Decision trees are widely used in data mining and well supported in R (R Core Team, 2014). Decision tree learning deploys a divide and conquer approach, known as recursive partitioning. It is usually implemented as a greedy search using information gain or the Gini index to select the best input variable on which to partition our dataset at each step.

This Module introduces rattle (Williams, 2014) and rpart (Therneau and Atkinson, 2014) for building decision trees. We begin with a step-by-step example of building a decision tree using Rattle, and then illustrate the process using R beginning with Section 14. We cover both classification trees and regression trees.

The required packages for this module include:

```
library(rattle)    # GUI for building trees and fancy tree plot
library(rpart)     # Popular decision tree algorithm
library(rpart.plot)# Enhanced tree plots
library(party)     # Alternative decision tree algorithm
library(partykit)  # Convert rpart object to BinaryTree
library(RWeka)     # Weka decision tree J48.
library(C50)       # Original C5.0 implementation.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command’s documentation and understand what the command does. You can ask for help using the ? command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the `help=` option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

Copyright © 2013-2014 Graham Williams. You can freely copy, distribute, or adapt this material, as long as the attribution is retained and derivative work is provided under the same license.
1 Start Rattle

After starting R (perhaps via RStudio) we can start up `rattle` (Williams, 2014) from the R Console prompt. Simply load the `rattle` package then invoke the `rattle()`, as in:

```r
library(rattle)
rattle()
```

We will see the following Window. Notice the row of buttons, below which we see a series of tabs that we will work through. Remember, in Rattle, that after we set up a particular tab we must press the Execute button to have the tab take effect.
2 Load Example Weather Dataset

Rattle provides a number of sample datasets. We can easily load them into Rattle. By default, Rattle will load the weather dataset.

We load the dataset in two simple steps

1. Click on the Execute button and an example dataset is offered.
2. Click on Yes to load the weather dataset.

We can use this dataset for predictive modelling to predict if it might rain tomorrow (aka statistical classification and supervised learning), or to predict how much rain we might get tomorrow (aka regression analysis).
3 Summary of the Weather Dataset

The weather dataset from rattle consists of daily observations of various weather related data over one year at one location (Canberra Airport). Each observation has a date and location. These are the id variables for this dataset.

The observations include the temperature during the day, humidity, the number of hours of sunshine, wind speed and direction, and the amount of evaporation. These are the input variables for this dataset.

Together with each day’s observations we record whether it rains the following day and how much rain was received. These will be the target variables for this dataset.

Scroll through the list of variables to notice that default roles have been assigned the variables.
4 Model Tab — Decision Tree

We can now click on the Model tab to display the modelling options. The default modelling option is to build a decision tree. Various options to tune the building of a decision tree are provided. Underneath rpart (Therneau and Atkinson, 2014) is used to build the tree, and many more options are available through using rpart() directly, as we will see later in this Module.
5 Build Tree to Predict RainTomorrow

We can simply click the Execute button to build our first decision tree. Notice the time taken to build the tree, as reported in the status bar at the bottom of the window. A summary of the tree is presented in the text view panel. We note that a classification model is built using `rpart()`.

The number of observations from which the model was built is reported. This is 70% of the observations available in the weather dataset. The weather dataset is quite a tiny dataset in the context of data mining, but suitable for our purposes here.

A legend for reading the information in the textual representation of the decision tree is then presented.

The legend indicates that each node is identified (numbered), followed by a split (which will usually be in the form of a test on the value of a variable), the number of entities \( n \) at that node, the number of entities that are incorrectly classified (the loss), the default classification for the node (the \( yval \)), and then the distribution of classes in that node (the \( yprobs \)). The next line indicates that a "*" denotes a terminal node of the tree (i.e., a leaf node—the tree is not split any further at that node).

The distribution is ordered by levels of the class and the order is the same for all nodes. The order here is: No, Yes.
6 Decision Tree Predicting RainTomorrow

Click the Draw button to display a tree (Settings → Advanced Graphics).
7 Evaluate Decision Tree—Error Matrix

Click Evaluate tab—options to evaluate model performance. Click Execute to display simple error matrix. Identify the True/False Positives/Negatives.

![Error Matrix Screenshot](image-url)
8 Decision Tree Risk Chart

Click the Risk type and then Execute.

Exercise: Research how to interpret a risk chart. Explain the risk chart in one or two paragraphs.
9 Decision Tree ROC Curve

Click the ROC type and then Execute.

Exercise: Research how to interpret an ROC curve. Explain the ROC curve in one or two paragraphs.
10 Other Evaluation Plots

Exercise: Research the cost curve, the Hand plots, the Lift chart and the Precision and Sensitivy plots. Produce an example of each and explain each one of them in one or two paragraphs.
11 Score a Dataset

Click the Score type to score a new dataset using model.

A model can be deployed on a dataset to obtain scores or classifications for each observation in the dataset.

By default the testing dataset (if any) will be scored. Otherwise the training dataset is scored. As an alternative, a CSV file can be loaded and scored. This choice of what is scored is controlled by the radio button options.

For binary models a probability score can be recorded. For regression models a value is recorded for each observation. Otherwise a class will be recorded for each observation. This can be controlled by the Class and Probability radio buttons.

The resulting CSV file will include just those variables having a role as Identifier (plus the Target and the Score), or else all of the variables.

The name of a CSV file into which the results will be written will be prompted for.
12 Log of R Commands

Click the Log tab for a history of all your interactions. Save the log contents as a script to repeat what we did.

```
# Rattle is Copyright (c) 2006-2013 Togaware Pty Ltd.
# Rattle timestamp: 2013-07-03 20:09:38 x86_64-pc-linux-gnu
# Rattle version 2.6.27 user 'gvv'
# Export this log textview to a file using the Export button or the Tools
# menu to save a log of all activity. This facilitates repeatability. Exporting
# to file 'myR01.R', for example, allows us to the type in the R Console
# the command source('myR01.R') to repeat the process automatically.
# Generally, we may want to edit the file to suit our needs. We can also directly
# edit this current log textview to record additional information before exporting.
# Saving and loading projects also retains this log.
library(rattle)
# This log generally records the process of building a model. However, with very
# little effort the log can be used to score a new dataset. The logical variable
# 'building' is used to toggle between generating transformations, as when building
# a model, and simply using the transformations, as when scoring a dataset.
building <- TRUE
scoring <- !building
# The colorspace package is used to generate the colours used in plots, if available.
library(colorspace)
```
13 From GUI to R—rpart()

The Log tab shows the call to `rpart()` to build the model. We can click on the Export button to save the script to file and that script can then be used to rerun this model building process, automatically within R.

The command to build the model is presented in the Log tab exactly as it is passed on to R to invoke the model building. It takes a little time to understand it, and the remainder of this module covers interacting directly with R to achieve the same results.

As we will soon learn we would write this sequence of command ourselves as:

```r
set.seed(42)
library(rattle)
library(rpart)

ds <- weather
target <- "RainTomorrow"
nobs <- nrow(ds)
form <- formula(paste(target, "~."))
train <- sample(nobs, 0.70 * nobs)
vars <- c(1,2,23)
model <- rpart(form, ds[train, vars], parms=list(split="information"))
```

Copyright © 2013-2014 Graham@togaware.com Module: DTreesO Page: 13 of 86
14 Prepare Weather Data for Modelling

See the separate Data and Model modules for template for preparing data and building models. In brief, we set ourselves up for modelling the weather dataset with the following commands, extending the simpler example we have just seen.

```r
set.seed(1426)
library(rattle)
data(weather)
dsn <- "weather"
ds <- get(dsn)
id <- c("Date", "Location")
target <- "RainTomorrow"
risk <- "RISK_MM"
ignore <- c(id, if (exists("risk")) risk)
(vars <- setdiff(names(ds), ignore))
## [1] "MinTemp" "MaxTemp" "Rainfall" "Evaporation"
## [5] "Sunshine" "WindGustDir" "WindGustSpeed" "WindDir9am"
## [9] "WindDir3pm" "WindSpeed9am" "WindSpeed3pm" "Humidity9am"
## [13] "Humidity3pm" "Pressure9am" "Pressure3pm" "Cloud9am"
....
inputs <- setdiff(vars, target)
(nobs <- nrow(ds))
## [1] 366
(numerics <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.numeric))]))
## [1] "MinTemp" "MaxTemp" "Rainfall" "Sunshine"
## [5] "WindDir9am" "WindDir3pm" "WindSpeed9am" "WindSpeed3pm"
## [9] "Humidity9am" "Humidity3pm" "Pressure9am" "Pressure3pm"
## [13] "Cloud9am" "Cloud3pm"
....
(categoricals <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.factor))]))
## [1] "Evaporation" "WindGustDir" "WindGustSpeed" "Temp9am"
## [5] "Temp3pm"
(form <- formula(paste(target, "~ .")))
## RainTomorrow ~ .
length(train <- sample(nobs, 0.7*nobs))
## [1] 256
length(test <- setdiff(seq_len(nobs), train))
## [1] 110
actual <- ds[test, target]
risks <- ds[test, risk]
```
15 Review the Dataset

It is always a good idea to review the data.

```r
dim(ds)
## [1] 366  24

names(ds)
## [1] "Date"   "Location" "MinTemp" "MaxTemp"
## [5] "Rainfall" "Evaporation" "Sunshine" "WindGustDir"
## [9] "WindGustSpeed" "WindDir9am" "WindDir3pm" "WindGust9am"
## [13] "WindSpeed3pm" "Humidity9am" "Humidity3pm" "Pressure9am"
## [17] "Pressure3pm" "Cloud9am" "Cloud3pm" "Temp9am"

head(ds)
##   Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2007-11-01 Canberra  8.0  24.3     0.0        3.4          6.3
## 2 2007-11-02 Canberra 14.0  26.9     3.6        4.4          9.7
## 3 2007-11-03 Canberra 13.7  23.4     3.6        5.8          3.3
## 4 2007-11-04 Canberra 13.3  15.5    39.8        7.2          9.1

tail(ds)
## 361 2008-10-26 Canberra  7.9  26.1     0.0        6.8          3.5
## 362 2008-10-27 Canberra  9.0  30.7     0.0        7.6         12.1
## 363 2008-10-28 Canberra  7.1  28.4     0.0       11.6         12.7
## 364 2008-10-29 Canberra 12.5  19.9     0.0        8.4          5.3

str(ds)
## 'data.frame': 366 obs. of 24 variables:
##  $ Date       : Date, format: "2007-11-01" "2007-11-02" ...
##  $ Location   : Factor w/ 49 levels "Adelaide","Albany",...: 10 10 10 10 ...
##  $ MinTemp    : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
##  $ MaxTemp    : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...

summary(ds)
##    Date          Location     MinTemp     MaxTemp
##  Min. :2007-11-01: 366  Min. : -5.30  Min. : 7.6
##  1st Qu.:2008-01-31:   0  1st Qu.: 2.30  1st Qu.:15.0
##  Median :2008-05-01:   0  Median : 7.45  Median :19.6
##  Mean :2008-05-01:   0  Mean : 7.27  Mean :20.6
```

Copyright © 2013-2014 Graham@togaware.com Module: DTreesO Page: 15 of 86
16 Build Decision Tree Model

Build a decision tree using \texttt{rpart()}. Once the different variables have been defined (form, ds, train, and vars) this same command can be re-used.

\begin{verbatim}
model <- rpart(formula=form, data=ds[train, vars])
\end{verbatim}

Notice in the above command we have named each of the arguments. If we have a look at the structure of \texttt{rpart} we see that the arguments are in their expected order, and hence the use of the argument names, \texttt{formula=} and \texttt{data=} is optional.

\begin{verbatim}
str(rpart)
## function (formula, data, weights, subset, na.action=na.rpart, method,
## model=FALSE, x=FALSE, y=TRUE, parms, control, cost, ...)
model <- rpart(form, ds[train, vars])
\end{verbatim}

Whilst they are optional, they can assist in reading the code, and so it is recommended that we use the argument names in function calls.

A textual presentation of the model is concise but informative, once we learn how to read it. Note this tree is different to the previous one, since we have randomly selected a different dataset to train the model.

\begin{verbatim}
model
## n= 256
##
## node), split, n, loss, yval, (yprob)
## * denotes terminal node
##
## 1) root 256 38 No (0.85156 0.14844)
## 2) Humidity3pm< 71 238 25 No (0.89496 0.10504)
## 4) Pressure3pm>=1010 208 13 No (0.93750 0.06250) *
## 5) Pressure3pm< 1010 30 12 No (0.60000 0.40000)
## 10) Sunshine>=9.95 14 1 No (0.92857 0.07143) *
## 11) Sunshine< 9.95 16 5 Yes (0.31250 0.68750) *
## 3) Humidity3pm>=71 18 5 Yes (0.27778 0.72222) *
\end{verbatim}

Refer to Section 5 for an explanation of the format of the textual presentation of the decision tree. The first few lines indicate the number of observation from which the tree was built ($n =$) and then a legend for reading the information in the textual representation of the tree.

The legend indicates that a node number will be provided, followed by a split (which will usually be in the form of a variable operation and value), the number of entities $n$ at that node, the number of entities that are incorrectly classified (the loss), the default classification for the node (the \texttt{yval}), and then the distribution of classes in that node (the \texttt{yprobs}). The distribution is ordered by class and the order is the same for all nodes. The next line indicates that a \texttt{*} denotes a terminal node of the tree (i.e., a leaf node—the tree is not split any further at that node).
17 Summary of the Model

```r
summary(model)
```

## Call:
## rpart(formula=form, data=ds[train, vars])
## n= 256
##
## CP nsplit rel error xerror xstd
## 1 0.21053 0 1.0000 1.0000 0.1497
## 2 0.07895 1 0.7895 0.9474 0.1464
## 3 0.01000 3 0.6316 1.0263 0.1513
##
## Variable importance
## Humidity3pm Sunshine Pressure3pm Temp9am Pressure9am Temp3pm
## 25 17 14 9 8 8
## Cloud3pm MaxTemp MinTemp
## 7 6 5
##
## Node number 1: 256 observations, complexity param=0.2105
## predicted class=No  expected loss=0.1484  P(node) =1
## class counts: 218 38
## probabilities: 0.852 0.148
## left son=2 (238 obs) right son=3 (18 obs)
## Primary splits:
## Humidity3pm < 71 to the left, improve=12.750, (0 missing)
## Pressure3pm < 1011 to the right, improve=11.240, (0 missing)
## Cloud3pm < 6.5 to the left, improve=11.010, (0 missing)
## Sunshine < 6.45 to the right, improve= 9.975, (2 missing)
## Pressure9am < 1018 to the right, improve= 8.381, (0 missing)
## Surrogate splits:
## Sunshine < 0.75 to the right, agree=0.949, adj=0.278, (0 split)
## Pressure3pm < 1002 to the right, agree=0.938, adj=0.111, (0 split)
## Temp3pm < 7.6 to the right, agree=0.938, adj=0.111, (0 split)
## Pressure9am < 1005 to the right, agree=0.934, adj=0.056, (0 split)
##
## Node number 2: 238 observations, complexity param=0.07895
## predicted class=No  expected loss=0.105  P(node) =0.9297
## class counts: 213 25
## probabilities: 0.895 0.105
## left son=4 (208 obs) right son=5 (30 obs)
## Primary splits:
## Pressure3pm < 1010 to the right, improve=5.973, (0 missing)
## Cloud3pm < 6.5 to the left, improve=4.475, (0 missing)
```

In the following pages we dissect the various components of this summary.
18 Complexity Parameter

We can print a table of optimal prunings based on a complexity parameter using `printcp()`. The data is actually stored as `model$cptable`.

```r
printcp(model)
```

```
##
## Classification tree:
## rpart(formula=form, data=ds[train, vars])
##
## Variables actually used in tree construction:
## [1] Humidity3pm Pressure3pm Sunshine
##
## Root node error: 38/256=0.15
##
## n= 256
##
## CP nsplit rel error xerror xstd
## 1 0.211 0 1.00 1.00 0.15
## 2 0.079 1 0.79 0.95 0.15
## 3 0.010 3 0.63 1.03 0.15
```

Exercise: Research what the complexity parameter does. Explain/illustrate it in one or two paragraphs.
19 Complexity Parameter Plot

The `plotcp()` plots the cross-validation results. Here we see a set of possible cost-complexity prunings of the tree. We might choose to prune using the leftmost complexity parameter which has a mean below the horizontal line.

```
plotcp(model)
```
20 More Interesting Complexity Behaviour

We don’t see much of the true behaviour of the complexity parameter with out small dataset. We can build a more interesting model based on the larger `weatherAUS` dataset from `rattle`.

If we build a default model then we can plot the complexity parameter as before.

```r
  tmodel <- rpart(form, weatherAUS[vars])
  plotcp(tmodel)
```

![Complexity Parameter Plot](image.png)
21 More Interesting Complexity Behaviour—cp=0

We can set the cp= argument to be 0, so that no pruning of the tree is performed.

```r
tmodel <- rpart(form, weatherAUS[vars], control=rpart.control(cp=0))
plotcp(tmodel)
```

Notice that as we continue to build the model, by recursive partitioning, the model gets more complex but the performance does not improve, and in fact over time the model performance starts to deteriorate because of overfitting.
22 More Interesting Complexity Behaviour—Numeric View

We can look at the raw data to have a more precise and detailed view of the data. Here we only list specific rows from the complexity parameter table.

```r
tmodel$cptable[c(1:5,22:29, 80:83),]
```

<table>
<thead>
<tr>
<th>#</th>
<th>CP</th>
<th>nsplit</th>
<th>rel error</th>
<th>xerror</th>
<th>xstd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.890e-01</td>
<td>0</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.006125</td>
</tr>
<tr>
<td>2</td>
<td>2.909e-02</td>
<td>1</td>
<td>0.8110</td>
<td>0.8175</td>
<td>0.005685</td>
</tr>
<tr>
<td>3</td>
<td>7.486e-03</td>
<td>3</td>
<td>0.7528</td>
<td>0.7563</td>
<td>0.005515</td>
</tr>
<tr>
<td>4</td>
<td>6.440e-03</td>
<td>4</td>
<td>0.7453</td>
<td>0.7538</td>
<td>0.005508</td>
</tr>
<tr>
<td>5</td>
<td>4.277e-03</td>
<td>8</td>
<td>0.7196</td>
<td>0.7323</td>
<td>0.005445</td>
</tr>
<tr>
<td>22</td>
<td>5.347e-04</td>
<td>57</td>
<td>0.6611</td>
<td>0.6912</td>
<td>0.005320</td>
</tr>
<tr>
<td>23</td>
<td>5.225e-04</td>
<td>59</td>
<td>0.6600</td>
<td>0.6898</td>
<td>0.005315</td>
</tr>
<tr>
<td>24</td>
<td>4.861e-04</td>
<td>63</td>
<td>0.6579</td>
<td>0.6894</td>
<td>0.005314</td>
</tr>
<tr>
<td>25</td>
<td>4.375e-04</td>
<td>67</td>
<td>0.6560</td>
<td>0.6894</td>
<td>0.005314</td>
</tr>
<tr>
<td>26</td>
<td>3.889e-04</td>
<td>69</td>
<td>0.6551</td>
<td>0.6897</td>
<td>0.005315</td>
</tr>
<tr>
<td>27</td>
<td>3.727e-04</td>
<td>72</td>
<td>0.6540</td>
<td>0.6901</td>
<td>0.005316</td>
</tr>
<tr>
<td>28</td>
<td>3.646e-04</td>
<td>75</td>
<td>0.6528</td>
<td>0.6899</td>
<td>0.005316</td>
</tr>
<tr>
<td>29</td>
<td>3.403e-04</td>
<td>77</td>
<td>0.6521</td>
<td>0.6899</td>
<td>0.005316</td>
</tr>
<tr>
<td>80</td>
<td>3.646e-05</td>
<td>1607</td>
<td>0.4535</td>
<td>0.7742</td>
<td>0.005566</td>
</tr>
<tr>
<td>81</td>
<td>3.472e-05</td>
<td>1640</td>
<td>0.4522</td>
<td>0.7780</td>
<td>0.005577</td>
</tr>
<tr>
<td>82</td>
<td>3.240e-05</td>
<td>1658</td>
<td>0.4516</td>
<td>0.7789</td>
<td>0.005580</td>
</tr>
<tr>
<td>83</td>
<td>3.093e-05</td>
<td>1679</td>
<td>0.4509</td>
<td>0.7809</td>
<td>0.005585</td>
</tr>
</tbody>
</table>
```

See how the relative error continues to decrease as the tree becomes more complex, but the cross validated error decreases and then starts to increase! We might choose a sensible value of \( cp = \) from this table.

**Exercise:** Choose some different values of \( cp = \) based on the table above and explore the effect. Explain what initially appears to be an oddity about the different looking trees we get.
23 Variable Importance

Exercise: Research how the variable importance is calculated. Explain it in one or two paragraphs.

```r
model$variable.importance

## Humidity3pm  Sunshine Pressure3pm Temp9am Pressure9am Temp3pm
## Cloud3pm     MaxTemp MinTemp Rainfall
## 3.6436       3.0330    2.8339    0.1991
```

24 Node Details and Surrogates

Exercise: In your own words explain how to read the node information. Research the concept of surrogates to handle missing values ad in one or two paragraphs, explain it.
25 Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```r
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual, risks)
```

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```r
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
## Predicted
## Actual No Yes
## No 65 9
## Yes 11 15
....
26 Visualise Decision Tree as Rules

We can use the following function to print the paths through the decision tree as rules.

```r
asRules.rpart <- function(model)
{
  if (!inherits(model, "rpart")) stop("Not a legitimate rpart tree")
  # Get some information.
  frm <- model$frame
  names <- row.names(frm)
  ylevels <- attr(model, "ylevels")
  ds.size <- model$frame[,1]$n
  # Print each leaf node as a rule.
  for (i in 1:nrow(frm))
  {
    if (frm[i,1] == "<leaf>")
    {
      # The following [,5] is hardwired - needs work!
      cat("\n")
      cat(sprintf(" Rule number: %s ", names[i]))
      cat(sprintf(" [yval=%s cover=%d (%.0f%%) prob=%0.2f]\n",
                   ylevels[frm[i,]$yval], frm[i,]$n,
                   round(100*frm[i,]$n/ds.size), frm[i,]$yval2[,5])
      pth <- path.rpart(model, nodes=as.numeric(names[i]), print.it=FALSE)
      cat(sprintf(" %s\n", unlist(pth)[-1]), sep=""))
    }
  }
}
```

```
asRules(model)
##
## Rule number: 4 [yval=No cover=208 (81%) prob=0.06]
##  Humidity3pm< 71
##  Pressure3pm>=1010
##
## Rule number: 10 [yval=No cover=14 (5%) prob=0.07]
##  Humidity3pm< 71
##  Pressure3pm< 1010
##  Sunshine>=9.95
##
## Rule number: 11 [yval=Yes cover=16 (6%) prob=0.69]
##  Humidity3pm< 71
##  Pressure3pm< 1010
##  Sunshine< 9.95
....
```

Copyright © 2013-2014 Graham@togaware.com Module: DTreesO Page: 26 of 86
27  Visualise Decision Trees

The default plot of the model is quite basic. In this plot we move to the left in the binary tree if the condition is true.

```r
plot(model)
text(model)
```
28 Visualise Decision Trees: Uniform

plot(model, uniform=TRUE)

text(model)
29 Visualise Decision Trees: Extra Information

plot(model, uniform=TRUE)

text(model, use.n=TRUE, all=TRUE, cex=.8)
30  Fancy Plot

The *rattle* package provides a fancy plot based on the functionality provided by *rpart.plot* (Miliborrow, 2014) and using colours from *RColorBrewer* (Neuwirth, 2011), tuned for use in *rattle*. The same options can be passed directly to *prp()* to achieve the same plot and colours, as we see in the following pages. The colours are specially constructed in *rattle*.
31 Enhanced Plots: Default

Stephen Milborrow’s `rpart.plot` provides a suite of enhancements to the basic `rpart.plot()` command. The following pages exhibit the various (and quite extensive) options provided by `rpart.plot` and specifically `prp()`.
32 Enhanced Plots: Favourite

This is a plot that I find particularly useful, neat, and informative, particularly for classification models.

The leaf nodes are each labelled with the predicted class. They are neatly lined up at the bottom of the figure (\texttt{fallen.leaves=TRUE}), to visually reinforce the structure. We can see the straight lines from the top to the bottom which lead to decisions quickly, whilst the more complex paths need quite a bit more information in order to make a decision.

Each node includes the probability for each class, and the percentage of observations associated with the node (\texttt{extra=104}). The node numbers are included (\texttt{nn=TRUE}) so we can cross reference each node to the text decision tree, or other decision tree plots, or a rule set generated from the decision tree.

Using a dotted line type (\texttt{branch.lty=3}) removes some of the focus from the heavy lines and back to the nodes, whilst still clearly identifying the links. The grey shadow is an optional nicety.

\begin{verbatim}
prp(model, type=2, extra=104, nn=TRUE, fallen.leaves=TRUE, faclen=0, varlen=0, shadow.col="grey", branch.lty=3)
\end{verbatim}
33  Enhanced Plot: With Colour

```r
col <- c("#FD8D3C", "#FD8D3C", "#FD8D3C", "#BCBDDC", "#FDD0A2", "#FD8D3C", "#BCBDDC")
prp(model, type=2, extra=104, nn=TRUE, fallen.leaves=TRUE, faclen=0, varlen=0, shadow.col="grey", branch.lty=3, box.col=col)
```

The `fancyRpartPlot()` function from `rattle` (Williams, 2014) generates scaled colour for colouring the boxes depending on the decision and the strength. The hard work is in generating the scaled colours for the nodes. Here we use other palettes for the colours rather than those used by `fancyRpartPlot()`.
34 Enhanced Plots: Label all Nodes

```
prp(model, type=1)
```

Here all nodes are labelled with the majority class.
35  Enhanced Plots: Labels Below Nodes

prp(model, type=2)
36 Enhanced Plots: Split Labels

Humidity < 71

Pressure >= 1010

Sunshine >= 9.9

prp(model, type=3)
37 Enhanced Plots: Interior Labels

```
prp(model, type=4)
```

---

Copyright © 2013-2014 Graham@togaware.com
38  Enhanced Plots: Number of Observations

```r
prp(model, type=2, extra=1)
```
39 Enhanced Plots: Add Percentage of Observations

```
prp(model, type=2, extra=101)
```
40 Enhanced Plots: Classification Rate

```r
prp(model, type=2, extra=2)
```

Copyright © 2013-2014 Graham@togaware.com  Module: DTreesO  Page: 40 of 86
41 Enhanced Plots: Add Percentage of Observations

prp(model, type=2, extra=102)

Notice the pattern? When we add 100 to the extra option then the percentage of observations located with each node is then included in the plot.
42 Enhanced Plots: Misclassification Rate

```
prp(model, type=2, extra=3)
```
Enhanced Plots: Probability per Class

```
prp(model, type=2, extra=4)
```
Enhanced Plots: Add Percentage Observations

```
prp(model, type=2, extra=104)
```

This is a particularly informative plot for classification models. Each node includes the probability of each class. Each node also includes the percentage of the training dataset that corresponds to that node.
45  Enhanced Plots: Only Probability Per Class

prp(model, type=2, extra=5)
Enhanced Plots: Probability of Second Class

\[ \text{Humidity} < 71 \]

\[ \text{Pressure} \geq 1010 \]

\[ \text{Sunshine} \geq 9.9 \]

\[ \text{No} \]

\[ 0.15 \]

\[ \text{No} \]

\[ 0.11 \]

\[ \text{No} \]

\[ 0.06 \]

\[ \text{No} \]

\[ 0.40 \]

\[ \text{No} \]

\[ 0.07 \]

\[ \text{Yes} \]

\[ 0.69 \]

\[ \text{Yes} \]

\[ 0.72 \]

\[ \text{Yes no} \]

\[
\text{prp(model, type=2, extra=6)}
\]

This is particularly useful for binary classification, as here, where the second class is usually the positive response.
Enhanced Plots: Add Percentage Observations

```
Humidity < 71
Pressure >= 1010
Sunshine >= 9.9
No 0.15 100%
No 0.11 93%
No 0.06 81%
No 0.40 12%
No 0.07 5%
Yes 0.69 6%
Yes 0.72 7%
```

```
prp(model, type=2, extra=106)
```

This is a particularly informative plot for binary classification tasks. Each node includes the probability of the second class, which is usually the positive class in a binary classification dataset. Each node also includes the percentage of the training dataset that corresponds to that node.
48 Enhanced Plots: Only Probability of Second Class

```
prp(model, type=2, extra=7)
```

```
Humidity < 71

Pressure >= 1010

Sunshine >= 9.9

yes

0.15

0.11

no

0.72

0.06

0.40

Sunshine >= 9.9

0.07

0.69
```
Enhanced Plots: Probability of the Class

```
prp(model, type=2, extra=8)
```
50  Enhanced Plots: Overall Probability

```
prp(model, type=2, extra=9)
```
51 Enhanced Plots: Percentage of Observations

prp(model, type=2, extra=100)
52 Enhanced Plots: Show the Node Numbers

We now take our favourite plot (type=2 and extra=106) and explore some of the other options available for prp() from rpart.plot.

Here we add the node numbers as they appear in the textual version of the model, and also often for rule sets generated from the decision tree.

```
prp(model, type=2, extra=106, nn=TRUE)
```
53 Enhanced Plots: Show the Node Indices

```
prp(model, type=2, extra=106, nn=TRUE, ni=TRUE)
```

These are the row numbers of the nodes within the model object’s `frame` component.
54  Enhanced Plots: Line up the Leaves

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE)
```
55  Enhanced Plots: Angle Branch Lines

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    branch=0.5)
```
56 Enhanced Plots: Do Not Abbreviate Factors

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE, faclen=0)
```
57   Enhanced Plots: Add a Shadow to the Nodes

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    shadow.col="grey")
```
Enhanced Plots: Draw Branches as Dotted Lines

The `branch.lty=` option allows us to specify the type of line to draw for the branches. A dotted line is attractive as it reduces the dominance of the branches whilst retaining the node connections. Other options are just the standard values for line type in R:

```
0 blank, 1 solid, 2 dashed "44", 3 dotted "13", 4 dotdash "1343",
5 longdash "73", 6 two-dash "2262"
```

The line type can also be specified as an even length string of up eight characters of the hex digits (0-9, a-f). The pairs specify the length in pixels of the line and the blank. Thus `lty="44"` is the same as `lty=2`:

```r
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty=2)
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty="dashed")
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty="44")
```
59  Enhanced Plots: Other Options

Exercise: Explore examples of split.cex=1.2, split.prefix="is "}, split.suffix="?", col=cols, border.col=cols, split.box.col="lightgray", split.border.col="darkgray", split.round=.5, and other parameters you might discover.
60  Party Tree

The party (Hothorn et al., 2013) package can be used to draw rpart decision trees using as.party() from partykit (Hothorn and Zeileis, 2014) which can be installed from R-Forge:

```r
install.packages("partykit", repos="http://R-Forge.R-project.org")
library(partykit)
class(model) ## [1] "rpart"
plot(as.party(model))
```

The textual presentation of an rpart decision tree can also be improved using party.

```r
print(as.party(model))
##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine + WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am + WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am + Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## | [2] Humidity3pm < 71
## | | [3] Pressure3pm >= 1010.25: No (n=208, err=6%)
## | | [4] Pressure3pm < 1010.25
## | | | [5] Sunshine >= 9.95: No (n=14, err=7%)
## | | | [6] Sunshine < 9.95: Yes (n=16, err=31%)
## | [7] Humidity3pm >= 71: Yes (n=18, err=28%)
##
```

The textual presentation of an rpart decision tree can also be improved using party.
61 Conditional Decision Tree

Note that we are using the newer version of the `ctree()` function as is provided by `partykit` (Hothorn and Zeileis, 2014). One advantage of the newer version is that `predict()` with `type="prob"` works just like other `predict()` methods (returns a matrix rather than a list).

```r
library(partykit)
model <- ctree(formula=form, data=ds[train, vars])
model
```

```
##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
##   WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +
##   WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
##   Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
##    [1] root
##    [2] Sunshine <= 6.4
##    | [3] Pressure3pm <= 1015.9: Yes (n=29, err=24%)
##    | [4] Pressure3pm > 1015.9: No (n=36, err=8%)
##    [5] Sunshine > 6.4
##    | [6] Pressure3pm <= 1010.2: No (n=25, err=28%)
##    | [7] Pressure3pm > 1010.2: No (n=166, err=4%)
##
## Number of inner nodes: 3
## Number of terminal nodes: 4
```
Conditional Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```r
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual, risks)
```

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```r
predicted <- predict(model, ds[test, vars], type="response")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2182
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
## Predicted
## Actual No Yes
## No 67 7
## Yes 15 11
```
63 CTree Plot

```r
plot(model)
```

![Decision Tree Diagram](image.png)
64  Weka Decision Tree

The suite of algorithms implemented in Weka are also available to R thanks to RWeka (Hornik, 2014).

```r
library(RWeka)
model <- J48(formula=form, data=ds[train, vars])
model
```

```
# J48 pruned tree
# ------------------
# 
# Humidity3pm <= 70
# | Pressure3pm <= 1015.8
# | | Sunshine <= 9.2
# | | | Evaporation <= 4.4: No (17.0/4.0)
# | | | Evaporation > 4.4: Yes (17.0/2.0)
# | | Sunshine > 9.2: No (49.0/3.0)
# | Pressure3pm > 1015.8: No (132.0/1.0)
# Humidity3pm > 70: Yes (16.0/4.0)
# 
# Number of Leaves : 5
# 
# Size of the tree : 9
```
65 Weka Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```r
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual, risks)
```

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```r
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
## Predicted
## Actual No Yes
## No 66 8
## Yes 12 14
```

Copyright © 2013-2014 Graham@togaware.com  Module: DTreesO  Page: 65 of 86
66  Weka Decision Tree Plot Using Party

plot(as.party(model))

We can also display a textual version using party

print(as.party(model))

```
##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
## WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm +WindSpeed9am +
## WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
## Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## |   [2] Humidity3pm <= 70
## |     |   [3] Pressure3pm <= 1015.8
## |     |     |   [4] Sunshine <= 9.2
## ....
```
67 The Original C5.0 Implementation

The C50 (Kuhn et al., 2014) package interfaces the original C code of the C5.0 implementation by Ross Quinlan, the developer of the decision tree induction algorithm.

```R
library(C50)
model <- C5.0(form, ds[train, vars])
```

```
## Call:
## C5.0.formula(formula=form, data=ds[train, vars])
##
## Classification Tree
## Number of samples: 256
## Number of predictors: 20
##
## Tree size: 8
##
## Non-standard options: attempt to group attributes
```

```R
C5imp(model)
```

```
## Overall
## Humidity3pm 100.00
## Pressure3pm  97.27
## Sunshine    34.77
## Evaporation 15.23
## WindGustSpeed 7.81
## WindDir3pm   7.03
## MinTemp      0.00
## MaxTemp      0.00
## Rainfall     0.00
## WindGustDir  0.00
## WindDir9am   0.00
## WindSpeed9am 0.00
## WindSpeed3pm 0.00
## Humidity9am  0.00
## Pressure9am  0.00
## Cloud9am     0.00
## Cloud3pm     0.00
## Temp9am      0.00
## Temp3pm      0.00
## RainToday    0.00
```
68  C5.0 Summary

```r
summary(model)
```

```r
## Call:
## C5.0.formula(formula=form, data=ds[train, vars])
##
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 256 cases (21 attributes) from undefined.data
##
## Decision tree:
##
## Humidity3pm > 70:
##  :...WindDir3pm in [N-ESE]: Yes (7)
##  :  WindDir3pm in [SE-NNW]:
##  : :...Pressure3pm <= 1014.5: Yes (7/1)
##  :  Pressure3pm > 1014.5: No (4)
## Humidity3pm <= 70:
##  :...Pressure3pm > 1015.8: No (149/3)
##  Pressure3pm <= 1015.8:
##  :...Sunshine > 9.2: No (50/3)
##   Sunshine <= 9.2:
##   :...Evaporation > 4.4: Yes (19/4)
##    Evaporation <= 4.4:
##    :...WindGustSpeed <= 30: Yes (2)
##    WindGustSpeed > 30: No (18/2)
##
## Evaluation on training data (256 cases):
##
## Decision Tree
## ----------------
## Size   Errors
## 8     13 (5.1%) <<
##
## (a)  (b) <-classified as ....
```
69  C5.0 Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```r
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual, risks)
```

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```r
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2182
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
# Table of error counts
```

---

Copyright © 2013-2014 Graham@togaware.com  Module: DTreesO  Page: 69 of 86
70  C5.0 Rules Model

```r
library(C50)
model <- C5.0(form, ds[train, vars], rules=TRUE)

model
```

```r
##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars], rules=TRUE)
##
## Rule-Based Model
## Number of samples: 256
## Number of predictors: 20
##
## Number of Rules: 7
##
## Non-standard options: attempt to group attributes
```

```r
C5imp(model)
```

```r
## Overall
## Humidity3pm  75.00
## Pressure3pm  74.61
## Sunshine  51.17
## WindDir3pm  44.53
## Evaporation  41.80
## WindGustSpeed  32.42
## MinTemp  0.00
## MaxTemp  0.00
## Rainfall  0.00
## WindGustDir  0.00
## WindDir9am  0.00
## WindSpeed9am  0.00
## WindSpeed3pm  0.00
## Humidity9am  0.00
## Pressure9am  0.00
## Cloud9am  0.00
## Cloud3pm  0.00
## Temp9am  0.00
## Temp3pm  0.00
## RainToday  0.00
```
71  C5.0 Rules Summary

```r
summary(model)
```

```r
## Call:
## C5.0.formula(formula=form, data=ds[train, vars], rules=TRUE)
##
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 256 cases (21 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (149/3, lift 1.1)
## Humidity3pm <= 70
## Pressure3pm > 1015.8
## -> class No [0.974]
##
## Rule 2: (107/3, lift 1.1)
## Sunshine > 9.2
## -> class No [0.963]
##
## Rule 3: (107/3, lift 1.1)
## WindDir3pm in [SE-NNW]
## Pressure3pm > 1014.5
## -> class No [0.963]
##
## Rule 4: (83/3, lift 1.1)
## Evaporation <= 4.4
## WindGustSpeed > 30
## Humidity3pm <= 70
## -> class No [0.953]
##
## Rule 5: (7, lift 6.0)
## WindDir3pm in [N-ESE]
## Humidity3pm > 70
## -> class Yes [0.889]
##
## Rule 6: (11/1, lift 5.7)
```

...
C5.0 Rules Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```r
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual, risks)
```

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```r
predicted <- predict(model, ds[test, vars], type="class")
sum(ds[test, target] != predicted)/length(predicted) # Overall error rate
## [1] 0.2273
round(100*table(ds[test, target], predicted, dnn=c("Actual", "Predicted"))/length(predicted))
## Predicted
##       Actual No Yes
##   No   66  8
##   Yes  15 11
```

Copyright © 2013-2014 Graham@togaware.com
73 Regression Trees

The discussion so far has dwelt on classification trees. Regression trees are similarly well catered for in R.

We can plot regression trees as with classification trees, but the node information will be different and some options will not make sense. For example, `extra= only makes sense for 100 and 101.

First we will build regression tree:

```r
target <- "RISK_MM"
vars <- c(inputs, target)
form <- formula(paste(target, "~ "))
(model <- rpart(formula=form, data=ds[train, vars]))
```

```
## n= 256
##
## node), split, n, deviance, yval
## * denotes terminal node
##
## 1) root 256 2580.000 0.9656
## 2) Sunshine>=0.3 248 1169.000 0.6460
## 4) WindGustSpeed< 60 233 343.600 0.3957
## 8) Cloud3pm< 6.5 173 84.110 0.1653 *
## 9) Cloud3pm>=6.5 60 223.900 1.0600
## 18) Pressure3pm>=1016 36 65.320 0.4333 *
## 19) Pressure3pm< 1016 24 123.200 2.0000
## 38) Sunshine>=9.15 7 2.777 0.3429 *
## 39) Sunshine< 9.15 17 93.280 2.6820 *
## 5) WindGustSpeed>=60 15 584.200 4.5330 *
## 3) Sunshine< 0.3 8 599.800 10.8800 *
```
74 Visualise Regression Trees

```r
plot(model)
text(model)
```
75 Visualise Regression Trees: Uniform

```r
plot(model, uniform=TRUE)
text(model)
```
76 Visualise Regression Trees: Extra Information

```r
plot(model, uniform=TRUE)
text(model, use.n=TRUE, all=TRUE, cex=.8)
```
77 Fancy Plot of Regression Tree

\texttt{fancyRpartPlot(model)}

- **Sunshine >= 0.3**
  - \texttt{yes}
  - \texttt{no}

- **WindGustSpeed < 60**

- **Cloud3pm < 6.5**
  - **Pressure3pm >= 1016**
    - \texttt{yes}
    - \texttt{no}

- **Sunshine >= 9.1**

Rattle 2014–Aug–03 17:59:21 gjw
Enhanced Plot of Regression Tree: Default

prp(model)
Enhanced Plot of Regression Tree: Favourite

\texttt{prp(model, type=2, extra=101, nn=TRUE, fallen.leaves=TRUE, faclen=0, varlen=0, shadow.col="grey", branch.lty=3)}
80 Party Regression Tree

The tree drawing facilities of party (Hothorn et al., 2013) can again be used to draw the rpart regression tree using as.party().

```r
class(model)
## [1] "rpart"
plot(as.party(model))
```

Notice the visualisation of the predictions—this is particularly informative. We have, in an instant, a view of the conditions when there is little or no rain, compared to significant rain.
81  Conditional Regression Tree

We can also build a regression tree using party (Hothorn et al., 2013).

```r
model <- ctree(formula=form, data=ds[train, vars])
model
```

```r
## Model formula:
## RISK_MM ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
##     WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +
##     WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
##     Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## | [2] Sunshine <= 0.2: 11 (n=8, err=600)
## | [3] Sunshine > 0.2
## | | [4] Pressure3pm <= 1006.5: 5 (n=11, err=445)
## | | [5] Pressure3pm > 1006.5
## | | | [6] Cloud3pm <= 6
## | | | | [7] Pressure3pm <= 1013.8
## | | | | | [8] Humidity9am <= 73: 0 (n=31, err=12)
## | | | | | [9] Humidity9am > 73: 2 (n=11, err=41)
## | | | | [10] Pressure3pm > 1013.8: 0 (n=133, err=5)
##
## Number of inner nodes: 5
## Number of terminal nodes: 6
```
82 CTree Plot

```r
plot(model)
```
83 Weka Regression Tree

Weka’s J48() does not support regression trees.
84 Further Reading and Acknowledgements

The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from http://datamining.togaware.com, including the Datamining Desktop Survival Guide.

This chapter is one of many chapters available from http://HandsOnDataScience.com. In particular follow the links on the website with a * which indicates the generally more developed chapters.

- http://www.milbo.org/rpart-plot/prp.pdf: Plotting rpart trees with prp, by Stephen Milborrow. Stephen (author of rpart.plot) showcases the options of prp() and was the inspiration for the showcase of prp() included here.
85 References


