Chapter 9
Decision Tree Divide and Conquer Classification

When classification needs to be apparent, kNN or naive Bayes we presented earlier may not be useful as they do not generate explicit classification rules. In some cases, we need to specify well stated rules for our decisions, just like a scoring criterion for driving ability or credit scoring for loan underwriting. The decisions in many situations actually require having a clear and easily understandable decision tree to follow the classification process start to finish.

In this chapter, we will (1) see a simple motivational example of decision trees based on the Iris data; (2) describe decision-tree divide and conquer methods; (3) examine certain measures quantifying classification accuracy; (4) show strategies for pruning decision trees; (5) work through a Quality of Life in Chronic Disease case-study; and (6) review the *One Rule* and *RIPPER* algorithms.

9.1 Motivation

Decision tree learners enable classification via tree structures modeling the relationships among all features and potential outcomes in the data. All decision trees begin with a trunk (all data are part of the same cohort), which is then split into narrower and narrower branches by forking decisions based on the intrinsic data structure. At each step, splitting the data into branches may include binary or multinomial classification. The final decision is obtained when the tree branching process terminates. The terminal (leaf) nodes represent the action to be taken as the result of the series of branching decisions. For predictive models, the leaf nodes provide the expected forecasting results given the series of events in the tree.

There are a number of `R` packages available for decision tree classification including `rpart`, `C5.0`, `party`, etc.

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_9

9.2 Hands-on Example: Iris Data

Let's start by seeing a simple example using the Iris dataset, which we saw in Chap. 3. The data features or attributes include `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`, and classes are represented by the `Species taxa` (setosa; versicolor; and virginica).

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1
1 1 1 1 ...

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

##
##     setosa versicolor  virginica
##         50         50         50
```

The `ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris)` function will build a decision tree (Figs. 9.1 and 9.2).

```
iris_ctree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Pe
tal.Width, data=iris)
print(iris_ctree)

##   Conditional inference tree with 4 terminal nodes
##
## Response:  Species
## Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
## Number of observations:  150
##
## 1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
##   2)*  weights = 50
## 1) Petal.Length > 1.9
##   3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894
##     4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865
##       5)*  weights = 46
##     4) Petal.Length > 4.8
##       6)*  weights = 8
##   3) Petal.Width > 1.7
##     7)*  weights = 46

plot(iris_ctree, cex=2)
```

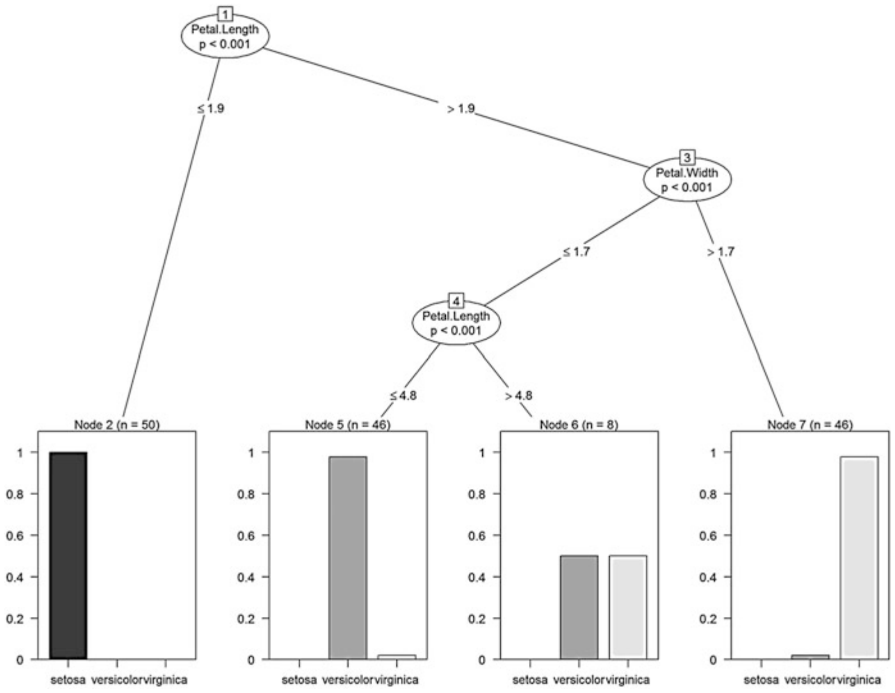

Fig. 9.1 Decision tree classification illustrating four leaf node labels corresponding to the three iris genera

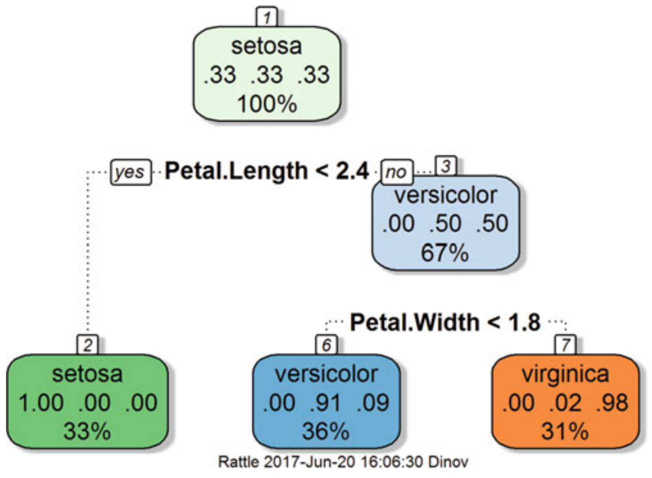

Fig. 9.2 An alternative decision tree classification of the iris flowers dataset

```
head(iris); tail(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

Similarly, we can demonstrate a classification of the *iris taxa* via `rpart`:

```
library(rpart)
iris_rpart = rpart(Species~., data=iris)
print(iris_rpart)

## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 50   0 setosa (1.000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 100  50 versicolor (0.000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 54  5 versicolor (0.000 0.90740741 0.09259259) *
##     7) Petal.Width>=1.75 46   1 virginica (0.000 0.02173913 0.97826087) *

# Use the `rattle::fancyRpartPlot` to generates an elegant plot
library(rattle)

## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

fancyRpartPlot(iris_rpart, cex = 1.5)
```

9.3 Decision Tree Overview

The decision tree algorithm represents an upside down tree with lots of tree branch bifurcations where a series of logical decisions are encoded as tree node splits. The classification begins at the root node and goes through many branches until it gets to the terminal nodes. This iterative process splits the data into different classes by rigid criteria.

9.3.1 Divide and Conquer

Decision trees involve recursive partitioning that uses data features and attributes to split the data into groups (nodes) of similar classes.

To make classification trees using data features, we need to observe the pattern between the data features and potential classes using training data. We can draw scatter plots and separate groups that are clearly clotted together. Each group is considered a segment of the data. After getting the approximate range of each feature value under each group, we can make the decision tree.

$$X = [X_1, X_2, X_3, \ldots, X_k] = \underbrace{\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} \end{pmatrix}}_{\text{features/attributes}} \left.\vphantom{\begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}}\right\} \textit{cases}$$

The decision tree algorithms use a top-down recursive divide-and-conquer approach (sometimes they may also use bottom up or mixed splitting strategies) to divide and evaluate the splits of a dataset D (input). The best split decision corresponds to the split with the **highest information gain**, reflecting a partition of the data into K subsets (using divide-and-conquer). The iterative algorithm terminates when some stopping criteria are reached. Examples of stopping conditions used to terminate the recursive process include:

- All the samples belong to the same class, that is they have the same label and the sample is already `pure`.
- Stop when majority of the points are already of the same class (relative to some error threshold).
- There are no remaining attributes on which the samples may be further partitioned.

One objective criteria for splitting or clustering data into groups is based on the **information gain measure**, or **impurity reduction**, which can be used to select the test attribute at each node in the decision tree. The attribute with the highest information gain (i.e., greatest entropy reduction) is selected as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions. There are three main indices to evaluate the impurity reduction: **Misclassification error**, **Gini index** and **Entropy**.

For a given table containing pairs of attributes and their class labels, we can assess the homology of the classes in the table. A table is pure (homogenous) if it only contains a single class. If a data table contains several classes, then we say that the table is impure or heterogeneous. This degree of impurity or heterogeneity can be quantitatively evaluated using impurity measures like entropy, Gini index, and misclassification error.

9.3.2 Entropy

The `Entropy` is an information measure of the amount of disorder or uncertainty in a system. Suppose we have a data set $D = (X_1, X_2, \ldots, X_n)$ that includes n features (variables) and suppose each of these features can take on any of k possible values (states). Then the cardinality of the entire system is k^n as each of the features are assumed to have k independent states, thus the total number of different datasets that can be expected is $\underbrace{k \times k \times \ldots \times k}_{n} = k^n$. Suppose p_1, p_2, …, p_n represent the proportions of each class (note: $\sum_i p_i = 1$) present in the child node that results from a split in a decision tree classifier. Then the entropy measure is defined by:

$$Entropy(D) = -\sum_i p_i \log_2 p_i.$$

If each of the $1 \le i \le k$ states for each feature is equally likely to be observed with probability $p_i = \frac{1}{k}$, then the entropy is maximized:

$$Entropy(D) = -\sum_{i=1}^{k} \frac{1}{k} \log \frac{1}{k} = \sum_{i=1}^{k} \frac{1}{k} \log k = \frac{1}{k} \sum_{i=1}^{k} 1 = 1.$$

In the other extreme, the entropy is minimized. Note that by L'Hopital's Rule $\lim_{x \to 0} x \times log(x) = \lim_{x \to 0} \frac{\frac{1}{x}}{-\frac{1}{x^2}} = \lim_{x \to 0} x = 0$) for a single class classification where the probability of one class is unitary ($p_{i_o} = 1$) and the other ones are trivial ($p_{i \neq i_o} = 0$):

$$Entropy(D) = -\sum_i \frac{1}{k} log\left(\frac{1}{k}\right) = p_{i_o} \times log\left(p_{i_o}\right) + \sum_{i \neq i_o} p_i \, log(p_i) =$$
$$= 1 \times \log(1) + \lim_{x \to 0} \sum_{i \neq i_o} x \, log(x) = 0 + 0 = 0.$$

In classification settings, higher entropy (i.e., more disorder) corresponds to a sample that has a **mixed collection of labels**. Conversely, lower entropy corresponds to a classification where we have mostly pure partitions. In general, the entropy of a sample $D = \{x_1, x_2, \ldots, x_n\}$ is defined by:

$$H(D) = -\sum_{i=1}^{k} P(c_i|D) \log P(c_i|D),$$

where $P(c_i|D)$ is the probability of a data point in D being labeled with class c_i, and k is the number of classes (clusters). $P(c_i|D)$ can be estimated from the observed data by:

$$P(c_i|D) = \frac{|\left\{x_j \in D | x_j \text{ has label } y_j = c_i\right\}|}{|D|}.$$

Observe that if the observations are evenly split amongst all k classes, then $P(c_i|D) = \frac{1}{k}$ and

$$H(D) = -\sum_{i=1}^{k} \frac{1}{k} \log \frac{1}{k} = 1.$$

At the other extreme, if all the observations are from one class then:

$$H(D) = -1 * log_k(1) = 0.$$

Also note that the base of the log function is somewhat irrelevant and can be used to normalize (scale) the range of the entropy $\left(log_b(x) = \frac{log_2(x)}{log_2(b)}\right)$.

The `Gain` is the expected reduction in entropy caused by knowing the value of an attribute.

9.3.3 Misclassification Error and Gini Index

Similar to the `Entropy` measure, the `Misclassification error` and the `Gini index` are also applied to evaluate information gain. The Misclassification error is defined by the formula:

$$ME = 1 - \max_k (p_k).$$

And the Gini index is expressed as:

$$GI = \sum^{k} p_k(1 - p_k) = 1 - \sum^{k} p_k^2.$$

9.3.4 C5.0 Decision Tree Algorithm

C5.0 algorithm is a popular implementation of decision trees.

To begin with, let's consider the term `purity`. If the segments of data contains a single class, they are considered `pure`. The `entropy` represents a mathematical formalism measuring purity of data segments.

$$Entropy(S) = -\sum_{i=1}^{c} p_i log_2(p_i),$$

where `entropy` is the measurement, c is the number of total class levels, and p_i refers to the proportion of observations that fall into each class (i.e., probability of a randomly selected data point to belong to the *ith* class level. For two possible classes, the `entropy` ranges from 0 to 1. For n classes, the entropy ranges from 0 to $log_2(n)$, where the minimum entropy corresponds to data that is purely homogeneous (completely deterministic/predictable) and the maximum entropy represents completely disordered data (stochastic or extremely noisy). You might wonder what is the benefit of using the entropy? Another way to say this is the smaller the entropy, the more information is contained in this split method. Systems (data) with high entropy indicate significant information content (randomness) and data with low entropy indicates highly-compressible data with structure in it.

If we only have one class in the segment, then $Entropy(S) = (-1) \times log_2(1) = 0$.

Let's try another example. If we have a segment of data that contains two classes, the first class contains 80% of the data and the second class contains the remaining 20%. Then, we have the following `entropy`:

$$Entropy(S) = -0.8log_2(0.8) - 0.2log_2(0.2) = 0.7219281.$$

The relationship for two class proportions and entropy are illustrated in Fig. 9.3, where x is the proportion for elements in one of the classes.

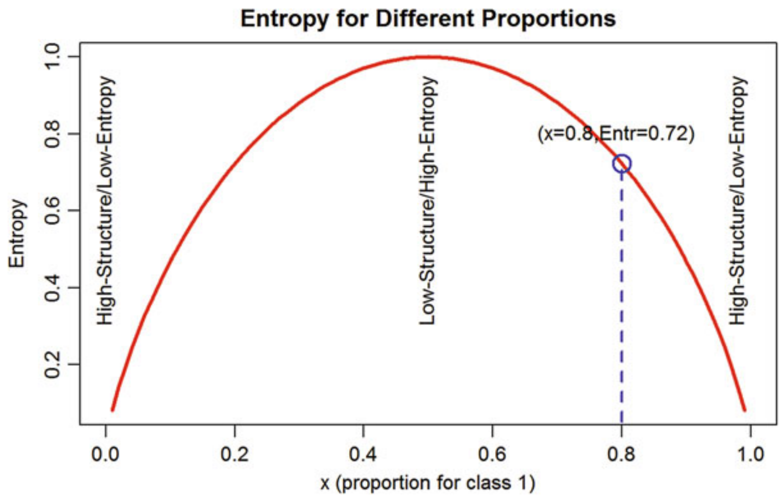

Fig. 9.3 Plot of the entropy of a (symmetric) binary process as a function of the proportion of class 1 cases

```
set.seed(1234)
x<-runif(100)
curve(-x*log2(x)-(1-x)*log2(1-x), col="red", main="Entropy for Different
Proportions", xlab = "x (proportion for class 1)", ylab = "Entropy", lwd=3)
```

The closer the binary proportion split is to 0.5, the greater the entropy. The more homogeneous the split (one class becomes the majority) the lower the entropy. **Decision trees** aim to find splits in the data that reduce the entropy, i.e., increasing the homogeneity of the elements within all classes.

This measuring mechanism could be used to measure and compare the information we get using different features as data partitioning characteristics. Let's consider this scenario. Suppose S and S_1 represent the entropy of the system before and after the splitting/partitioning of the data according to a specific data feature attribute (F). Denote the entropies of the original and the derived partition by $Entropy(S)$ and $Entropy(S_1)$, respectively. The **information we gained** from partitioning the data using this specific feature (F) is calculated as a change in the entropy:

$$Gain(F) = Entropy(S) - Entropy(S_1).$$

Note that smaller entropy $Entropy(S_1)$ corresponds with better classification and more information gained. A more complicated case would be that the partitions create multiple segments. Then, the entropy for each partition method is calculated by the following formula:

$$Entropy(S) = \sum_{i=1}^{n} w_i Entropy(P_i) = \sum_{i=1}^{n} w_i \left(\sum_{j=1}^{c} -p_i log_2(p_i) \right),$$

where w_i is the proportion of examples falling in that segment and P_i is segment i. Thus, the total entropy of a partition method is calculated by a weighted sum of entropies for each segment created by this method.

When we get the maximum reduction in entropy with a feature (F), then the $Gain(F) = Entropy(S)$, since $Entropy(S_1) = 0$. On the contrary, if we gain no information with this feature, we have $Gain(F) = 0$.

9.3.5 Pruning the Decision Tree

While making a decision tree, we can classify those observations using as many splits as we want. This eventually might over classify our data. An extreme example of this would be that we make each observation as a class, which is meaningless.

So how do we control the size of the decision tree? One possible solution is to make a cutoff for the number of decisions that a decision tree could make. Similarly, we can control the number of examples in each segment to be not too small. This method is called *early stopping* or *pre-pruning* the decision tree. However, this might make the decision procedure stop prematurely, before some important partition occurs.

Another solution *post-pruning* is that we begin with growing a big decision tree and subsequently reduce the branches based on error rates with penalty at the nodes. This is often more effective than the pre-prunning solution.

The `C5.0 algorithm` uses the *post-pruning* method to control the size of the decision tree. It first grows an overfitting large tree to contain all the possibilities of partitioning. Then, it cuts out nodes and branches with little effect on classification errors.

9.4 Case Study 1: Quality of Life and Chronic Disease

9.4.1 *Step 1: Collecting Data*

In this Chapter, we are using the Quality of life and chronic disease dataset, `Case06_QoL_Symptom_ChronicIllness.csv`. This dataset has 41 variables. Detailed description for each variable is provided here (https://umich.instructure.com/files/399150/download?download_frd=1).

Important variables:

- **Charlson Comorbidity Index**: ranging from 0 to 10. A score of 0 indicates no comorbid conditions. Higher scores indicate a greater level of comorbidity.
- **Chronic Disease Score**: A summary score based on the presence and complexity of prescription medications for select chronic conditions. A high score in decades the patient has severe chronic diseases. Entries stored as −9 indicate missing value.

9.4.2 *Step 2: Exploring and Preparing the Data*

Let's load the data first.

```
qol<-read.csv("https://umich.instructure.com/files/481332/download?download_
frd=1")
str(qol)

## 'data.frame':    2356 obs. of  41 variables:
##  $ ID                  : int  171 171 172 179 180 180 181 182 183 186 ...
##  $ INTERVIEWDATE       : int  0 427 0 0 0 42 0 0 0 0 ...
##  $ LANGUAGE            : int  1 1 1 1 1 1 1 1 1 2 ...
##  $ AGE                 : int  49 49 62 44 64 64 52 48 49 78 ...
##  $ RACE_ETHNICITY      : int  3 3 3 7 3 3 3 3 3 4 ...
##  $ SEX                 : int  2 2 2 2 1 1 2 1 1 1 ...
##  $ QOL_Q_01            : int  4 4 3 6 3 3 4 2 3 5 ...
##  $ QOL_Q_02            : int  4 3 3 6 2 5 4 1 4 6 ...
##  $ QOL_Q_03            : int  4 4 4 6 3 6 4 3 4 4 ...
##  $ QOL_Q_04            : int  4 4 2 6 3 6 2 2 5 2 ...
##  $ QOL_Q_05            : int  1 5 4 6 2 6 4 3 4 3 ...
##  $ QOL_Q_06            : int  4 4 2 6 1 2 4 1 2 4 ...
##  $ QOL_Q_07           : int  1 2 5 -1 0 5 8 4 3 7 ...
##  $ QOL_Q_08           : int  6 1 3 6 6 6 3 1 2 4 ...
##  $ QOL_Q_09           : int  3 4 3 6 2 2 4 2 2 4 ...
##  $ QOL_Q_10           : int  3 1 3 6 3 6 3 2 4 3 ...
##  $ MSA_Q_01           : int  1 3 2 6 2 3 4 1 1 2 ...
##  $ MSA_Q_02           : int  1 1 2 6 1 6 4 3 2 4 ...
##  $ MSA_Q_03           : int  2 1 2 6 1 2 3 3 1 2 ...
##  $ MSA_Q_04           : int  1 3 2 6 1 2 1 4 1 5 ...
##  $ MSA_Q_05           : int  1 1 1 6 1 2 1 6 3 2 ...
##  $ MSA_Q_06           : int  1 2 2 6 1 2 1 1 2 2 ...
##  $ MSA_Q_07           : int  2 1 3 6 1 1 1 1 1 5 ...
##  $ MSA_Q_08           : int  1 1 1 6 1 1 1 1 2 1 ...
##  $ MSA_Q_09           : int  1 1 1 6 2 2 4 6 2 1 ...
##  $ MSA_Q_10           : int  1 1 1 6 1 1 1 1 1 3 ...
##  $ MSA_Q_11           : int  2 3 2 6 1 1 2 1 1 5 ...
##  $ MSA_Q_12           : int  1 1 2 6 1 1 2 6 1 3 ...
##  $ MSA_Q_13           : int  1 1 1 6 1 6 2 1 4 2 ...
##  $ MSA_Q_14           : int  1 1 1 6 1 2 1 1 3 1 ...
##  $ MSA_Q_15           : int  2 1 1 6 1 1 3 2 1 3 ...
##  $ MSA_Q_16           : int  2 3 5 6 1 2 1 2 1 2 ...
##  $ MSA_Q_17           : int  2 1 1 6 1 1 1 1 1 3 ...
##  $ PH2_Q_01           : int  3 2 1 5 1 1 3 1 2 3 ...
##  $ PH2_Q_02           : int  4 4 1 5 1 2 1 1 4 2 ...
##  $ TOS_Q_01           : int  2 2 2 4 1 1 2 2 1 1 ...
##  $ TOS_Q_02           : int  1 1 1 4 4 4 1 2 4 4 ...
##  $ TOS_Q_03           : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ TOS_Q_04           : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ CHARLSONSCORE      : int  2 2 3 1 0 0 2 8 0 1 ...
##  $ CHRONICDISEASESCORE: num  1.6 1.6 1.54 2.97 1.28 1.28 1.31 1.67 2.21 2
.51 ...
```

Most of the coded variables like `QOL_Q_01`(heath rating) have ordinal values (1 = excellent, 2 = very good, 3 = good, 4 = fair, 5 = poor, 6 = no answer). We can use the `table()` function to see their distributions. We also have some numerical variables in the dataset like `CHRONICDISEASESCORE`. We can take a look at it by using `summary()`.

Our variable of interest `CHRONICDISEASESCORE` has some missing data. A simple way to address this is just deleting those observations with missing values. You could also try to impute the missing value using various imputation methods mentioned in Chap. 3.

```
table(qol$QOL_Q_01)

## 
##   1   2   3   4   5   6 
##  44 213 801 900 263 135

qol<-qol[!qol$CHRONICDISEASESCORE==-9, ]
summary(qol$CHRONICDISEASESCORE)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.000   0.880   1.395   1.497   1.970   4.760
```

Let's create two classes using variable CHRONICDISEASESCORE. We classify the patients with CHRONICDISEASESCORE < mean (CHRONICDISEASESCORE) as having minor disease and the rest as having severe disease. This dichotomous classification (qol$cd) may not be perfect and we will talk about alternative classification strategies in the practice problem in the end of the chapter.

```
qol$cd<-qol$CHRONICDISEASESCORE>1.497
qol$cd<-factor(qol$cd, levels=c(F, T), labels = c("minor_disease", "severe_d
isease"))
```

Data Preparation: Creating Random Training and Test Datasets

To make the qol data more organized, we can order the data by the variable ID.

```
qol<-qol[order(qol$ID), ]

# Remove ID (col=1) # the clinical Diagnosis (col=41) will be handled later
qol <- qol[ , -1]
```

Then, we are able to subset *training* and *testing* datasets. Here is an example of a *non-random split* of the entire data into training (2114) and testing (100) sets:

```
qol_train<-qol[1:2114, ]
qol_test<-qol[2115:2214, ]
```

And here is an example of *random assignments* of cases into training and testing sets (80–20% slit):

```
set.seed(1234)
train_index <- sample(seq_len(nrow(qol)), size = 0.8*nrow(qol))
qol_train<-qol[train_index, ]
qol_test<-qol[-train_index, ]
```

We can quickly inspect the distributions of the training and testing data to ensure they are not vastly different. We can see that the classes are split fairly equal in training and testing datasets.

```
prop.table(table(qol_train$cd))
##  minor_disease severe_disease
##      0.5279503      0.4720497

prop.table(table(qol_test$cd))

##  minor_disease severe_disease
##       0.503386       0.496614
```

9.4.3 Step 3: Training a Model On the Data

In this section, we are using the `C5.0()` function from the `C50` package.

The function `C5.0()` has following components:

```
m<-C5.0(train, class, trials=1, costs=NULL)
```

- train: data frame containing numeric training data (features).
- class: factor vector with the class for each row in the training data.
- trials: an optional number to control the boosting iterations (default = 1).
- costs: an optional matrix to specify the costs of false positive and false negative.

You could delete the # in the following code and run it in R to install and load the `C50` package.

```
# install.packages("C50")
library(C50)
```

In the `qol` dataset (ID column is already removed), column 41 is the class vector (`qol$cd`), and column 40 is the numerical version of vector 41 (`qol$CHRONICDISEASESCORE`). We need to delete these two columns to create our training data that only contains features.

```
summary(qol_train[,-c(40, 41)])

##  INTERVIEWDATE        LANGUAGE           AGE         RACE_ETHNICITY
##  Min.    :  0.00   Min.   :1.000   Min.   :20.00   Min.   :1.000
##  1st Qu.:  0.00   1st Qu.:1.000   1st Qu.:52.00   1st Qu.:3.000
##  Median :  0.00   Median :1.000   Median :59.00   Median :3.000
##  Mean   : 21.68   Mean   :1.217   Mean   :58.74   Mean   :3.614
##  3rd Qu.:  0.00   3rd Qu.:1.000   3rd Qu.:67.00   3rd Qu.:4.000
##  Max.   :440.00   Max.   :2.000   Max.   :90.00   Max.   :7.000
##       SEX            QOL_Q_01        QOL_Q_02        QOL_Q_03
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:3.000   1st Qu.:3.000   1st Qu.:3.000
##  Median :1.000   Median :4.000   Median :3.000   Median :4.000
##  Mean   :1.422   Mean   :3.661   Mean   :3.408   Mean   :3.714
##  3rd Qu.:2.000   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :2.000   Max.   :6.000   Max.   :6.000   Max.   :6.000
…

##     TOS_Q_03        TOS_Q_04      CHARLSONSCORE
##  Min.   :1.000   Min.   :1.000   Min.   :-9.0000
##  1st Qu.:4.000   1st Qu.:5.000   1st Qu.: 0.0000
##  Median :4.000   Median :5.000   Median : 1.0000
##  Mean   :3.787   Mean   :4.686   Mean   : 0.8826
##  3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.: 1.0000
##  Max.   :5.000   Max.   :6.000   Max.   :10.0000

set.seed(1234)
qol_model<-C5.0(qol_train[,-c(40, 41)], qol_train$cd)
qol_model
```

```
##
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd)
##
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
##
## Tree size: 25
##
## Non-standard options: attempt to group attributes
```

```
summary(qol_model)
```

```
##
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd)
##
##
## C5.0 [Release 2.07 GPL Edition]      Tue Jun 20 16:09:16 2017
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 1771 cases (40 attributes) from undefined.data
##
## Decision tree:
##
## CHARLSONSCORE <= 0: minor_disease (665/180)
## CHARLSONSCORE > 0:
```

```
## :...AGE <= 47:
##     :...MSA_Q_08 > 2: severe_disease (15/4)
##     :   MSA_Q_08 <= 2:
##     :   :...MSA_Q_14 <= 1: minor_disease (86/20)
##     :       MSA_Q_14 > 1:
##     :       :...MSA_Q_10 > 4: minor_disease (6)
##     :           MSA_Q_10 <= 4:
##     :           :...TOS_Q_03 > 4: severe_disease (8)
##     :               TOS_Q_03 <= 4:
##     :               :...MSA_Q_17 > 2: minor_disease (8/1)
##     :                   MSA_Q_17 <= 2:
##     :                   :...QOL_Q_01 <= 2: minor_disease (4)
##     :                       QOL_Q_01 > 2: severe_disease (38/13)
##     AGE > 47:
##     :...RACE_ETHNICITY > 3:
##         :...QOL_Q_07 > 5: severe_disease (133/26)
##         :   QOL_Q_07 <= 5:
##         :   :...QOL_Q_10 > 5: severe_disease (24/2)
##         :       QOL_Q_10 <= 5:
##         :       :...MSA_Q_14 <= 5: severe_disease (202/72)
##         :           MSA_Q_14 > 5: minor_disease (11/2)
##         RACE_ETHNICITY <= 3:
##         :...QOL_Q_01 <= 2: minor_disease (50/20)
##             QOL_Q_01 > 2:
##             :...CHARLSONSCORE > 1: severe_disease (184/58)
##                 CHARLSONSCORE <= 1:
##                 :...MSA_Q_04 > 5: minor_disease (27/8)
##                     MSA_Q_04 <= 5:
##                     :...QOL_Q_07 <= 5:
##                         :...QOL_Q_05 <= 2:
##                         :   :...TOS_Q_04 <= 2: minor_disease (5)
##                         :   :   TOS_Q_04 > 2: severe_disease (52/15)
##                         :   QOL_Q_05 > 2:
##                         :   :...MSA_Q_06 <= 5: minor_disease (119/46)
##                         :       MSA_Q_06 > 5: severe_disease (10/2)
##                         QOL_Q_07 > 5:
##                         :...QOL_Q_09 <= 2: severe_disease (18/1)
##                             QOL_Q_09 > 2:
##                             :...RACE_ETHNICITY <= 2: minor_disease (12/5)
##                                 RACE_ETHNICITY > 2:
##                                 :...MSA_Q_17 > 3: severe_disease (19/2)
##                                     MSA_Q_17 <= 3:
##                                  ...PH2_Q_01 <= 3: severe_disease (50/14)
##                                         PH2_Q_01 > 3:
##                                    ...MSA_Q_14 <= 3: minor_disease (21/6)
##                                          MSA_Q_14 > 3: severe_disease (4)
##
##
## Evaluation on training data (1771 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##      25  497(28.1%)   <<
##
##     (a)   (b)    <-classified as
##    ----  ----
##     726   209    (a): class minor_disease
##     288   548    (b): class severe_disease
```

```
##
##
##  Attribute usage:
##
##  100.00% CHARLSONSCORE
##   62.45% AGE
##   53.13% RACE_ETHNICITY
##   38.40% QOL_Q_07
##   34.61% QOL_Q_01
##   21.91% MSA_Q_14
##   19.03% MSA_Q_04
##   13.38% QOL_Q_10
##   10.50% QOL_Q_05
##    9.32% MSA_Q_08
##    8.13% MSA_Q_17
##    7.28% MSA_Q_06
##    7.00% QOL_Q_09
##    4.23% PH2_Q_01
##    3.61% MSA_Q_10
##    3.27% TOS_Q_03
##    3.22% TOS_Q_04
```

The output of `qol_model` indicates that we have a tree that has 25 terminal nodes. `summary(qol_model)` suggests that the classification error for decision tree is 28% in the training data.

9.4.4 Step 4: Evaluating Model Performance

Now we can make predictions using the decision tree that we just built. The `predict()` function we will use is the same as the one we showed in earlier chapters, e.g., Chaps. 3 and 8. In general, `predict()` is extended by each specific type of regression, classificaiton, clustering, or forecasting machine learning technique. For example, `randomForest::predict.randomForest()` is invoked by:

```
predict(RF_model, newdata, type="response", norm.votes=TRUE,
predict.all=FALSE, proximity=FALSE, nodes=FALSE, cutoff, ...),
```

where `type` represents type of prediction output to be generated - "response" (equivalent to "class"), "prob" or "votes". Thus, the predicted values are either predicted "response" class labels, matrix of class probabilities, or vote counts.

This time we are going to introduce the `confusionMatrix()` function under package `caret` as the evaluation method. When we combine it with a `table()` function, the output of the evaluation is very straight forward.

```
qol_pred<-predict(qol_model, qol_test[ ,-c(40, 41)])  # removing the last 2
columns CHRONICDISEASESCORE and cd, which represent the clinical outcomes we
are predicting!
# install.packages("caret")
library(caret)

confusionMatrix(table(qol_pred, qol_test$cd))

## Confusion Matrix and Statistics
##
## qol_pred         minor_disease severe_disease
##   minor_disease            149             89
##   severe_disease            74            131
##
##                Accuracy : 0.6321
##                  95% CI : (0.5853, 0.6771)
##     No Information Rate : 0.5034
##     P-Value [Acc > NIR] : 3.317e-08
##
##                   Kappa : 0.2637
##  Mcnemar's Test P-Value : 0.2728
##
##             Sensitivity : 0.6682
##             Specificity : 0.5955
##          Pos Pred Value : 0.6261
##          Neg Pred Value : 0.6390
##              Prevalence : 0.5034
##          Detection Rate : 0.3363
##    Detection Prevalence : 0.5372
##       Balanced Accuracy : 0.6318
##
##        'Positive' Class : minor_disease
```

The Confusion Matrix shows that the testing data prediction accuracy is about 63%. However, this may vary (see the corresponding confidence interval).

9.4.5 Step 5: Trial *Option*

The `C5.0` function includes, an option `trials=`, which is an integer specifying the number of boosting iterations. The default value of one indicates that a single model is used, and we can specify a larger number of iterations, for instance `trials=6`.

```
set.seed(1234)
qol_boost6<-C5.0(qol_train[ , -c(40, 41)], qol_train$cd, trials=6) # try alt
ernative values for the trials option
qol_boost6

##
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd, trials = 6)
##
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
##
## Number of boosting iterations: 6
## Average tree size: 11.7
##
## Non-standard options: attempt to group attributes
```

We can see that the size of the tree reduced to about 12 (this may vary at each run).

Since this is a fairly small tree, we can visualize it by the function `plot()`. We also use the option `type="simple"` to make the tree look more condensed (Fig. 9.4).

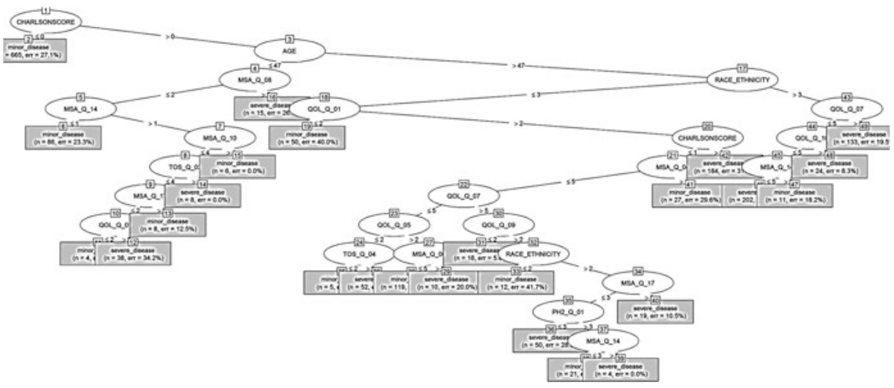

Fig. 9.4 Classification tree plot of the quality of lofe (QoL) data

```
plot(qol_boost6, type="simple")
```

Caution The plotting of decision trees will fail if you have columns that start with numbers or special characters (e.g., "*5variable*", "*!variable*"). In general, avoid spaces, special characters, and other non-terminal symbols in column/row names.

The next step would be making predictions and testing the corresponding accuracy.

```
qol_boost_pred6 <- predict(qol_boost6, qol_test[ ,-c(40, 41)])
confusionMatrix(table(qol_boost_pred6, qol_test$cd))

## Confusion Matrix and Statistics
## qol_boost_pred6  minor_disease severe_disease
##   minor_disease            140             75
##   severe_disease            83            145
##
##                Accuracy : 0.6433
##                  95% CI : (0.5968, 0.688)
##     No Information Rate : 0.5034
##     P-Value [Acc > NIR] : 1.987e-09
##                   Kappa : 0.2868
##  Mcnemar's Test P-Value : 0.5776
##             Sensitivity : 0.6278
##             Specificity : 0.6591
##          Pos Pred Value : 0.6512
##          Neg Pred Value : 0.6360
##              Prevalence : 0.5034
##          Detection Rate : 0.3160
##    Detection Prevalence : 0.4853
##       Balanced Accuracy : 0.6434
##
##        'Positive' Class : minor_disease
```

The accuracy is about 64%. However, this may vary each time we run the experiment (mind the confidence interval). In some studies, the *trials* option provides significant improvement to the overall accuracy. A good choice for this option is `trials` = `10`.

9.4.6 *Loading the Misclassification Error Matrix*

Suppose we want to reduce the *false negative rate*, in this case, misclassifying a severe case as minor. False negative (failure to detect a severe disease case) may be more costly than false positive (misclassifying a minor disease case as severe). Misclassification errors can be expressed as a matrix:

```
error_cost<-matrix(c(0, 1, 4, 0), nrow = 2)
error_cost

##      [,1] [,2]
## [1,]    0    4
## [2,]    1    0
```

Let's build a decision tree with the option `cpsts=error_cost`.

```
set.seed(1234)
qol_cost<-C5.0(qol_train[-c(40, 41)], qol_train$cd, costs=error_cost)
qol_cost_pred<-predict(qol_cost, qol_test)
confusionMatrix(table(qol_cost_pred, qol_test$cd))

## Confusion Matrix and Statistics
##
##
## qol_cost_pred    minor_disease severe_disease
##   minor_disease             60             17
##   severe_disease           163            203
##
##                Accuracy : 0.5937
##                  95% CI : (0.5463, 0.6398)
##     No Information Rate : 0.5034
##     P-Value [Acc > NIR] : 8.352e-05
##
##                   Kappa : 0.1909
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.2691
##             Specificity : 0.9227
##          Pos Pred Value : 0.7792
##          Neg Pred Value : 0.5546
##              Prevalence : 0.5034
##          Detection Rate : 0.1354
##    Detection Prevalence : 0.1738
##       Balanced Accuracy : 0.5959
##
##        'Positive' Class : minor_disease
```

Although the overall accuracy decreased, the false negative cell labels were reduced from 75 (without specifying a cost matrix) to 17 (when specifying a non-trivial (loaded) cost matrix). This comes at the cost of increasing the rate of false-positive labeling (minor disease cases misclassified as severe).

9.4.7 Parameter Tuning

There are multiple choices to plot trees fitted by `rpart`, `C50`.

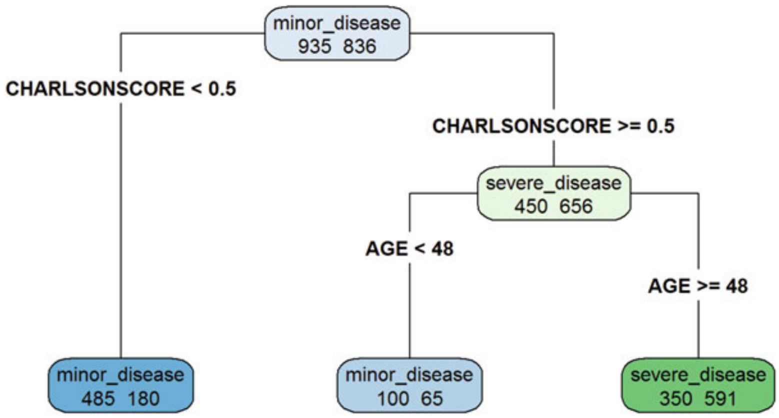

Fig. 9.5 Decision tree classification of the QoL data

```
library("rpart")
# remove CHRONICDISEASESCORE, but keep *cd* label
set.seed(1234)
qol_model<-rpart(cd~., data=qol_train[, -40], cp=0.01)
# here we use rpart::cp = *complexity parameter* = 0.01
qol_model

## n= 1771
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 1771 836 minor_disease (0.5279503 0.4720497)
##   2) CHARLSONSCORE< 0.5 665 180 minor_disease (0.7293233 0.2706767) *
##   3) CHARLSONSCORE>=0.5 1106 450 severe_disease (0.4068716 0.5931284)
##     6) AGE< 47.5 165  65 minor_disease (0.6060606 0.3939394) *
##     7) AGE>=47.5 941 350 severe_disease (0.3719447 0.6280553) *
```

You can also plot directly using `rpart.plot` (Fig. 9.5).

```
library(rpart.plot)
rpart.plot(qol_model, type = 4,extra = 1,clip.right.labs = F)
```

We can use `fancyRpartPlot` (Figs. 9.6).

```
library("rattle")
fancyRpartPlot(qol_model, cex = 1)
```

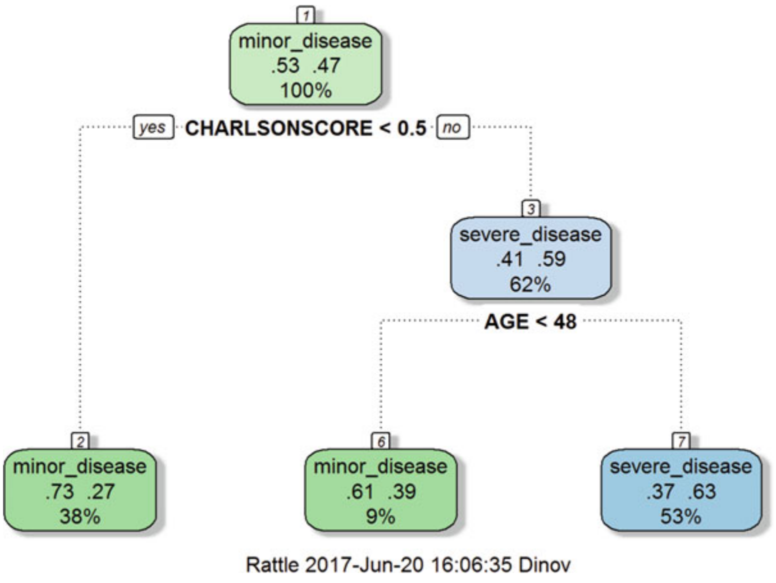

Fig. 9.6 Another decision tree classification of the QoL data, compare to Fig. 9.5

```
qol_pred<-predict(qol_model, qol_test,type = 'class')
confusionMatrix(table(qol_pred, qol_test$cd))
## Confusion Matrix and Statistics
## qol_pred          minor_disease severe_disease
##   minor_disease             133             64
##   severe_disease             90            156
##
##                  Accuracy : 0.6524
##                    95% CI : (0.606, 0.6967)
##     No Information Rate : 0.5034
##     P-Value [Acc > NIR] : 1.759e-10
##                     Kappa : 0.3053
##  Mcnemar's Test P-Value : 0.04395
##               Sensitivity : 0.5964
##               Specificity : 0.7091
##            Pos Pred Value : 0.6751
##            Neg Pred Value : 0.6341
##                Prevalence : 0.5034
##            Detection Rate : 0.3002
##      Detection Prevalence : 0.4447
##         Balanced Accuracy : 0.6528
##          'Positive' Class : minor_disease
```

These results are consistent with their counterparts reported using `C5.0`. How can we tune the parameters to further improve the results? (Fig. 9.7).

```
set.seed(1234)
control = rpart.control(cp = 0.000, xxval = 100, minsplit = 2)
qol_model= rpart(cd ~ ., data = qol_train[ , -40], control = control)
plotcp(qol_model)
```

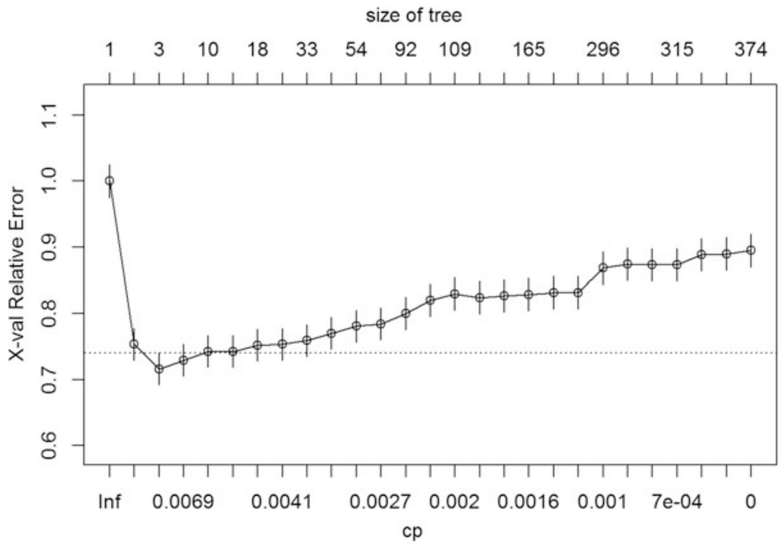

Fig. 9.7 Tuning the decision tree classification by reducing the error across the spectrum of cost-complexity pruning parameter (cp) and tree size

```
printcp(qol_model)

## Classification tree:
## rpart(formula = cd ~ ., data = qol_train[, -40], control = control)
##
## Variables actually used in tree construction:
##  [1] AGE            CHARLSONSCORE  INTERVIEWDATE  LANGUAGE
##  [5] MSA_Q_01       MSA_Q_02       MSA_Q_03       MSA_Q_04
##  [9] MSA_Q_05       MSA_Q_06       MSA_Q_07       MSA_Q_08
## [13] MSA_Q_09       MSA_Q_10       MSA_Q_11       MSA_Q_12
## [17] MSA_Q_13       MSA_Q_14       MSA_Q_15       MSA_Q_16
## [21] MSA_Q_17       PH2_Q_01       PH2_Q_02       QOL_Q_01
## [25] QOL_Q_02       QOL_Q_03       QOL_Q_04       QOL_Q_05
## [29] QOL_Q_06       QOL_Q_07       QOL_Q_08       QOL_Q_09
## [33] QOL_Q_10       RACE_ETHNICITY SEX            TOS_Q_01
## [37] TOS_Q_02       TOS_Q_03       TOS_Q_04
##
## Root node error: 836/1771 = 0.47205
##
## n= 1771
##
##             CP nsplit rel error  xerror     xstd
## 1  0.24641148      0 1.0000000 1.00000 0.025130
## 2  0.04186603      1 0.7535885 0.75359 0.024099
## 3  0.00717703      2 0.7117225 0.71651 0.023816
## 4  0.00657895      3 0.7045455 0.72967 0.023920
## 5  0.00598086      9 0.6543062 0.74282 0.024020
## 6  0.00478469     14 0.6244019 0.74282 0.024020
## 7  0.00418660     17 0.6100478 0.75239 0.024090
## 8  0.00398724     21 0.5933014 0.75359 0.024099
## 9  0.00358852     32 0.5466507 0.75957 0.024141
## 10 0.00318979     41 0.5143541 0.77033 0.024215
## 11 0.00299043     53 0.4665072 0.78110 0.024286
## 12 0.00239234     59 0.4485646 0.78469 0.024309
## 13 0.00209330     91 0.3708134 0.80024 0.024406
## 14 0.00199362     95 0.3624402 0.82057 0.024522
## 15 0.00191388    108 0.3349282 0.83014 0.024574
## 16 0.00179426    122 0.2978469 0.82416 0.024542
## 17 0.00159490    151 0.2416268 0.82656 0.024555
## 18 0.00153794    164 0.2177033 0.82895 0.024567
## 19 0.00149522    171 0.2069378 0.83134 0.024580
## 20 0.00119617    182 0.1866029 0.83134 0.024580
## 21 0.00089713    295 0.0514354 0.86842 0.024758
## 22 0.00079745    306 0.0406699 0.87440 0.024783
## 23 0.00071770    309 0.0382775 0.87321 0.024778
## 24 0.00068353    314 0.0346890 0.87321 0.024778
## 25 0.00059809    321 0.0299043 0.88876 0.024841
## 26 0.00039872    367 0.0023923 0.88995 0.024846
## 27 0.00000000    373 0.0000000 0.89474 0.024864
```

Now, we can prune the tree according to the optimal `cp`, *complexity parameter* to which the `rpart` object will be trimmed. Instead of using the *real error* (e.g., $1 - R^2$, RMSE) to capture the discrepancy between the observed labels and the model-predicted labels, we will use the `xerror`, which averages the discrepancy between observed and predicted classifications using *cross-validation*, see Chap. 21. Figs. 9.8, 9.9, and 9.10 show some alternative decision tree prunning results.

```
set.seed(1234)
selected_tr <- prune(qol_model, cp= qol_model$cptable[which.min(qol_model$cp
table[,"xerror"]),"CP"])
fancyRpartPlot(selected_tr, cex = 1)
```

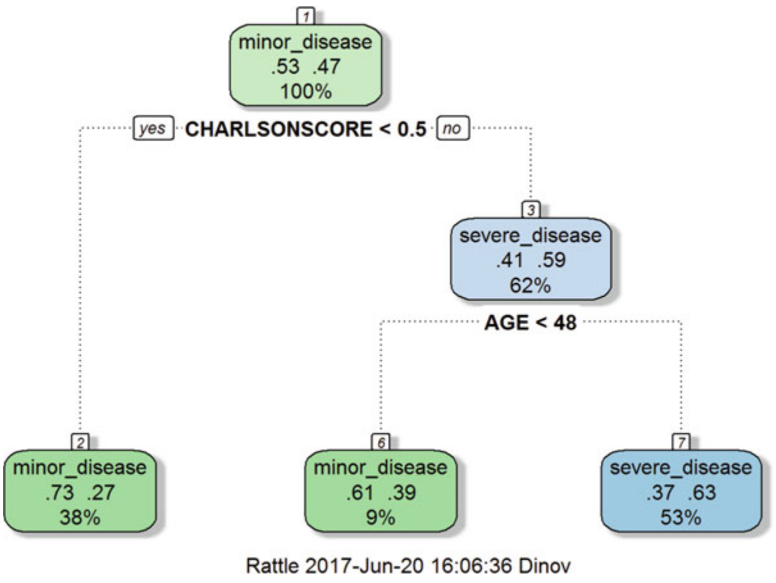

Fig. 9.8 Prunned decision tree classification for the QoL data, compare to Figs. 9.5 and 9.6

```
qol_pred_tune<-predict(selected_tr, qol_test,type = 'class')
confusionMatrix(table(qol_pred_tune, qol_test$cd))
```

```
## Confusion Matrix and Statistics
## qol_pred_tune    minor_disease severe_disease
##   minor_disease            133             64
##   severe_disease            90            156
##
##                Accuracy : 0.6524
##                  95% CI : (0.606, 0.6967)
##     No Information Rate : 0.5034
##     P-Value [Acc > NIR] : 1.759e-10
##                   Kappa : 0.3053
##  Mcnemar's Test P-Value : 0.04395
##             Sensitivity : 0.5964
##             Specificity : 0.7091
##          Pos Pred Value : 0.6751
##          Neg Pred Value : 0.6341
##              Prevalence : 0.5034
##          Detection Rate : 0.3002
##    Detection Prevalence : 0.4447
##       Balanced Accuracy : 0.6528
##        'Positive' Class : minor_disease
```

The result is roughly same as that of `C5.0`. Despite the fact that there is no substantial classification improvement, the tree-pruning process generates a graphical representation of the decision making protocol (`selected_tr`) that is much simpler and intuitive compared to the original (un-pruned) tree (`qol_model`):

```
fancyRpartPlot(qol_model, cex = 0.1)
```

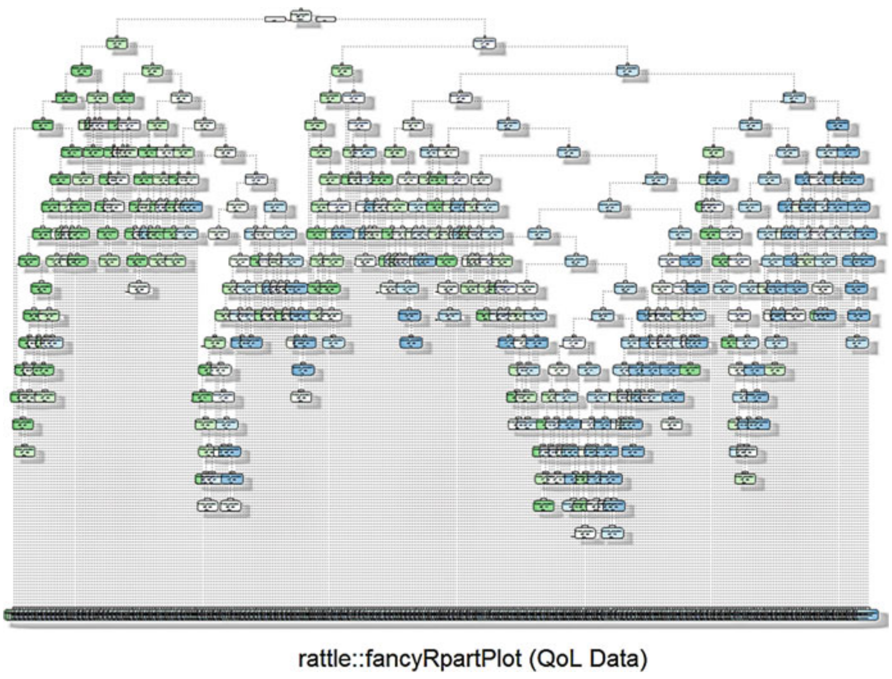

Fig. 9.9 Testing data (QoL dataset) decision tree prediction results (for chronic disease, CD)

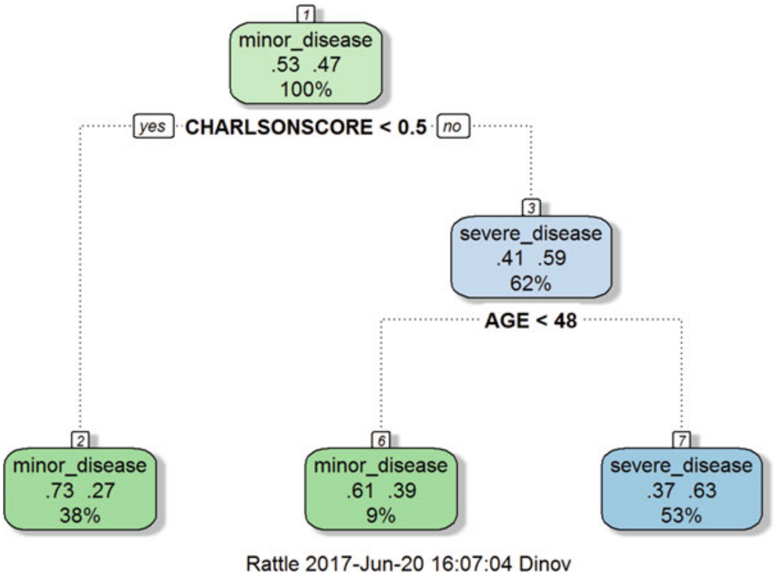

Fig. 9.10 Training data QoL decision tree plot

9.5 Compare Different Impurity Indices

We can change *split* = *"entropy"* to "error" or "gini" to apply an alternative information gain index. Experiment with these setting and compare the results.

```
set.seed(1234)
qol_model = rpart(cd ~ ., data=qol_train[ , -40],
parms = list(split = "entropy"))
fancyRpartPlot(qol_model, cex = 1)
```

```
# Modify and test using "error" and "gini"
# qol_pred<-predict(qol_model, qol_test,type = 'class')
# confusionMatrix(table(qol_pred, qol_test$cd))
```

9.6 Classification Rules

In addition to the classification trees we just saw, we can explore *classification rules* that utilize `if-else` logical statements to assign classes to unlabeled data. Below we review three classification rule strategies.

9.6.1 Separate and Conquer

Separate and conquer repeatedly splits the data (and subsets of the data) by rules that cover a subset of examples. This procedure is very similar to the *Divide and conquer* approach. However, a notable difference is that each rule can be independent, and yet, each decision node in a tree has to be linked to past decisions.

9.6.2 The One Rule Algorithm

To understand the `One Rule (OneR)` algorithm, we need to know about its "sibling" - `ZeroR` rule. `ZeroR` rule means that we assign the mode class to unlabeled test observations regardless of its feature value. The One rule algorithm is an improved version of `ZeroR` that uses a single rule for classification. In other words, `OneR` splits the training dataset into several segments based on feature values. Then, it assigns the modes of the classes with in each segment to related observations in the unlabeled test data. In practice, we first test multiple rules and pick the rule with the smallest error rate to be our `One Rule`. Remember, these rules may be subjective.

9.6.3 The RIPPER Algorithm

The *Repeated Incremental Pruning to Produce Error Reduction* algorithm is a combination of the ideas behind decision tree and classification rules. It consists of a three-step process:

- **Grow**: add conditions to a rule until it cannot split the data into more segments.
- **Prune**: delete some of the conditions that have large error rates.
- **Optimize**: repeat the above two steps until we cannot add or delete any of the conditions.

9.7 Case Study 2: QoL in Chronic Disease (Take 2)

Let’s take another look at the same dataset as Case Study 1 - this time applying *classification rules*. Naturally, we will skip over the first two data handling steps and go directly to step three.

9.7.1 Step 3: Training a Model on the Data

Let’s start by using the `OneR()` function in the `RWeka` package. Before installing the package you might want to check that the Java program in your computer is up to date. Also, its version has to match the version of R (i.e., 64bit R needs 64bit Java).

The function `OneR()` has the following invocation protocol:

```
m<-OneR(class~predictors, data=mydata)
```

- class: factor vector with the class for each row in **mydata**.
- predictors: feature variables in **mydata**. If we want to include x_1, x_2 as predictors and y as the class label variable, we do $y \sim x_1 + x_2$. To specify a full model, we use this notation: `y ~ .`, which includes all of the column variables as predictors.
- mydata: the dataset where the features and labels can be found.

```
# install.packages("RWeka")
library(RWeka)
# just remove the CHRONICDISEASESCORE but keep cd
set.seed(1234)
qol_1R<-OneR(cd~., data=qol[ , -40])
qol_1R

## CHARLSONSCORE:
##  < -4.5  -> severe_disease
##  < 0.5   -> minor_disease
##  < 5.5   -> severe_disease
##  < 8.5   -> minor_disease
##  >= 8.5  -> severe_disease
## (1453/2214 instances correct)
```

Note that 1,453 out of 2,214 cases are correctly classified, 66%, by the "one rule".

9.7.2 Step 4: Evaluating Model Performance

```
summary(qol_1R)

##
## === Summary ===
##
## Correctly Classified Instances        1453               65.6278 %
## Incorrectly Classified Instances       761               34.3722 %
## Kappa statistic                          0.3206
## Mean absolute error                      0.3437
## Root mean squared error                  0.5863
## Relative absolute error                 68.8904 %
## Root relative squared error            117.3802 %
## Total Number of Instances             2214
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  609 549 |   a = minor_disease
##  212 844 |   b = severe_disease
```

We obtained a single rule that correctly specifies 66% of the patients, which is in line with the prior decision tree classification results. Due to algorithmic stochasticity, it's normal that these results may vary each time you run the algorithm, albeit we used `seed(1234)` to ensure some result reproducibility.

9.7.3 Step 5: Alternative Model1

Another possible option for the classification rules would be the RIPPER rule algorithm that we discussed earlier in the chapter. In R we use the Java based function JRip() to invoke this algorithm.

JRip() function has the same components as the OneR() function:

```
m<-JRip(class~predictors, data=mydata)
```

```
set.seed(1234)
qol_jrip1<-JRip(cd~., data=qol[ , -40])
qol_jrip1

## JRIP rules:
## ===========
## (CHARLSONSCORE >= 1) and (RACE_ETHNICITY >= 4) and (AGE >= 49) => cd=seve
re_disease (448.0/132.0)
## (CHARLSONSCORE >= 1) and (AGE >= 53) => cd=severe_disease (645.0/265.0)
##  => cd=minor_disease (1121.0/360.0)
##
## Number of Rules : 3

summary(qol_jrip1)

## Correctly Classified Instances        1457               65.8085 %
## Incorrectly Classified Instances       757               34.1915 %
## Kappa statistic                          0.3158
## Mean absolute error                      0.4459
## Root mean squared error                  0.4722
## Relative absolute error                 89.3711 %
## Root relative squared error             94.5364 %
## Total Number of Instances             2214
## === Confusion Matrix ===
##    a   b   <-- classified as
##  761 397 |   a = minor_disease
##  360 696 |   b = severe_disease
```

This JRip() classifier uses only three rules and has a relatively similar accuracy 66%. As each individual has unique characteristics, classification in real world data is rarely perfect (close to 100% accuracy).

9.7.4 Step 5: Alternative Model2

Another idea is to repeat the generation of trees multiple times, predict according to each tree's performance, and finally ensemble those weighted votes into a combined classification result. This is precisely the idea behind random forest classification, see Chap. 15 (Figs. 9.11 and 9.12).

```
require(randomForest)

set.seed(12)
# rf.fit <- tuneRF(qol_train[ , -40], qol_train[ , 40], stepFactor=1.5)
rf.fit <- randomForest(cd~. , data=qol_train[ , -40],importance=TRUE,ntree=2
000,mtry=26)
varImpPlot(rf.fit); print(rf.fit)
```

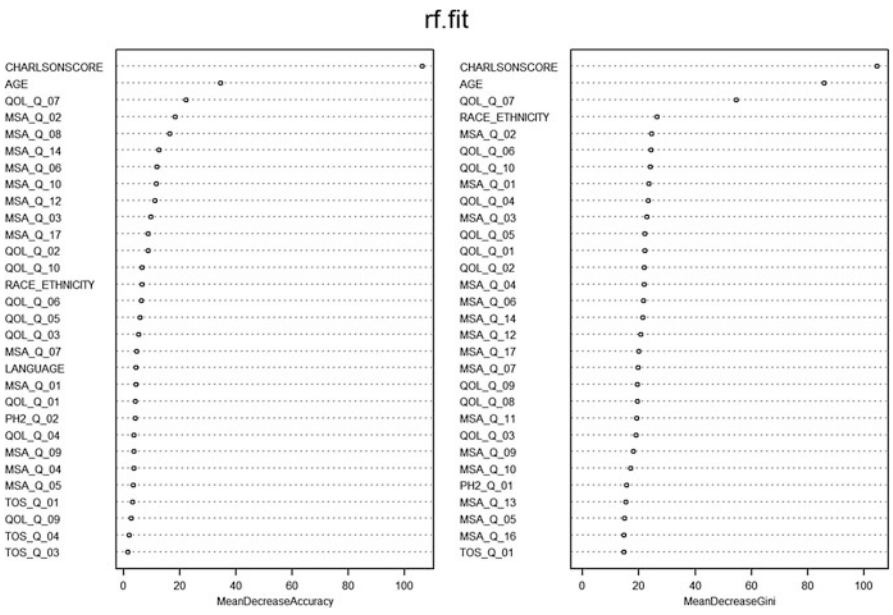

Fig. 9.11 Variable importance plots of random forest classification of the QoL CD variable using accuracy (left) and Gini index (right) as evaluation metrics

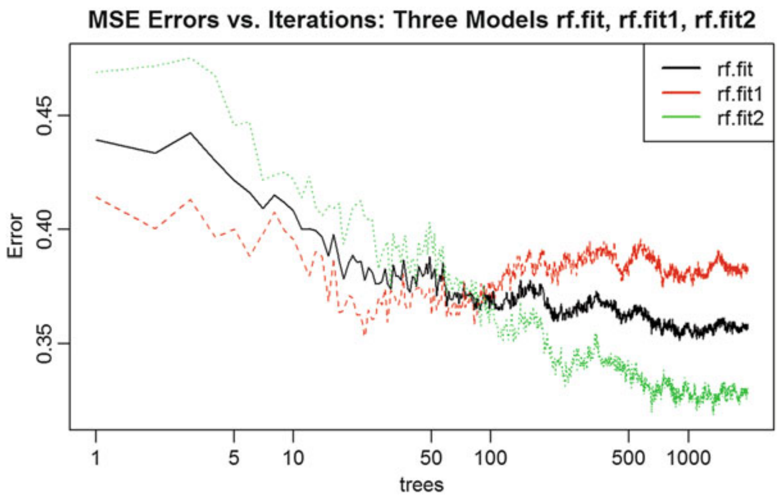

Fig. 9.12 Error plots of the random forest prediction of CD (QoL chronic disease) using three different trees models

```
## Call:
##  randomForest(formula = cd ~ ., data = qol_train[, -40], 
importance = TRUE,      ntree = 2000, mtry = 26) 
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 26
## 
##         OOB estimate of  error rate: 35.86%
## Confusion matrix:
##                  minor_disease severe_disease class.error
## minor_disease              576            359   0.3839572
## severe_disease             276            560   0.3301435

rf.fit1 <- randomForest(cd~. , data=qol_train[ , -40],importance=TRUE,ntree=
2000,mtry=26)
rf.fit2 <- randomForest(cd~. , data=qol_train[ , -40], importance=TRUE, node
size=5, ntree=5000, mtry=26)

plot(rf.fit,log="x",main="rf.fit (Black), rf.fit1 (Red), rf.fit2 (Green)")
points(1:5000, rf.fit1$mse, col="red", type="l")
points(1:5000, rf.fit2$mse, col="green", type="l")
```

```
qol_pred<-predict(rf.fit2, qol_test, type = 'class')
confusionMatrix(table(qol_pred, qol_test$cd))

## Confusion Matrix and Statistics
## qol_pred         minor_disease severe_disease
##   minor_disease            138             69
##   severe_disease            85            151
##                                           
##                Accuracy : 0.6524          
##                  95% CI : (0.606, 0.6967) 
##     No Information Rate : 0.5034          
##     P-Value [Acc > NIR] : 1.759e-10       
##                   Kappa : 0.305           
##  Mcnemar's Test P-Value : 0.2268          
##             Sensitivity : 0.6188          
##             Specificity : 0.6864          
##          Pos Pred Value : 0.6667          
##          Neg Pred Value : 0.6398          
##              Prevalence : 0.5034          
##          Detection Rate : 0.3115          
##    Detection Prevalence : 0.4673          
##       Balanced Accuracy : 0.6526          
##        'Positive' Class : minor_disease   
```

These variable importance plots (`varplot`) show the rank order of importance of the features according to the specific index (Accuracy, left, and Gini, right). More information about random forests is available in Chap. 15: Improving Model Performance.

In random forest (RF) classification, the node size (`nodesize`) refers to the smallest node that can be split, i.e., nodes with fewer cases than the `nodesize` are never subdivided. Increasing the node size leads to smaller trees, which may compromise previous predictive power. On the flip side, increasing the tree size (maxnodes) and the number of trees (ntree) tends to increase the predictive accuracy. However, there are tradeoffs between increasing node-size and tree-size simultaneously. To optimize the RF predictive accuracy, try smaller node sizes and more trees. Ensembling (forest) results from a larger number of trees will likely generate better results.

9.8 Practice Problem

In the previous case study, we classified the CHRONICDISEASESCORE into two groups. What will happen if we use three groups? Let's separate CHRONICDISEASESCORE evenly into three groups. Recall the quantile() function that we talked about in Chap. 3. We can use it to get the cut-points for classification. Then, a for loop will help us split the variable CHRONICDISEASESCORE into three categories.

```
quantile(qol$CHRONICDISEASESCORE, probs = c(1/3, 2/3))

## 33.33333% 66.66667%
##      1.06      1.80

for(i in 1:2214){
  if(qol$CHRONICDISEASESCORE[i]>0.7&qol$CHRONICDISEASESCORE[i]<2.2){
    qol$cdthree[i]=2
  }
  else if(qol$CHRONICDISEASESCORE[i]>=2.2){
    qol$cdthree[i]=3
  }
  else{
    qol$cdthree[i]=1
  }
}

qol$cdthree<-factor(qol$cdthree, levels=c(1, 2, 3), labels =
c("minor_disease", "mild_disease", "severe_disease"))
```

After labeling the three categories in the new variable cdthree, our job of preparing the class variable is done. Let's follow along the earlier sections in the chapter to determine how well the tree classifiers and the rule classifiers perform in the three-category case. First, try to build a tree classifier using C5.0() with 10 boost trials. One small tip is that in the training dataset, we cannot have column 40 (CHRONICDISEASESCORE), 41 (cd), and now 42 (cdthree) because they all contain class outcome related variables.

```
# qol_train1<-qol[1:2114, ]
# qol_test1<-qol[2115:2214, ]
train_index <- sample(seq_len(nrow(qol)), size = 0.8*nrow(qol))
qol_train1<-qol[train_index, ]
qol_test1<-qol[-train_index, ]

prop.table(table(qol_train1$cdthree))

##
##  minor_disease   mild_disease severe_disease
##      0.1699605      0.6459627      0.1840768

prop.table(table(qol_test1$cdthree))

##
##  minor_disease   mild_disease severe_disease
##      0.1760722      0.6478555      0.1760722

set.seed(1234)
qol_model1<-C5.0(qol_train1[ , -c(40, 41, 42)], qol_train1$cdthree,
trials=10)
qol_model1

##
## Call:
## C5.0.default(x = qol_train1[, -c(40, 41, 42)], y =
##  qol_train1$cdthree, trials = 10)
##
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
##
## Number of boosting iterations: 10
## Average tree size: 230.5
##
## Non-standard options: attempt to group attributes

qol_pred1<-predict(qol_model1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
##                   qol_pred1
##                    minor_disease mild_disease severe_disease
##   minor_disease               12           58              8
##   mild_disease                23          239             25
##   severe_disease               3           61             14
##
## Overall Statistics
##
##                Accuracy : 0.5982
##                  95% CI : (0.5509, 0.6442)
##     No Information Rate : 0.8081
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0923
##  Mcnemar's Test P-Value : 4.174e-07
##
## Statistics by Class:
##
```

```
##                      Class: minor_disease Class: mild_disease
## Sensitivity                       0.31579              0.6676
## Specificity                       0.83704              0.4353
## Pos Pred Value                    0.15385              0.8328
## Neg Pred Value                    0.92877              0.2372
## Prevalence                        0.08578              0.8081
## Detection Rate                    0.02709              0.5395
## Detection Prevalence              0.17607              0.6479
## Balanced Accuracy                 0.57641              0.5514
##                      Class: severe_disease
## Sensitivity                         0.2979
## Specificity                         0.8384
## Pos Pred Value                      0.1795
## Neg Pred Value                      0.9096
## Prevalence                          0.1061
## Detection Rate                      0.0316
## Detection Prevalence                0.1761
## Balanced Accuracy                   0.5681
```

We can see that the prediction accuracy with three categories is way lower than the one we did with two categories.

Next, try to build a rule classifier with `OneR()`.

```
set.seed(1234)
qol_1R1<-OneR(cdthree~., data=qol[ , -c(40, 41)])
qol_1R1

## INTERVIEWDATE:
##  < 3.5   -> mild_disease
##  < 28.5  -> severe_disease
##  < 282.0 -> mild_disease
##  < 311.5 -> severe_disease
##  >= 311.5    -> mild_disease
## (1436/2214 instances correct)
summary(qol_1R1)

##
## === Summary ===
##
## Correctly Classified Instances        1436               64.86   %
## Incorrectly Classified Instances       778               35.14   %
## Kappa statistic                          0.022
## Mean absolute error                      0.2343
## Root mean squared error                  0.484
## Relative absolute error                 67.5977 %
## Root relative squared error            116.2958 %
## Total Number of Instances             2214
##
## === Confusion Matrix ===
##
##     a    b    c   <-- classified as
##     0  375    4 |    a = minor_disease
##     0 1422    9 |    b = mild_disease
##     0  390   14 |    c = severe_disease

qol_pred1<-predict(qol_1R1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
```

```
##                qol_pred1
##                 minor_disease mild_disease severe_disease
##   minor_disease             0           78              0
##   mild_disease              0          285              2
##   severe_disease            0           76              2
##
## Overall Statistics
##
##                Accuracy : 0.6479
##                  95% CI : (0.6014, 0.6923)
##     No Information Rate : 0.991
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.012
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: minor_disease Class: mild_disease
## Sensitivity                            NA             0.64920
## Specificity                        0.8239             0.50000
## Pos Pred Value                         NA             0.99303
## Neg Pred Value                         NA             0.01282
## Prevalence                         0.0000             0.99097
## Detection Rate                     0.0000             0.64334
## Detection Prevalence               0.1761             0.64786
## Balanced Accuracy                      NA             0.57460
##                      Class: severe_disease
## Sensitivity                       0.500000
## Specificity                       0.826879
## Pos Pred Value                    0.025641
## Neg Pred Value                    0.994521
## Prevalence                        0.009029
## Detection Rate                    0.004515
## Detection Prevalence              0.176072
## Balanced Accuracy                 0.663440
```

The `OneRule` classifier that is purely based on the value of the `INTERVIEWDATE` has 65% internal classification accuracy, and also 65% external (validation data) prediction accuracy. Although, the latter assessment is a bit misleading, as the vast majority of external validation data are classified in only one class - *mild_disease*.

Finally, let's revisit the `JRip()` classifier with the same three class labels according to `cdthree`.

```
set.seed(1234)
qol_jrip1<-JRip(cdthree~., data=qol[ , -c(40, 41)])
qol_jrip1

## JRIP rules:
## ===========
## (CHARLSONSCORE <= 0) and (AGE <= 50) and (MSA_Q_06 <= 1) and
(QOL_Q_07 >= 1) and (MSA_Q_09 <= 1) => cdthree=minor_disease (35.0/11.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_10 >= 4) and (QOL_Q_07 >= 9) =>
cdthree=severe_disease (54.0/20.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_02 >= 5) and (MSA_Q_09 <= 4) and
(MSA_Q_04 >= 3) => cdthree=severe_disease (64.0/30.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_02 >= 4) and (PH2_Q_01 >= 3) and
(QOL_Q_10 >= 4) and (RACE_ETHNICITY >= 4) => cdthree=severe_disease
(43.0/19.0)
##  => cdthree=mild_disease (2018.0/653.0)
##
## Number of Rules : 5

summary(qol_jrip1)

## === Summary ===
##
## Correctly Classified Instances        1481               66.8925 %
## Incorrectly Classified Instances       733               33.1075 %
## Kappa statistic                          0.1616
## Mean absolute error                      0.3288
## Root mean squared error                  0.4055
## Relative absolute error                 94.8702 %
## Root relative squared error             97.42   %
## Total Number of Instances             2214
## === Confusion Matrix ===
##
##     a    b    c   <-- classified as
##    24  342   13 |    a = minor_disease
##    10 1365   56 |    b = mild_disease
##     1  311   92 |    c = severe_disease

qol_pred1<-predict(qol_jrip1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
##                   qol_pred1
##                    minor_disease mild_disease severe_disease
##   minor_disease                5           70              3
##   mild_disease                 2          275             10
##   severe_disease               0           61             17
##
## Overall Statistics
##
##                Accuracy : 0.6704
##                  95% CI : (0.6245, 0.7141)
##     No Information Rate : 0.9165
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1583
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##                      Class: minor_disease Class: mild_disease
## Sensitivity                    0.71429           0.6773
## Specificity                    0.83257           0.6757
## Pos Pred Value                 0.06410           0.9582
## Neg Pred Value                 0.99452           0.1603
## Prevalence                     0.01580           0.9165
## Detection Rate                 0.01129           0.6208
## Detection Prevalence           0.17607           0.6479
## Balanced Accuracy              0.77343           0.6765
##                          Class: severe_disease
## Sensitivity                     0.56667
## Specificity                     0.85230
## Pos Pred Value                  0.21795
## Neg Pred Value                  0.96438
## Prevalence                      0.06772
## Detection Rate                  0.03837
## Detection Prevalence            0.17607
## Balanced Accuracy               0.70948
```

In terms of the predictive accuracy on the testing data (`qol_test1$cdthree`), we can see from these outputs that the `RIPPER` algorithm performed better (67%) than the `C5.0` decision tree (60%) and similarly to the `OneR` algorithm (65%), which suggests that simple algorithms might outperform complex methods for certain real world case-studies. Later, in Chap. 15, we will provide more details about optimizing and improving classification and prediction performance.

Try to replicate these results with other data from the list of our Case-Studies.

9.9 Assignments 9: Decision Tree Divide and Conquer Classification

9.9.1 Explain These Concepts

- Information Gain Measure
- Impurity
- Entropy
- Gini

9.9.2 Decision Tree Partitioning

Use the SOCR Neonatal Pain data to build and display a decision tree recursively partitioning the data using the provided features and attributes to split the data into similar classes.

- Collect and preprocess the data, e.g., data conversion and variable selection.
- Randomly split the data into training and testing sets.
- Train decision tree models on the data using C5.0 and rpart.

- Evaluate and compare the two models.
- Tune the rpart parameter and repeat the evaluation and comparison again.
- Assess the prediction accuracy and report the confusion matrix.
- Comment on different aspects of the prediction performance.
- Use various *impurity* measures and re-estimate the models.
- Try to use the RWeka package to train decision models and compare the results.
- Try to apply *Random Forest* and obtain variables importance plot.

References

Fischetti, T, Lantz, B, Abedin, J, Mittal, HV, Makhabel, B, Berlinger, E, Illes, F, Badics, M, Banai, A, Daroczi, G (2016) *R: Data Analysis and Visualization*, Packt Publishing Ltd, ISBN 1786460483, 9781786460486.

Liu, H, Gegov, A, Cocea, M. (2015) *Rule Based Systems for Big Data: A Machine Learning Approach*, Springer, Volume 13 (Studies in Big Data), ISBN 3319236962, 9783319236964.

Witten, IH, Frank, E, Hall, MA, Pal, CJ. (2016) *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, Series in Data Management Systems, ISBN 0128043571, 9780128043578.