MTH 3270 Notes 7

7 Statistical Foundations (Cont'd) (7, E.1, E.2, E.5)

7.8 Model Complexity and Overfitting

- We'll want to know how good a given model is:
  - How well does it **predict** the response variable in **original data set** (that was used to build the model)?

    Can use $\sqrt{\text{MSE}}$, R^2, etc.
  - How well does it **predict** the response for **new** observations (**not** part of the original data set)?

    Can use *cross-validation* (we'll see how later).
- A model ***overfits*** the **original data** if it predicts the *those* responses well, but *not* responses of **new** observations.

  Overfitting results when the ***complexity*** of the model is *too high*.

  (In the context of regression, **complexity** refers to the number of explanatory variables in the model.)
- For example, here are data on **lengths** and **weights** of **nine** snakes (different from the Class Notes 6 data):

```
Ln <- c(85.7, 64.5, 84.1, 82.5, 78.0, 81.3, 71.0, 86.7, 78.7)
Wt <- c(331.9, 121.5, 382.2, 287.3, 224.3, 245.2, 208.2, 393.4, 228.3)

snakes <- data.frame(Length = Ln, Weight = Wt)
```

```
g <- ggplot(snakes, aes(x = Length, y = Weight)) + geom_point()
g
```

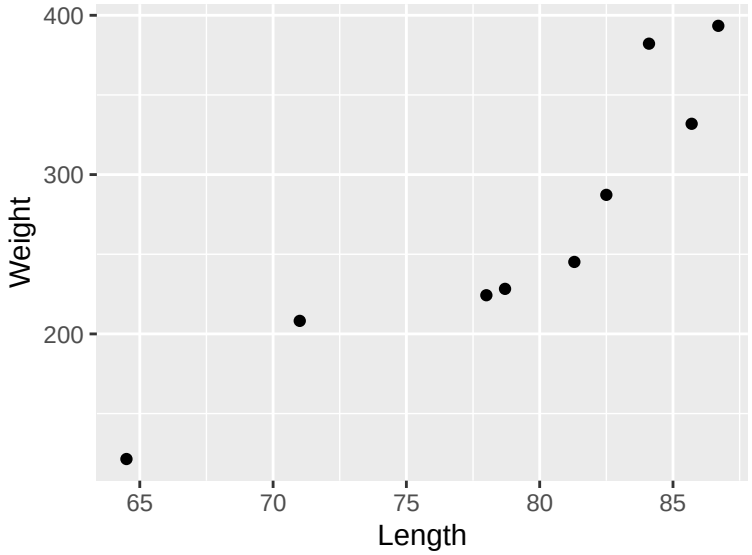


Figure 1

We fit each of these ***polynomial regression models*** to the data:

$$
\begin{aligned}
\text{Model 0}: \quad Y &= \hat{\beta}_0 \\
\text{Model 1}: \quad Y &= \hat{\beta}_0 + \hat{\beta}_1 X \\
\text{Model 2}: \quad Y &= \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 \\
\text{Model 3}: \quad Y &= \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 \\
\text{Model 4}: \quad Y &= \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 + \hat{\beta}_4 X^4 \\
\text{Model 5}: \quad Y &= \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2 + \hat{\beta}_3 X^3 + \hat{\beta}_4 X^4 + \hat{\beta}_5 X^5
\end{aligned}
$$

where Y is the **weight** and X the **length** of a snake.

```
g + stat_smooth(method = "lm", formula = y ~ 1, se = F) +
ggtitle(label = "Model 0")
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 1), se = F) +
ggtitle(label = "Model 1")
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 2), se = F) +
ggtitle(label = "Model 2")
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 3), se = F) +
ggtitle(label = "Model 3")
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 4), se = F) +
ggtitle(label = "Model 4")
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 5), se = F) +
ggtitle(label = "Model 5")
```

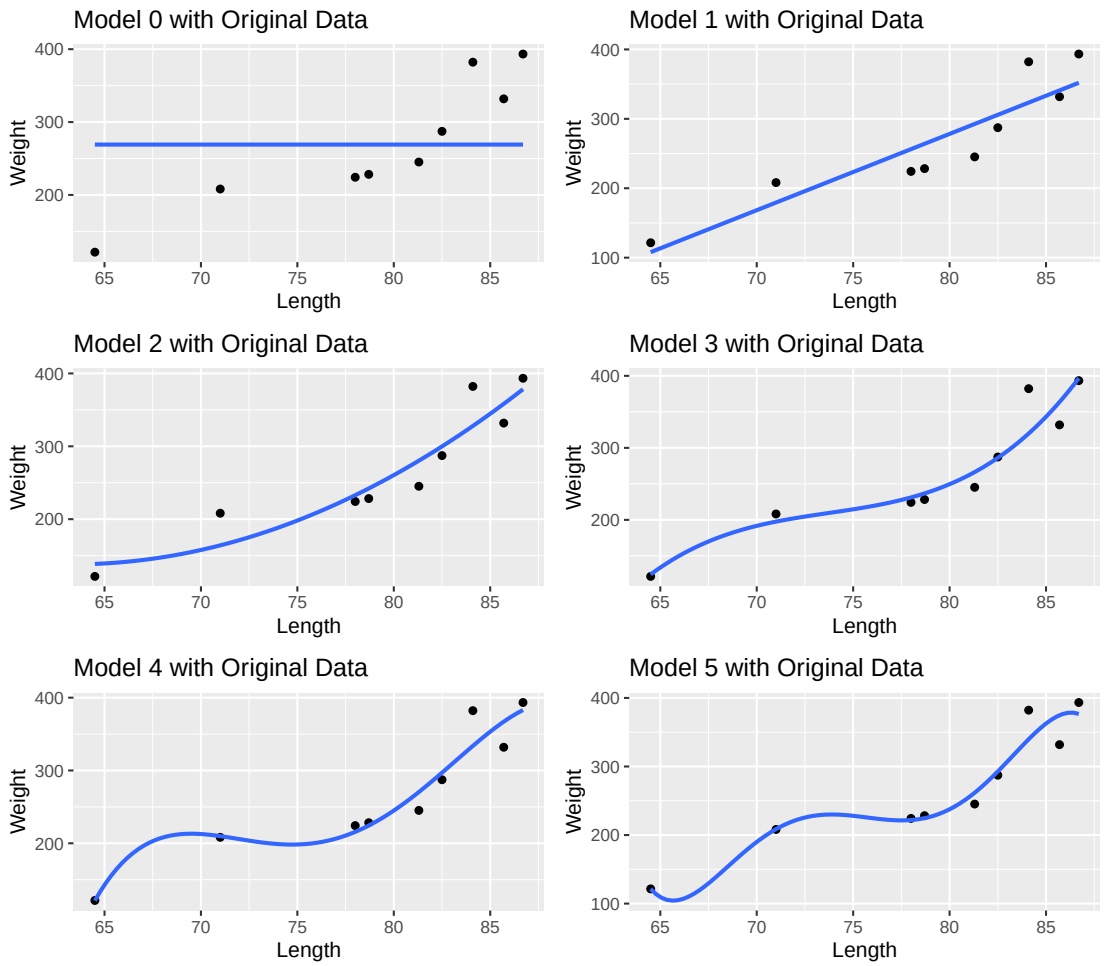


Figure 2

The models fit the **original** data progressively **better** as the model **complexity** gets **higher**, but they **don't necessarily** predict **new** observations better.

For example, here are five **new** snakes:

```
newSnakes <- data.frame(Length = c(67, 72, 77, 81, 86),
                        Weight = c(127.9, 153.7, 204.7, 300.6, 291.4))
newSnakes
```

```
##   Length Weight
## 1     67  127.9
## 2     72  153.7
## 3     77  204.7
## 4     81  300.6
## 5     86  291.4
```

The **fifth-degree** polynomial performs well in ***in-sample testing*** (i.e. it fits the **original data** well), but not so well in ***out-of-sample testing*** (i.e. it doesn't predict **new** observations very well).

The the **linear** model does better in **out-of-sample testing**. See Fig. 3.

```
g <- ggplot(snakes, aes(x = Length, y = Weight)) + geom_point(alpha = 0.05)
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 1), se = F) +
ggtitle(label = "Model 5 with New Snakes") +
geom_point(data = newSnakes)
```

```
g + stat_smooth(method = "lm", formula = y ~ poly(x, 5), se = F) +
ggtitle(label = "Model 5 with New Snakes") +
geom_point(data = newSnakes)
```

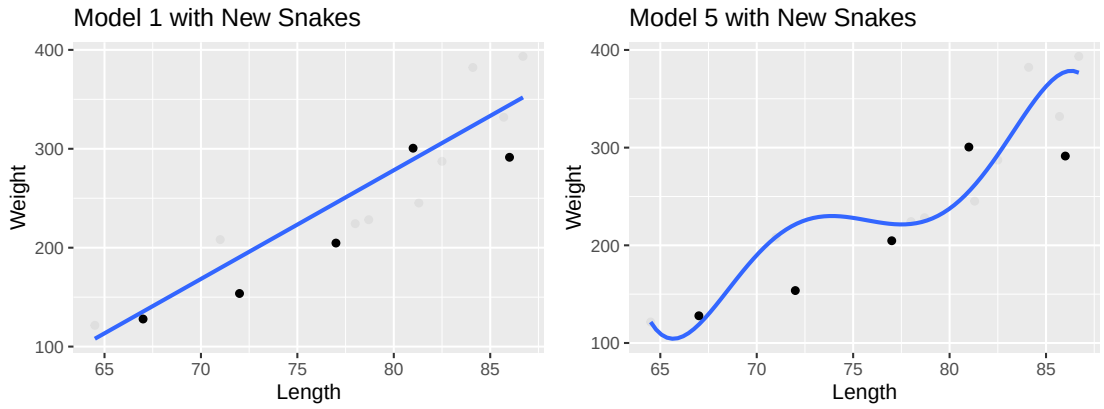


Figure 3

8 Statistical (Machine) Learning and Predictive Analytics (8)

8.1 Introduction to Machine Learning

- ***Machine learning*** is suite of methods for data-based **prediction** and **classifying** and **grouping observations**.

- There are two varieties of **machine learning**:

  1. ***Supervised learning***: Used for *predicting* values of a response variable from the values of explanatory variables. When the response variable is *categorical*, prediction is called ***classification***.

     For supervised learning, the data must contain values of **explanatory** variables *and* a **response** variable.

     (Regression is an example of supervised learning.)

  2. ***Unsupervised learning*** – The goal is *identifying groups* (i.e. *clusters*) of individuals based on the values of explanatory variables.

     For unsupervised learning, the data must contain values of **explanatory** variables, but *not* a response variable.

     (Cluster analysis is an example of unsupervised learning.)

8.2 Supervised Learning

- Supervised learning methods involve developing a **function** of the explanatory variables $X_1, X_2, \ldots, X_p$ that can be used to **predict** a response value Y.

8.2.1 Classifiers

- Recall that **classification** can be viewed as **predicting** a **categorical** response variable from the values of explanatory variables.
- We'll look at several **classifiers**:
  - Decision trees
  - Random forests
  - Nearest neighbor
  - Naive Bayes
  - Artificial Neural Networks

Decision Trees

- Suppose a **pet** is **9** inches tall and weighs **10** pounds. Can we **predict** whether it's a **dog** or **cat**?

  To make it easier, suppose also we have data on **weights** (pounds) and **heights** (inches) of **11 other pets**:

```
type <- c("dog", "dog", "cat", "dog", "cat", "dog", "cat", "dog", "cat",
          "dog", "cat", "dog", "dog", "cat", "dog", "cat", "dog", "cat", "dog")
wt <- c(8, 17, 8, 18, 7, 22, 6, 16, 7, 20, 10, 15, 14, 11, 13, 13, 15, 17, 10)
ht <- c(7.5, 10, 8, 15, 7, 15, 7, 13, 11, 16, 7, 10.5, 9, 9.5, 9, 8, 9, 8, 12)
pets <- data.frame(Type = type, Ht = ht, Wt = wt)
```

```
pets

##    Type   Ht Wt
## 1   dog  7.5  8
## 2   dog 10.0 17
## 3   cat  8.0  8
## 4   dog 15.0 18
## 5   cat  7.0  7
## 6   dog 15.0 22
## 7   cat  7.0  6
## 8   dog 13.0 16
## 9   cat 11.0  7
## 10  dog 16.0 20
## 11  cat  7.0 10
## 12  dog 10.5 15
## 13  dog  9.0 14
## 14  cat  9.5 11
## 15  dog  9.0 13
## 16  cat  8.0 13
## 17  dog  9.0 15
## 18  cat  8.0 17
## 19  dog 12.0 10
```

and we want to use the data to **predict** the **type** of the 9-inch, 10-pound animal. In other words, we want to **classify** the animal as either a dog or a cat.

(We'll return to this example a bit later.)

- Starting with the complete set of rows (or "records") of a data frame, a ***decision tree*** is produced by *recursively* using values of explanatory variables to **split** sets of rows (or "nodes") into smaller subsets (which become the new "nodes"), so that the subsets are "purer" with respect to the categorical response variable than the original sets.
- The `"rpart"` package has functions for developing **classification trees** in R. Among them are the following.

```
rpart()              # Recursive partitioning for decision trees (and
                     # regression trees).
predict.rpart()      # Predict the response variable of a new observation
                     # using a tree.  This is the "rpart" method for the
                     # generic predict() function.
plot.rpart()         # Plot a decision tree.  This is the "rpart" method
                     # for the generic plot() function.
text.rpart()         # Add text to the plot of a decision tree.  This is
                     # the "rpart" method for the generic text() function.
```

- The main arguments passed to `rpart()` are a *formula* indicating the response and explanatory variables and a *data frame* in which to find those variables.
- For example, using the `pets` data set:

```
library(rpart)
```

```
my.tree <- rpart(Type ~ Wt + Ht, data = pets,
                 control = rpart.control(minsplit = 7))
```

(The `rpart.control()` function is used, via the `control` argument, to set various parameters that control details of the decision tree `rpart()` fits to the data, in this case the minimum number of observations that must exist in a node in order for a split to be attempted.)

The object `my.tree` contains the results:

```
my.tree
```

```
## n= 19
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 19 8 dog (0.4210526 0.5789474)
##   2) Ht< 8.5 7 1 cat (0.8571429 0.1428571) *
##   3) Ht>=8.5 12 2 dog (0.1666667 0.8333333)
##     6) Wt< 12 3 1 cat (0.6666667 0.3333333) *
##     7) Wt>=12 9 0 dog (0.0000000 1.0000000) *
```

This tree uses two explanatory variables (`Wt` and `Ht`) to partition the `pets` data set into three parts (or ***terminal nodes***):

1. Pets shorter than 8.5 inches.
2. Pets taller than 8.5 inches but weighing less than 12 pounds.
3. Pets taller than 8.5 inches and weighing more than 12 pounds.

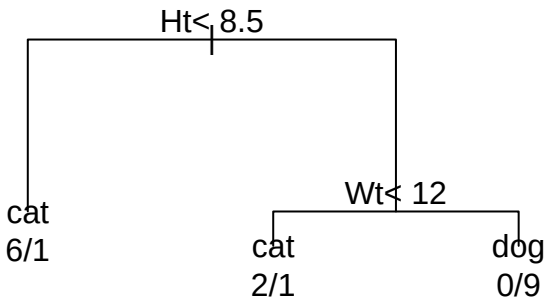
The object `my.tree` belongs to the `"rpart"` *class* of objects:

```
class(my.tree)
```

```
## [1] "rpart"
```

We can plot the tree using `plot()` and add text using `text()` (both *generic* functions that pass `my.tree` along to the `plot.rpart()` and `text.rpart()` *methods* from the `"rpart"` package):

```
par(xpd = TRUE)
plot(my.tree, compress = TRUE)
text(my.tree, use.n = TRUE)
```



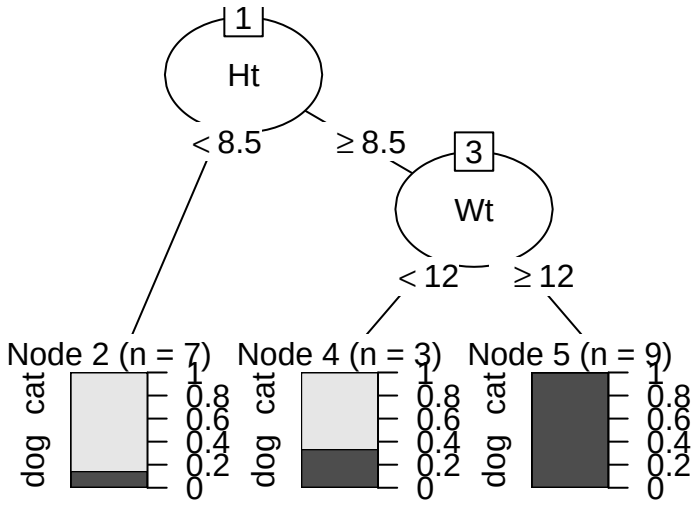

```
par(xpd = FALSE)          # Reset xpd to its default value.
```

(Setting the graphical parameter `xpd` to `TRUE` allows the tree diagram to extend outside the normal plot region.)

Another way to plot the tree is to use the `plot.party()` *method* from the `"partykit"` package (after converting `my.tree` to the `"party"` *class*):

```
library(partykit)
```

```
my.party.tree <- as.party(my.tree)
plot(my.party.tree)
```



In the plots, the numbers (or proportions) of **cats** and **dogs** are displayed at each **terminal node**.

The following sequence of scatterplots shows the **splits** used.

```
## Scatterplot of pets heights and weights:
g1 <- ggplot(data = pets, mapping = aes(x = Ht, y = Wt, color = Type)) +
  geom_point() +
  labs(title = "Wts and Hts of Pets")
g1
```

```
## Scatterplot with first split-line:
splitHt <- 8.5
g2 <- g1 + geom_vline(xintercept = splitHt, color = "dodgerblue", lty = 2)
g2
```

```
## Scatterplot with first and second split-lines:
splitWt <- 12
splitLines <- data.frame(x1 = splitHt, x2 = 16, y1 = splitWt, y2 = splitWt)
g3 <- g2 + geom_segment(data = splitLines,
                        mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
                        color = "rosybrown", lty = 2)
g3
```

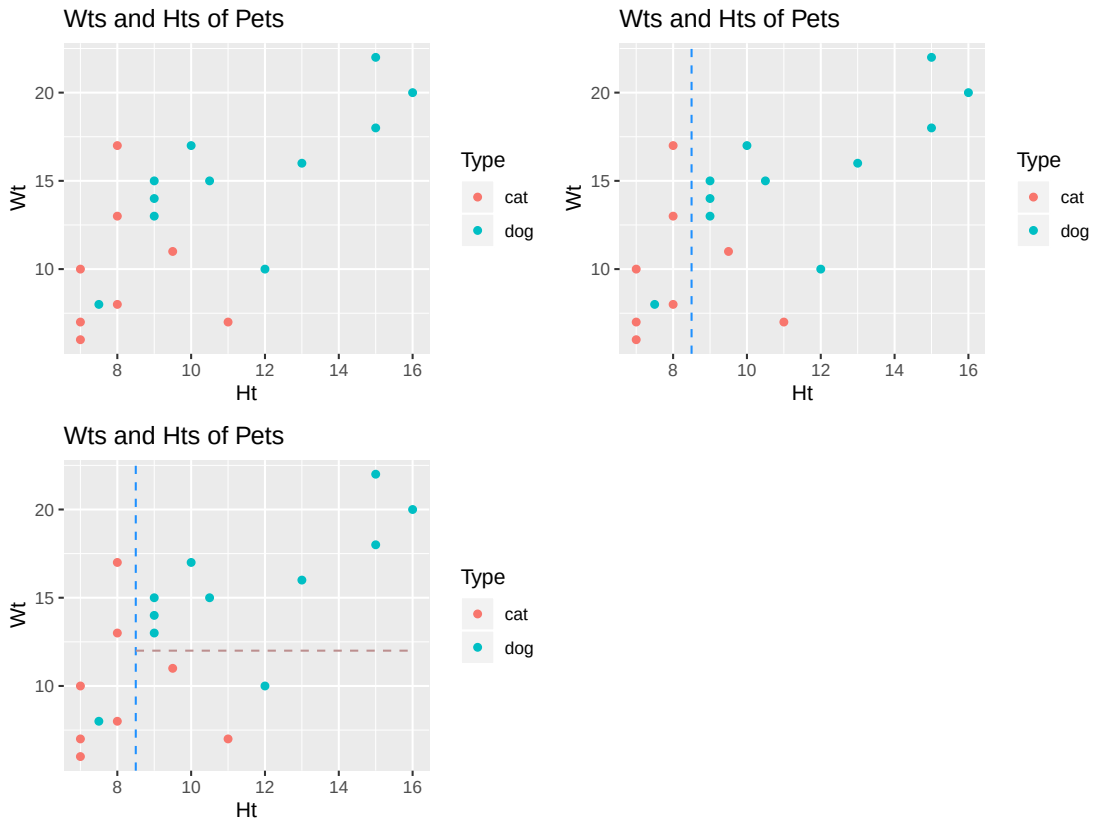


Figure 4

To get a more detailed summary of the fitted tree, we could use `summary()`:

```
summary(my.tree)
```

(Output not shown.)

- The `predict()` *generic* function passes the `"rpart"` object to the `predict.rpart()` *method*, which returns the tree's **predicted** type for each animal in `pets` (based on its `Ht` and `Wt`):

```
predType <- predict(my.tree, type = "class")
```

We can compare the **predictions** to the **actual** types from `pets`:

```
types <- data.frame(Actual = pets$Type, Predicted = predType)
types

##    Actual Predicted
## 1     dog       cat
```

```
## 2     dog       dog
## 3     cat       cat
## 4     dog       dog
## 5     cat       cat
## 6     dog       dog
## 7     cat       cat
## 8     dog       dog
## 9     cat       cat
## 10    dog       dog
## 11    cat       cat
## 12    dog       dog
## 13    dog       dog
## 14    cat       cat
## 15    dog       dog
## 16    cat       cat
## 17    dog       dog
## 18    cat       cat
## 19    dog       cat
```

The predicted types of the first and last animals in `pets` were wrong, but all others were right.

- The the so-called ***confusion matrix*** summarizes the correct/incorrect predictions:

```
confusion <- table(types)
confusion
```

```
##       Predicted
## Actual cat dog
##    cat   8   0
##    dog   2   9
```

The **accuracy** of the tree can be measured by it's ***correct classification rate***, which is **89.47%**:

$$\frac{8+9}{19} = 0.8947,$$

or by it's ***misclassification rate***, which is **10.53%**:

$$\frac{2+0}{19} = 0.1053.$$

In R, the **correct classification rate** is obtained via:

```
sum(diag(confusion)) / nrow(types)        # (8 + 9) / 19
```

```
## [1] 0.8947368
```

- At each step, for each subset of rows, all variables are tested for whether to split the set of rows on that variable.

  The **optimal split** value is the one for which the (average) **"purity"** of the resulting subsets of rows is highest.

  "Impurity" is measured by the ***Gini index***:

$$G = 1 - \sum_{i=1}^{k} p_i^2 \tag{1}$$

  where $\boldsymbol{k}$ is the number of categories of the response variable, and $\boldsymbol{p_i}$ is the proportion of rows, among the set under consideration for splitting, in the ith category of the response variable.

  Each p_i^2 is the *probability* that two randomly selected rows will *both* be in the *ith category* of the response. Their sum is the *probability* that they'll *both* be in the *same category* (one or another of the k categories).

  G takes values between 0 and 1, with **0** implying perfect **"purity"** (i.e. all the same category) and values closer to **1** implying more **"impurity"** (diversity).*

  * The maximum value of G is $1 - 1/k$.

- By default, in `rpart()` a split is only performed if it decreases the misclassification rate by at least 1%. This so-called ***tuning parameter*** can be adjusted (see Exercise 6). A *smaller* value of the **tuning parameter** results in a tree that *fits* the **original data** better (i.e. is more **complex**, having more **terminal nodes**), but risks **overfitting**.

Section 8.2 Exercises

Exercise 1 Refer to the **decision tree** based on the `pets` data (above). Answer the following questions *without* using `predict()`.

a) What type of animal (**cat** or **dog**) would you **predict** a **9**-inch, **10**-pound pet to be?

b) What type of animal (**cat** or **dog**) would you **predict** a **14**-inch, **21**-pound pet to be?

Exercise 2 The function `predict()` can be used with an object of class `"rpart"` to **predict** the categorical response of a new observation.

Create the `pets` data frame (above), then fit the **decision tree** using:

```
library(rpart)
```

```
my.tree <- rpart(Type ~ Wt + Ht, data = pets,
                 control = rpart.control(minsplit = 7))
```

Create a data frame with *new* height and weight values for which we want **predict** the pet type:

```
newPets <- data.frame(Ht = c(9, 14), Wt = c(10, 21))
newPets
```

```
##   Ht Wt
## 1  9 10
## 2 14 21
```

(The variable names need to be the same as in the data frame used to build the tree, `Ht` and `Wt` in this case.)

Now type:

```
predict(my.tree, newdata = newPets, type = "class")
```

a) What type of animal (**cat** or **dog**) would you **predict** the **9**-inch, **10**-pound pet to be?

b) What type would you **predict** the **14**-inch, **21**-pound pet to be?

c) Are your answers using `predict()` consistent with your answers to Exercise 1?

Exercise 3 Consider the (built-in) `iris` data set. Fit the following **decision tree** for predicting `Species` from `Petal.Length` and `Petal.Width`:

```
library(rpart)
```

```
my.tree <- rpart(Species ~ Petal.Length + Petal.Width, data = iris)
my.tree
```

a) How many **terminal nodes** does the tree have?

b) Now create the **confusion matrix**:

```
predSpecies <- predict(my.tree, type = "class")
species <- data.frame(Actual = iris$Species, Predicted = predSpecies)
species
```

```
confusion <- table(species)
confusion
```

What is the tree's **correct classification rate** (as a percent)? What is it's **misclassification rate** (as a percent)?

c) Look at the following two plots.

```
## First plot:
par(xpd = TRUE)
plot(my.tree, compress = TRUE)
text(my.tree, use.n = TRUE)
par(xpd = FALSE)
```

```
## Second plot:
splitPetal.Length <- 2.45
splitPetal.Width <- 1.75
splitLines <- data.frame(x1 = splitPetal.Length, x2 = 7,
                         y1 = splitPetal.Width, y2 = splitPetal.Width)
g <- ggplot(data = iris,
            mapping = aes(x = Petal.Length, y = Petal.Width,
            color = Species)) +
     geom_point() +
     labs(title = "Widths and Lengths of Petals") +
     geom_vline(xintercept = splitPetal.Length,
                      color = "dodgerblue",
                      lty = 2) +
     geom_segment(data = splitLines,
                      mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
                      color = "rosybrown", lty = 2)
g
```

Use the plots (*not* `predict()`) to **predict** the `Species` of the following flowers:

A flower whose `Petal.Length` is **3.0 cm** and whose `Petal.Width` is **1.5 cm**.

A flower whose `Petal.Length` is **4.0 cm** and whose `Petal.Width` is **2.1 cm**.

d) Create the following data frame of two *new* iris flowers:

```
newIris <- data.frame(Petal.Length = c(3.0, 4.0),
                      Petal.Width = c(1.5, 2.1))
newIris
```

```
##   Petal.Length Petal.Width
## 1            3         1.5
## 2            4         2.1
```

Use `predict()`, with `my.tree`, to **predict** the species of the flowers in the `newIris` data set. Report the **two predictions** and compare them to those of Part *e*.

Exercise 4 Consider again the (built-in) `iris` data set. This time fit the **decision tree** for predicting `Species` from `Sepal.Length` and `Sepal.Width`:

```
my.tree <- rpart(Species ~ Sepal.Length + Sepal.Width, data = iris)
my.tree
```

a) How many **terminal nodes** does this tree have?

b) Now create the **confusion matrix**:

```
predSpecies <- predict(my.tree, type = "class")
species <- data.frame(Actual = iris$Species, Predicted = predSpecies)
species
```

```
confusion <- table(species)
confusion
```

What is the tree's **correct classification rate** (as a percent)? What is it's **misclassification rate** (as a percent)?

c) Look at the following two plots.

```
## First plot:
par(xpd = TRUE)
plot(my.tree, compress = TRUE)
text(my.tree, use.n = TRUE)
par(xpd = FALSE)
```

```
## Second plot:
split1Sepal.Length <- 5.45
split1Sepal.Width <- 2.8
splitLines1 <- data.frame(x1 = 3, x2 = split1Sepal.Length,
                          y1 = split1Sepal.Width, y2 = split1Sepal.Width)
split2Sepal.Length <- 6.15
split2Sepal.Width <- 3.1
splitLines2 <- data.frame(x1 = split1Sepal.Length, x2 = split2Sepal.Length,
                          y1 = split2Sepal.Width, y2 = split2Sepal.Width)

g <- ggplot(data = iris,
            mapping = aes(x = Sepal.Length, y = Sepal.Width,
            color = Species)) +
     geom_point() +
     labs(title = "Widths and Lengths of Sepals") +
     geom_vline(xintercept = split1Sepal.Length,
                color = "dodgerblue",
                lty = 2)  +
     geom_vline(xintercept = split2Sepal.Length,
                color = "purple",
                lty = 2)  +
     geom_segment(data = splitLines1,
                  mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
                  color = "brown", lty = 2) +
     geom_segment(data = splitLines2,
                  mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
                  color = "red", lty = 2)
g
```

Use the plots (*not* `predict()`) to **predict** the `Species` of the following flowers:

A flower whose `Sepal.Length` is **6.0 cm** and whose `Sepal.Width` is **3.5 cm**?

A flower whose `Sepal.Length` is **7.0 cm** and whose `Sepal.Width` is **3.0 cm**?

d) Create the following data frame of two *new* iris flowers:

```
newIris <- data.frame(Sepal.Length = c(6.0, 7.0),
                      Sepal.Width = c(3.5, 3.0))
newIris
```

```
##   Sepal.Length Sepal.Width
## 1            6         3.5
## 2            7         3.0
```

Use `predict()`, with `my.tree`, to **predict** the species of the flowers in the `newIris` data set. Report the **two predictions** and compare them to those of Part *e*.

Exercise 5 The **Gini index** takes values between 0 and 1, with 0 implying perfect "purity" (i.e. all individuals belonging to the same category) and larger values implying less "purity" (i.e. more diversity).

a) Guess which of the two data sets, `y1` and `y2`, is "purer" according to the **Gini index**, then check your answer by computing G using expression 1 with the proportions p_1, p_2, and p_3 shown below.

```
y1 <- c("A", "B", "C", "C", "C", "C", "C", "C", "C", "C", "C", "C")
round(prop.table(table(y1)), digits = 1)

## y1
##   A   B   C
## 0.1 0.1 0.8

y2 <- c("A", "B", "C", "C", "A", "C", "B", "A", "A", "B", "C", "B")
round(prop.table(table(y2)), digits = 1)

## y2
##   A   B   C
## 0.3 0.3 0.3
```

b) What is the **Gini index** value for the following data set?

```
y <- c("A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A")
round(prop.table(table(y)), digits = 1)

## y
## A
## 1
```

Exercise 6 In constructing a tree, optimal split values are chosen so that the (average) **"purity"** of the resulting subsets of rows is highest, i.e. so that the (average) **Gini index** is *lowest*.

But in `rpart()`, by default the split is only performed if it decreases the **misclassification rate** by at least **1%**.

This so-called **tuning parameter** can be adjusted (see pg 180 of the textbook).

A *smaller* value of this **tuning parameter** results in a tree that *fits* the **original data** better (i.e. is more **complex**, having more **terminal nodes**), but risks **overfitting**.

Here's how to *lower* the tuning parameter to **0.2%** using the `control` argument:

```
my.tree <- rpart(Species ~ Sepal.Length + Sepal.Width, data = iris,
                 control = rpart.control(cp = 0.002))
my.tree
```

```
par(xpd = TRUE)
plot(my.tree, compress = TRUE)
text(my.tree, use.n = TRUE)
par(xpd = FALSE)
```

Here's how to *raise* it to **5%**:

```
my.tree <- rpart(Species ~ Sepal.Length + Sepal.Width, data = iris,
    control = rpart.control(cp = 0.05))
my.tree
```

```
par(xpd = TRUE)
plot(my.tree, compress = TRUE)
text(my.tree, use.n = TRUE)
par(xpd = FALSE)
```

Which threshold value, 0.2% or 5%, resulted in more **terminal nodes** in the tree?

Random Forests and Bagging

- A ***random forest*** is a collection of **bootstrapped decision trees**. Predictions of the response variable (classification) for new observations are based on **majority rule** (i.e. a "vote") of the trees.

  Each tree is based on a **bootstrap sample** of **observations** (rows) from the original data frame. Furthermore:

  - ***Bagging*** ("bootstrap aggregating") is when each **bootstrapped tree** is based on **all** the **variables** (columns) in the original data frame.
  - ***Random forest*** is when each **bootstrapped tree** is based on its own **random sample** of **variables** (columns) from the original data frame.*

  *A separate random sample of variables is taken for consideration to be split upon at each split (node).

- The following function, from the `"randomForest"` package, will carry out the procedures.

  ```
  randomForest()      # Carry out a random forest procedure on the data
                      # in a data frame.
  ```

  To construct a random forest:

  1. Choose `ntree`, the desired number of **trees** to make (e.g. 500), and choose `mtry`, the desired number of variables (columns) to use in each tree.
  2. Randomly select n rows from the data frame *with replacement* (where n is the number of rows in the original data frame), i.e. take a *bootstrap* sample of rows.

3. Randomly select `mtry` variables (columns) *without replacement*.
4. Build a tree on the resulting randomly selected data set.
5. Repeat this procedure `ntree` times.

A **prediction** (classification) for a new observation is made by **majority rule** (i.e. a "vote") of the predictions of all `ntree` trees in the forest.

Using `mtry = ncol(x)`, where `x` is the original data frame, gives **bagging**.

- For example, using the `iris` data set:

```
library(randomForest)
```

```
my.forest <- randomForest(Species ~ Sepal.Length + Sepal.Width +
                                    Petal.Length + Petal.Width,
                          data = iris,
                          ntree = 500,
                          mtry = 2)
my.forest

##
## Call:
##  randomForest(formula = Species ~ Sepal.Length + Sepal.Width +      Petal.Length + Petal.
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 4%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

- The class of `my.forest` is `"randomForest"` (and also `"randomForest.formula"`):

```
class(my.forest)

## [1] "randomForest.formula" "randomForest"
```

Objects in this class are actually just a *lists*:

```
is.list(my.forest)

## [1] TRUE
```

To see the names of the objects stored in `my.forest`, type:

```
names(my.forest)
```

(Output not shown.)

- The **confusion matrix** is obtained via:

```
my.forest$confusion

##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

and the **correct classification rate** via:

```
sum(diag(my.forest$confusion)) / nrow(iris)

## [1] 0.96
```

The procedure has a **96% correct classification rate** for predicting an iris `Species`.

- The **importance** of each explanatory variable in **predicting** the iris `Species` is obtained by typing:

```
importance(my.forest)

##              MeanDecreaseGini
## Sepal.Length        10.075713
## Sepal.Width          2.022187
## Petal.Length        43.162783
## Petal.Width         43.963557
```

`Petal.Width` is the *most* important for predicting `Species`, followed closely by `Petal.Length`, `Sepal.Length`, and `Sepal.Width` (*least* important).

This is analogous to using p-values in multiple regression for deciding which explanatory variables are most important.

Section 8.2 Exercises

Exercise 7 Recall that the argument `mtry` in `randomForest()` is the number of variables (columns) randomly sampled as candidates at each split. It can be thought of as a **tuning parameter** – larger `mtry` values produce trees that conform to the data better but risk **overfitting**.

One measure of how well the trees fit the data is the **correct classification rate**. Com-

pute the **correct classification rate** for each of the following **random forests** for predicting `Species` using the `iris` data:

a) Using `mtry = 1` in `randomForest()`:

```
my.forest <- randomForest(Species ~ Sepal.Length + Sepal.Width +
                                    Petal.Length + Petal.Width,
                          data = iris,
                          ntree = 500,
                          mtry = 1)
```

b) Using `mtry = 3` in `randomForest()`:

```
my.forest <- randomForest(Species ~ Sepal.Length + Sepal.Width +
                                    Petal.Length + Petal.Width,
                          data = iris,
                          ntree = 500,
                          mtry = 3)
```

Exercise 8 When **all** of the variables (columns) in a data set are used in each *bootstrapped* tree, the procedure is called ***bagging***.

Carry out the **bagging** procedure for predicting `Species` using the `iris` data by setting `mtry = 4` in `randomForest()`:

```
my.forest <- randomForest(Species ~ Sepal.Length + Sepal.Width +
                                    Petal.Length + Petal.Width,
                          data = iris,
                          ntree = 500,
                          mtry = 4)
```

a) Then look at the **importance** of each explanatory variable:

```
importance(my.forest)
```

Which of the four explanatory variables is *most* important for predicting `Species`? Which is *least* important?

b) There is a `predict()` *method* for the `"randomForest"` class of objects called `predict.randomForest()`.

Create the following data frame of three *new* iris flowers:

```
newIris <- data.frame(Petal.Length = c(3.0, 2.2, 2.7),
                      Petal.Width = c(1.2, 2.1, 1.6),
                      Sepal.Length = c(5.5, 5.1, 5.9),
                      Sepal.Width = c(3.0, 2.7, 3.2))
newIris
```

```
##   Petal.Length Petal.Width Sepal.Length Sepal.Width
## 1          3.0         1.2          5.5         3.0
## 2          2.2         2.1          5.1         2.7
## 3          2.7         1.6          5.9         3.2
```

Use `predict()`, with `my.forest`, to **predict** the species of the three flowers in the `newIris` data set:

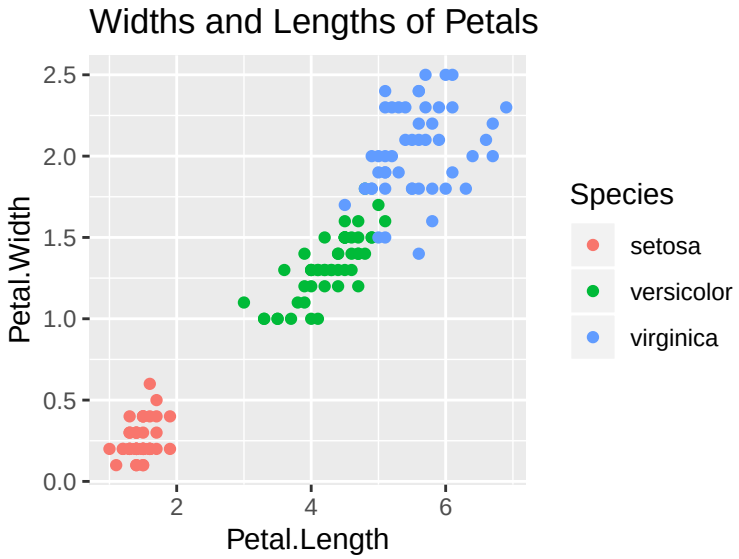
```
predict(my.forest, newdata = newIris, type = "class")
```

Report the **three species predictions**.

Nearest Neighbor

- We can think of an observation (row) with ***p* explanatory variables (*X*'s)** as a point in p-dimensional space. For example, a flower with given `Petal.Length` and `Petal.Width` is a point in 2-dimensional space:

```
ggplot(data = iris,
       mapping = aes(x = Petal.Length, y = Petal.Width,
       color = Species)) +
  geom_point() +
  labs(title = "Widths and Lengths of Petals")
```



Likewise, a flower with given `Petal.Length`, `Petal.Width`, `Sepal.Length`, and `Sepal.Width` is a point in 4-dimensional space.

The (Euclidean) **distance** between any two observations in p-dimensional space can be determined. For example, in 2-dimensional space, the **distance** between (x_1, y_1) and (x_2, y_2) is

$$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

(the *Pythagorean Theorem*), and in 3-dimensional space, the **distance** between (x_1, y_1, z_1) and (x_2, y_2, z_2) is

$$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

- A ***k nearest neighbor*** procedure for **predicting** the (categorical) response for a new observation (i.e. for **classifying** the observation) is based on the **majority rule** (i.e. a "vote") of the ***k* nearest observations** (in Euclidean distance) to the new observation.

For example, a *new* flower with **petal length 4.35 cm** and **petal width 1.65 cm** is shown in Fig. 5 (black dot).

```
newIris <- data.frame(Petal.Length = 4.35,
                      Petal.Width = 1.65)
```

```
g <- ggplot(data = iris,
            mapping = aes(x = Petal.Length, y = Petal.Width)) +
       geom_point(mapping = aes(color = Species)) +
       labs(title = "Widths and Lengths of Petals") +
       geom_point(data = newIris, color = "black")
g
```

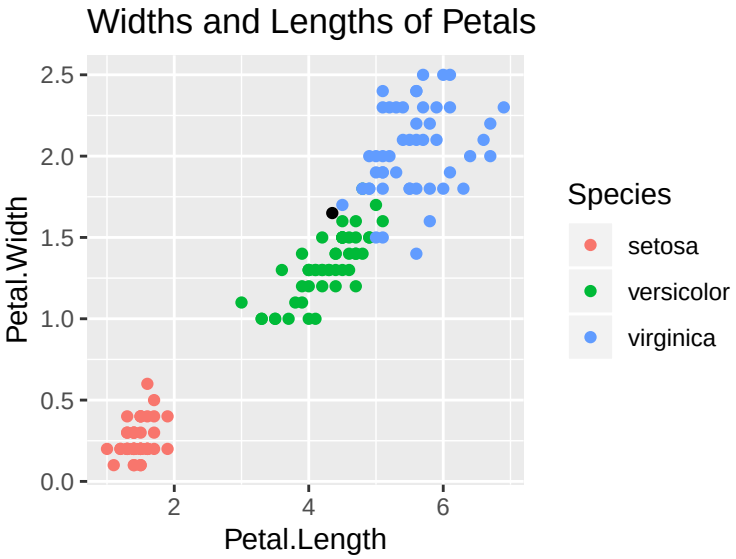


Figure 5

Of the $\boldsymbol{k = 3}$ nearest flowers (shown in the circle in Fig. 6), two are Versicolor, one is Setosa, and none are Virginica.

By a **majority rule** ("vote"), the *new* flower is **predicted** to be **Versicolor**.

```
g + geom_point(data = newIris, size = 9.1, shape = 1, color = "gold4")
```

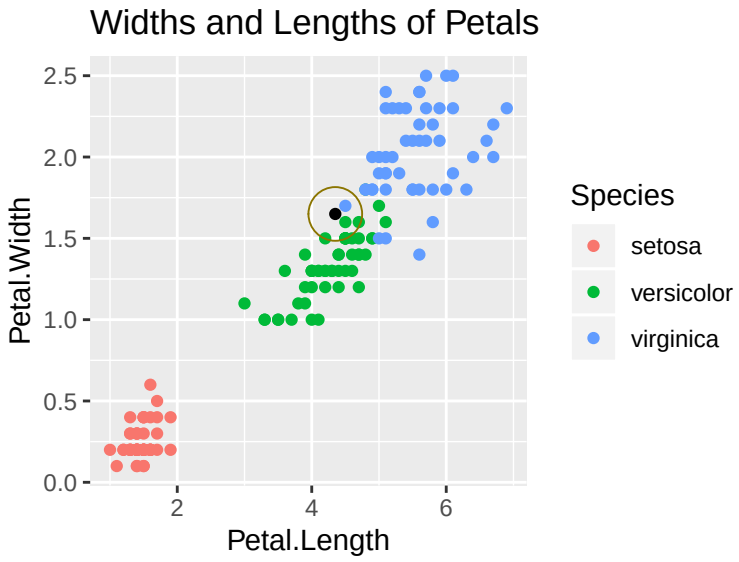


Figure 6

- The following function, from the `"class"` package, will carry out the ***k* nearest neighbor** classification procedure.

```
knn()     # Carry out a k nearest neighbor procedure on the data
          # in a data frame.
```

To carry out ***k* nearest neighbor** for predicting the (categorical) response for a **new observation** (i.e. for classifying that observation):

1. Choose k.
2. Find the k observations in the data set that are closest to the new observation (in p-dimensional space, where $\boldsymbol{p}$ is the number of explanatory variables).
3. **Predict** for the new observation the (categorical) response value shared by the **majority** (or plurality) of its k nearest neighbors.

The number of neighbors, $\boldsymbol{k}$, is a **tuning parameter**. A smaller value of k leads to the procedure conforming more closely to the data, but runs the risk of **overfitting**.

- For example, using the data set `iris`:

```
library(class)
library(dplyr)
```

```
# Distance can only be computed for numerical explanatory variables,
# so remove Species. Also, for this example, remove Sepal.Length
# and Sepal.Width:
train_iris <- select(iris, -c(Species, Sepal.Length, Sepal.Width))

# Data frame containing new flower for prediction:
newIris <- data.frame(Petal.Length = 4.35,
                      Petal.Width = 1.65)

# k nearest neighbor classification procedure:
my.knn <- knn(train = train_iris, test = newIris, cl = iris$Species, k = 3)
my.knn

## [1] versicolor
## Levels: setosa versicolor virginica
```

As expected (from Fig. 6), the **4.35 cm** long, **1.65 cm** wide flower is **predicted** to be **Versicolor**.

- We can determine the **correct classification rate** for **prediction** of `Species` for flowers in the *original data set* (`train_iris` from above) by typing:

```
my.knn <- knn(train = train_iris, test = train_iris, cl = iris$Species, k = 3)

# Save actual and predicted species:
species <- data.frame(Actual =  iris$Species, Predicted = my.knn)
head(species)

##   Actual Predicted
## 1 setosa    setosa
## 2 setosa    setosa
## 3 setosa    setosa
## 4 setosa    setosa
## 5 setosa    setosa
## 6 setosa    setosa
```

```
# Obtain correct classification rate:
confusion <- table(species)
confusion

##             Predicted
## Actual       setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          2        48

sum(diag(confusion)) / nrow(iris)

## [1] 0.9733333
```

Thus the **correct classification rate** is **97.3%**.

Section 8.2 Exercises

Exercise 9 The number of neighbors ***k*** is a **tuning parameter** in the **nearest neighbor** procedure. Using $k = 1$ predicts a new observation to be the same class as its (one) nearest neighbor. A *larger* k uses more data in each prediction.

Using $k = 1$ leads to a **100% correct classification rate** for the observations in the *original data set*, but won't predict *new* observations very well due to **overfitting**. Up to a point, a *larger* k will have a **lower** correct classification rate (for the *original data*), but will predict *new* observations **better**. Beyond that point, the predictive accuracy for new observations deteriorates with larger k values.

(The optimal k can be determined using ***cross-validation***.)

a) Experiment with a few different values of k by editing the code below. **Report** the **correct classification rate** (for the observations in the original data set) for the different values of k you chose.

```
library(class)
library(dplyr)

# Change this value to try different k, then run the code
# below for each choice of k:
ktry <- 3

# Distance can only be computed for numerical explanatory variables,
# so remove Species. Also, for this example, remove Sepal.Length
# and Sepal.Width:
train_iris <- select(iris, -c(Species, Sepal.Length, Sepal.Width))

# k nearest neighbor classification procedure:
my.knn <- knn(train = train_iris,
              test = train_iris,
              cl = iris$Species,
              k = ktry)
```

```
# Save actual and predicted species:
species <- data.frame(Actual =  iris$Species, Predicted = my.knn)

# Obtain correct classification rate:
confusion <- table(species)
confusion
sum(diag(confusion)) / nrow(iris)
```

b) Do your **predictions** for the **4.35 cm** long, **1.65 cm** wide flower change with the choice of k? Edit the code below to find out.

```r
# Change this value to try different k, then run the code
# below for each choice of k:
ktry <- 3

# Data frame containing new flower for prediction:
newIris <- data.frame(Petal.Length = 4.35,
                      Petal.Width = 1.65)

# k nearest neighbor classification procedure:
my.knn <- knn(train = train_iris,
              test = newIris,
              cl = iris$Species,
              k = ktry)
my.knn
```