```
# CHUNK 7
cp.min <- dt$cptable[which.min(dt$cptable[, "xerror"]), "CP"]
cp.min

## [1] 0.726

dt.pruned <- prune(dt, cp = cp.min)
rpart.plot(dt.pruned)
```


Figure 5.2.3: The pruned regression tree for `Y` with `X1` and `X2` as predictors.

branches of `dt` that do not satisfy the impurity reduction requirement prescribed by the optimal complexity parameter. (If you find the coding here too hard, you can also manually set the `cp` parameter of the `prune()` function to the optimal complexity parameter, 0.726, or any greater value.) As we expect, the pruned tree in this case is the no-split tree with only the root node. This suggests that the fitted decision tree is, to our dismay, not very useful, perhaps due to the small sample size.

## 5.3 Case Study: Classification Trees

Now that we are comfortable with reading the output of a tree-based model in R and have some experience with fitting decision trees, in this section we examine a more sophisticated case study involving classification trees, which are technically more complex (and thus more likely to be tested!) than regression trees. Both base and ensemble trees will be constructed. At the completion of this case study, you should be able to:

- Build base classification trees and control their complexity by pruning.
- Build ensemble trees using the `train()` and `trainControl()` functions in the `caret` package and tune the model parameters.
- Quantify the prediction accuracy of (base or ensemble) classification trees constructed.
- Compare the pros and cons of decision trees relative to GLMs.

| Variable | Description | Characteristics |
|---|---|---|
| `ID` | Sample code number | Positive integer 61634-13454352 |
| `thickness` | Clump thickness | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `size` | Uniformity of cell size | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `shape` | Uniformity of cell shape | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `adhesion` | Marginal adhesion | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `eSize` | Single epithelial cell size | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `bNuclei` | Bare nuclei | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic, ? = missing data) |
| `chromatin` | Bland chromatin | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `nNucleoli` | Normal nucleoli | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `mitosis` | Mitoses | Positive integer 1-10 (1 = closest to benign, 10 = most anaplastic) |
| `class` | Class of breast cancer | Integer 0-1 (0 = benign, 1 = malignant) |

Table 5.2: Data dictionary for the `breast` data.

### 5.3.1 Problem Set-up and Preparatory Steps

**Data description.** This case study revolves around the Original Wisconsin Breast Cancer dataset available from the UCI Machine Learning Repository. Obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg, this dataset has been saved as a pre-processed[xii] CSV file named `breast.csv` for your convenience. The file contains the information of $n = 699$ fine-needle aspirate samples, where 458 (65.52%) are benign and 241 (34.48%) are malignant. For each sample, nine cytological characteristics possibly related to breast cancer malignancy are available. These characteristics are described in Table 5.2. The goal of the case study is to identify the key characteristics affecting malignancy and develop tree-based classifiers that accurately predict malignancy using a combination of these characteristics. Such classifiers allow early treatment to be prescribed and have considerable benefits to medical professionals as well as patients. Because the end users of these classifiers are individuals who may not have the expertise in predictive analytics, ease of interpretation and communication of a model will be a big plus.

For starters, run CHUNK 1 to load the dataset and print a summary for each variable.

[xii] See `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)` for the original dataset.

```
# CHUNK 1
breast <- read.csv("breast.csv")
summary(breast)

##        ID              thickness          size            shape
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  870688   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1071704   Mean   : 4.418   Mean   : 3.134   Mean   : 3.207
##  3rd Qu.: 1238298   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##     adhesion          eSize           bNuclei      chromatin
##  Min.   : 1.000   Min.   : 1.000   1      :402   Min.   : 1.000
##  1st Qu.: 1.000   1st Qu.: 2.000   10     :132   1st Qu.: 2.000
##  Median : 1.000   Median : 2.000   2      : 30   Median : 3.000
##  Mean   : 2.807   Mean   : 3.216   5      : 30   Mean   : 3.438
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3      : 28   3rd Qu.: 5.000
##  Max.   :10.000   Max.   :10.000   8      : 21   Max.   :10.000
##                                    (Other): 56
##    nNucleoli         mitosis          class
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000
##  Median : 1.000   Median : 1.000   Median :0.0000
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
##
```

The summary suggests that all of the variables in the dataset at this stage are treated as numeric variables with the exception of `bNuclei`, which has its frequency table printed. This will be taken care of in the first task.

**TASK 1: Data pre-processing**

Remove any questionable observations and determine whether any variables should undergo a type conversion before the dataset is ready for analysis.

The target variable is the last variable, `class`, representing the type of breast cancer. The first variable is simply a column of ID numbers which contribute no useful information for predicting malignancy, so it will be dropped.

```
# CHUNK 2
# Drop the ID variable
breast$ID <- NULL
```

The nine characteristics, viewed as predictors for `class`, are all integer-valued on a scale from 1 (closest to benign) to 10 (most anaplastic), except `bNuclei`, which is treated technically as an

unordered factor due to the 16 missing observations coded as a question mark "?" (not NA). Due to their small number and the absence of evidence that they are systematically generated, the deletion of these 16 observations with missing values of bNuclei is unlikely to bias or impair the effectiveness of the predictive models we are going to construct. The deletion is done in CHUNK 3 via logical subsetting introduced in Section 1.3.

```
# CHUNK 3
class(breast$bNuclei)

## [1] "factor"

table(breast$bNuclei)

##
##   ?   1  10   2   3   4   5   6   7   8   9
##  16 402 132  30  28  19  30   4   8  21   9

# Drop the missing observations for bNuclei
breast <- breast[!(breast$bNuclei == "?"), ]
nrow(breast)

## [1] 683
```

The logical test breast$bNuclei == "?" identifies all of the rows in the breast data with a missing value for bNuclei. Feeding the complement of this set of rows into the row index allows us to filter out these 16 rows, with 683 observations remaining in the breast data. Following the removal of the missing observations, bNuclei is converted to an integer variable to recognize its ordered nature. Otherwise, R would still treat bNuclei as an unordered factor and the classification trees we will construct might make a strange split using $\{\texttt{bNuclei} = 1, 3, 5, 7, 9\}$, leading to nonsensical model results. This conversion is achieved by a two-step procedure:

1. It is first converted to an ordered factor with its levels (integers from 1 to 10) explicitly specified and with the option order = TRUE added (recall the use of the levels and order arguments as discussed on page 20).

2. The ordered factor is then converted to an integer via the as.integer() function.

These two steps are done in CHUNK 4.

```
# CHUNK 4
# Convert bNuclei to integers
breast$bNuclei <- factor(breast$bNuclei, levels = 1:10, order = TRUE)
breast$bNuclei <- as.integer(breast$bNuclei)
class(breast$bNuclei)

## [1] "integer"

table(breast$bNuclei)

##
##   1   2   3   4   5   6   7   8   9  10
## 402  30  28  19  30   4   8  21   9 132
```

Note that the first step is necessary. Had we applied the `as.integer()` function directly to the original unordered version of `bNuclei`, we would have got the following undesirable result:

```
breast$bNuclei <- as.integer(breast$bNuclei)
class(breast$bNuclei)
table(breast$bNuclei)
## [1] "integer"
##
##   2   3   4   5   6   7   8   9  10  11
## 402 132  30  28  19  30   4   8  21   9
```

Not only would the values of `bNuclei` become 2 to 11 rather than 1 to 10, more importantly the ordered structure of `bNuclei` would also be destroyed, with the original level 10 (which would become level 3) preceding the original levels 2-9 (which would become levels (4-11)).

Finally, we convert `class` from an integer variable to a factor in CHUNK 5. This will force R to acknowledge the categorical nature of `class` and will help with not only data visualization, but also model development.

```
# CHUNK 5
# Convert class to a factor
breast$class <- as.factor(breast$class)
summary(breast)

##    thickness          size            shape          adhesion
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.00
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.00
##  Median : 4.000   Median : 1.000   Median : 1.000   Median : 1.00
##  Mean   : 4.442   Mean   : 3.151   Mean   : 3.215   Mean   : 2.83
##  3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000   3rd Qu.: 4.00
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.00
##      eSize           bNuclei         chromatin        nNucleoli
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.00
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.00
##  Median : 2.000   Median : 1.000   Median : 3.000   Median : 1.00
##  Mean   : 3.234   Mean   : 3.545   Mean   : 3.445   Mean   : 2.87
##  3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 4.00
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.00
##     mitosis       class
##  Min.   : 1.000   0:444
##  1st Qu.: 1.000   1:239
##  Median : 1.000
##  Mean   : 1.603
##  3rd Qu.: 1.000
##  Max.   :10.000
```

Overall, we have removed 16 rows of observations involving missing values for `bNuclei`, and the summary shows that `bNuclei` and `class` have been correctly converted to an integer and a factor (with 65.01% and 34.99% of the observations belonging to the two levels `"0"` and `"1"`), respectively.

**EXAM NOTE**

The December 2018 PA exam solutions says that "[f]ew candidates summarized the total effects of data cleaning." At a minimum, point out how many observations and variables are left after cleaning the data. If you have made any particularly significant and somewhat controversial changes, e.g., removing a large number of observations, be sure to make a mention as well.

**TASK 2: Explore the relationship of each variable to `class`**

Use graphical displays and/or summary statistics to form preliminary conclusions regarding which variables are likely to have significant predictive power.

Our target variable `class` is binary and all predictors are numeric, so a good way to explore the relationship between each predictor and `class` is to create box plots for the predictor split by the two levels of `class`. In CHUNK 6, we use a `for` loop to construct the split box plots for each of the nine numeric predictors, with the plots for `thickness`, `size`, `bNuclei`, and `mitosis` shown in Figure 5.3.1. We can see that all characteristics, except for `mitosis`, stand out as important predictors for malignancy, with malignant observations conspicuously associated with much larger and more volatile values of the predictors (you can also use summary statistics and look at the means of the predictors grouped by the two levels of `class`, but the relationships in CHUNK 6 are so strong that summary statistics are not warranted). The challenge in this case study is to develop predictive models that identify the most useful variables among these possibly highly correlated predictors, which we will address in later tasks.

### 5.3.2 Construction and Evaluation of Base Classification Trees

**TASK 3: Construct base classification trees**

Construct two classification trees of different levels of complexity to predict malignancy. Be sure to explain the tree development methodology. Do not use an ensemble method here.

Prior to fitting any trees, we split the data into the training (70%) and test (30%) sets again with the use of the `createDataPartition()` function, which uses stratified sampling to mitigate the effects that random sampling could have on model outcomes.

```
# CHUNK 7
library(caret)
set.seed(2019)
partition <- createDataPartition(y = breast$class, p = .7, list = FALSE)
train <- breast[partition, ]
test <- breast[-partition, ]

table(train$class)/nrow(train)
table(test$class)/nrow(test)
```

```
# CHUNK 6
library(ggplot2)

var <- names(breast)[-10] # Extract all variables except class

for (i in var) {
  plot <- ggplot(breast, aes(x = class, y = breast[, i])) +
    geom_boxplot() +
    labs(y = i)
  print(plot)
}
```

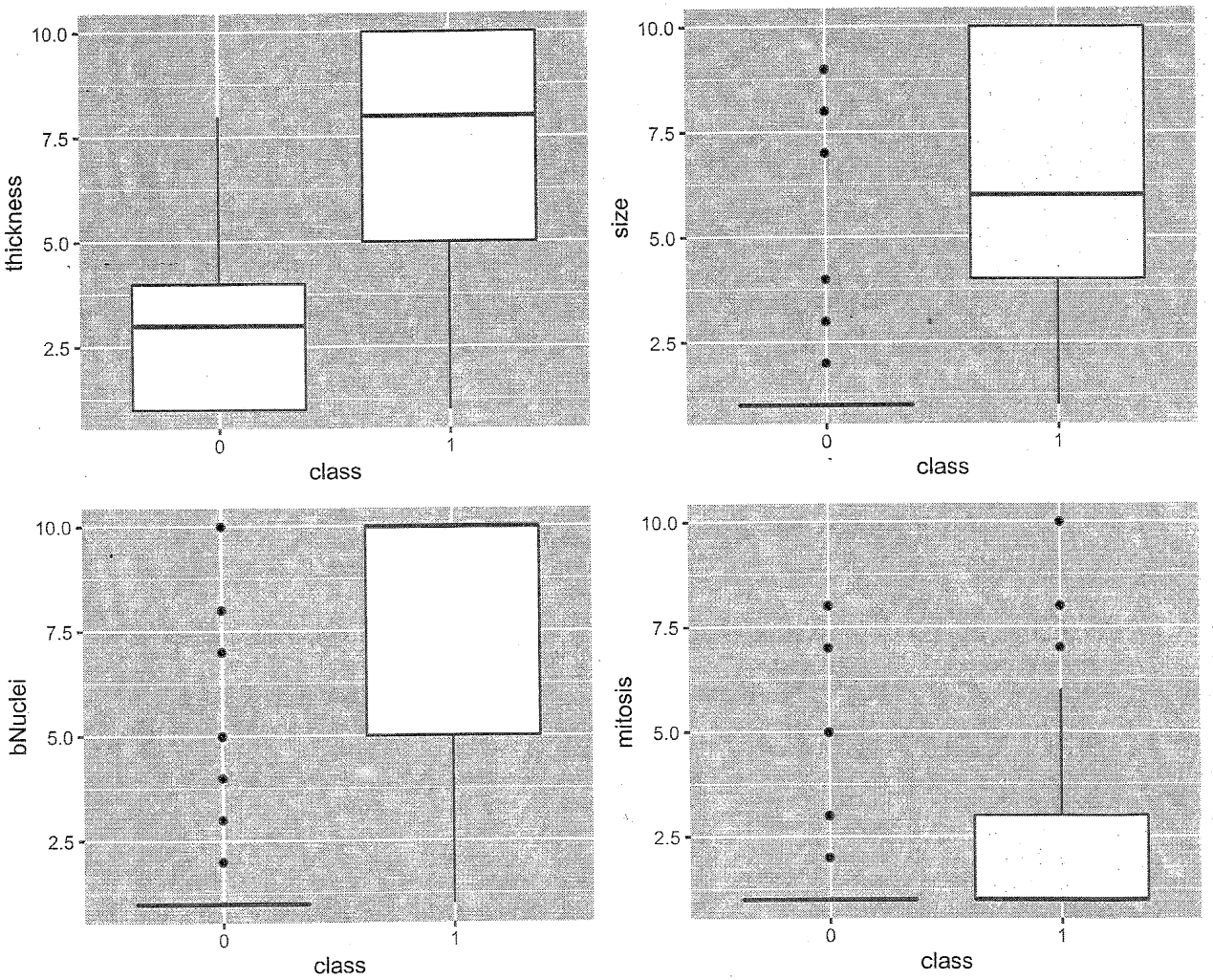


Figure 5.3.1: Split box plots for `thickness`, `size`, `bNuclei`, and `mitosis` by `class` in the `breast` dataset.

```
##
##         0         1
## 0.6492693 0.3507307
##
##         0         1
## 0.6519608 0.3480392
```

To check that the two sets are representative, we note that the proportion of the malignant observations is 35.07% in the training set and 34.80% in the test set. While these two proportions do not exactly match, they are close enough, showing the effectiveness of the built-in stratification of the target variable. We will use the same training/test partition for all models in this case study.

**Tree 1: Fitting a large, complex tree.** To begin with, we build a large, complex classification tree using all of the nine characteristics to predict malignancy. The control parameters are deliberately set so that a large tree can be grown. The complex tree will be pruned at a later stage to achieve the optimal level of tree complexity.

- `minbucket`: The minimum number of observations in any terminal node is 2, which is reasonable in view of the relatively small size of the data.
- `cp`: The complexity parameter is set to 0.0005. A split will be made as long as it lowers the overall node impurity (which is measured by Gini, due to the command `parms = list(split = "gini")`) strictly by 0.0005. The low value of `cp` allows us to construct a sufficiently large tree while pre-pruning branches that are not worthwhile to pursue.
- `maxdepth`: The maximum depth of the tree is 20.

**EXAM NOTE**

When constructing a decision tree, be sure to describe the rationale behind the choice of the control parameters as they play a pivotal role in controlling tree complexity!

This full tree, referred to as Tree 1 in the sequel, is constructed in CHUNK 8, saved as an `rpart` object named `dt.1` (to distinguish it from another tree we will grow), and depicted in Figure 5.3.2.

```
# CHUNK 8
library(rpart)
library(rpart.plot)
set.seed(123)
# method = "class" ensures that target is treated as a categorical variable
dt.1 <- rpart(class ~ ., data = train, method = "class",
              control = rpart.control(minbucket = 2,
                                      cp = 0.0005,
                                      maxdepth = 20),
              parms = list(split = "gini"))
dt.1
```

```
## n= 479
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 479 168 0 (0.64926931 0.35073069)
##    2) size< 3.5 328  25 0 (0.92378049 0.07621951)
##      4) bNuclei< 5.5 311  10 0 (0.96784566 0.03215434)
##        8) nNucleoli< 3.5 304   5 0 (0.98355263 0.01644737)
##         16) bNuclei< 2.5 282   0 0 (1.00000000 0.00000000) *
##         17) bNuclei>=2.5 22   5 0 (0.77272727 0.22727273)
##           34) thickness< 3.5 15   0 0 (1.00000000 0.00000000) *
##           35) thickness>=3.5 7   2 1 (0.28571429 0.71428571)
##             70) eSize< 2.5 3   1 0 (0.66666667 0.33333333) *
##             71) eSize>=2.5 4   0 1 (0.00000000 1.00000000) *
##        9) nNucleoli>=3.5 7   2 1 (0.28571429 0.71428571)
##         18) thickness< 4.5 2   0 0 (1.00000000 0.00000000) *
##         19) thickness>=4.5 5   0 1 (0.00000000 1.00000000) *
##      5) bNuclei>=5.5 17   2 1 (0.11764706 0.88235294) *
##    3) size>=3.5 151   8 1 (0.05298013 0.94701987)
##      6) size< 4.5 29   6 1 (0.20689655 0.79310345)
##       12) bNuclei< 7.5 13   6 1 (0.46153846 0.53846154)
##         24) adhesion< 3.5 5   1 0 (0.80000000 0.20000000) *
##         25) adhesion>=3.5 8   2 1 (0.25000000 0.75000000)
##           50) shape< 4.5 3   1 0 (0.66666667 0.33333333) *
##           51) shape>=4.5 5   0 1 (0.00000000 1.00000000) *
##       13) bNuclei>=7.5 16   0 1 (0.00000000 1.00000000) *
##      7) size>=4.5 122   2 1 (0.01639344 0.98360656) *
```

Note that we have preceded the call of the `rpart()` function with a random seed. This is to make sure that the internal cross-validation results returned by `rpart()` are reproducible. (No random seed was used in CHUNK 6 of Section 5.2 because the number of folds used equals the number of training observations, making the training/validation splits non-random.)

**Interpreting the output of a classification tree.** When we type the name of a classification tree, we will have at a single glance a lot of useful information about all of the tree splits made. Compared to a regression tree, we no longer have the RSS, which is replaced by the number of misclassifications in each node (in the `loss` column) along with the proportions of observations in each class of the categorical target variable (in the `(yprob)` column). As shown in CHUNK 8, Tree 1 starts with the root node, where there are 479 observations, 168 of which are misclassified—the majority of the observations are benign and so benign (0) is the predicted class (see the `yval` column), but these 168 observations are malignant. Then `"size < 3.5"` represents the first split, meaning that out of all the $p = 9$ predictors, partitioning the data on the basis of `size` leads to the greatest reduction in impurity at this point. Of all the 479 observations in the training set, those with `size < 3.5` will be sent to the left branch, where the predicted class is benign, while those with `size >= 3.5` will be sent to the right branch, where the predicted class is malignant. At this point, you can see that the number of misclassifications decreases drastically from 168 to

```
# CHUNK 8 (Cont.)
rpart.plot(dt.1)
```

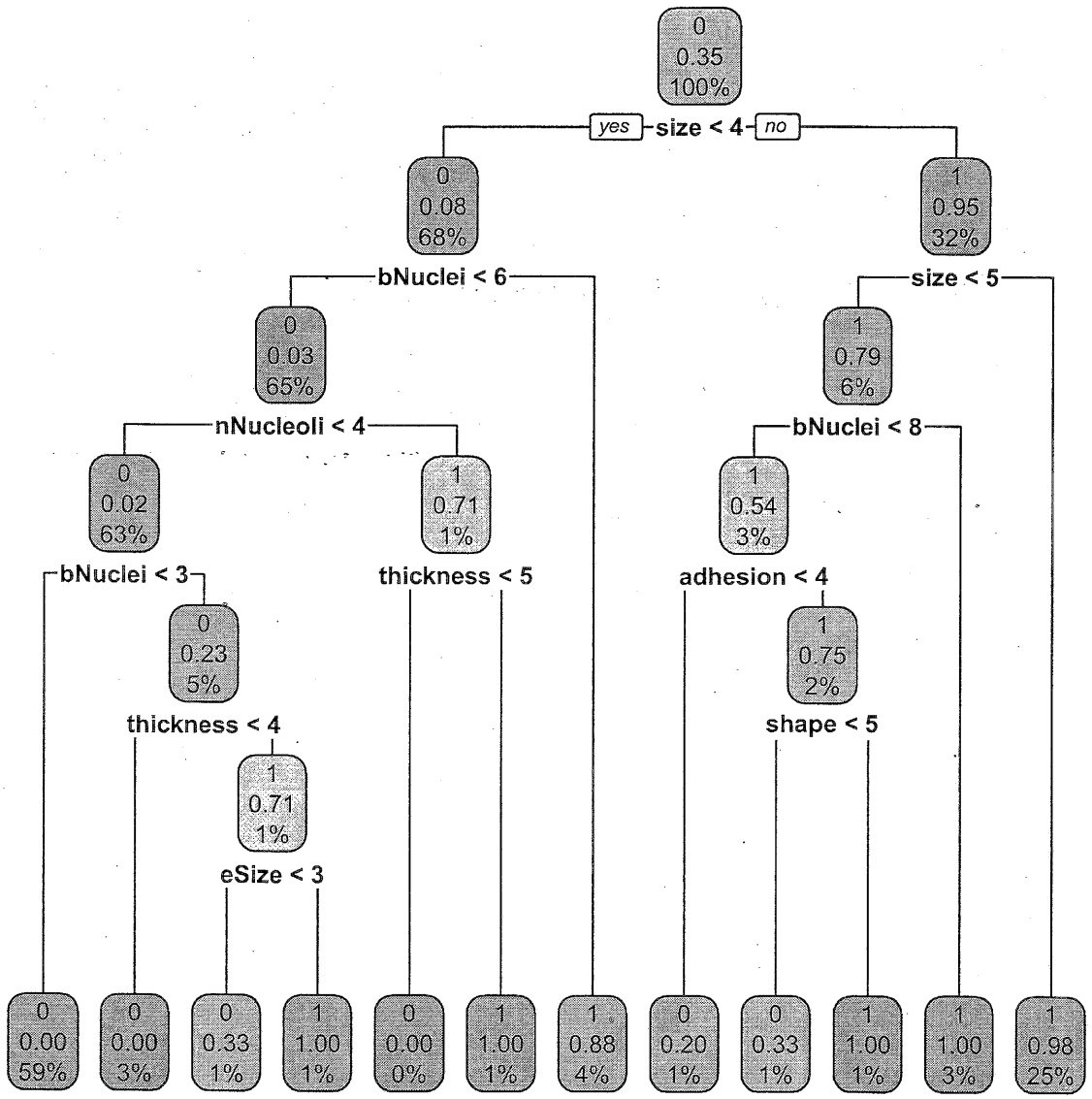


Figure 5.3.2: The first classification tree constructed for the `breast` data.

$25 + 8 = 33$. The splitting continues until the impurity of the model cannot be improved by more than the `cp` parameter. We can see that some terminal nodes such as nodes 18, 19, 24, 50, 51, 70, and 71 contain fewer than 10 observations, which may suggest that this tree is overfitted.

Figure 5.3.2 visualizes the fitted classification tree and shows, for each node, the predicted class (top value), the proportion of observations in that node that are malignant (middle number), and the proportion of training observations belonging to the node (bottom number). For instance, node 3, which is the right branch coming out of the root node and contains 32% of the training observations, has a predicted class of `"1"` since 95% of the observations there are malignant.

**Tree 2: Pruning Tree 1 using the one-standard-error rule.** As you can see in Figure 5.3.2, Tree 1 has a total of 11 splits (equivalently, 12 terminal nodes) and seems too complex to be used for interpretation and/or predictions. Remember that the classification tree is intended for use by doctors, who may not be familiar with predictive analytics, so interpretability should be one of the key factors when deciding on which model to use, in addition to prediction accuracy. To produce a classification tree of reasonable size for both predictions and interpretation, let's look at the `cptable` component of Tree 1, as in CHUNK 9.

```
# CHUNK 9
dt.1$cptable
```

```
##           CP nsplit  rel error    xerror       xstd
## 1 0.803571429      0 1.00000000 1.0000000 0.06216670
## 2 0.077380952      1 0.19642857 0.2261905 0.03520741
## 3 0.017857143      2 0.11904762 0.1488095 0.02897483
## 4 0.011904762      3 0.10119048 0.1309524 0.02727046
## 5 0.008928571      4 0.08928571 0.1130952 0.02542604
## 6 0.005952381      6 0.07142857 0.1130952 0.02542604
## 7 0.000500000     11 0.04166667 0.1011905 0.02410285
```

Just like a regression tree, the `cptable` of a classification tree shows how the relative error (which, in a classification setting, is the classification error of the corresponding tree scaled by the classification error of the tree with no split) and scaled cross-validation error vary with the complexity parameter. As the complexity parameter decreases (i.e., the tree becomes more complex), the relative error decreases sharply, as shown in the `rel error` column. We know that, however, it is the cross-validation error that is an accurate measure of the predictive performance of trees of different sizes. A good way to visualize the behavior of the cross-validation error is to look at the *complexity parameter plot*, which exhibits how the cross-validation error of a decision tree varies with the complexity parameter. Run CHUNK 10 to use the `plotcp()` function to generate such a plot for Tree 1 (Figure 5.3.3). The top of the plot shows the size of the tree measured by the number of terminal nodes (which equals number of splits + 1) associated with different values of the complexity parameter.

In this case study, the cross-validation error happens to follow a decreasing shape (if you set the value of `minbucket` to 1, then the cross-validation error eventually rises, albeit slowly), so the complexity parameter that gives rise to the lowest cross-validation error happens to be the smallest value, `cp = 0.0005`, which corresponds to Tree 1, with 11 splits. However, we can observe that the trees with 4 and 6 splits have a cross-validation error which is comparable to that of Tree 1 while having a much smaller size. To identify a tree that is much more interpretable and is comparably predictive, one common practice is to employ the *one-standard-error rule*:

```
# CHUNK 10
plotcp(dt.1)
```

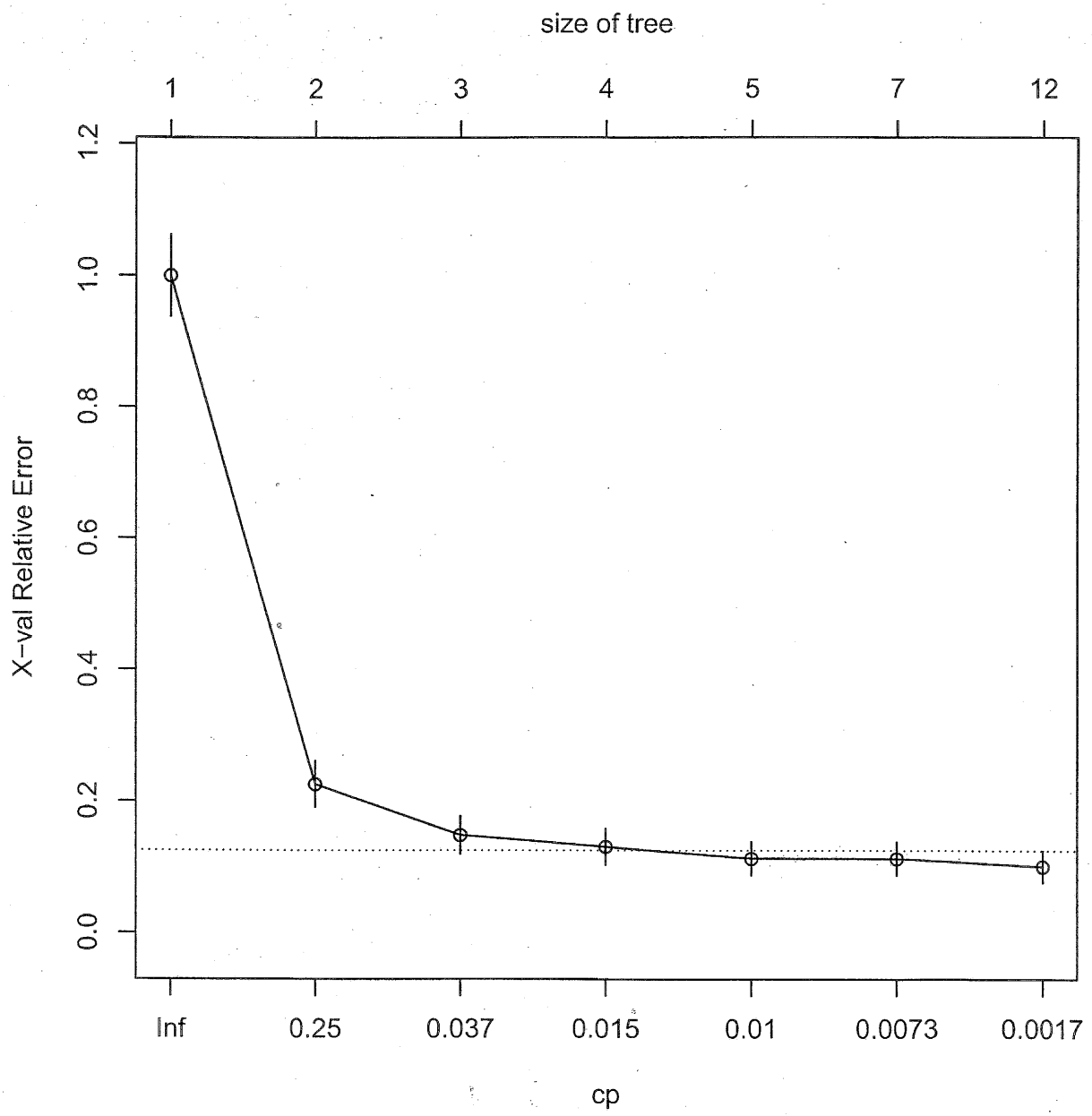


Figure 5.3.3: Complexity parameter plot for Tree 1 in the `breast` data.

To select the smallest tree whose cross-validation error is *within one standard error from the minimum cross-validation error.*

In the current setting, this cutoff level equals $0.1011905 + 0.02410285 = 0.125293$, which is also shown in the complexity parameter plot (Figure 5.3.3) as a dotted line. Among all trees with a cross-validation error within one standard error from the minimum cross-validation error, the simplest tree is the 4-split (i.e., 5-terminal-node) tree, as we can see from Figure 5.3.3, with a complexity parameter of 0.008928571. In CHUNK 11, we pass this value to the `cp` argument in the `prune()` function along with `dt.1` to produce a much simpler tree, called Tree 2 and coded `dt.2`.

Compared to Tree 1, Tree 2 is considerably simpler and more interpretable. It only has four splits and five terminal nodes, starting with `size`, then further splitting the larger bucket (68% of the observations) by `bNuclei`, `nNucleoli`, and `thickness`. That `size` appears in the first split indicates that it is the most informative predictor for malignancy, which makes intuitive sense. It also appears that there are interactions between these four characteristics as there are further splits on the left branch of the root node, but not on the right branch.

**EXAM NOTE**

The December 2018 PA model solutions says that "only some [candidates] interpreted [a decision tree] fully." At a minimum, you can comment on how many splits the tree has, which variables appear to be the most informative or influential as shown in the first few splits, and whether that makes sense or not (hopefully so!).

**TASK 4: Select a classification tree**

Compare the two trees constructed in Task 3 and make a recommendation as to which tree to use. Do not base your recommendation solely on predictive performance.

**Comparing the predictive performance of the two trees.** Now that we have constructed two different classification trees with rather different degrees of complexity on the training set, it is time to evaluate and rank their predictive performance on the test set. The first task is to generate predictions for the observations on the test set. As you can expect, this task is fulfilled again by the super versatile `predict()` function we have been using. When the `predict()` function is applied to a classification tree fitted by `rpart()`, the `type` argument can be used to indicate the type of predictions:

- `"prob"`: To produce a matrix of predicted class probabilities, one column for each class of the target variable. This is the default for a classification tree fitted by `rpart()`.
- `"class"`: To produce a vector of predicted class labels.

Run CHUNK 12 to produce the predicted classes and use them to construct the confusion matrices (defined in Subsection 4.1.3) for the two classification trees.

```
# CHUNK 11
dt.2 <- prune(dt.1, cp = dt.1$cptable[5, "CP"])
rpart.plot(dt.2)
```

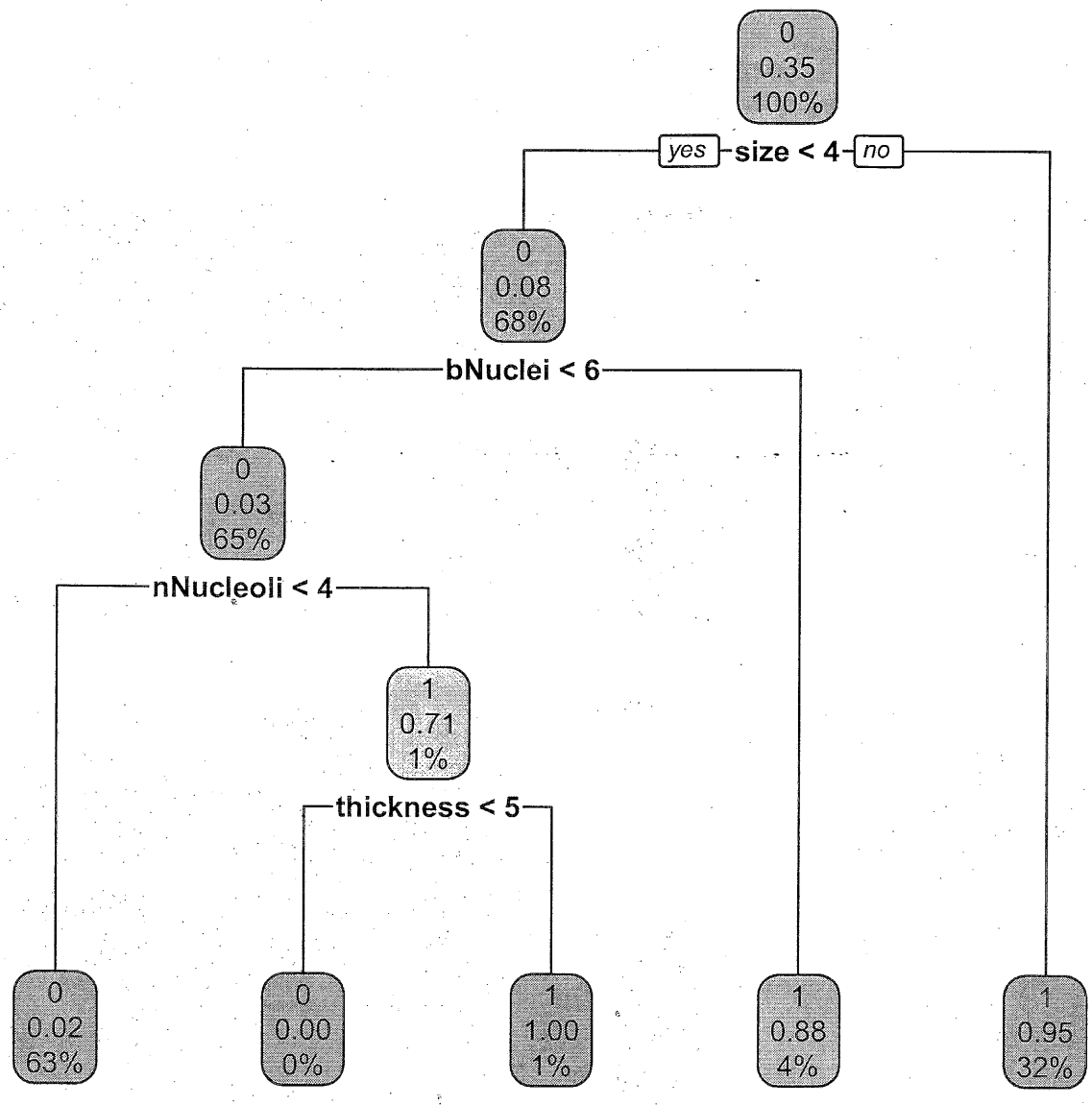


Figure 5.3.4: The second classification tree resulting from the one-standard-error rule in the `breast` data.

```
# CHUNK 12
pred.1.class <- predict(dt.1, newdata = test, type = "class")
pred.2.class <- predict(dt.2, newdata = test, type = "class")

# The positive class should be "1"
confusionMatrix(pred.1.class, test$class, positive = "1")

## Confusion Matrix and Statistics
##
##          Reference
## Prediction   0   1
##          0 129   2
##          1   4  69
##
##                Accuracy : 0.9706
##                  95% CI : (0.9371, 0.9891)
##     No Information Rate : 0.652
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9356
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9718
##             Specificity : 0.9699
##          Pos Pred Value : 0.9452
##          Neg Pred Value : 0.9847
##              Prevalence : 0.3480
##          Detection Rate : 0.3382
##    Detection Prevalence : 0.3578
##       Balanced Accuracy : 0.9709
##
##        'Positive' Class : 1
##

confusionMatrix(pred.2.class, test$class, positive = "1")

## Confusion Matrix and Statistics
##
##          Reference
## Prediction   0   1
##          0 129   2
##          1   4  69
##
##                Accuracy : 0.9706
##                  95% CI : (0.9371, 0.9891)
##     No Information Rate : 0.652
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9356
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9718
##             Specificity : 0.9699
##          Pos Pred Value : 0.9452
##          Neg Pred Value : 0.9847
##              Prevalence : 0.3480
##          Detection Rate : 0.3382
##    Detection Prevalence : 0.3578
##       Balanced Accuracy : 0.9709
##
##        'Positive' Class : 1
##
```

Despite their different degrees of complexity, the two classification trees share the same accuracy $(0.9706 = (129 + 69)/204)$, which is an important metric that has practical significance in this medical diagnosis context and can be easily communicated to medical professionals. Their sensitivity and specificity are 0.9718 and 0.9699, respectively, both of which are very close to 1.

**Example 5.3.1. (Ranking the two trees by AUC)** Compare the two classification trees with respect to AUC.

*Solution.* Instead of ranking the two trees on the basis of accuracy, sensitivity, or specificity based on a given cutoff, we can compare them using a single metric such as the AUC on the test set. In CHUNK 13, we first generate the predicted probabilities for the positive class `"1"` on the test set and feed these predicted probabilities into the `roc()` function (we first used this function in Section 4.3).

```
# CHUNK 13
library(pROC)

# Extract the predicted probabilities for the second class (malignant)
pred.1 <- predict(dt.1, newdata = test)[, 2]
pred.2 <- predict(dt.2, newdata = test)[, 2]

roc(test$class, pred.1, auc = TRUE)

##
## Call:
## roc.default(response = test$class, predictor = pred.1, auc = TRUE)
##
## Data: pred.1 in 133 controls (test$class 0) < 71 cases (test$class 1).
## Area under the curve: 0.9682

roc(test$class, pred.2, auc = TRUE)

##
## Call:
## roc.default(response = test$class, predictor = pred.2, auc = TRUE)
##
## Data: pred.2 in 133 controls (test$class 0) < 71 cases (test$class 1).
## Area under the curve: 0.9602
```

As mentioned above, the `predict()` function when applied to a classification tree produces, by default, a matrix of predicted probabilities, so we use the subscript `[, 2]` to extract the predicted probabilities for the second class, `"1"`, which is the positive class. We have also set the option `auc = TRUE` in the `roc()` function so that the AUC of the two trees will be automatically returned.

The two AUCs, 0.9682 and 0.9602, are very close to each other, meaning that Trees 1 and 2 have comparable prediction performance, despite Tree 2 having much fewer splits and using much less information. Considering both prediction accuracy and interpretability, it seems reasonable to recommend Tree 2, which is predictive and easy to interpret, to medical professionals. □

**Remark: Using the `caret` package to fit decision trees.** Besides the `rpart` package, the `caret` package, which we have been using to generate training and test sets and confusion matrices, can also be used to fit decision trees and automate the process of selecting the complexity parameter. To fit a single tree, the `rpart` package is, in my opinion, easier to use and good enough to produce almost all of the information we need in Exam PA. The syntax of the tree-building functions in the `caret` package, called `trainControl()` and `train()`, is more complex, but we will see in the next subsection that the `caret` package is very useful for tuning parameters when *multiple* decision trees are fitted and used together to make predictions.

### 5.3.3 Construction and Evaluation of Ensemble Trees

**TASK 5: Investigate ensemble trees**

Construct two ensemble trees and evaluate their test set performance. Would you recommend using any of these ensemble trees or the base classification trees in Task 4? As in Task 4, do not base your recommendation solely on predictive performance.

In this subsection, we construct two ensemble classification trees, a random forest and a boosted tree, and see if the use of ensemble methods will lead to a significant improvement on predictive performance over the base trees fitted in Subsection 5.3.2 (we, of course, hope for a good, if not significant improvement!). Coding-wise, we will switch from the `rpart` package to the `caret` package. The code will be more cumbersome, but it is (very) likely that code chunks will be provided on the exam, as in the Student Success sample project (if not, consult `?trainControl` and `?train`!). When studying this section, pay more attention to what tuning parameters you can control.

**Ensemble Tree 1: Random forest.** In R, there are several packages for fitting a random forest, such as `randomForest` and `caret`, both of which are available on the exam. Here we will use the `caret` package as its tree-building functions work not only for random forests, but also for boosted trees, and they have powerful features for tuning tree parameters by cross-validation.

In the `caret` package, a generic function for fitting a number of classification and regression models is `train()`, but it is often preceded by the `trainControl()` function, which allows you to control the way cross-validation is done to tune parameters. In the first part of CHUNK 14, we request that 5-fold cross-validation (`number = 5`) repeated 3 times (`repeats = 3`) for robustness be conducted for tuning the random forest parameters and *undersampling* (`sampling = "down"`) be applied. Used with unbalanced data (i.e., the two proportions of the categorical target variable are widely different), undersampling is meant to undersample the negative class (benign here) and retain all of the positive class observations (malignant here) to produce roughly balanced data for training. This will improve the model's ability to capture the signal leading to malignancy. (You can also try oversampling by specifying `sampling = "up"`)

```
# CHUNK 14
set.seed(1)  # because cross-validation is done
control <- trainControl(method = "repeatedcv",
                        number = 5,
                        repeats = 3,
                        sampling = "down")
```

Given the manner in which cross-validation will be performed, we can set up a "grid" containing all possible values of the parameters we are interested in tuning. Technically, the grid should be a data frame whose variables are named in the same way as the tuning parameters of the predictive model to be used (random forest here) and will be iterated across to find the optimal combination. Among all the parameters of a random forest, the number of features considered in each split, represented by the `mtry` parameter, is arguably the most important (note that the `caret` package only supports the tuning of `mtry` for a random forest model). Its default value is $\sqrt{p}$ for a classification tree and $p/3$ for a regression tree, where $p$ is the number of features. In the second part of CHUNK 14, we use the `expand.grid()` function to set up a data frame known as

`rf.grid` containing the possible values of `mtry` (in this case, all integers from 1 to 9) that we would like to consider. (Here we follow the PA e-learning modules and use the `expand.grid()` function, which is not needed for this simple case. This function will prove useful in CHUNK 18 below though and will be explained there.)

```
# CHUNK 14 (Cont.)
rf.grid <- expand.grid(mtry = 1:9)
rf.grid

##   mtry
## 1    1
## 2    2
## 3    3
## 4    4
## 5    5
## 6    6
## 7    7
## 8    8
## 9    9
```

The control parameters and grid are passed to the `trControl` and `tuneGrid` arguments of the `train()` function, respectively, to construct a random forest on the training set of the `breast` data, as in the last part of CHUNK 14.

```
# CHUNK 14 (Cont.)
rf <- train(class ~ .,
            data = train,
            method = "rf",
            ntree = 200,
            importance = TRUE,
            trControl = control,
            tuneGrid = rf.grid)
rf

## Random Forest
##
## 479 samples
##   9 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 383, 383, 384, 382, 384, 384, ...
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
```

```
##   1     0.9749264  0.9455864
##   2     0.9728355  0.9409089
##   3     0.9721339  0.9394974
##   4     0.9679744  0.9301140
##   5     0.9735155  0.9425218
##   6     0.9644874  0.9224921
##   7     0.9672728  0.9288399
##   8     0.9672725  0.9289430
##   9     0.9638074  0.9211887
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 1.
```

Besides the usual `formula` and `data` arguments, we have specified `method = "rf"` to instruct the `train()` function to construct a random forest (which is what `"rf"` stands for). The `ntree` parameter determines the number of trees to be grown. The default value is 500. Note that it is desirable to have as many trees as possible, as long as computational resources allow, to ensure that each observation and each feature is represented at least once in the random forest. If your dataset is large, then you may want to set `ntree` to a smaller value to ease computation burden (e.g., the dataset in Section 7.3 of the PA e-learning modules has about $n = 350,000$ observations, for which `ntree = 500` will freeze your computer for a day!). To save time, here we set `ntree` to 200 (still it will take a few seconds to load the code).

**EXAM NOTE**

To test the construction of ensemble trees, a real exam will probably instruct you to set the tuning parameters carefully so that an ensemble tree can be fitted in only a few seconds.

Unlike a base decision tree, typing the name of a random forest will not return how the splits are defined (remember, there are as many as `ntree` trees fitted!); instead, some basic information about the random forest like the number of observations (479), predictors (9), levels of the target variable (`"0"` and `"1"`), and the resampling method used (5-fold cross-validation, repeated 3 times) is shown. Most importantly, the optimal value of `mtry` is 1, which means that only one predictor should be sampled and automatically used in every split of every tree built.

In CHUNK 15, we apply the random forest to classify each case in the test set, generate its confusion matrix, and compute its AUC. In contrast to an `rpart` object, for which the `predict()` function outputs predicted probabilities by default, the default type of predictions returned by `predict()` applied to a random forest is the predicted class.

```
# CHUNK 15
pred.rf.class <- predict(rf, newdata = test)
confusionMatrix(pred.rf.class, test$class, positive = "1")

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 129   0
##          1   4  71
##
##                Accuracy : 0.9804
##                  95% CI : (0.9506, 0.9946)
##     No Information Rate : 0.652
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9574
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 1.0000
##             Specificity : 0.9699
##          Pos Pred Value : 0.9467
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3480
##          Detection Rate : 0.3480
##    Detection Prevalence : 0.3676
##       Balanced Accuracy : 0.9850
##
##        'Positive' Class : 1
##
```

```
# Add the type = "prob" option to return predicted probabilities
pred.rf.prob <- predict(rf, newdata = test, type = "prob")[, 2]
roc(test$class, pred.rf.prob, auc = TRUE)
```

```
##
## Call:
## roc.default(response = test$class, predictor = pred.rf.prob,     auc = TRUE)
##
## Data: pred.rf.prob in 133 controls (test$class 0) < 71 cases (test$class 1).
## Area under the curve: 0.9963
```

The accuracy of the random forest is 0.9804, which is (slightly) higher than that of Trees 1 and 2 (0.9706) fitted in Subsection 5.3.2. It is remarkable that all of the malignant test observations are correctly predicted to be malignant, leading to a unit sensitivity. In this medical diagnosis context, it is more important to correctly classify malignant cases so that patients with malignant breast cancer can be given proper treatment, than to correctly classify benign cases. In this regard, the unit sensitivity is desirable. The random forest also outperforms Trees 1 and 2 in terms of AUC; its AUC is 0.9963, compared favorably to that of Trees 1 and 2 (0.9682 and 0.9602, respectively). Even though we are losing the ability to interpret our model using an intuitive tree-like representation, the gain in predictive power is non-negligible.

**Variable importance plots.** Although random forests cannot produce a series of easy-to-understand classification rules like what a single decision tree does, they have graphical devices that try to overcome this model opacity issue. One useful tool is a *variable importance plot*, which ranks the predictors according to their contribution to the model, with the importance score for a particular predictor computed by totaling the drop in node impurity due to that predictor, averaged over all the trees in the random forest. Variables higher up in the list lead to larger improvements in the splitting criterion when they were used as splits and therefore are more important.

In CHUNK 16, we use the `varImp()` function to compute the variable importance of each of the nine predictors and make a variable importance plot (see Figure 5.3.5). To facilitate comparison, the importance scores have been scaled so that the most important predictor has a score of 100 and the predictors are sorted in descending order. The plot shows that `bNuclei` is the most important predictor of malignancy, followed by `size` and `thickness`. While `thickness` and `bNuclei` also appear in Figure 5.3.4, the order of importance suggested by Tree 2 (`size`, `bNuclei`, `nNucleoli`, and `thickness`, in this order) is not quite the same. Which set of findings is more reliable? In most cases, the order indicated by the random forest should be more credible as it is based on a total of `ntree` (200) trees fitted rather than a single tree, which is known to be unstable.

**Ensemble Tree 2: Boosted Model.** The second ensemble tree we will fit is a boosted model, also by the `train()` and `trainControl()` functions in the `caret` package. Compared to a random forest, a boosted model has a lot more parameters to tune and requires more coding effort. To begin with, we first change the two levels of the target variable `class`, `"0"` and `"1"`, to characters `"B"` (meaning "benign") and `"M"` (meaning "malignant"), respectively, via an `ifelse()` statement. This conversion is needed due to `caret`'s limitation when implementing the `xgboost` (standing for "extreme gradient boosting") package, which is for fitting boosted models. (If you ignore CHUNK 17 and run CHUNK 18 directly, you will run into an error!)

```
# CHUNK 17
# Rename the two levels of class
breast$class <- ifelse(breast$class == "0", "B", "M")
breast$class <- as.factor(breast$class)

# Reproduce the training and test sets with the "transformed" target
train <- breast[partition, ]
test <- breast[-partition, ]
```

```
# CHUNK 16
imp <- varImp(rf)
plot(imp, main = "Variable Importance of Classification Random Forest")
```

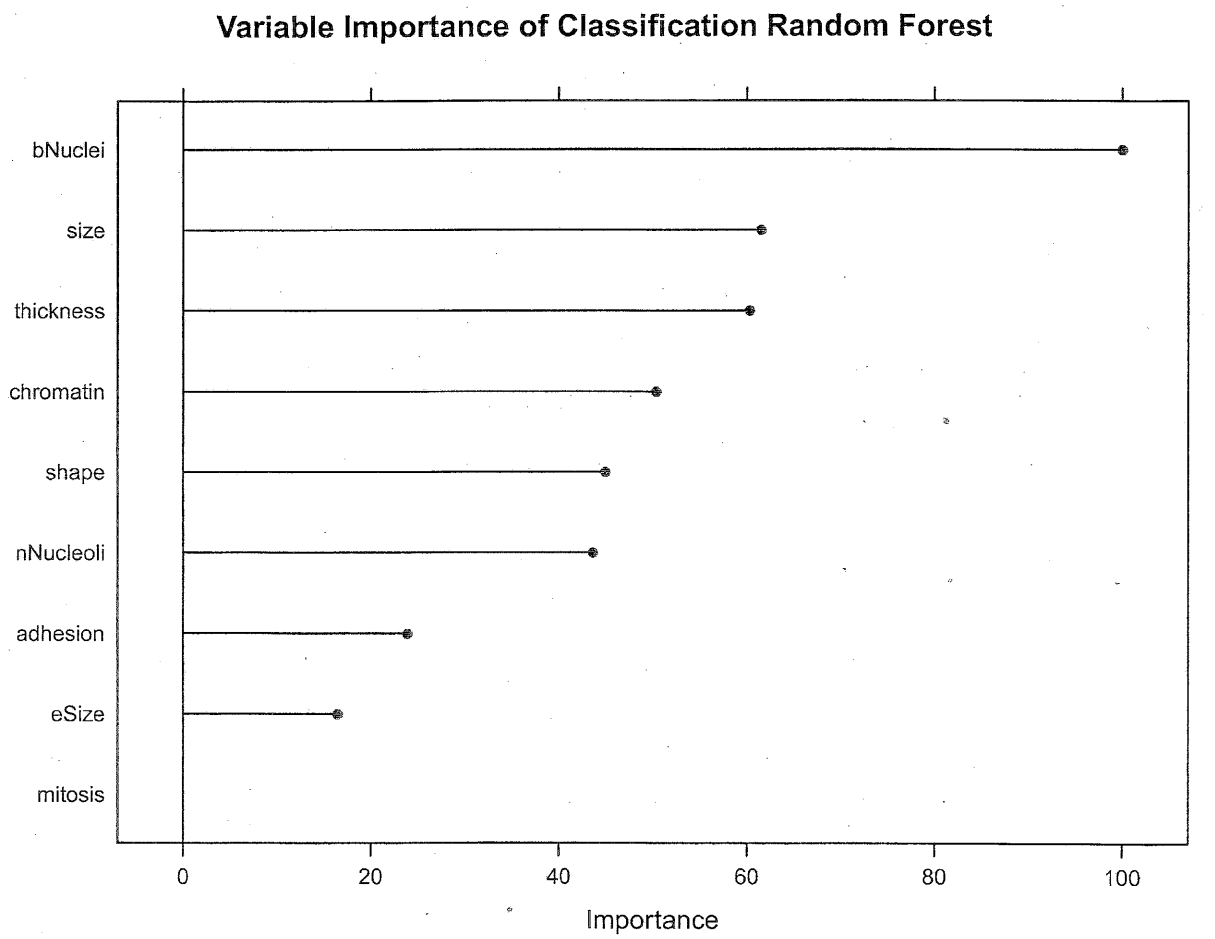


Figure 5.3.5: Variable importance plot for the classification random forest for the `breast` data.

Then we set up the grid of tuning parameters, as in the first part of CHUNK 18.

```
# CHUNK 18
xgb.grid <- expand.grid(max_depth = c(1, 3, 7),
                        min_child_weight = 1,
                        gamma = 0,
                        nrounds = 1000,
                        eta = c(0.01, 0.05, 0.1),
                        colsample_bytree = c(0.6, 0.9),
                        subsample = 0.6)
xgb.grid
```

```
##    max_depth min_child_weight gamma nrounds  eta colsample_bytree
## 1          1                1     0    1000 0.01              0.6
## 2          3                1     0    1000 0.01              0.6
## 3          7                1     0    1000 0.01              0.6
## 4          1                1     0    1000 0.05              0.6
## 5          3                1     0    1000 0.05              0.6
## 6          7                1     0    1000 0.05              0.6
## 7          1                1     0    1000 0.10              0.6
## 8          3                1     0    1000 0.10              0.6
## 9          7                1     0    1000 0.10              0.6
## 10         1                1     0    1000 0.01              0.9
## 11         3                1     0    1000 0.01              0.9
## 12         7                1     0    1000 0.01              0.9
## 13         1                1     0    1000 0.05              0.9
## 14         3                1     0    1000 0.05              0.9
## 15         7                1     0    1000 0.05              0.9
## 16         1                1     0    1000 0.10              0.9
## 17         3                1     0    1000 0.10              0.9
## 18         7                1     0    1000 0.10              0.9
##    subsample
## 1        0.6
## 2        0.6
## 3        0.6
## 4        0.6
## 5        0.6
## 6        0.6
## 7        0.6
## 8        0.6
## 9        0.6
## 10       0.6
## 11       0.6
## 12       0.6
## 13       0.6
## 14       0.6
## 15       0.6
```

```
## 16       0.6
## 17       0.6
## 18       0.6
```

Note that all of the seven parameters need to be specified; missing one of them will lead to an error. Parameters that we are interested in tuning (e.g., `max_depth`, `nrounds`, `eta`) are given a vector of possible values and those that are not of interest are assigned a single value. These parameters are described below: (On the exam, you can type `?xgboost` to learn more about these parameters. There is no need to memorize their definitions.)

- `max_depth`, `min_child_weight`, `gamma`: These parameters control the complexity of the underlying trees. At least one of these parameters should be tuned, but more can be tuned if computational resources allow.
- `colsample_bytree`, `subsample`: These two parameters determine the proportion of features and observations used in each individual tree, respectively. Typically, at least the proportion of features should be tuned.
- `nrounds`: This parameter controls the maximum number of rounds or iterations in the model fitting process. It is often set around 1,000, high enough so that sufficient trees are grown to produce a good fit. When a good fit has been achieved, the algorithm may stop early.
- `eta`: This is the learning rate parameter, a scalar multiple between 0 and 1 that applies to the contribution of each tree. The higher the learning rate, the faster the model will reach optimality and the fewer the number of iterations required, though the resulting model will more likely overfit. Typically, `eta` is set to values between 0.01 and 0.2.

The `expand.grid()` function creates a data frame called `xgb.grid` from all combinations of the possible values of the tuning parameters. Passing this grid and the control parameters to the `train()` function, we build the boosted model on the training set of the `breast` data in the remainder of CHUNK 18. (It may take a few minutes to complete the fitting. Be patient!)

```
# CHUNKN 18 (Cont.)
control <- trainControl(method = "cv",
                        number = 5,
                        sampling = "down")

xgb.tuned <- train(class ~ .,
                   data = train,
                   method = "xgbTree",
                   trControl = control,
                   tuneGrid = xgb.grid)
xgb.tuned

## eXtreme Gradient Boosting
##
## 479 samples
##   9 predictor
##   2 classes: 'B', 'M'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 383, 383, 383, 383, 384
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   eta   max_depth  colsample_bytree  Accuracy   Kappa
##   0.01  1          0.6               0.9707895  0.9361896
##   0.01  1          0.9               0.9686842  0.9318446
##   0.01  3          0.6               0.9707675  0.9362334
##   0.01  3          0.9               0.9686842  0.9318446
##   0.01  7          0.6               0.9728509  0.9406236
##   0.01  7          0.9               0.9707675  0.9363520
##   0.05  1          0.6               0.9603509  0.9139218
##   0.05  1          0.9               0.9666009  0.9268119
##   0.05  3          0.6               0.9707675  0.9358995
##   0.05  3          0.9               0.9686842  0.9310448
##   0.05  7          0.6               0.9645175  0.9225921
##   0.05  7          0.9               0.9728509  0.9406611
##   0.10  1          0.6               0.9603509  0.9128839
##   0.10  1          0.9               0.9582675  0.9087453
##   0.10  3          0.6               0.9707675  0.9364837
##   0.10  3          0.9               0.9645175  0.9229567
##   0.10  7          0.6               0.9624342  0.9179970
##   0.10  7          0.9               0.9707675  0.9366766
##
## Tuning parameter 'nrounds' was held constant at a value of 1000
##
## Tuning parameter 'min_child_weight' was held constant at a value of
##  1
## Tuning parameter 'subsample' was held constant at a value of 0.6
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 1000, max_depth =
##  7, eta = 0.01, gamma = 0, colsample_bytree = 0.6, min_child_weight =
##  1 and subsample = 0.6.
```

The optimal values of `max_depth`, `eta`, and `colsample_bytree` are 7, 0.01, and 0.6, respectively.

The test set performance of the boosted model is evaluated in CHUNK 19.[xiii] Note that in the `confusionMatrix()` function, we have specified that the positive class is `"M"` rather than `"1"`.

```
# CHUNK 19
pred.xgb.class <- predict(xgb.tuned, newdata = test)
confusionMatrix(pred.xgb.class, test$class, positive = "M")
```

[xiii] It appears that CHUNK 39 of Section 7.3 of the PA e-learning modules neglects to convert the predicted classes into predicted probabilities with the `type = "prob"` option.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   B   M
##          B 127   0
##          M   6  71
##
##                Accuracy : 0.9706
##                  95% CI : (0.9371, 0.9891)
##     No Information Rate : 0.652
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9364
##
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
##             Specificity : 0.9549
##          Pos Pred Value : 0.9221
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3480
##          Detection Rate : 0.3480
##    Detection Prevalence : 0.3775
##       Balanced Accuracy : 0.9774
##
##        'Positive' Class : M
##
```

```
pred.xgb.prob <- predict(xgb.tuned, newdata = test, type = "prob")[, 2]
roc(as.numeric(test$class), pred.xgb.prob, auc = TRUE)
```

```
##
## Call:
## roc.default(response = as.numeric(test$class), predictor = pred.xgb.prob,      auc = T
##
## Data: pred.xgb.prob in 133 controls (as.numeric(test$class) 1) < 71 cases (as.numeric
## Area under the curve: 0.9948
```

The accuracy of the boosted model on the test set is 0.9706, which is the same as that of the two base decision trees. However, all of the malignant cases are correctly classified to be malignant and the AUC rises to 0.9948, an indication that the boosted model is superior in terms of prediction performance.

**Ensemble trees vs. base trees: Which one to use here?** You probably have this feeling when fitting the random forest and boosted model: What do they look like and what are they really trying to do? This is typical of the "black box" nature of ensemble methods, which often fare well in terms of prediction accuracy, but their users tend to have little clue as to how they work and how to

interpret the model results. In this case study, the random forest and boosted model do result in an improvement in prediction accuracy, but the improvement appears to be incremental compared to base decision trees, which perform almost equally well but lend themselves to easy interpretation. Since one of the key considerations when selecting a predictive model in this case study is ease of interpretation, it seems judicious to recommend a base tree like Tree 2 to medical professionals.