Data Science with R
Decision Trees

Graham.Williams@togaware.com

6th April 2014

Visit `http://onepager.togaware.com/` for more OnePageR's.

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 begining 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.

## Error:  there is no package called 'RWeka'

library(C50)          # Original C5.0 implementation.
```

As we work through this module, 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)
```

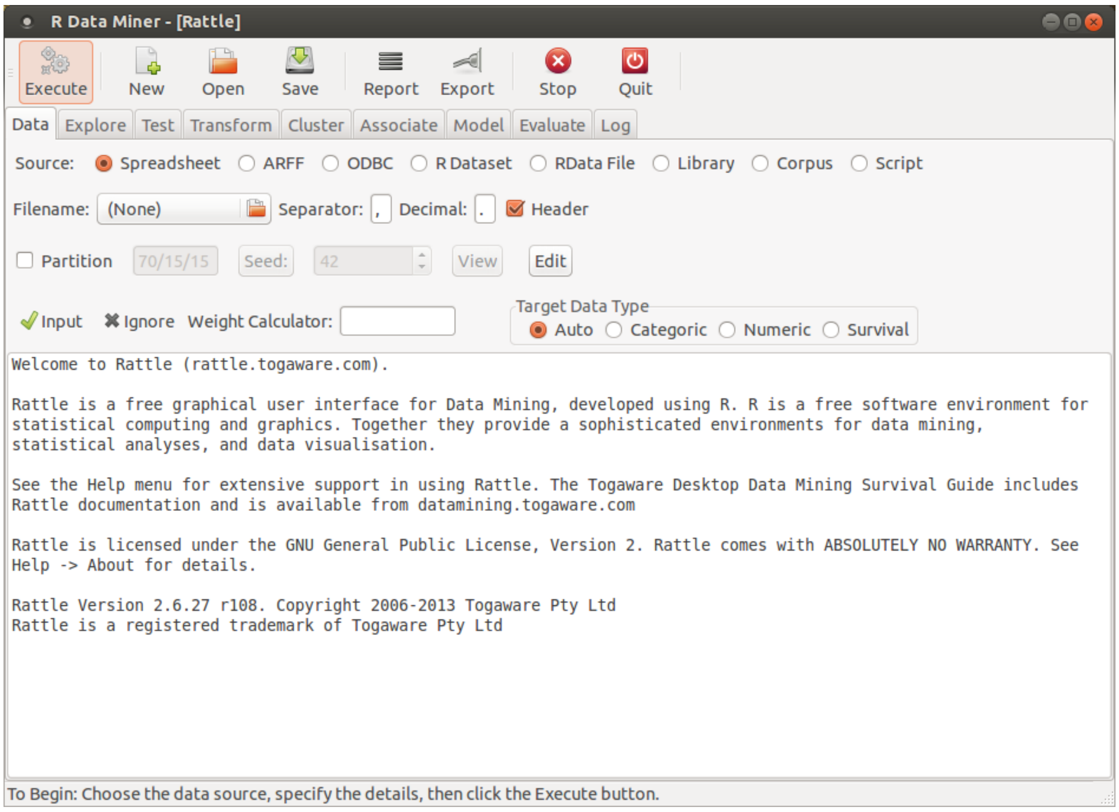
This present module is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

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:

```
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. Remeber, in Rattle, that after we set up a particular tab we must press the Execute button to have the tab take effect.



Source: [✓ Spreadsheet] ARFF ODBC R Dataset RData File Library Corpus Script

Filename: (None) Separator: , Decimal: . [✓ Header]

Partition 70/15/15 Seed: 42 View Edit

Input Ignore Weight Calculator:

Target Data Type [✓ Auto] Categoric Numeric Survival

```
Welcome to Rattle (rattle.togaware.com).

Rattle is a free graphical user interface for Data Mining, developed using R. R is a free software environment for
statistical computing and graphics. Together they provide a sophisticated environments for data mining,
statistical analyses, and data visualisation.

See the Help menu for extensive support in using Rattle. The Togaware Desktop Data Mining Survival Guide includes
Rattle documentation and is available from datamining.togaware.com

Rattle is licensed under the GNU General Public License, Version 2. Rattle comes with ABSOLUTELY NO WARRANTY. See
Help -> About for details.

Rattle Version 2.6.27 r108. Copyright 2006-2013 Togaware Pty Ltd
Rattle is a registered trademark of Togaware Pty Ltd
```

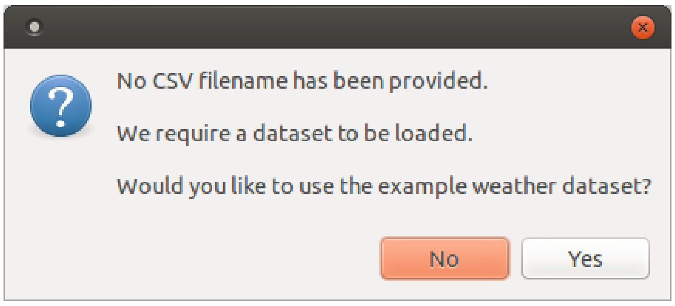
To Begin: Choose the data source, specify the details, then click the Execute button.

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.


No CSV filename has been provided.

We require a dataset to be loaded.

Would you like to use the example 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).

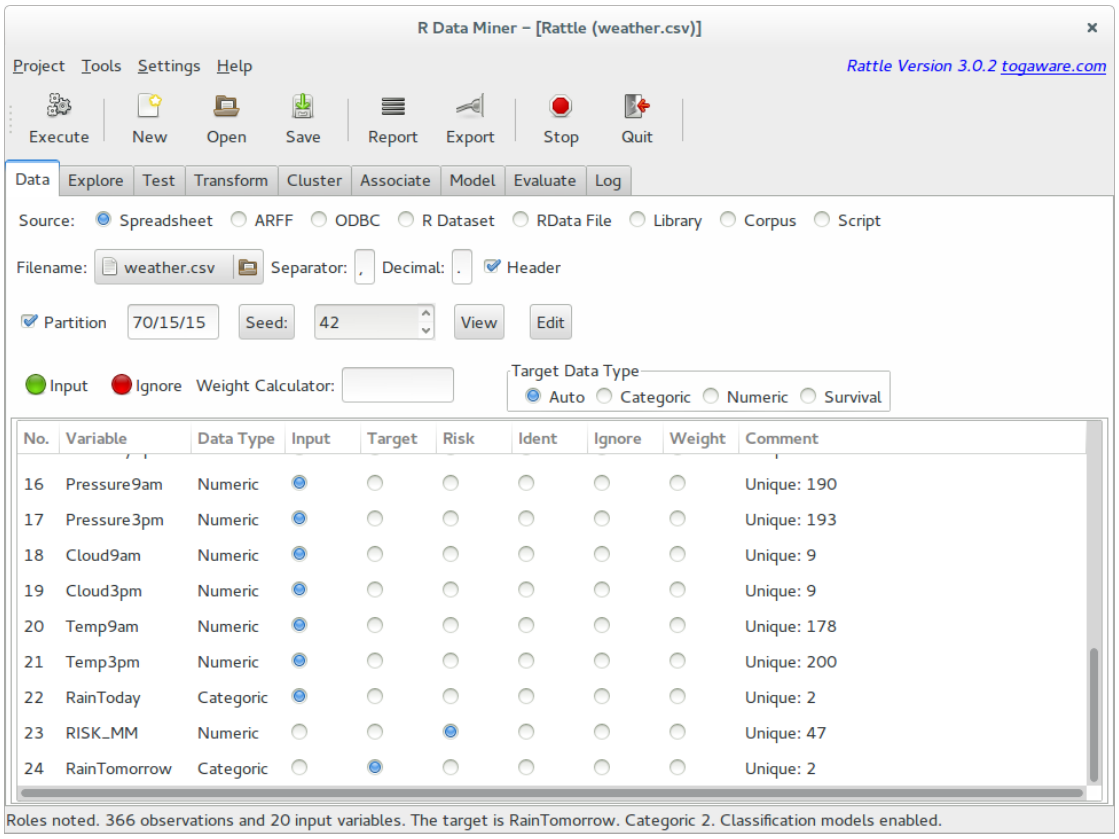
3 Summary of the Weather Dataset

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.


R Data Miner – [Rattle (weather.csv)]
Project Tools Settings Help
Rattle Version 3.0.2 togaware.com
Execute New Open Save Report Export Stop Quit
Data Explore Test Transform Cluster Associate Model Evaluate Log

Source: [✓ Spreadsheet] ARFF ODBC R Dataset RData File Library Corpus Script

Filename: weather.csv Separator: , Decimal: . [✓ Header]

[✓ Partition] 70/15/15 Seed: 42 View Edit

Input Ignore Weight Calculator:

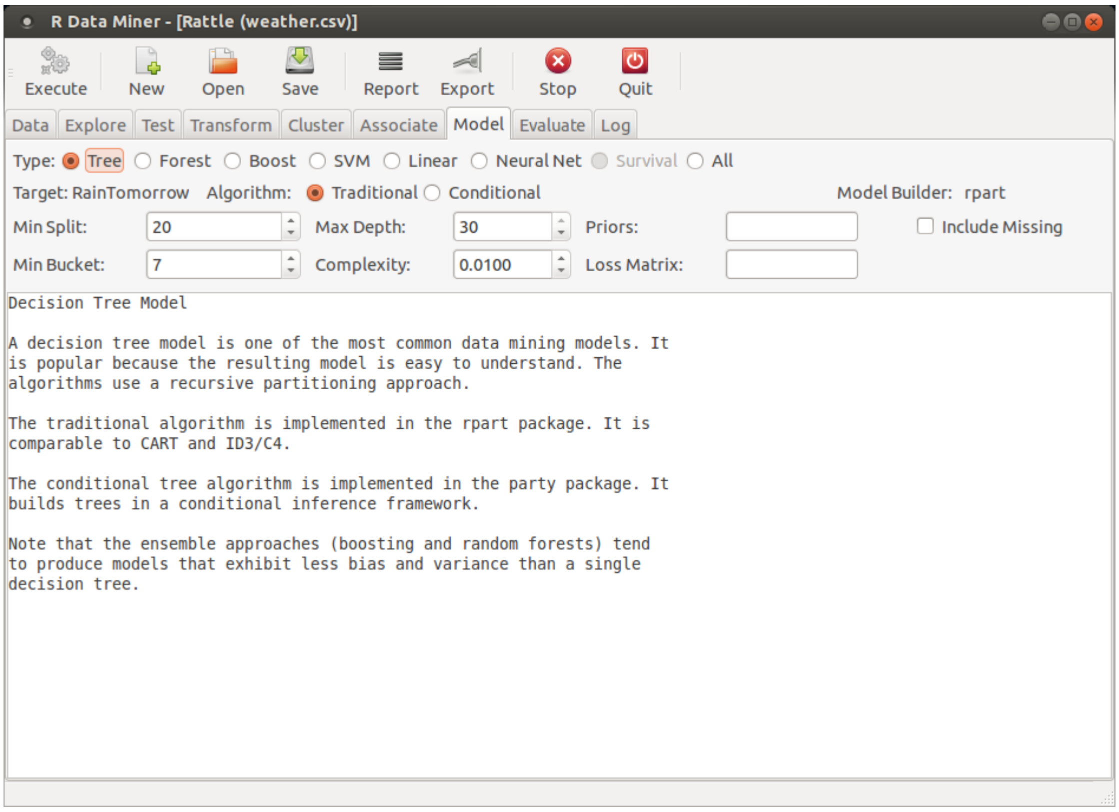
Target Data Type: [✓ Auto] Categoric Numeric Survival

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
16	Pressure9am	Numeric	●	○	○	○	○	○	Unique: 190
17	Pressure3pm	Numeric	●	○	○	○	○	○	Unique: 193
18	Cloud9am	Numeric	●	○	○	○	○	○	Unique: 9
19	Cloud3pm	Numeric	●	○	○	○	○	○	Unique: 9
20	Temp9am	Numeric	●	○	○	○	○	○	Unique: 178
21	Temp3pm	Numeric	●	○	○	○	○	○	Unique: 200
22	RainToday	Categoric	●	○	○	○	○	○	Unique: 2
23	RISK_MM	Numeric	○	○	●	○	○	○	Unique: 47
24	RainTomorrow	Categoric	○	●	○	○	○	○	Unique: 2

Roles noted. 366 observations and 20 input variables. The target is RainTomorrow. Categoric 2. Classification models enabled.

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.



Type: [✓ Tree] Forest Boost SVM Linear Neural Net Survival All

Target: RainTomorrow Algorithm: [✓ Traditional] Conditional Model Builder: rpart

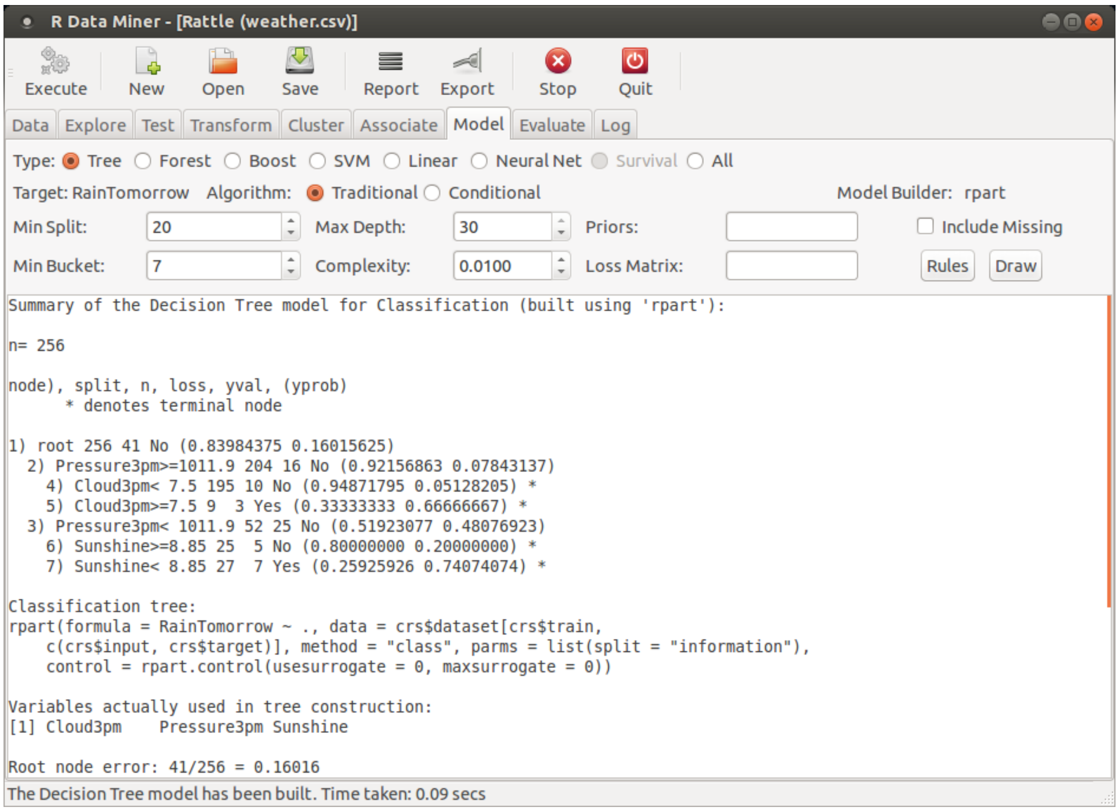
Min Split: 20 Max Depth: 30 Priors: Include Missing

Min Bucket: 7 Complexity: 0.0100 Loss Matrix:

```
Decision Tree Model

A decision tree model is one of the most common data mining models. It
is popular because the resulting model is easy to understand. The
algorithms use a recursive partitioning approach.

The traditional algorithm is implemented in the rpart package. It is
comparable to CART and ID3/C4.

The conditional tree algorithm is implemented in the party package. It
builds trees in a conditional inference framework.

Note that the ensemble approaches (boosting and random forests) tend
to produce models that exhibit less bias and variance than a single
decision tree.
```

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()`.



Type: [✓ Tree] Forest Boost SVM Linear Neural Net Survival All

Target: RainTomorrow Algorithm: [✓ Traditional] Conditional Model Builder: rpart

Min Split: 20 Max Depth: 30 Priors: Include Missing

Min Bucket: 7 Complexity: 0.0100 Loss Matrix:

```
Summary of the Decision Tree model for Classification (built using 'rpart'):

n= 256

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 256 41 No (0.83984375 0.16015625)
  2) Pressure3pm>=1011.9 204 16 No (0.92156863 0.07843137)
    4) Cloud3pm< 7.5 195 10 No (0.94871795 0.05128205) *
    5) Cloud3pm>=7.5 9  3 Yes (0.33333333 0.66666667) *
  3) Pressure3pm< 1011.9 52 25 No (0.51923077 0.48076923)
    6) Sunshine>=8.85 25  5 No (0.80000000 0.20000000) *
    7) Sunshine< 8.85 27  7 Yes (0.25925926 0.74074074) *

Classification tree:
rpart(formula = RainTomorrow ~ ., data = crs$dataset[crs$train,
    c(crs$input, crs$target)], method = "class", parms = list(split = "information"),
    control = rpart.control(usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Cloud3pm    Pressure3pm Sunshine

Root node error: 41/256 = 0.16016
```

The Decision Tree model has been built. Time taken: 0.09 secs

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.

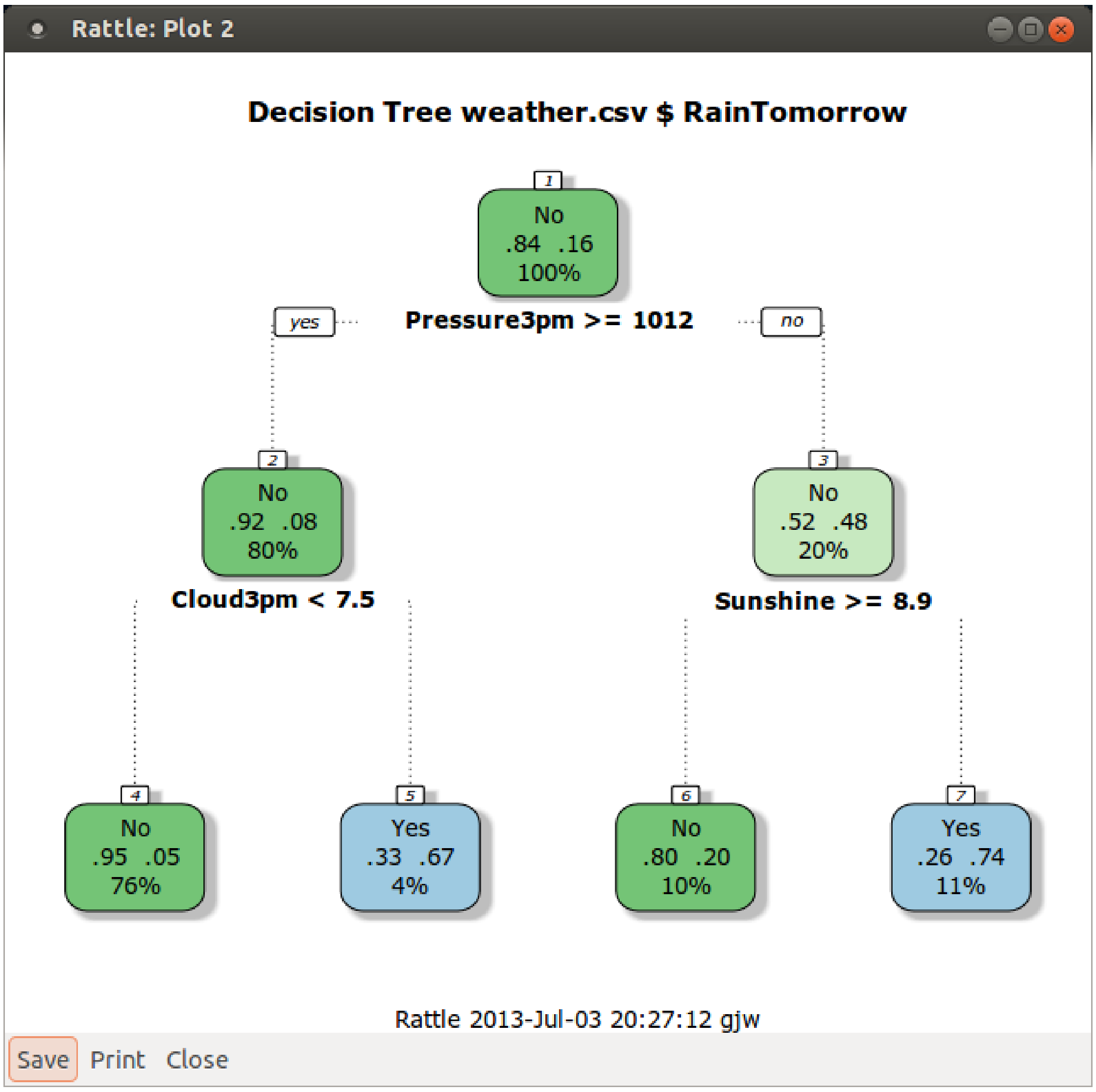
A legend for reading the information in the textual representation of the decision tree is then presented.

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.

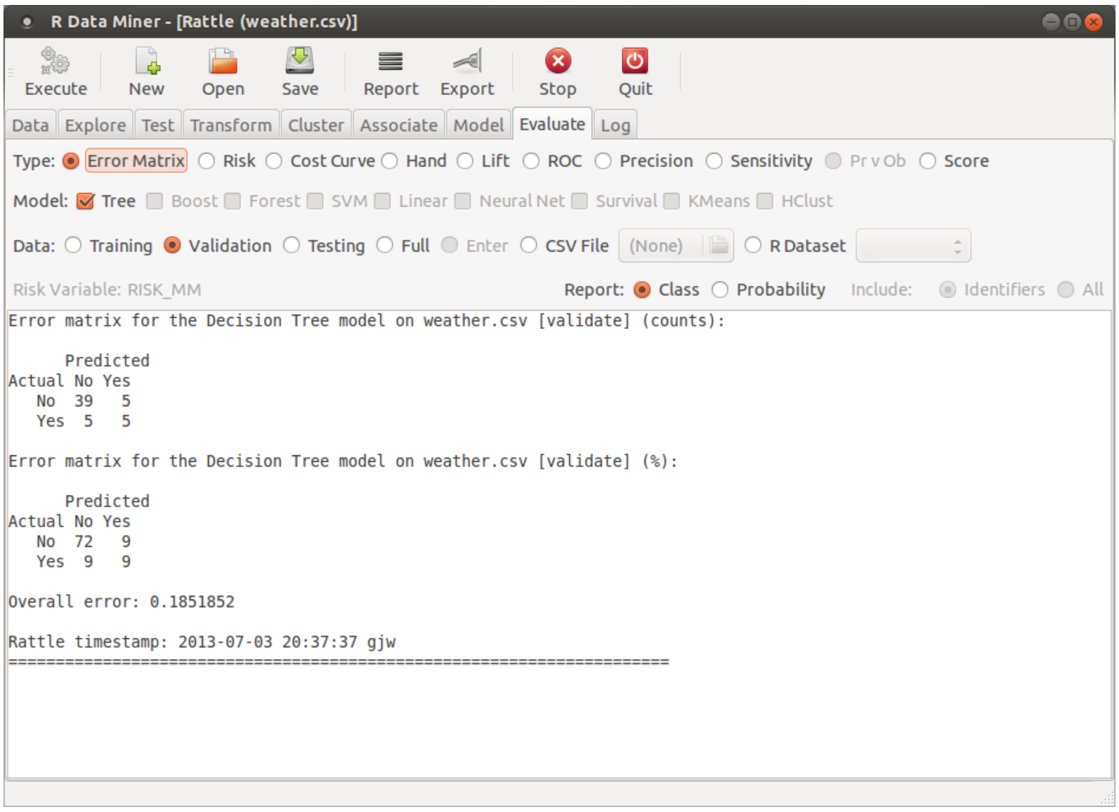
6 Decision Tree Predicting RainTomorrow

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.



Type: [✓ Error Matrix] Risk Cost Curve Hand Lift ROC Precision Sensitivity Pr v Ob Score

Model: [✓ Tree] Boost Forest SVM Linear Neural Net Survival KMeans HClust

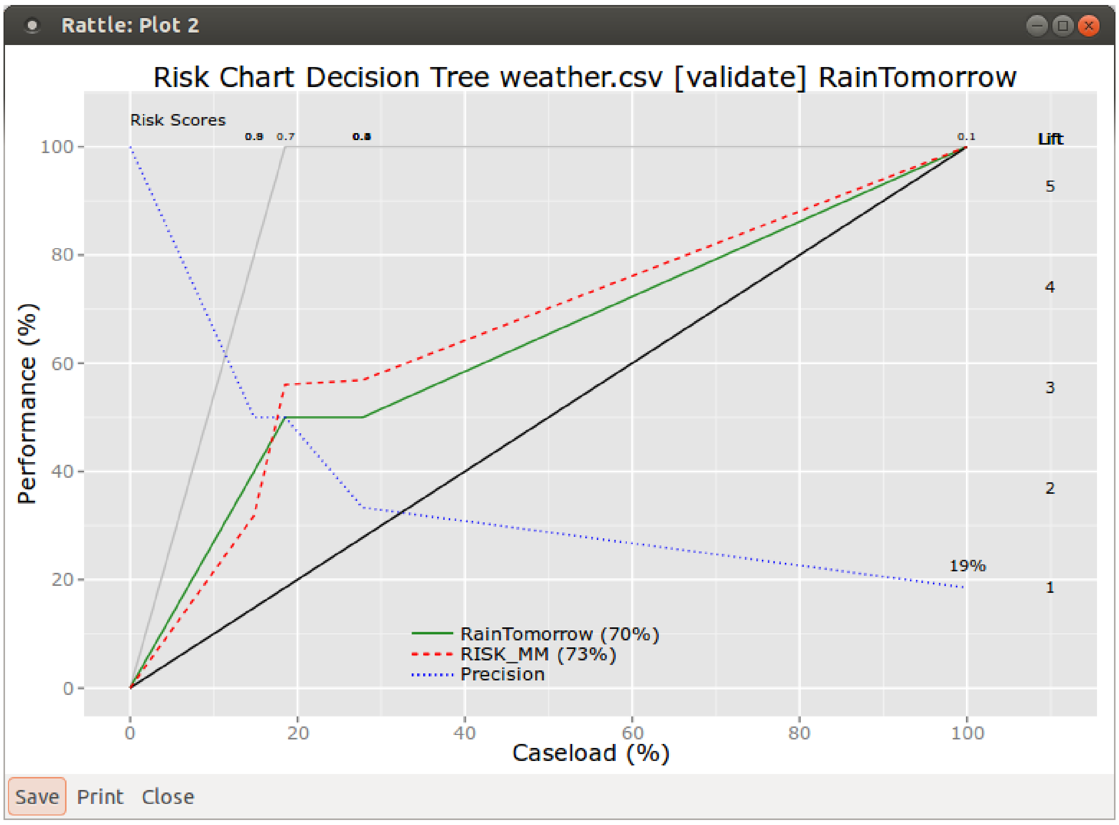
Data: Training [✓ Validation] Testing Full Enter CSV File (None) R Dataset

Risk Variable: RISK_MM Report: [✓ Class] Probability Include: [✓ Identifiers] All

```
Error matrix for the Decision Tree model on weather.csv [validate] (counts):

      Predicted
Actual No Yes
   No  39   5
   Yes  5   5

Error matrix for the Decision Tree model on weather.csv [validate] (%):

      Predicted
Actual No Yes
   No  72   9
   Yes  9   9

Overall error: 0.1851852

Rattle timestamp: 2013-07-03 20:37:37 gjw
======================================================================
```

8 Decision Tree Risk Chart

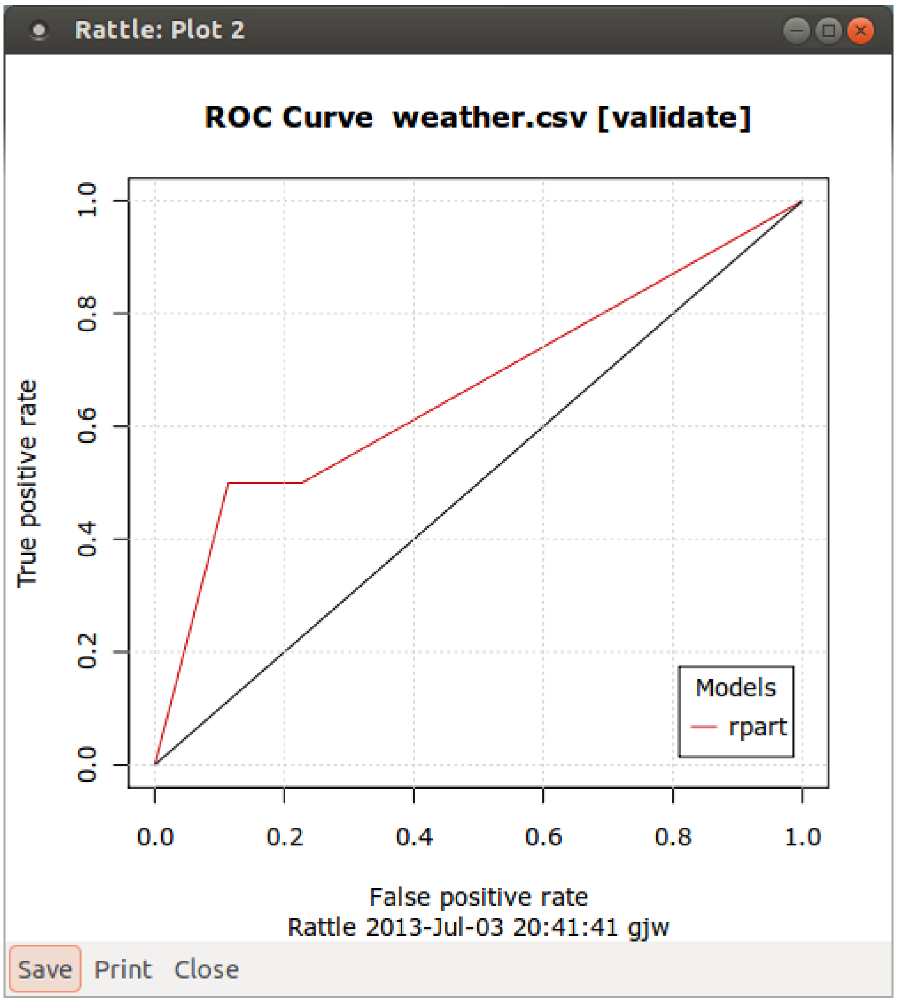
Click the Risk type and then Execute.



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.



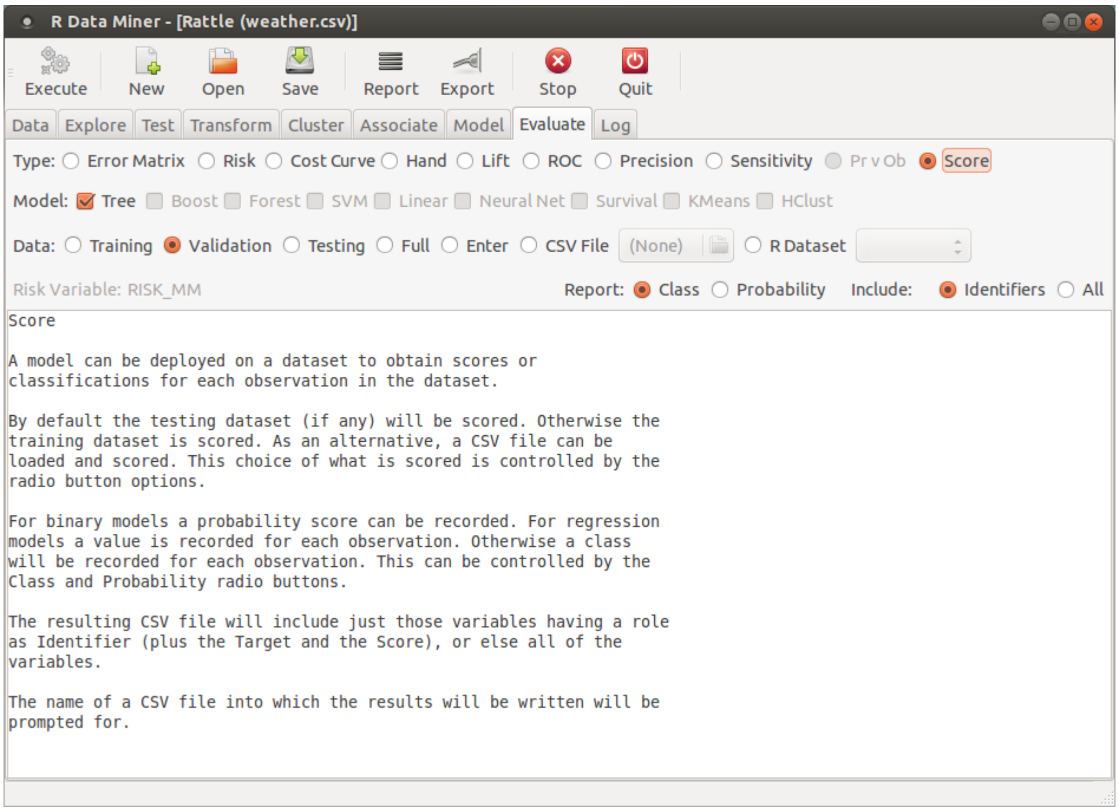
Exercise: Research how to interpret an ROC curve. Explain the ROC curve in one or two paragraphs.

10 Other Evaluation Plots

Exercise: Research the cost curve, the Hand plots, the Lift chart and the Precision and Sensitivity 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.



Type: Error Matrix, Risk, Cost Curve, Hand, Lift, ROC, Precision, Sensitivity, Pr v Ob, [✓ Score]

Model: [✓ Tree] Boost, Forest, SVM, Linear, Neural Net, Survival, KMeans, HClust

Data: Training, [✓ Validation] Testing, Full, Enter, CSV File (None), R Dataset

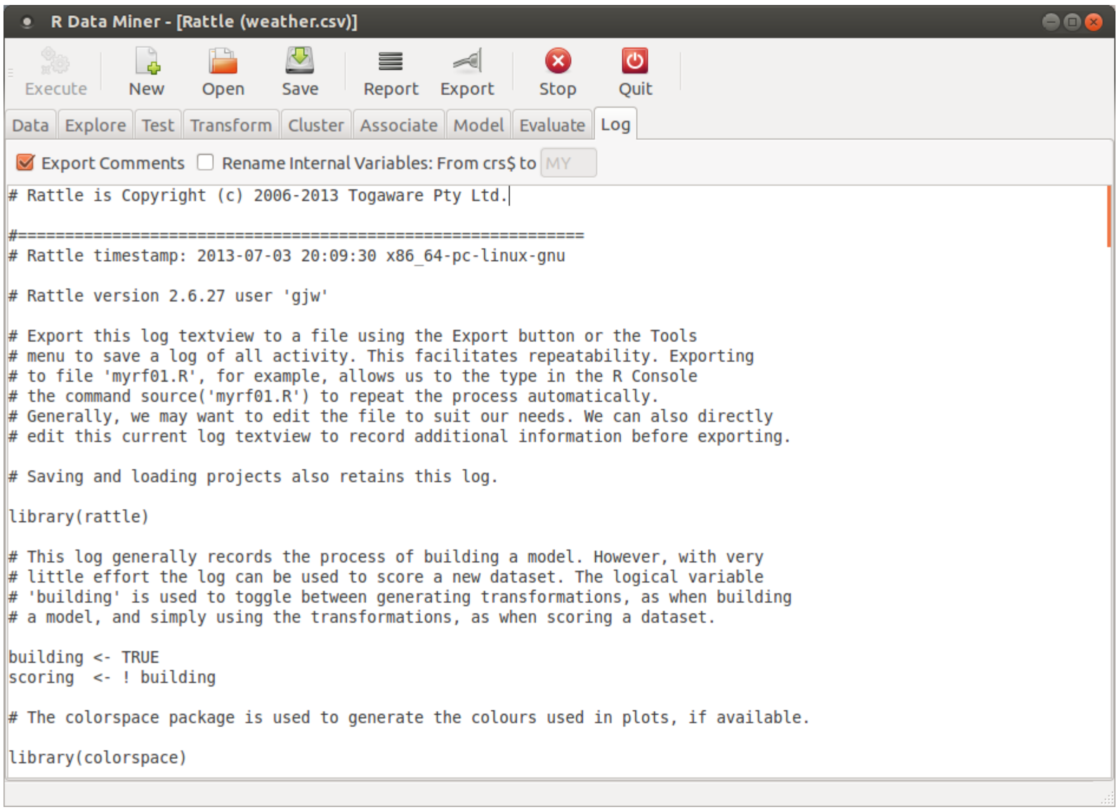
Risk Variable: RISK_MM

Report: [✓ Class] Probability Include: [✓ Identifiers] All

```
Score

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.

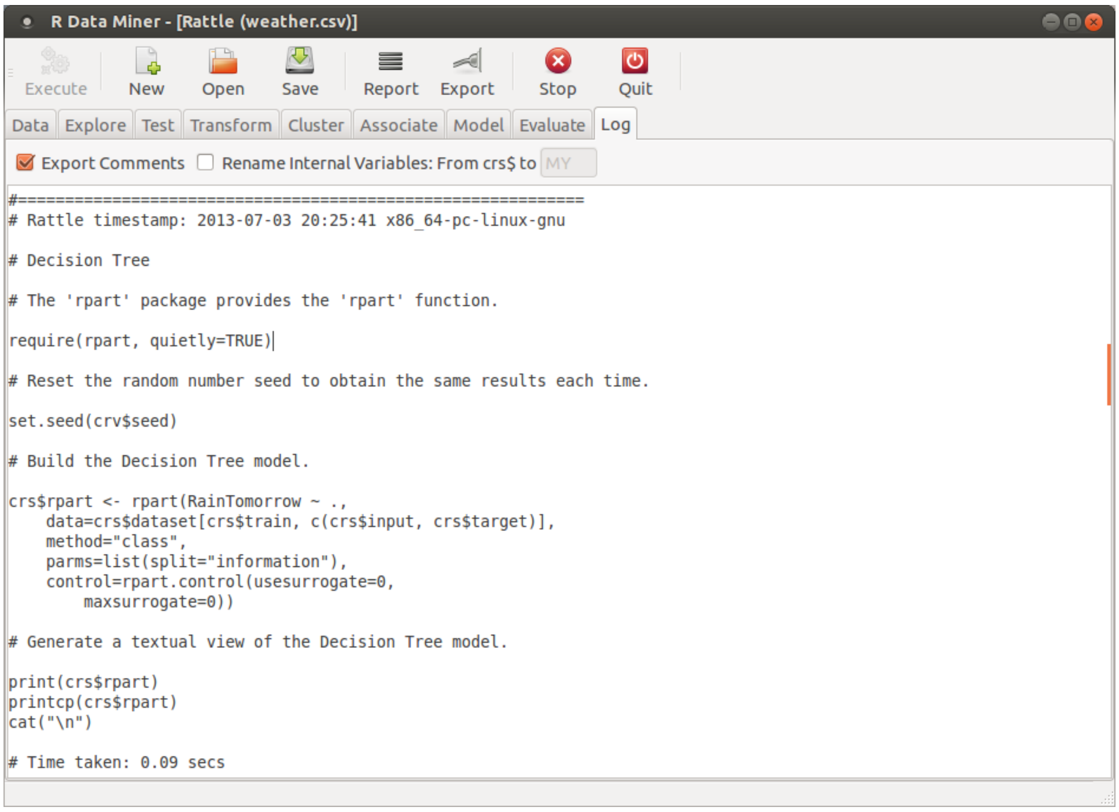


```
# Rattle is Copyright (c) 2006-2013 Togaware Pty Ltd.

#============================================================
# Rattle timestamp: 2013-07-03 20:09:30 x86_64-pc-linux-gnu

# Rattle version 2.6.27 user 'gjw'

# 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 'myrf01.R', for example, allows us to the type in the R Console
# the command source('myrf01.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.



```
#============================================================
# Rattle timestamp: 2013-07-03 20:25:41 x86_64-pc-linux-gnu

# Decision Tree

# The 'rpart' package provides the 'rpart' function.

require(rpart, quietly=TRUE)

# Reset the random number seed to obtain the same results each time.

set.seed(crv$seed)

# Build the Decision Tree model.

crs$rpart <- rpart(RainTomorrow ~ .,
    data=crs$dataset[crs$train, c(crs$input, crs$target)],
    method="class",
    parms=list(split="information"),
    control=rpart.control(usesurrogate=0,
        maxsurrogate=0))

# Generate a textual view of the Decision Tree model.

print(crs$rpart)
printcp(crs$rpart)
cat("\n")

# Time taken: 0.09 secs
```

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

```
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"))
```

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.

```
set.seed(1426)
library(rattle)
data(weather)
dsname       <- "weather"
ds           <- get(dsname)
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"
....

(categorics  <- 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.

```
dim(ds)

## [1] 366  24

names(ds)

##  [1] "Date"          "Location"      "MinTemp"       "MaxTemp"
##  [5] "Rainfall"      "Evaporation"   "Sunshine"      "WindGustDir"
##  [9] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
## [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)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 361 2008-10-26 Canberra     7.9    26.1        0         6.8      3.5
## 362 2008-10-27 Canberra     9.0    30.7        0         7.6     12.1
## 363 2008-10-28 Canberra     7.1    28.4        0        11.6     12.7
## 364 2008-10-29 Canberra    12.5    19.9        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/ 46 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   Canberra      :366   Min.   :-5.30   Min.   : 7.6
##  1st Qu.:2008-01-31   Adelaide      :  0   1st Qu.: 2.30   1st Qu.:15.0
##  Median :2008-05-01   Albany        :  0   Median : 7.45   Median :19.6
##  Mean   :2008-05-01   Albury        :  0   Mean   : 7.27   Mean   :20.6
....
```

16 Build Decision Tree Model

Buld a decision tree using `rpart()`. Once the different variables have been defined (form, ds, train, and vars) this some command can be re-used.

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

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

```
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])
```

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.

```
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) *
```

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 yval), and then the distribution of classes in that node (the yprobs). The distribution is ordered by class and the order is the same for all nodes. 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).

17 Summary of the Model

```
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`.

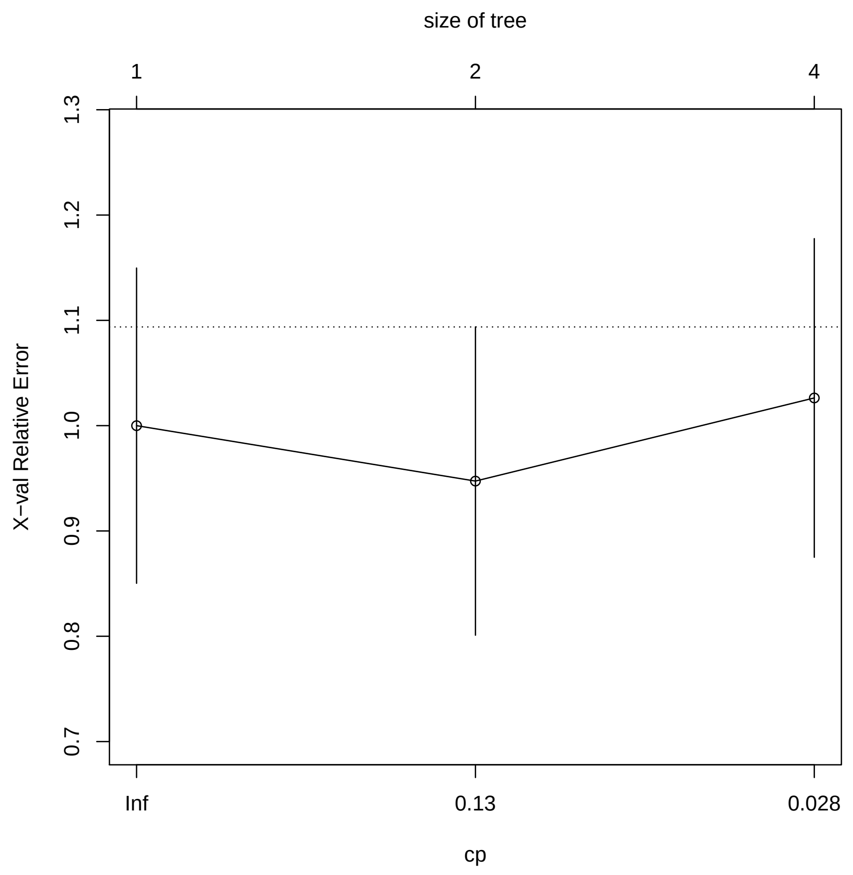
```
printcp(model)

##
## 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)
```

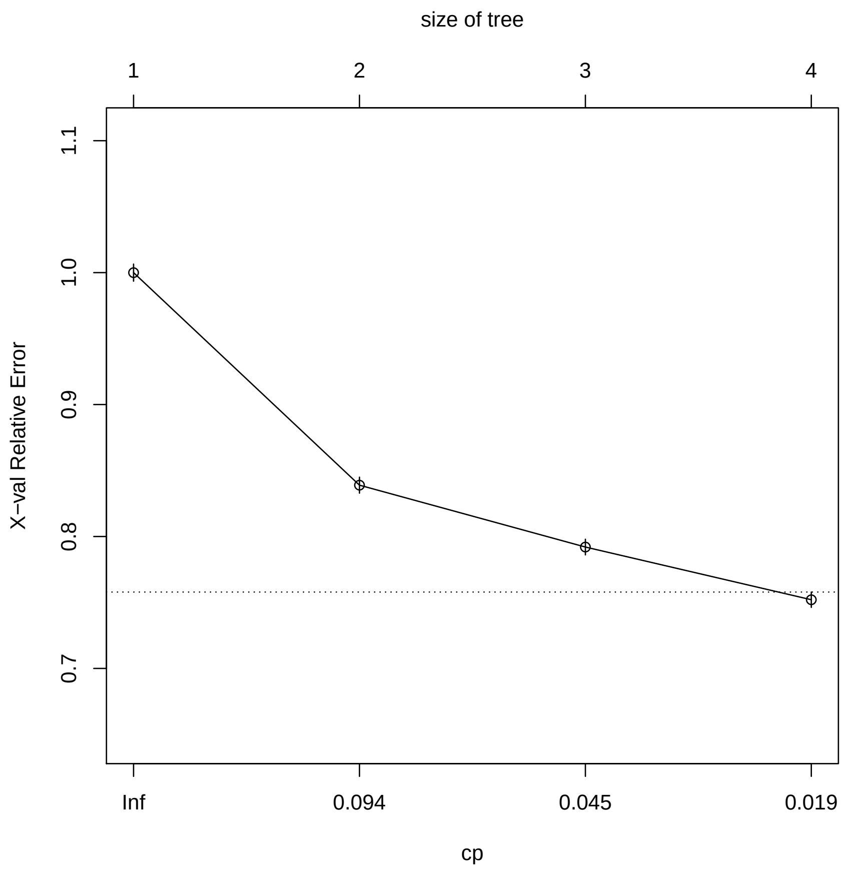


20 More Interesting Complexity Behaviour

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.

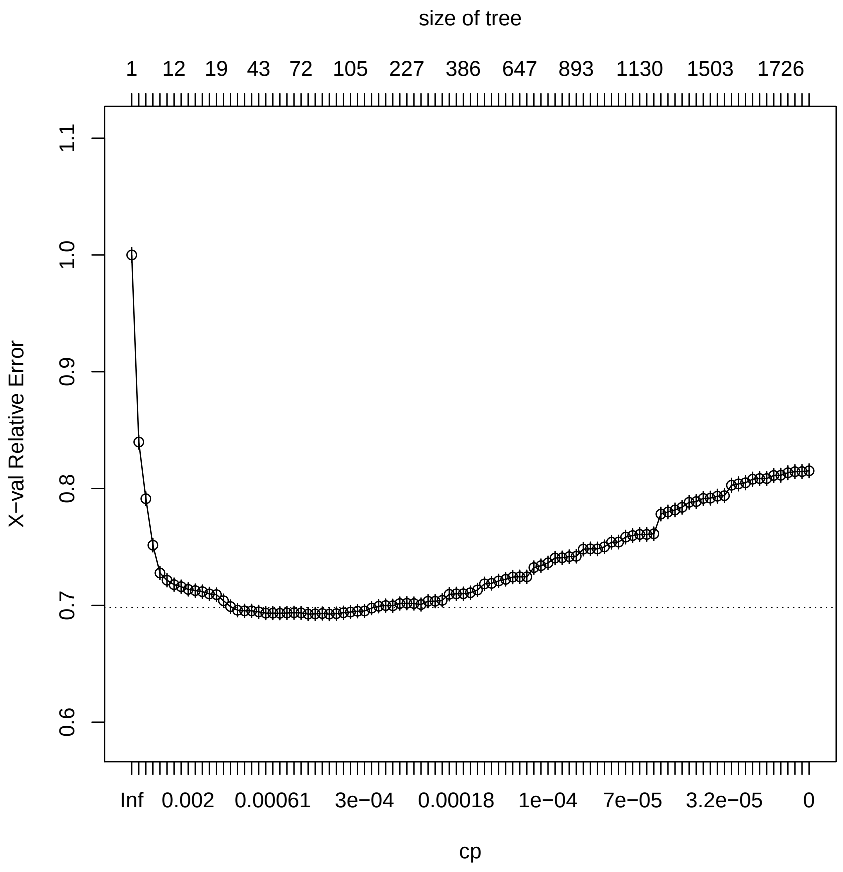
```
tmodel <- rpart(form, weatherAUS[vars])
plotcp(tmodel)
```



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.

```
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.

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

##           CP nsplit rel error xerror     xstd
## 1  1.587e-01      0    1.0000 1.0000 0.006487
## 2  5.606e-02      1    0.8413 0.8398 0.006084
## 3  3.666e-02      2    0.7852 0.7913 0.005945
## 4  5.557e-03      3    0.7486 0.7515 0.005826
## 5  4.485e-03      7    0.7264 0.7277 0.005752
## 22 5.448e-04     52    0.6682 0.6932 0.005640
## 23 5.175e-04     64    0.6598 0.6935 0.005641
## 24 5.085e-04     66    0.6588 0.6937 0.005641
## 25 4.903e-04     71    0.6561 0.6936 0.005641
## 26 4.576e-04     75    0.6541 0.6925 0.005638
## 27 4.358e-04     80    0.6518 0.6927 0.005638
## 28 3.995e-04     86    0.6491 0.6930 0.005639
## 29 3.813e-04     90    0.6474 0.6925 0.005638
## 80 4.358e-05   1436    0.4575 0.7882 0.005936
## 81 4.086e-05   1441    0.4573 0.7888 0.005938
## 82 3.891e-05   1488    0.4549 0.7916 0.005946
## 83 3.632e-05   1502    0.4542 0.7918 0.005947
```

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.

```
model$variable.importance

## Humidity3pm    Sunshine Pressure3pm     Temp9am Pressure9am     Temp3pm
##     13.1468      9.2091      7.3894      4.6458      4.2920      4.2504
##    Cloud3pm     MaxTemp     MinTemp    Rainfall
##      3.6436      3.0330      2.8339      0.1991
....
```

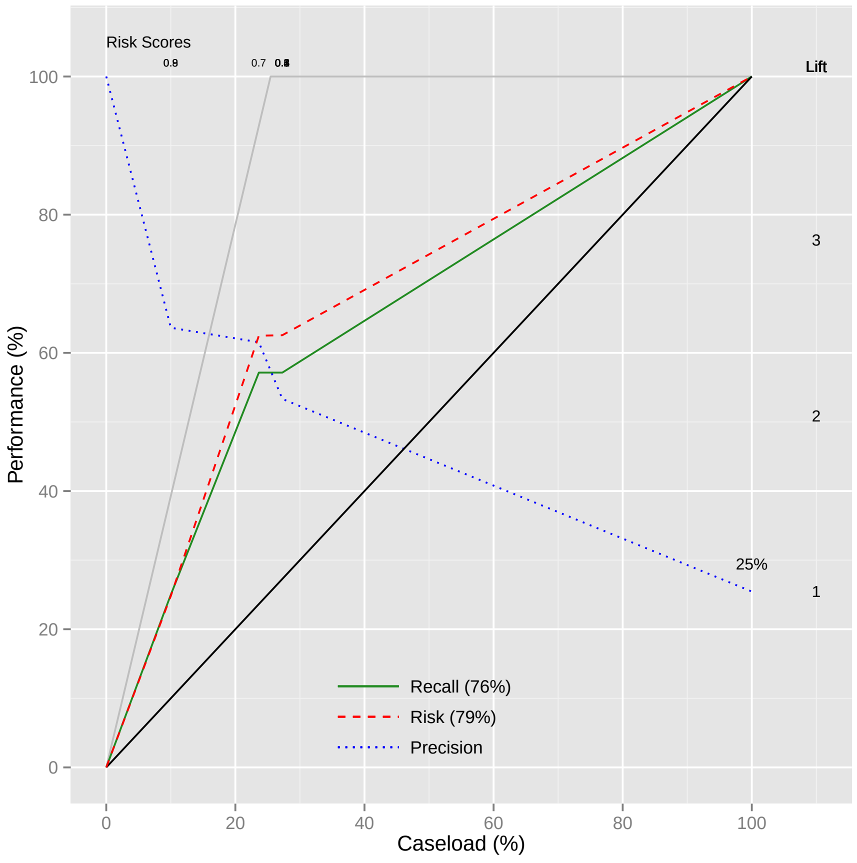
24 Node Details and Surrogates

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.

```
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.

```
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

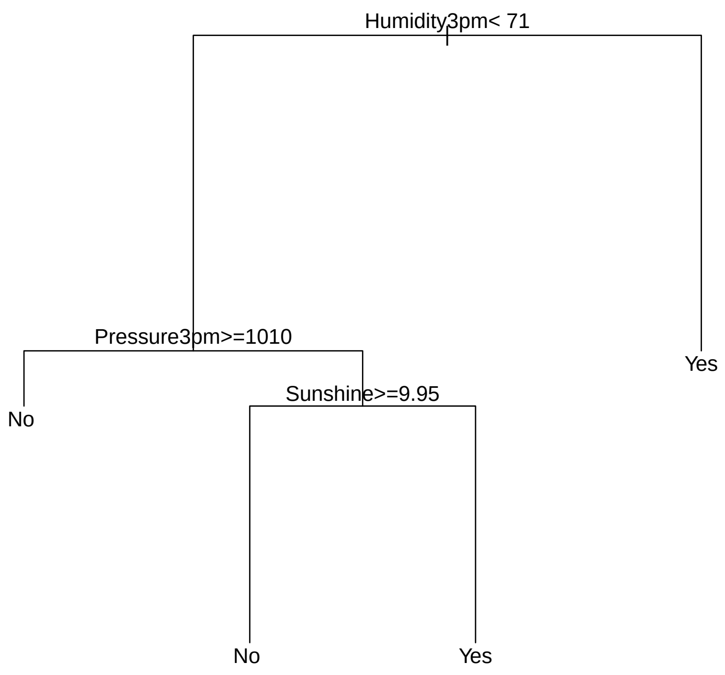
We can use the following function to print the paths through the decision tree as rules.

```
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
....
```

27 Visualise Decision Trees

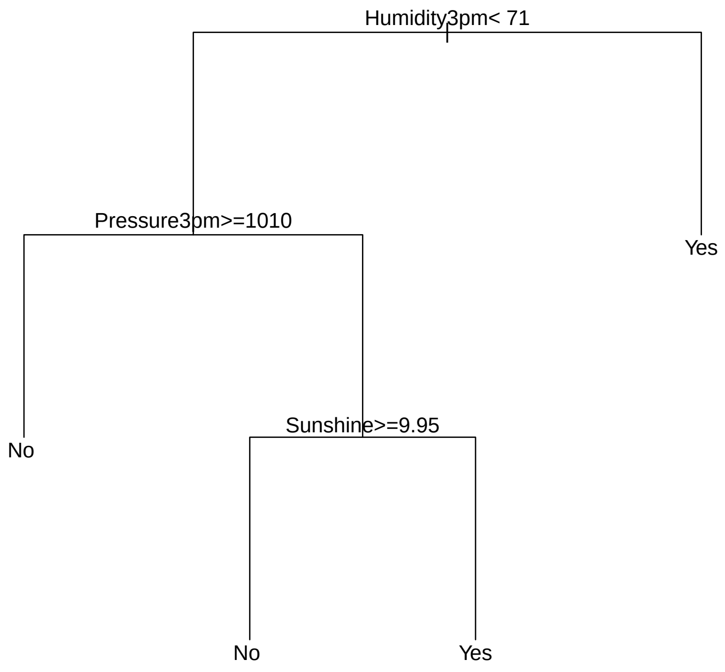
```r
plot(model)
text(model)
```



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.

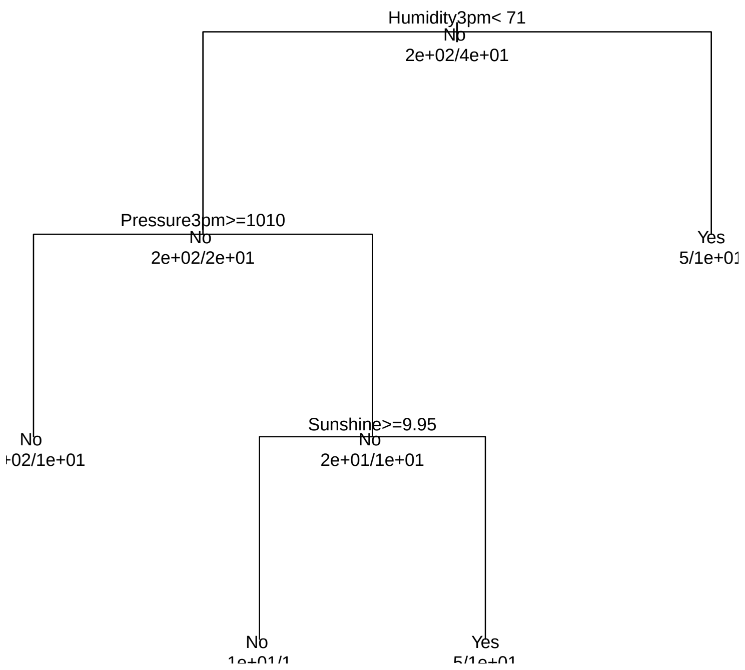
28 Visualise Decision Trees: Uniform

```
plot(model, uniform=TRUE)
text(model)
```



29 Visualise Decision Trees: Extra Information

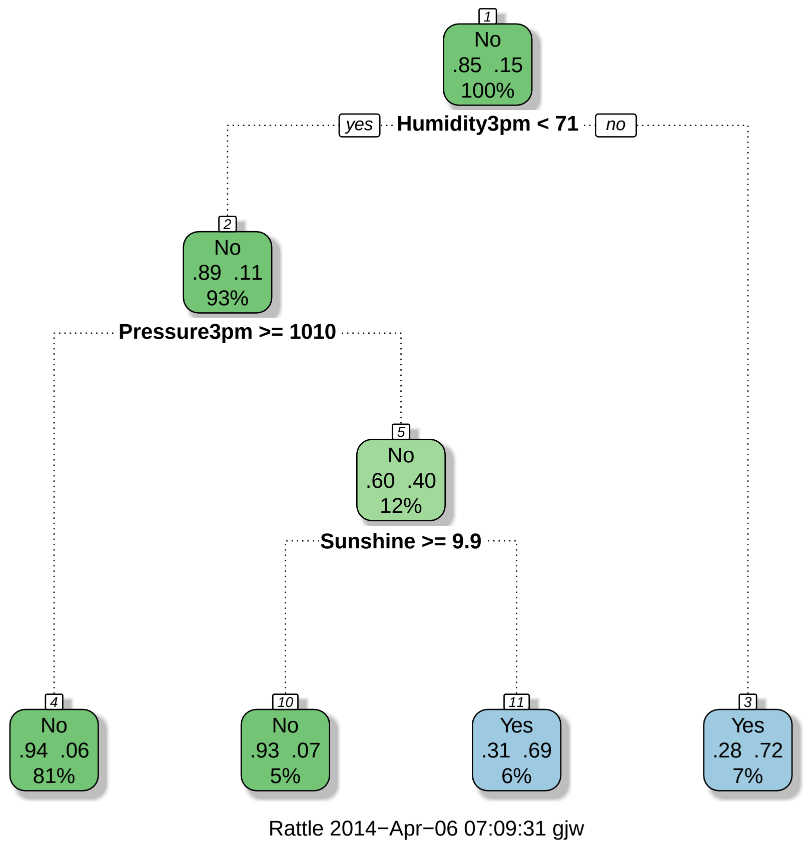
```
plot(model, uniform=TRUE)
text(model, use.n=TRUE, all=TRUE, cex=.8)
```



30 Fancy Plot

The rattle package provides a fancy plot based on the functionality provided by rpart.plot (Milborrow, 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.

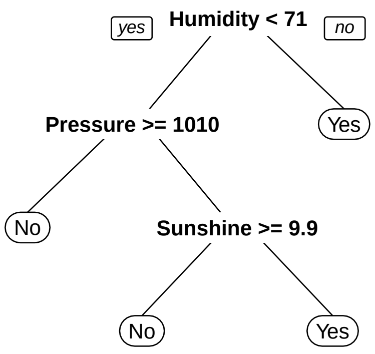
```
fancyRpartPlot(model)
```



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()`.

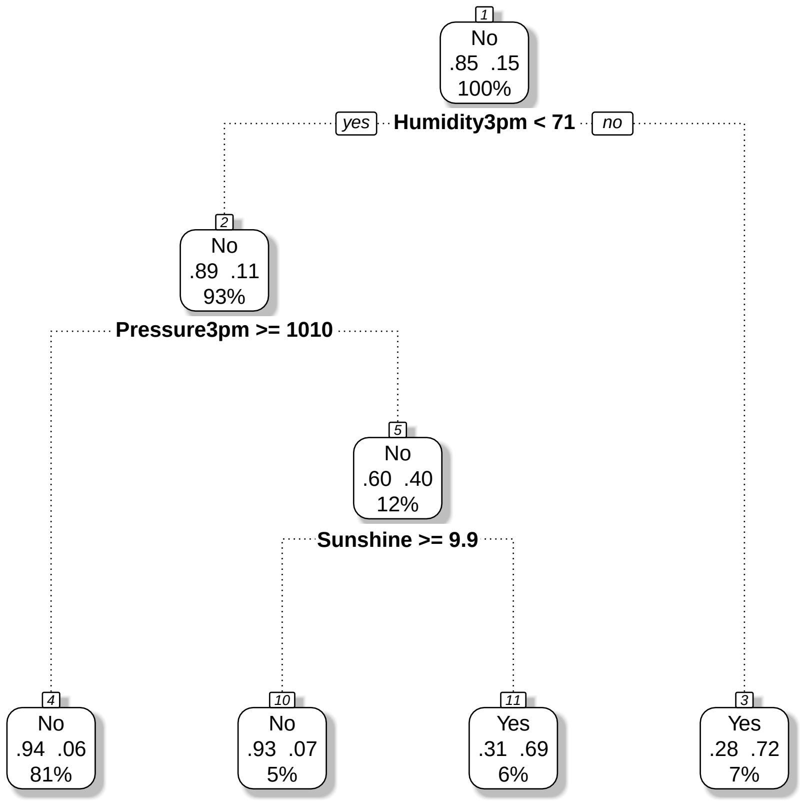
```
prp(model)
```



32 Enhanced Plots: Favourite

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

```
prp(model, type=2, extra=104, nn=TRUE, fallen.leaves=TRUE,
    faclen=0, varlen=0, shadow.col="grey", branch.lty=3)
```



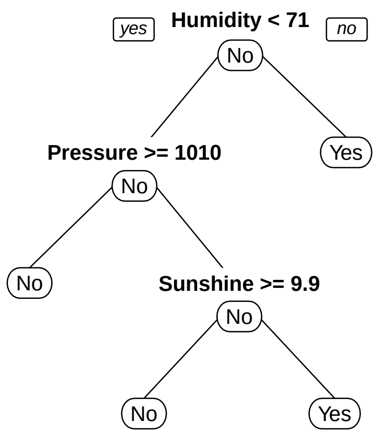
The leaf nodes are each labelled with the predicted class. They are neatly lined up at the bottom of the figure (`fallen.leaves=TRUE`), to visually reinforce the structure. We can see the straight lines from the top to the bottom which lead to decisions quickly, whilst the more complex paths need quite 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 (`extra=104`). The node numbers are included (`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 (`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.

33 Enhanced Plots: Label all Nodes

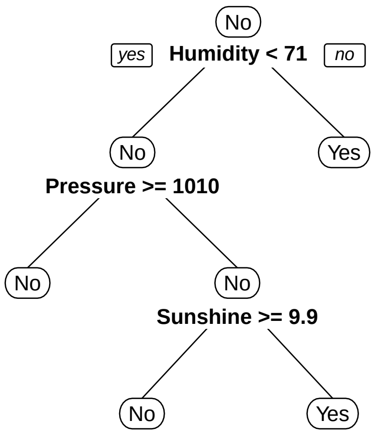
```
prp(model, type=1)
```



Here all nodes are labelled with the majority class.

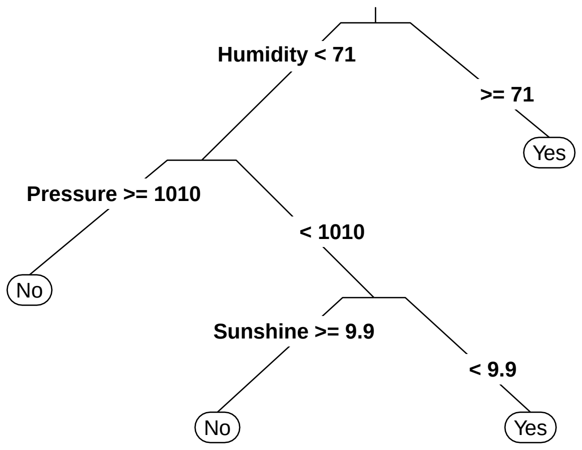
34 Enhanced Plots: Labels Below Nodes

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



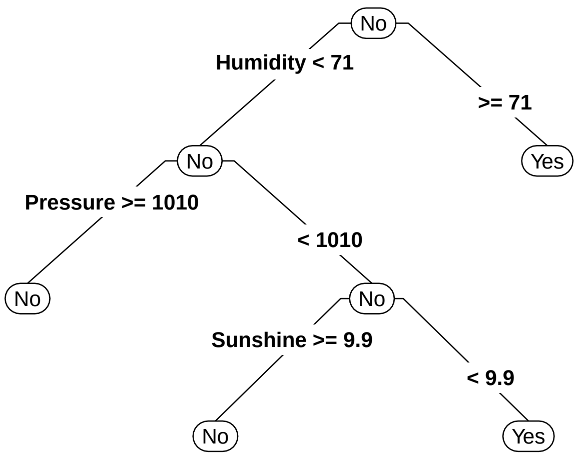
35 Enhanced Plots: Split Labels

```
prp(model, type=3)
```



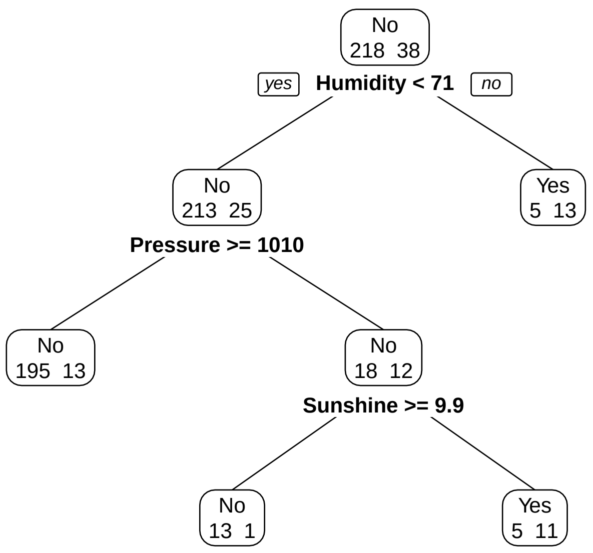
36 Enhanced Plots: Interior Labels

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



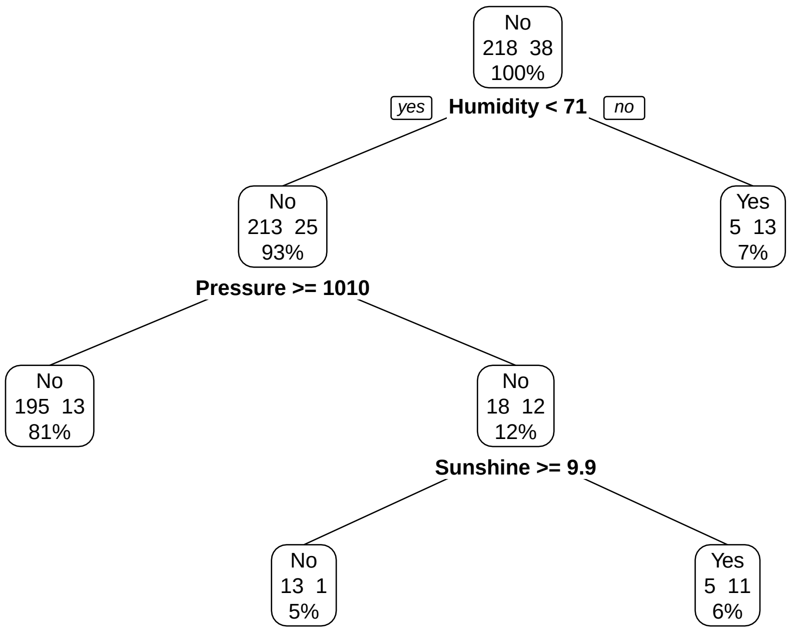
37 Enhanced Plots: Number of Observations

```r
prp(model, type=2, extra=1)
```



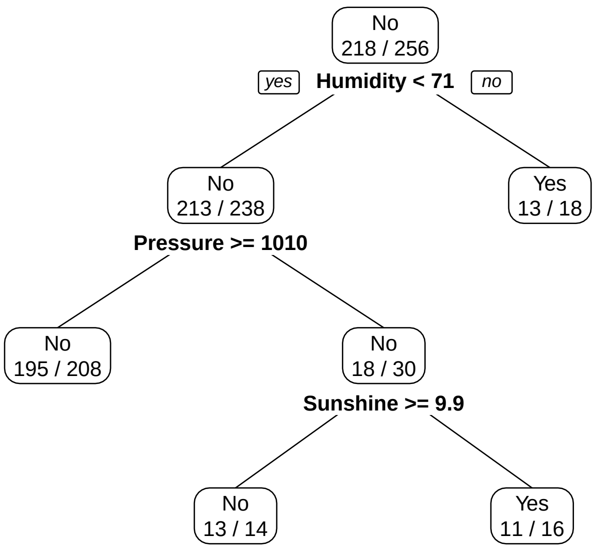
38 Enhanced Plots: Add Percentage of Observations

```
prp(model, type=2, extra=101)
```



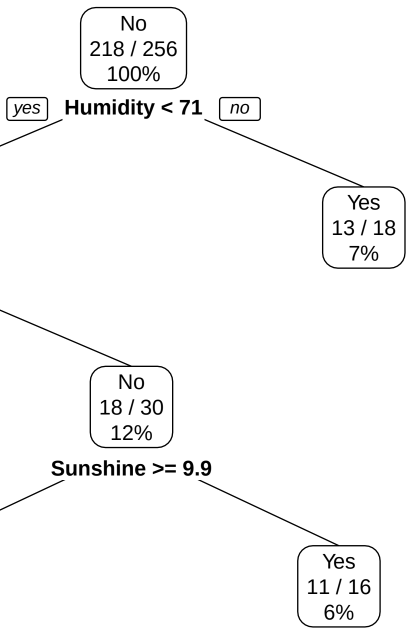
39 Enhanced Plots: Classification Rate

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



40 Enhanced Plots: Add Percentage of Observations

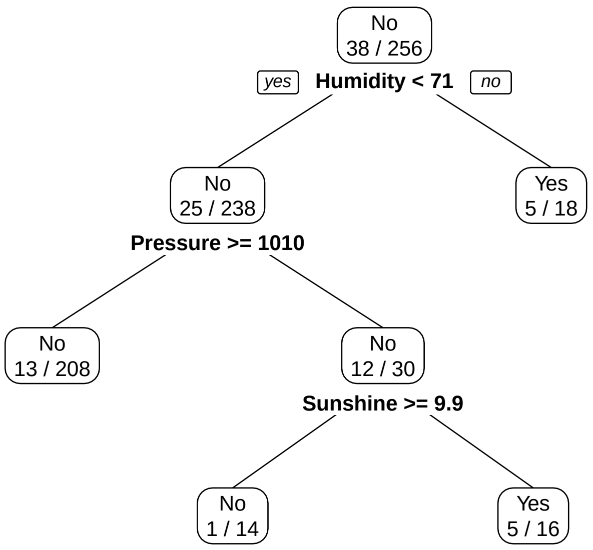
```
prp(model, type=2, extra=102)
```



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.

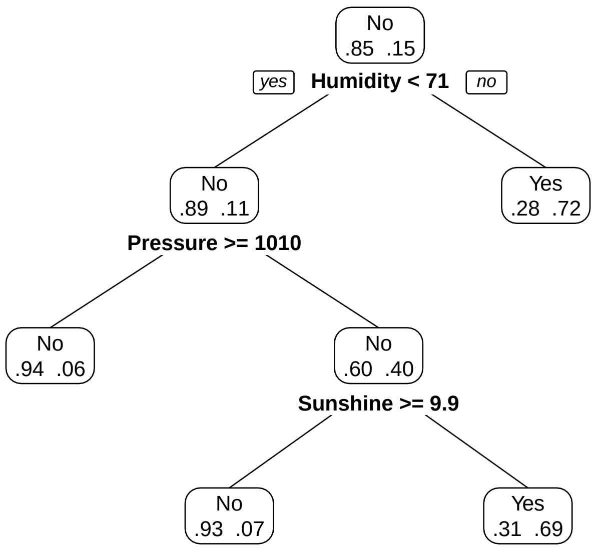
41 Enhanced Plots: Misclassification Rate

```
prp(model, type=2, extra=3)
```



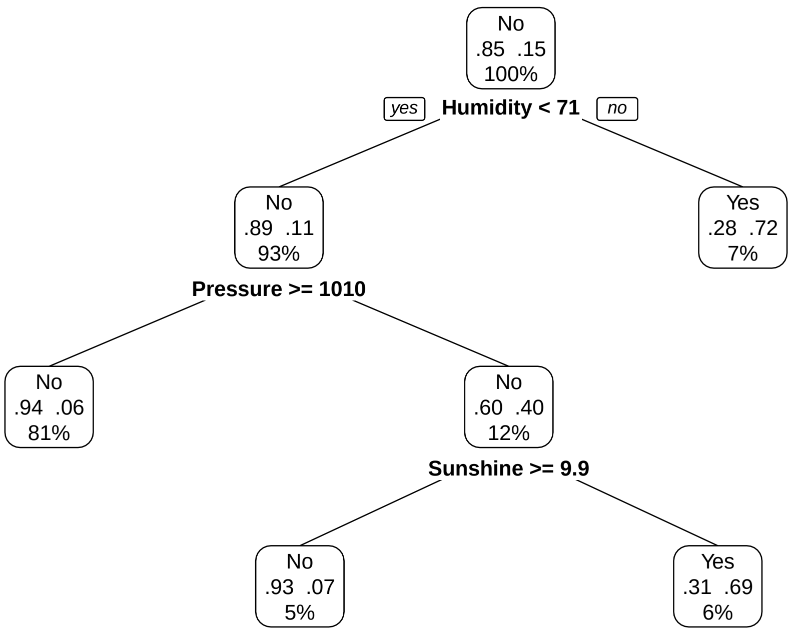
42 Enhanced Plots: Probability per Class

```r
prp(model, type=2, extra=4)
```



43 Enhanced Plots: Add Percentage Observations

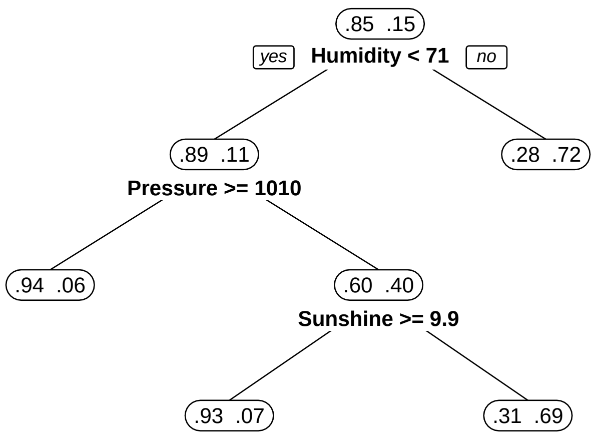
```
prp(model, type=2, extra=104)
```



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.

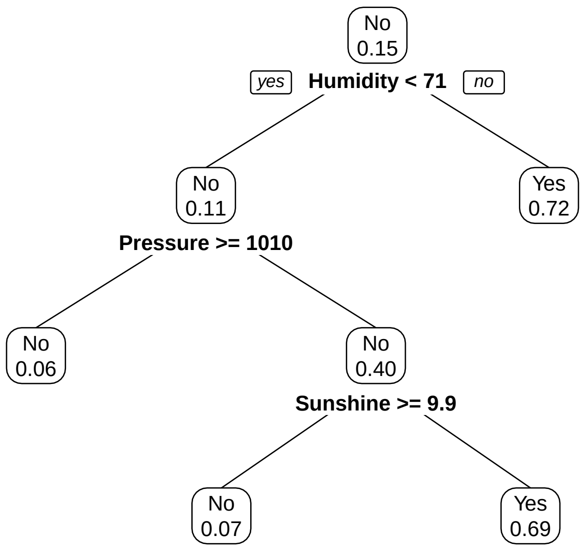
44 Enhanced Plots: Only Probability Per Class

```
prp(model, type=2, extra=5)
```



45 Enhanced Plots: Probability of Second Class

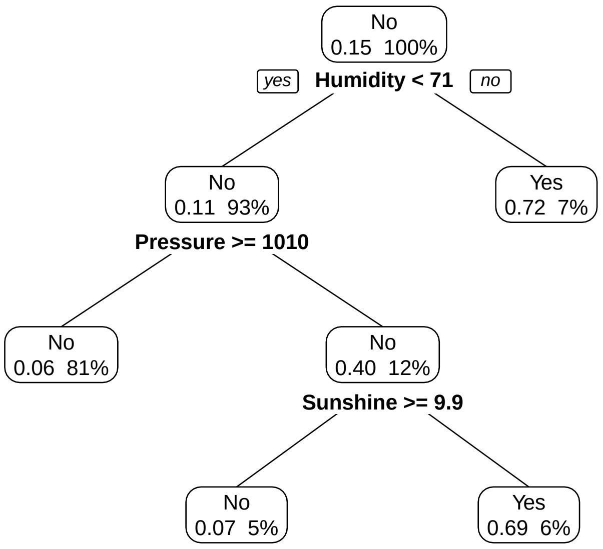
```
prp(model, type=2, extra=6)
```



This is particularly useful for binary classification, as here, where the second class is usually the positive response.

46 Enhanced Plots: Add Percentage Observations

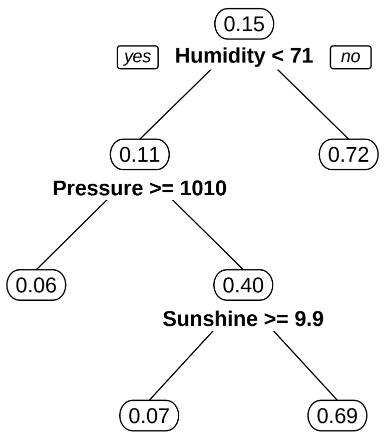
```
prp(model, type=2, extra=106)
```



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.

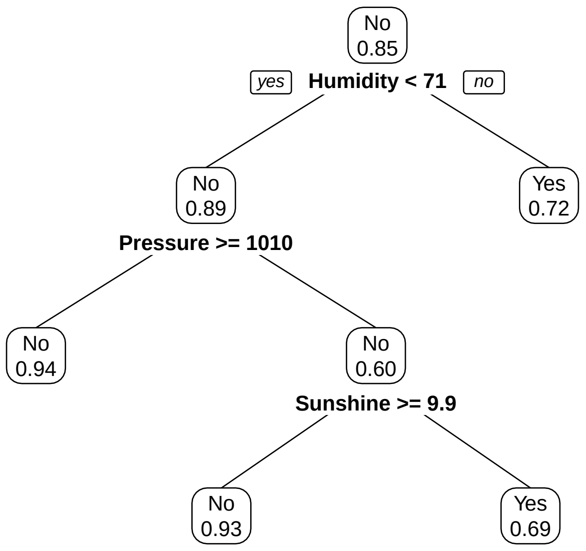
47 Enhanced Plots: Only Probability of Second Class

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



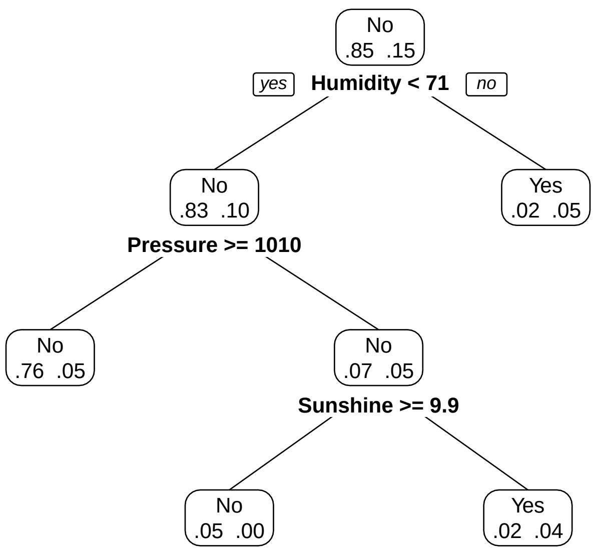
48 Enhanced Plots: Probability of the Class

```r
prp(model, type=2, extra=8)
```



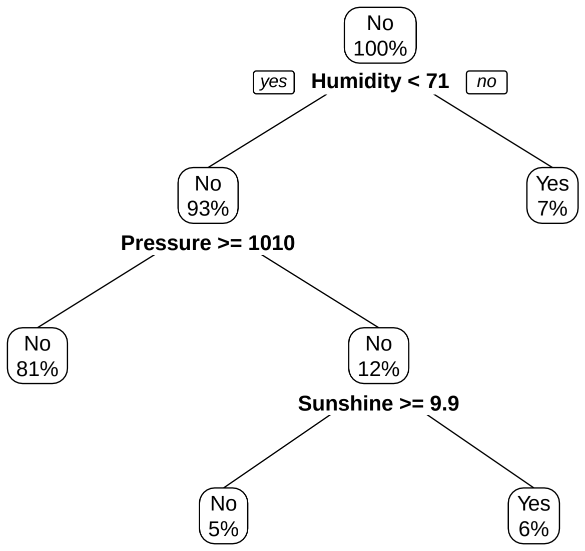
49 Enhanced Plots: Overall Probability

```
prp(model, type=2, extra=9)
```



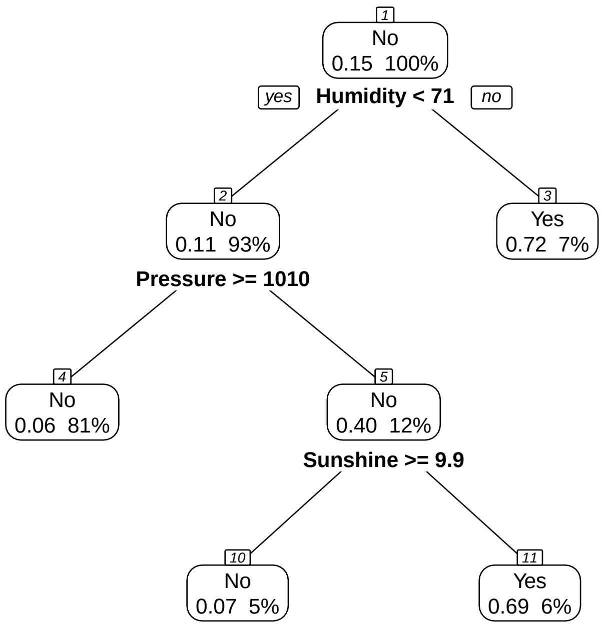
50 Enhanced Plots: Percentage of Observations

```
prp(model, type=2, extra=100)
```



51 Enhanced Plots: Show the Node Numbers

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

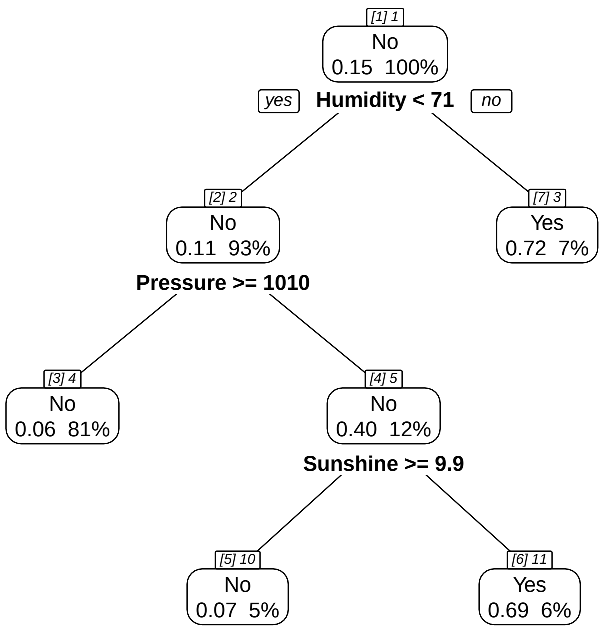


We now take our favourite plot (`type=2` and `extra=106`) and explore some of the other options available for `prp()` from `rpart.plot`.

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.

52 Enhanced Plots: Show the Node Indicies

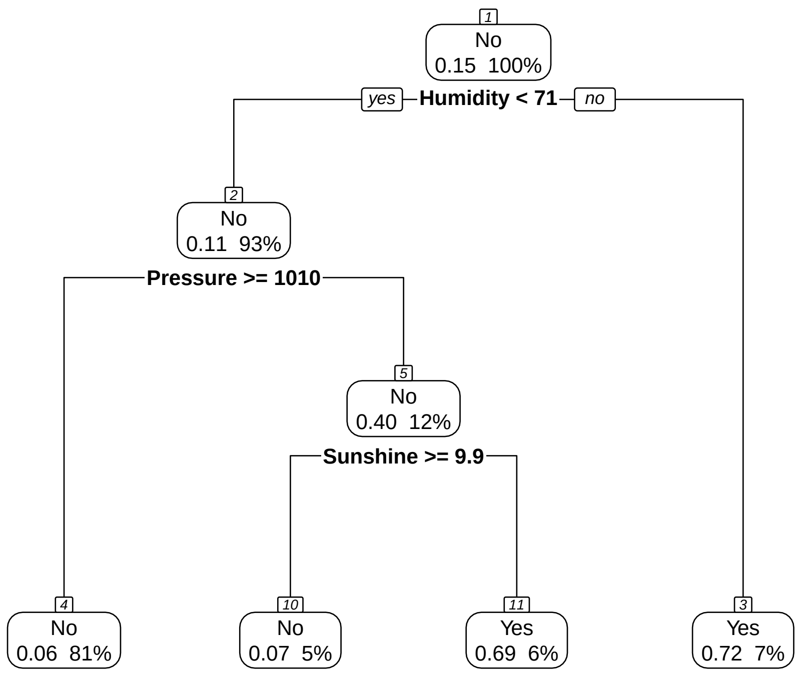
```r
prp(model, type=2, extra=106, nn=TRUE, ni=TRUE)
```



These are the row numbers of the nodes within the model object's `frame` component.

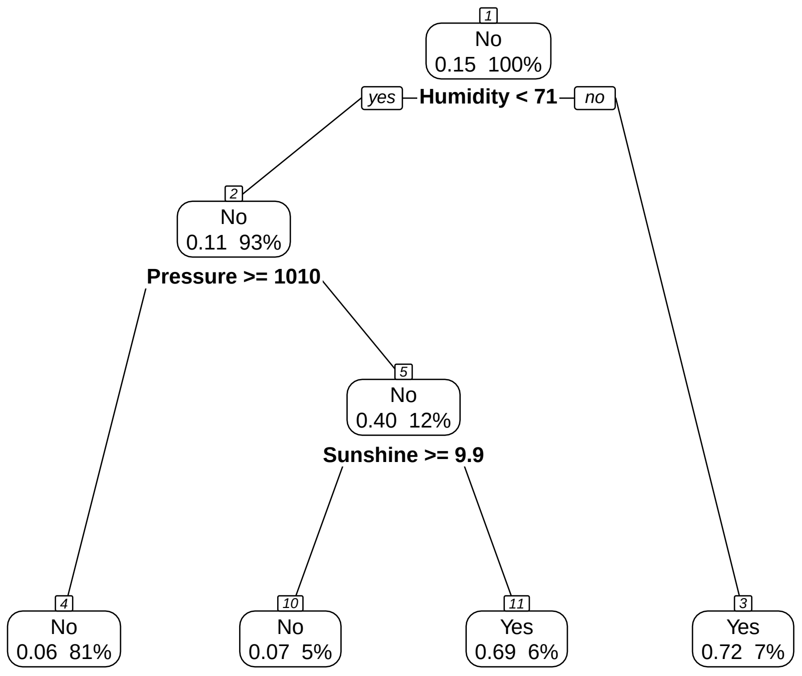
53 Enhanced Plots: Line up the Leaves

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE)
```



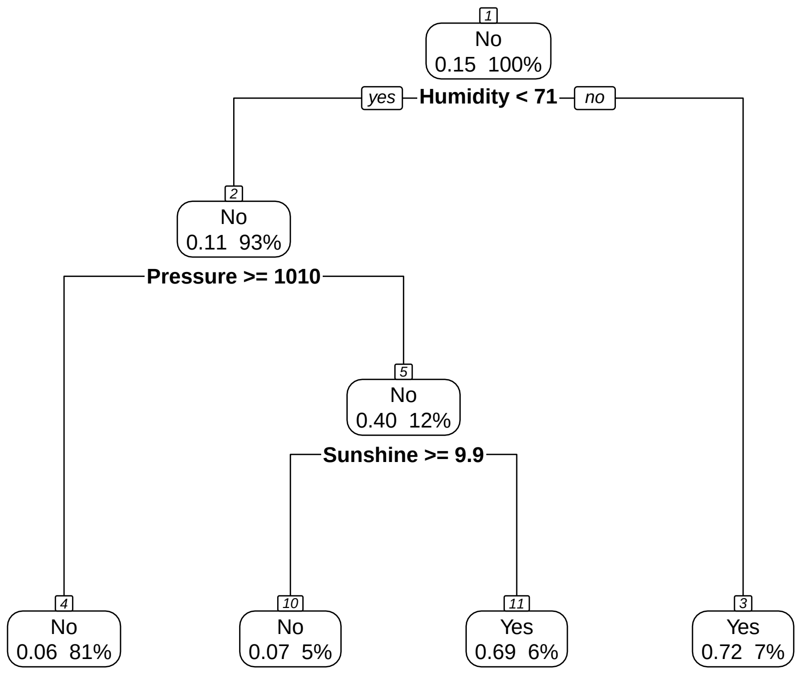
54 Enhanced Plots: Angle Branch Lines

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    branch=0.5)
```



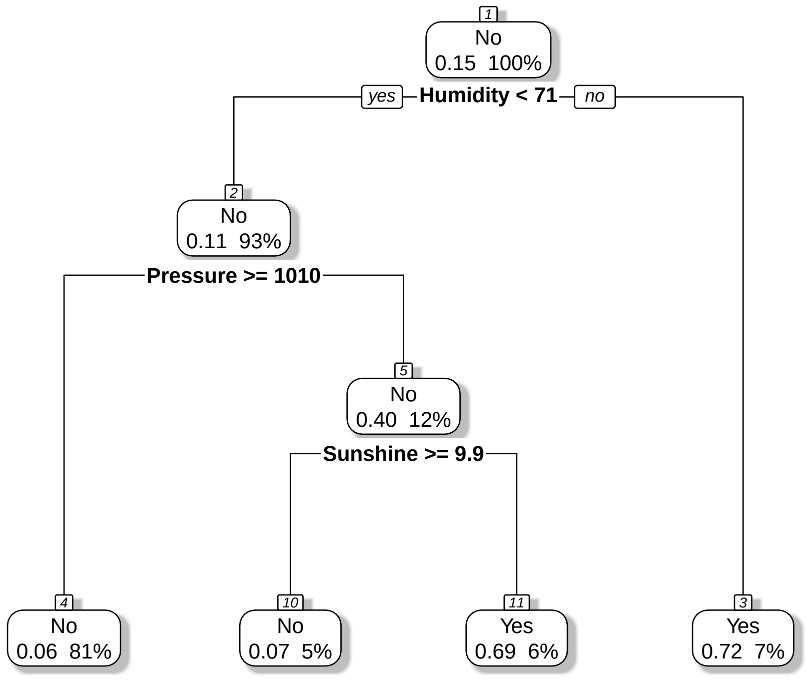
55 Enhanced Plots: Do Not Abbreviate Factors

```r
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    faclen=0)
```



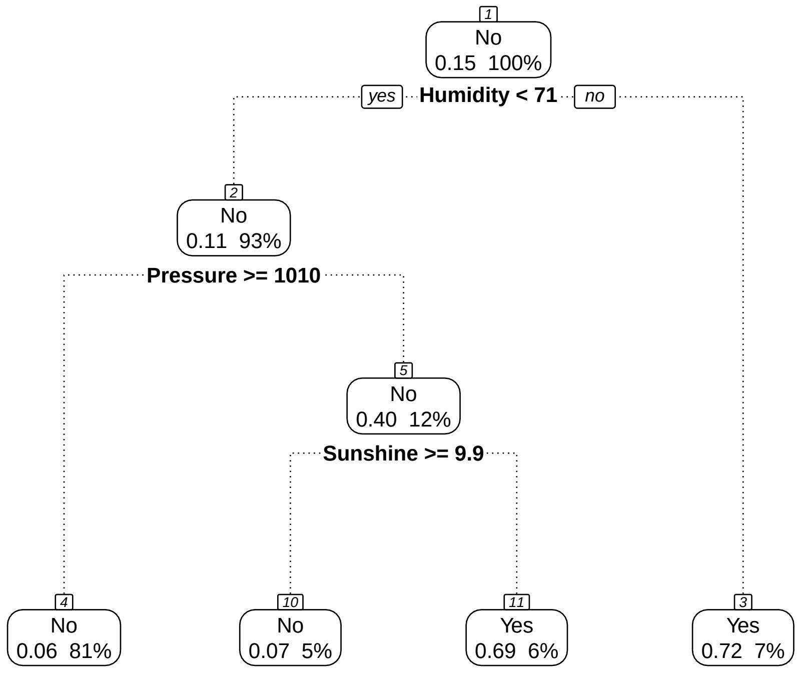
56 Enhanced Plots: Add a Shadow to the Nodes

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    shadow.col="grey")
```



57 Enhanced Plots: Draw Branches as Dotted Lines

```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
    branch.lty=3)
```



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:

More appropriate in Plots module.

0	blank	1	solid	2	dashed "44"	3	dotted "13"	4	dotdash "1343"	5	longdash "73"	6	twodash "2262"

Add line example into each cell of table.

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`:

```
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")
```

58 Enhanced Plots: Other Options

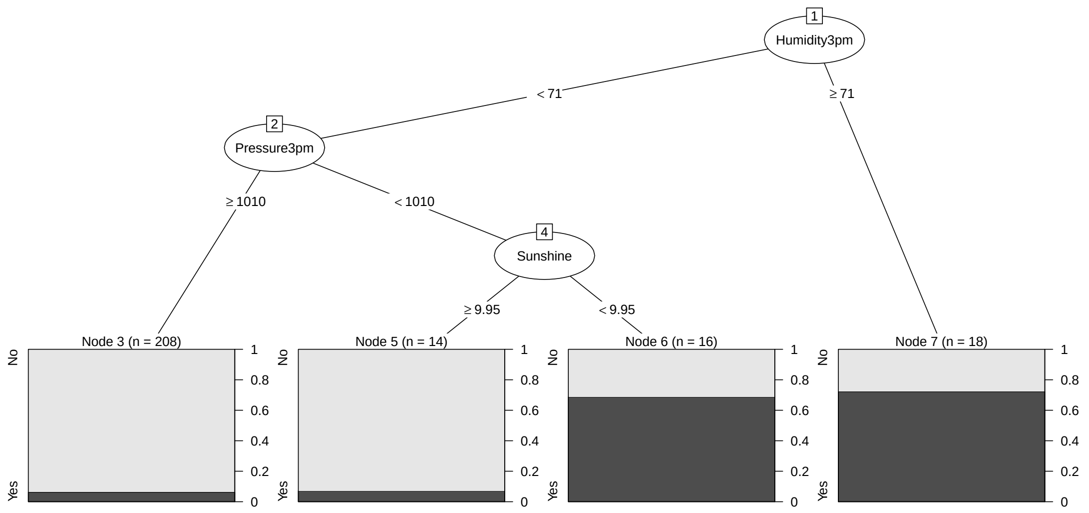
Exercise: Explore examples of split.cex=1.2, split.prefix="is ", split.suffix="?", col=cols, border.col=cols, split.box.col="lightgray", split.border.col="darkgray", split.round=.5, and other parameters you might discover.

59 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:

```
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.

```
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%)
....
```

60 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).

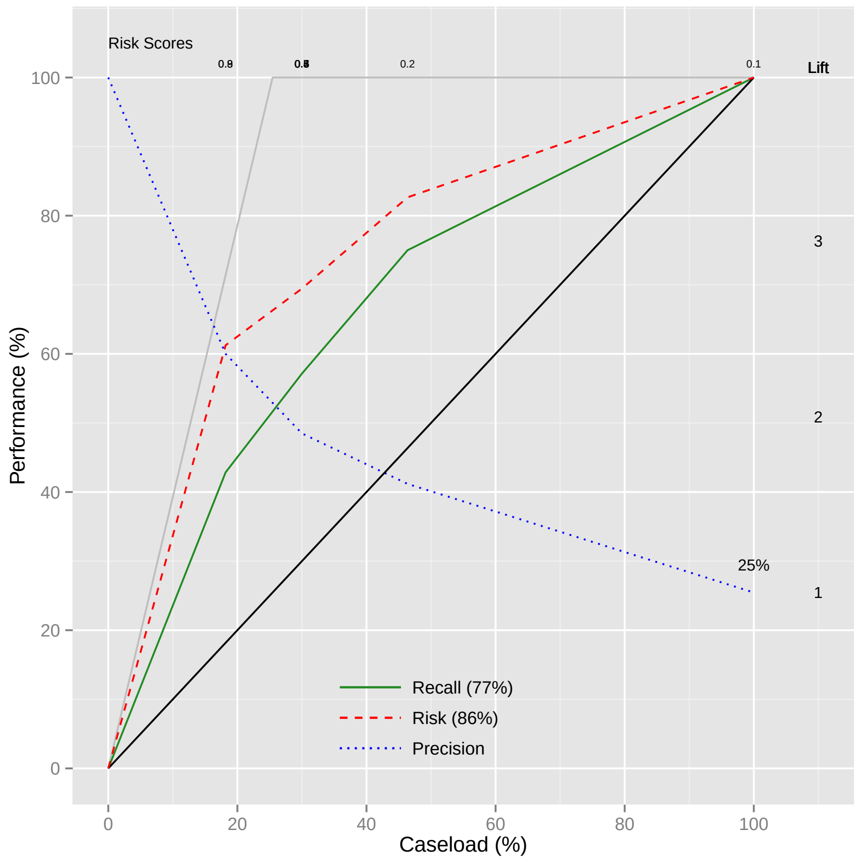
```
library(partykit)
model <- ctree(formula=form, data=ds[train, vars])
```

```
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=37, err=11%)
## |   [5] Sunshine > 6.4
## |   |   [6] Pressure3pm <= 1010.2: No (n=25, err=28%)
## |   |   [7] Pressure3pm > 1010.2: No (n=165, err=3%)
##
## Number of inner nodes:    3
## Number of terminal nodes: 4
```

61 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.

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

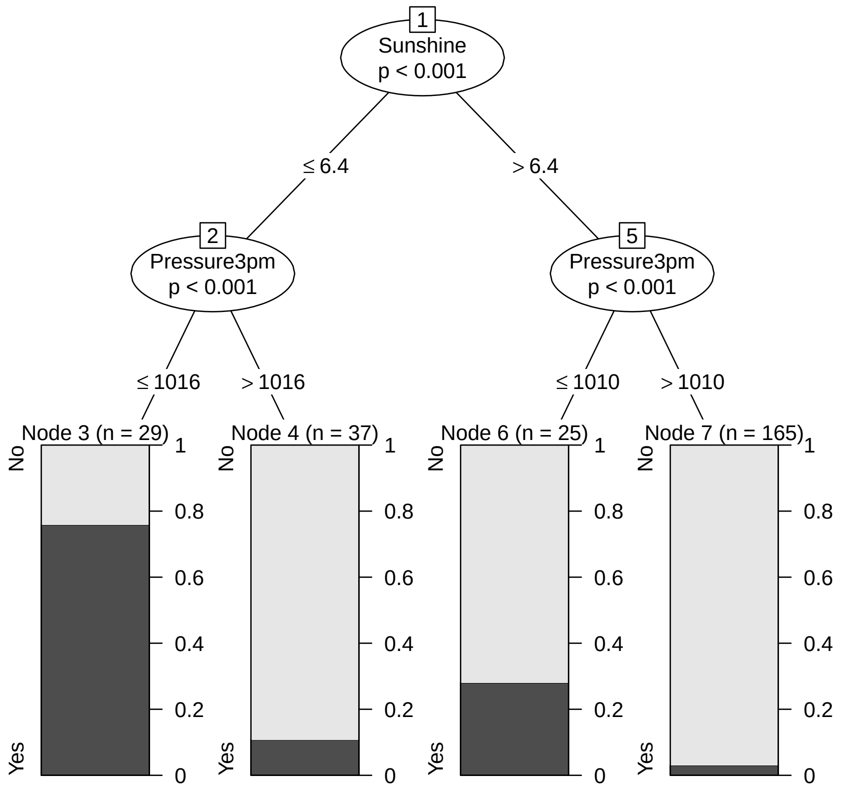


An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
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
....
```

62 CTree Plot

```
plot(model)
```



63 Weka Decision Tree

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

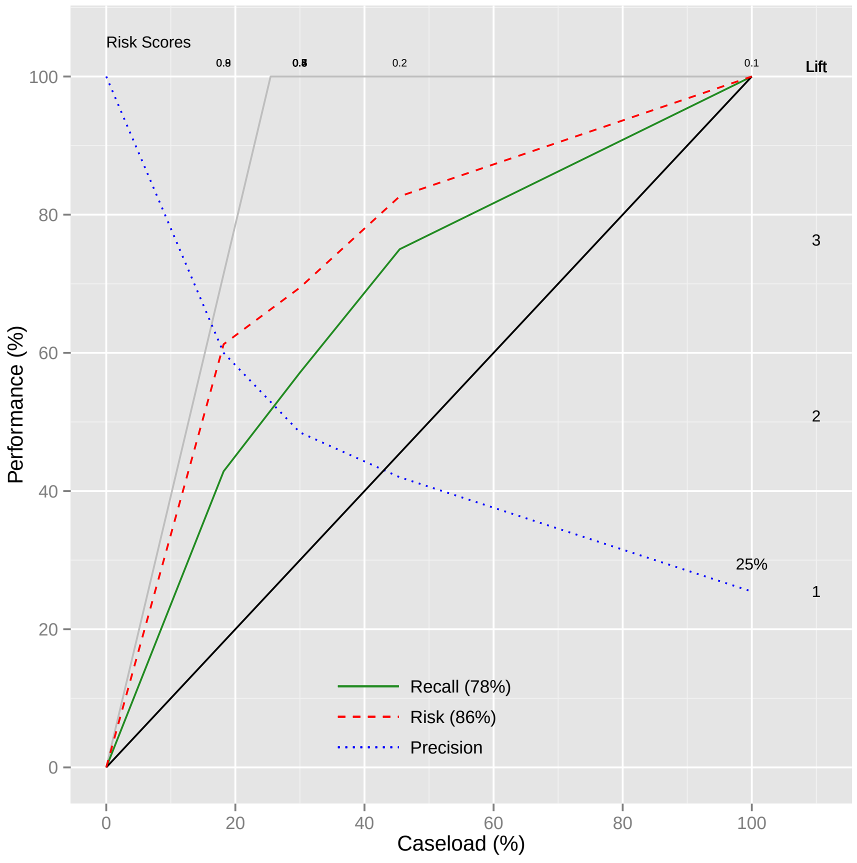
```
library(RWeka)

## Error:  there is no package called 'RWeka'

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

## Error:  could not find function "J48"
```

```
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=37, err=11%)
## |   [5] Sunshine > 6.4
## |   |   [6] Pressure3pm <= 1010.2: No (n=25, err=28%)
## |   |   [7] Pressure3pm > 1010.2: No (n=165, err=3%)
##
## Number of inner nodes:    3
## Number of terminal nodes: 4
```

64 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.

```
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.

```
predicted <- predict(model, ds[test, vars], type="class")

## Error:  'arg' should be one of "response", "prob", "quantile", "density", "node"

sum(actual != predicted)/length(predicted) # Overall error rate

## [1] 1

round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))

##       Predicted
## Actual 0.0303030303030303 0.108108108108108 0.28 0.758620689655172
##    No                  48                11    8                 7
##    Yes                    6                   5    4                  11
....
```

65 Weka Decision Tree Plot Using Party

```
plot(as.party(model))

## Error:  error in evaluating the argument 'x' in selecting a method for function
'plot':  Error in UseMethod("as.party") :
##  no applicable method for 'as.party' applied to an object of class "c('constparty',
'party')"
## Calls:  as.party
```

We can also display a textual version using party

```
print(as.party(model))

## Error:  no applicable method for 'as.party' applied to an object of class "c('constparty',
'party')"
```

66 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.

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

```
model

##
## 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
```

```
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
```

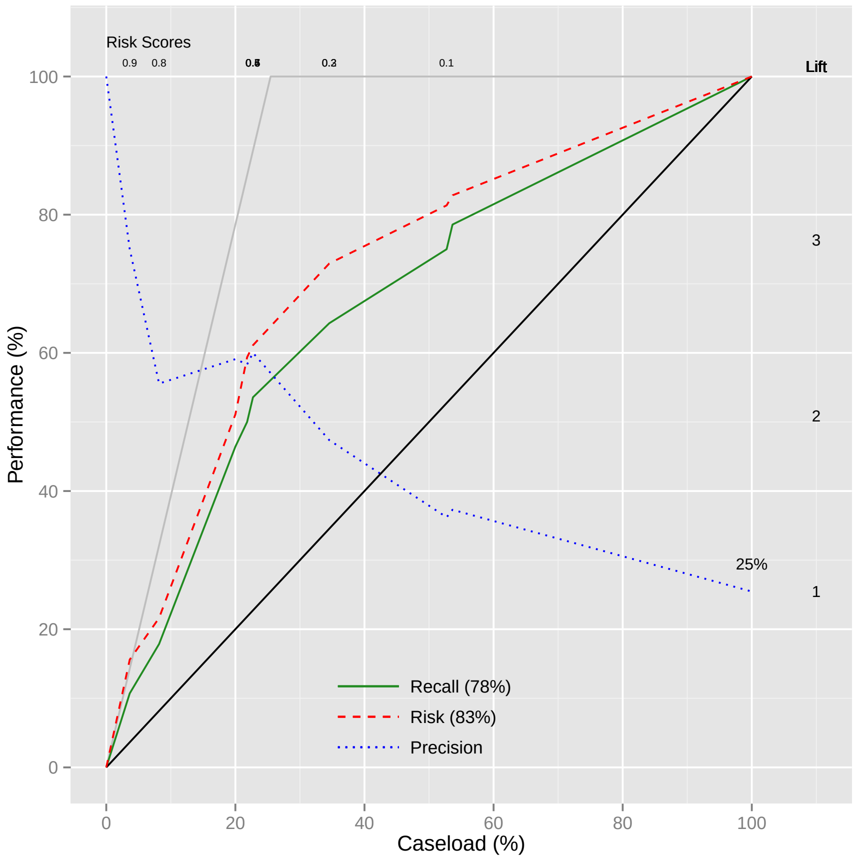
67 C5.0 Summary

```
summary(model)
##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars])
##
##
## C5.0 [Release 2.07 GPL Edition]   Sun Apr  6 07:09:38 2014
## -------------------------------
##
## 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
....
```

68 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.

```
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.

```
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))

##       Predicted
## Actual No Yes
##    No  65   9
##    Yes 13  13
....
```

69 C5.0 Rules Model

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

```
model

##
## 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
```

```
C5imp(model)

##                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
```

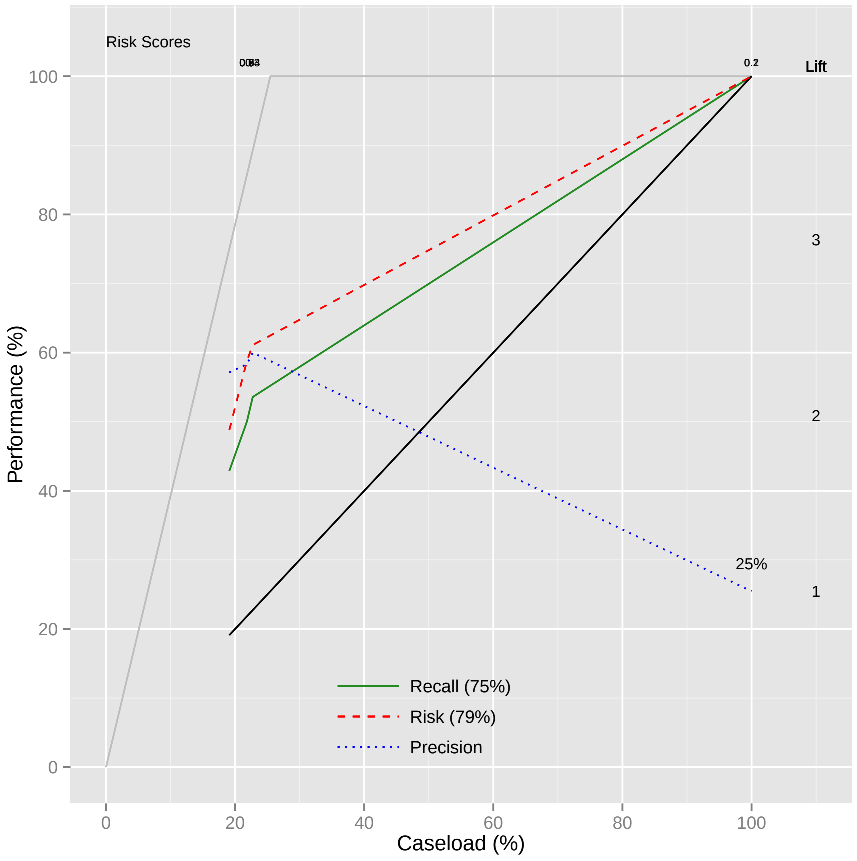
70 C5.0 Rules Summary

```
summary(model)
##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars], rules=TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]   Sun Apr  6 07:09:39 2014
## -------------------------------
##
## 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)
....
```

71 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.

```
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.

```
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
....
```

72 Regression Trees

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

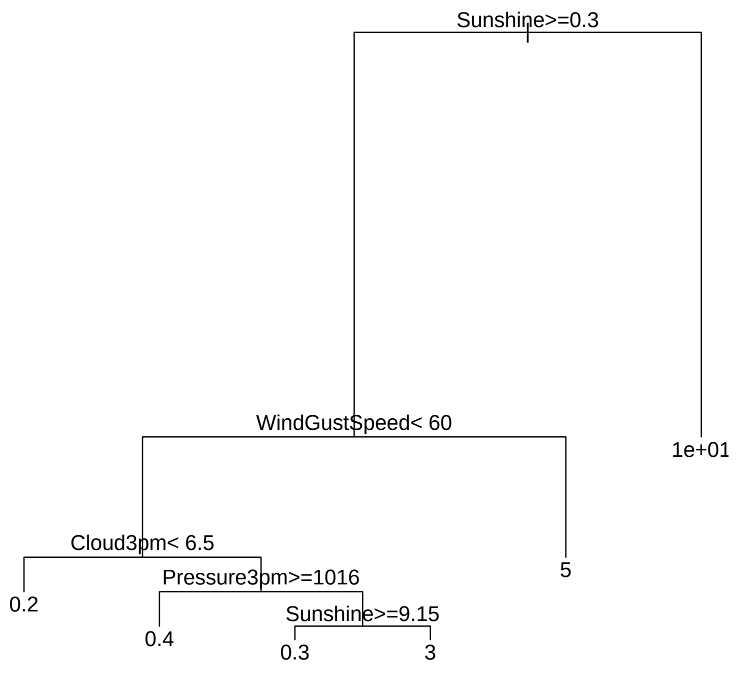
We can plot regression trees as with classification trees, but the node information will be different and some options will not make sense. For example, `extra=` only makes sense for 100 and 101.

First we will build regression tree:

```
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 *
```

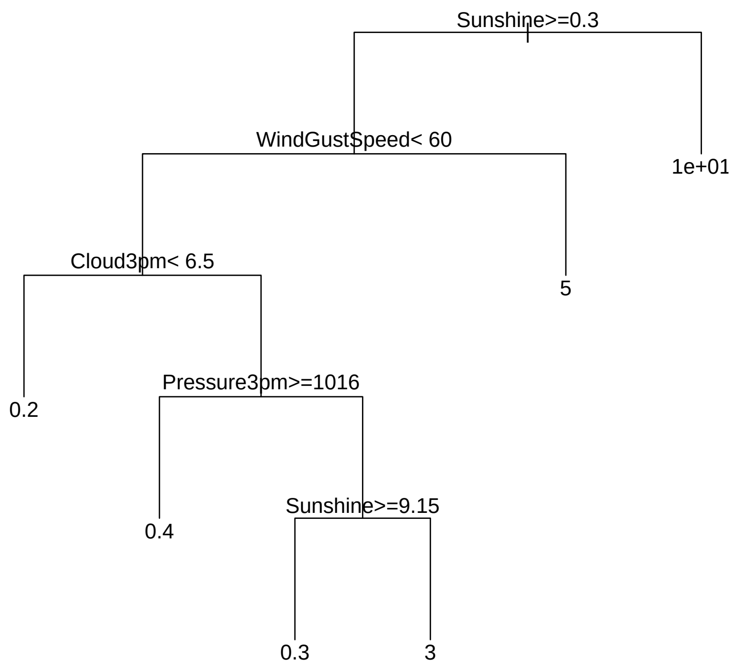
73 Visualise Regression Trees

```
plot(model)
text(model)
```



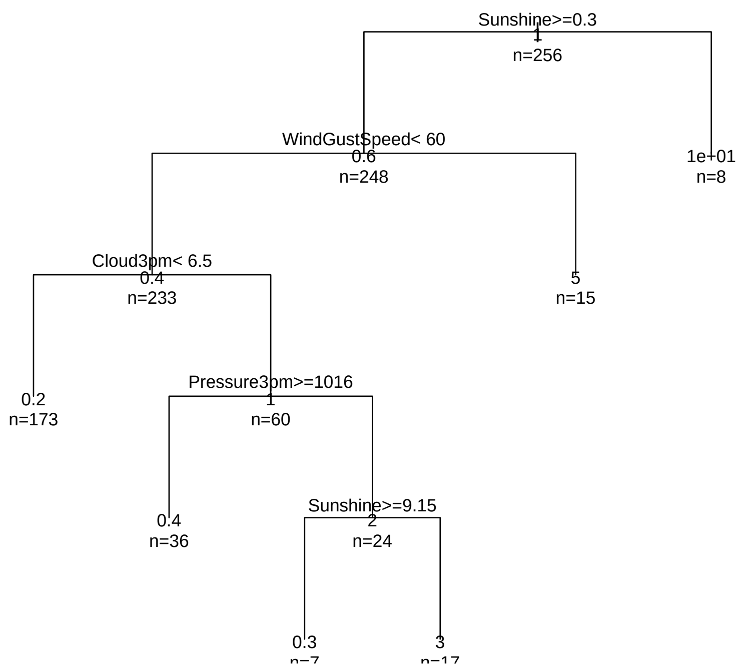
74 Visualise Regression Trees: Uniform

```
plot(model, uniform=TRUE)
text(model)
```



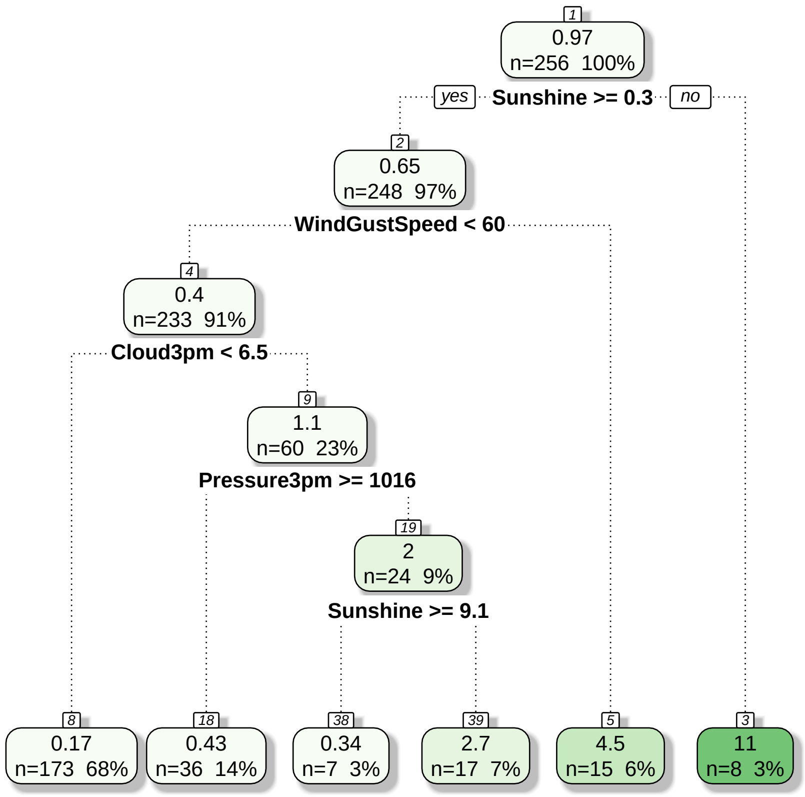
75 Visualise Regression Trees: Extra Information

```
plot(model, uniform=TRUE)
text(model, use.n=TRUE, all=TRUE, cex=.8)
```



76 Fancy Plot of Regression Tree

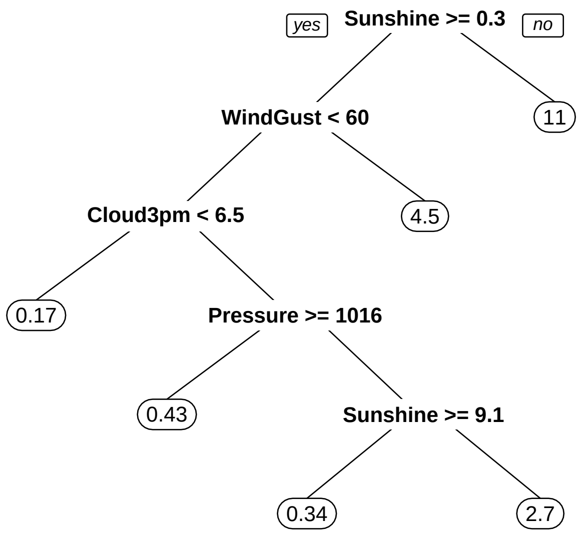
```
fancyRpartPlot(model)
```



Rattle 2014−Apr−06 07:09:41 gjw

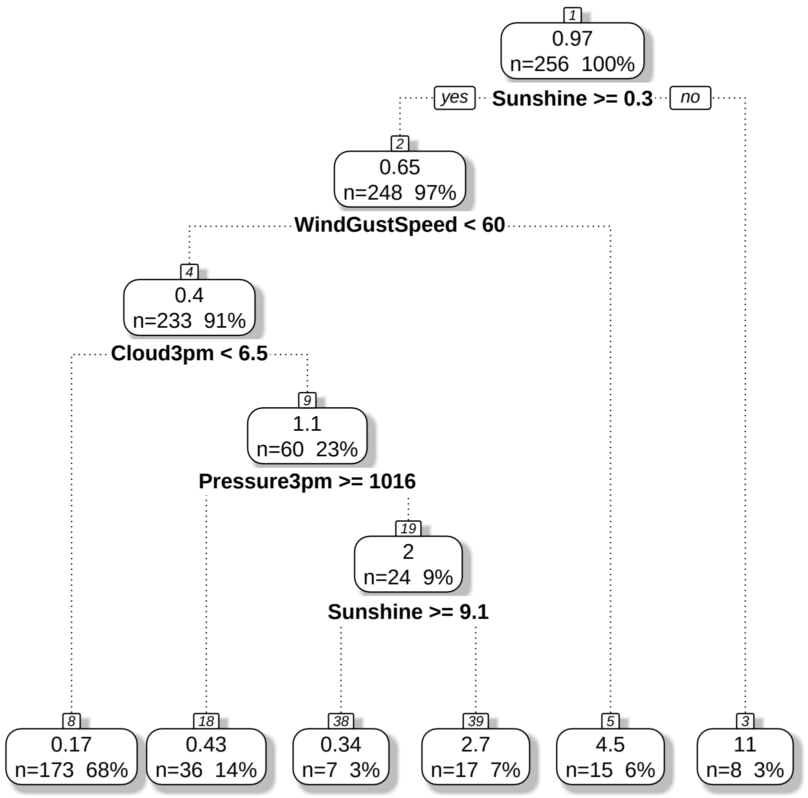
77 Enhanced Plot of Regression Tree: Default

```
prp(model)
```



78 Enhanced Plot of Regression Tree: Favourite

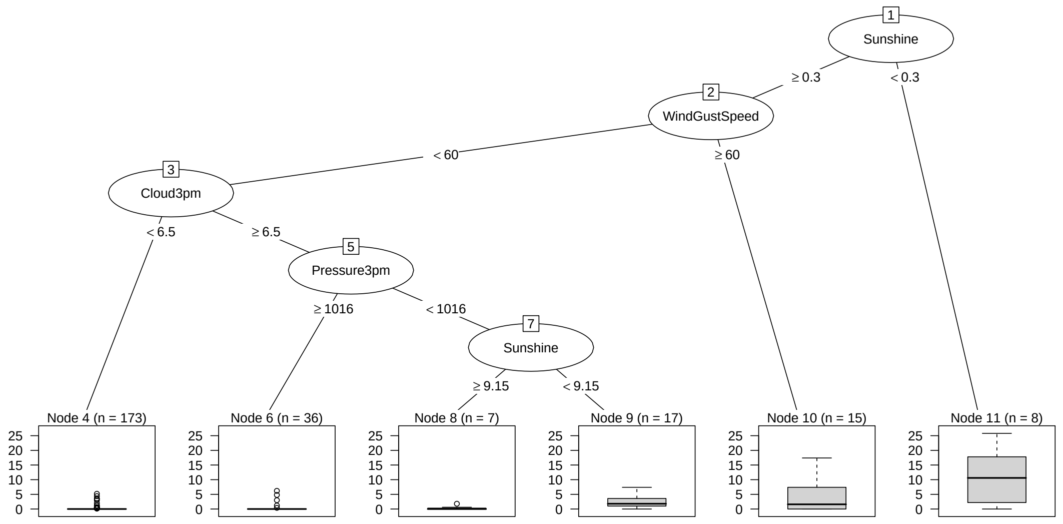
```
prp(model, type=2, extra=101, nn=TRUE, fallen.leaves=TRUE,
    faclen=0, varlen=0, shadow.col="grey", branch.lty=3)
```



79 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()`.

```
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.

80 Conditional Regression Tree

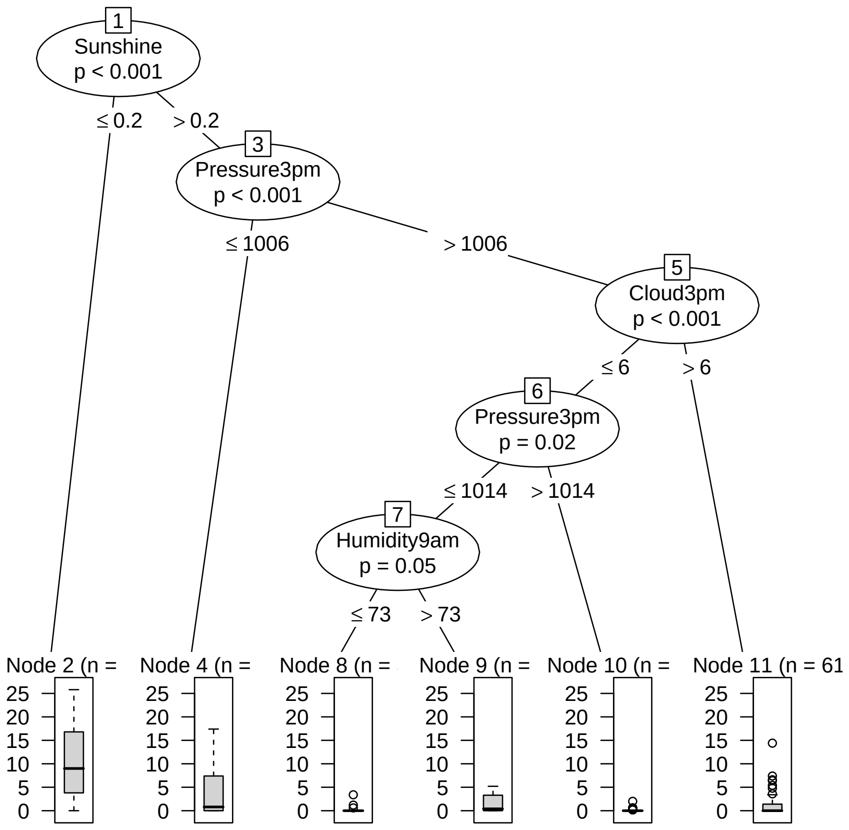
We can also build a regression tree using party (Hothorn *et al.*, 2013).

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

```
model

##
## 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: 10 (n=9, err=619)
## |   [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)
## |   |   |   [11] Cloud3pm > 6: 1 (n=61, err=375)
##
## Number of inner nodes:    5
## Number of terminal nodes: 6
```

81 CTree Plot

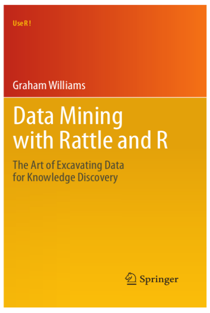
```
plot(model)
```



82 Weka Regression Tree

Weka's `J48()` does not support regression trees.

83 Further Reading


Use R!
Graham Williams
Data Mining with Rattle and R
The Art of Excavating Data for Knowledge Discovery
Springer

- 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 module is one of many OnePageR modules available from `http://onepager.togaware.com`. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

- `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.

84 References

Hothorn T, Hornik K, Strobl C, Zeileis A (2013). *party: A Laboratory for Recursive Partytioning*. R package version 1.0-9, URL http://CRAN.R-project.org/package=party.

Hothorn T, Zeileis A (2014). *partykit: A Toolkit for Recursive Partytioning*. R package version 0.8-0, URL http://CRAN.R-project.org/package=partykit.

Kuhn M, Weston S, Coulter N, Quinlan JR (2014). *C50: C5.0 Decision Trees and Rule-Based Models*. R package version 0.1.0-16, URL http://CRAN.R-project.org/package=C50.

Milborrow S (2014). *rpart.plot: Plot rpart models. An enhanced version of plot.rpart*. R package version 1.4-4, URL http://CRAN.R-project.org/package=rpart.plot.

Neuwirth E (2011). *RColorBrewer: ColorBrewer palettes*. R package version 1.0-5, URL http://CRAN.R-project.org/package=RColorBrewer.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Therneau TM, Atkinson B (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8, URL http://CRAN.R-project.org/package=rpart.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

Williams GJ (2014). *rattle: Graphical user interface for data mining in R*. R package version 3.0.4, URL http://rattle.togaware.com/.

This document, sourced from DTreesO.Rnw revision 332, was processed by KnitR version 1.5 of 2013-09-28 and took 37.8 seconds to process. It was generated by gjw on nyx running Ubuntu 13.10 with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-04-06 07:09:43.