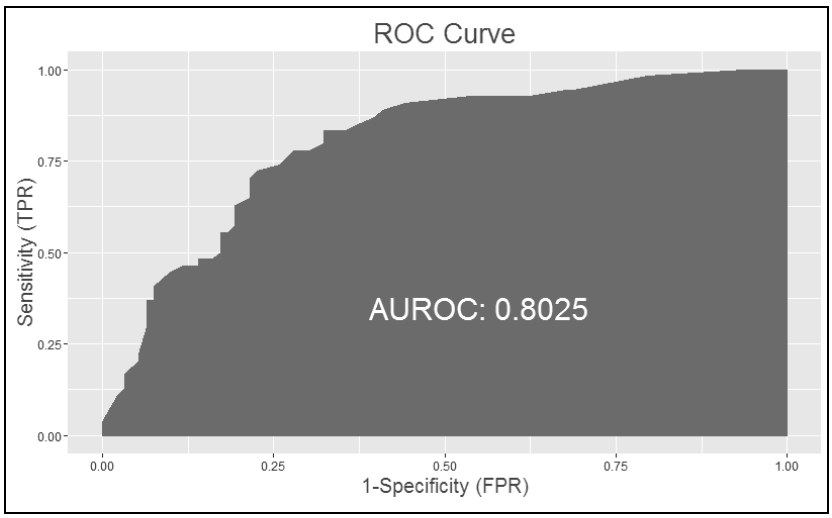
下面看看这个模型在测试集上的表现。与训练集一样，测试集数据也要转换为矩阵。再次使用InformationValue包中的工具来帮助我们完成这个任务。下面的代码先加载程序库，再生成预测结果来分析模型性能：

```
> library(InformationValue)
> pred <- predict(xgb.fit, x)
> optimalCutoff(y, pred)
[1] 0.3899574
> pima.testMat <- as.matrix(pima.test[, 1:7])
> xgb.pima.test <- predict(xgb.fit, pima.testMat)
> y.test <- ifelse(pima.test$type == "Yes", 1, 0)
> confusionMatrix(y.test, xgb.pima.test, threshold = 0.39)
  0 1
0 72 16
1 20 39
> 1 - misClassError(y.test, xgb.pima.test, threshold = 0.39)
[1] 0.7551
```

你注意到我在代码中使用`optimalCutoff()`函数做的事情了吗？这个Information包中的函数可以找出使误差最小化的最优概率阈值。顺便说一下，模型误差大概是25%，还是没有超过SVM模型。再悄悄告诉你，可以看看ROC曲线，AUC超过了0.8。以下代码可以生成ROC曲线：

```
> plotROC(y.test, xgb.pima.test)
```

代码输出如下页图。



6.2.2 模型选择

回忆一下我们在本章的基本目标——使用基于树的学习方法提高前几章中得到的模型的预测能力。我们有什么收获？首先，在具有定量响应变量的前列腺癌数据集上，第4章已经得到了一个线性模型，但我们的努力没有改善这个模型。其次，随机森林在威斯康星乳腺癌数据集上的表现超过了第3章中的逻辑斯蒂回归模型。最后，我必须失望地宣布，在皮玛印第安人糖尿病数据集上，使用提升树不能得到比SVM模型更好的结果。

综上，使我感到欣慰的是，对于前列腺癌数据集和乳腺癌数据集，我们得到了比较好的模型。第7章将介绍神经网络和深度学习的概念，我们会再次尝试改善糖尿病模型。开始下一章之前，我想介绍一种使用随机森林技术的强大的特征消除方法。

6.2.3 使用随机森林进行特征选择

到现在为止，我们已经介绍了几种特征选择技术，比如正则化、最优子集以及递归特征消除。下面要介绍一种对于分类问题非常有效的特征选择方法，这种方法是通过随机森林实现的，需要使用Boruta包。有一篇论文详细介绍了这种方法如何选择所有相关特征：

Kursa M., Rudnicki W. (2010), Feature Selection with the Boruta Package, *Journal of Statistical Software*, 36(11), 1 - 13

在这一节，我会简单介绍这种算法，然后将其应用于一个特征非常多的数据集。我发现这种算法非常有效，但有人警告我，说这种算法非常消耗计算能力，可能达不到预想的效果。但它确实可以有效消除不重要的特征，使我们可以集中精力构建一个更简洁、更有效，也更有意义的模

型，还是值得花时间学习的。

在更高的层次上，算法会复制所有输入特征，并对特征中的观测顺序进行重新组合，以去除相关性，从而创建**影子特征**。然后使用所有输入特征建立一个随机森林模型，并计算每个特征（包括影子特征）的正确率损失均值的Z分数。如果某个特征的Z分数显著高于影子特征的Z分数，那么这个特征就被认为是**重要**的；反之，这个特征就被认为是**不重要**的。然后，去除掉影子特征和那些已经确认了重要性的特征，重复上面的过程，直到所有特征都被赋予一个表示重要性的值。你可以设定随机森林迭代的最大次数。算法结束之后，每个初始特征都会被标记为**确认**、**待定**或**拒绝**。对于待定的特征，你必须自己确定是否要包括在下一次建模中。根据具体情况，你可以有以下几种选择：

- 改变随机数种子，重复运行算法多次（*k*次），然后只选择那些在*k*次运行中都标记为“确认”的属性；
- 将你的训练数据分为*k*折，在每折数据上分别进行算法迭代，然后选择那些在所有*k*折数据上都标记为“确认”的属性。

请注意，所有这些工作都可以用几行代码完成。下面查看代码，并将其应用于Sonar数据集，这个数据集来自mlbench包。数据集中有208个观测，60个输入特征，以及1个用于分类的标号向量。标号是个因子，如果sonar对象是岩石，标号就是R；如果sonar对象是矿藏，标号则是M。首先加载数据，并快速进行数据探索：

```
> data(Sonar, package="mlbench")
> dim(Sonar)
[1] 208 61
> table(Sonar$Class)
  M R
111 97
```

要运行这个算法，你只须加载Boruat包，然后在`boruta()`函数中创建一条模型公式。请记住，标号必须是因子类型，否则算法不会正常执行。如果想跟踪算法的进程，可以设定`doTrace = 1`。还有，不要忘了设定随机数种子：

```
> library(Boruta)
> set.seed(1)
> feature.selection <- Boruta(Class ~ ., data = Sonar, doTrace = 1)
```

正如我们前面提到过的，这个算法需要大量的计算能力。在我的老式笔记本电脑上，需要以下这么长的时间：

```
> feature.selection$timeTaken
Time difference of 25.78468 secs
```

从一个简单的表格可以得出最终重要决策的计数。可以看出，我们完全能够去除大约一半特征：

```
> table(feature.selection$finalDecision)
Tentative Confirmed Rejected
    12        31       17
```

根据以上结果，可以很容易地使用我们选择的特征创建一个新的数据框。从getSelected Attributes()函数开始，这个函数可以找出特征名称。在下面的例子中，只选择那些“确认”的特征。如果想包括“确认”和“待定”的特征，只需在函数中指定withTentative=TRUE：

```
  > fNames <- getSelectedAttributes(feature.selection) # withTentative =
TRUE
  > fNames
[1] "V1" "V4" "V5" "V9" "V10" "V11" "V12" "V13" "V15" "V16"
[11] "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V27" "V28" "V31"
[21] "V35" "V36" "V37" "V44" "V45" "V46" "V47" "V48" "V49" "V51"
[31] "V52"
```

使用这些特征名称，可以创建一个Sonar数据集的子集：

```
> Sonar.features <- Sonar[, fNames]
> dim(Sonar.features)
[1] 208 31
```

一切搞定！数据框Sonar.features包含了boruta算法选择的所有“确认”特征。现在可以使用它进行更深入、更有意义的数据探索。通过几行代码和一些对算法运行的耐心等待，即可显著提高建模工作和知识生成的水平。

6.3 小结

你在本章既体会了基于树的学习方法在处理分类和回归问题方面的威力，也看到了它们的局限性。单个树模型虽然易于构建和解释，但对于我们试图解决的多数问题不具备必需的预测能力。要想提高模型的预测能力，建议使用随机森林和梯度提升树。通过随机森林方法可以建立几十棵甚至几百棵树，这些树的结果会聚集成一个综合的预测。随机森林中的每棵树都是使用数据抽样建立的，抽样的方法称为自助法，预测变量也同样进行抽样。对于梯度提升方法，先创建一棵初始的、相对小规模的树，之后，基于残差或误分类会生成一系列树。这种技术的预期结果是建立一系列树，后面的树可以对前面的树的缺点加以改善，最终降低偏差和方差。我们还知道，在R中，可以使用随机森林作为特征选择的方法。

尽管这些方法确实非常强大，但在机器学习世界中它们不是万能的。不同的数据集要求分析者根据实际情况判断应该使用什么分析技术。对于分析者来说，技术的选择和调优参数的选择同等重要，有些预测模型是好的，但有些预测模型是伟大的，这种不断调整和细化的过程决定了二者之间的所有区别。

下一章的主要任务是，使用R建立神经网络和深度学习模型。