# Data Science with R
# Building Models—A Template

Graham.Williams@togaware.com

20th August 2014

Visit http://HandsOnDataScience.com/ for more Chapters.

In this module we introduce a generic template for building models using R. The intention is that this document and the included scripts serve as a template for building a model and evaluating the model. This module builds on the Data module where we developed a template for preparing a dataset for analysing. The **weather** dataset from rattle (Williams, 2014) is used and for modeller we use rpart (Therneau and Atkinson, 2014).

The required packages for this module include:

```
library(rpart)          # Build decision tree model with rpart().
library(randomForest)   # Build model with randomForest().
library(ada)            # Build boosted trees model with ada().
library(rattle)         # Display tree model with fancyRpartPlot().
library(ROCR)           # Use prediction() for evaluation.
library(party)          # Build conditional tree models with ctree() and cforest().
library(ggplot2)        # Display evaluations.
```

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.

# 1 Getting Started—Load the Dataset

In the Data module we loaded the **weather** dataset, processed it, and saved it to file. Here we load the dataset and review its contents.

```
(load("weather_130704.RData"))

##  [1] "ds"     "dsname" "dspath" "dsdate" "target" "risk"   "id"
##  [8] "ignore" "vars"   "nobs"   "omit"   "inputi" "inputc" "numi"
## [15] "numc"   "cati"   "catc"

dsname

## [1] "weather"

dspath

## [1] "/home/gjw/R/x86_64-pc-linux-gnu-library/3.1/rattle/csv/weather.csv"

dsdate

## [1] "_130704"

dim(ds)

## [1] 366  24

id

## [1] "date"     "location"

target

## [1] "rain_tomorrow"

risk

## [1] "risk_mm"

ignore

## [1] "date"          "location"      "risk_mm"       "temp_3pm"
## [5] "pressure_9am" "temp_9am"

vars

##  [1] "min_temp"        "max_temp"        "rainfall"
##  [4] "evaporation"     "sunshine"        "wind_gust_dir"
##  [7] "wind_gust_speed" "wind_dir_9am"    "wind_dir_3pm"
## [10] "wind_speed_9am"  "wind_speed_3pm"  "humidity_9am"
## [13] "humidity_3pm"    "pressure_3pm"    "cloud_9am"
## [16] "cloud_3pm"       "rain_today"      "rain_tomorrow"
```

## 2 Step 4: Prepare—Formula to Describe the Goal

We continue on from the Data module where we had Steps 1, 2, and 3 and the beginnings of Step 4 of a data mining process.

The next step is to describe the model to be built by way of writing a formula to capture our intent. The formula describes the model to be built as being constructed to predict the target variable based on the other (suitable) variables available in the dataset. The notation used to express this is to name the target (*rain_tomorrow*), followed by a tilde (˜) followed by a period (.) to represent all other variables (these variables will be listed in *vars* in our case).

```
(form <- formula(paste(target, "~ .")))

## rain_tomorrow ~ .
```

The formula indicates that we will fit a model to predict *rain_tomorrow* from all of the other variables.

# 3 Step 4: Prepare—Training and Testing Datasets

A common methodology for model building is to randomly partition the available data into a **training** dataset and **testing** dataset. We sometimes also introducing a third dataset called the **validation** dataset, used during the building of the model, but for now we will use just the two.

First we (optionally) initiate the random number sequence with a randomly selected seed, and report what the seed is so that we could repeat the experiments presented here if required. For consistency in this module we use a particular seed of 123.

```
(seed <- sample(1:1000000, 1))

## [1] 534896

seed <- 123
set.seed(seed)
```

Next we partition the dataset into two subsets. The first is a 70% random sample for building the model (the training dataset) and the second is the remainder, used to evaluate the performance of the model (the testing dataset).

```
length(train <- sample(nobs, 0.7*nobs))

## [1] 256

length(test  <- setdiff(seq_len(nobs), train))

## [1] 110
```

Any modelling we do will thus build the model on the 70% training dataset. Our model will then be surely quite good at predicting these observations. But how will the model perform in general when we use it to predict other, yet unseen, observations? The model's performance on the data on which it was trained will be a very optimistic (or biased) estimate of the true performance of the model on other datasets.

The testing dataset is a hold-out dataset in that it has not been used at all for building the model. So when we apply the model to this dataset we would expect it to have a lesser performance (e.g., a higher error rate). This is what we will generally observe, and we will see this in the following sections.

The overall error rate measured on the training dataset will be shown to be less than the error rate calculated on the testing dataset.

The error rate (or the performance measure in general) calculated on the testing dataset is closer to what we will obtain in general when we begin to use the model. It is an unbiased estimate of the true performance of the model.

We also record the actual outcomes and the risks.

```
actual.train <- ds[train, target]
actual       <- ds[test, target]
risks        <- ds[test, risk]
```

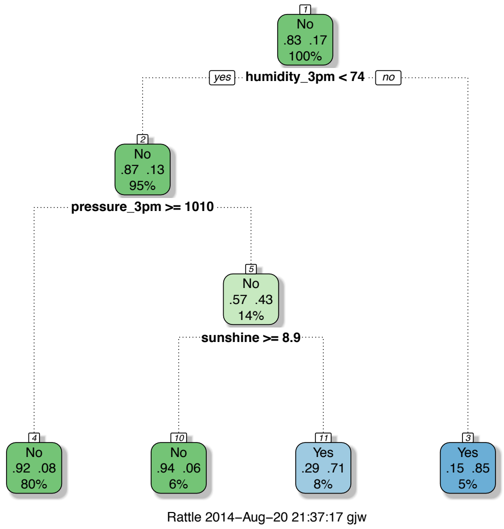
# 4 Step5: Build—Decision Tree

Now we build an `rpart()` decision tree, as an example model builder.

```
ctrl <- rpart.control(maxdepth=3)
system.time(model <- m.rp <- rpart(form, ds[train, vars], control=ctrl))

##    user  system elapsed
##   0.017   0.000   0.017

model

## n= 256
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 256 43 No (0.83203 0.16797)
##    2) humidity_3pm< 73.5 243 32 No (0.86831 0.13169)
##      4) pressure_3pm>=1010 206 16 No (0.92233 0.07767) *
##      5) pressure_3pm< 1010 37 16 No (0.56757 0.43243)
##       10) sunshine>=8.85 16  1 No (0.93750 0.06250) *
##       11) sunshine< 8.85 21  6 Yes (0.28571 0.71429) *
##    3) humidity_3pm>=73.5 13  2 Yes (0.15385 0.84615) *

mtype <- "rpart" # Record the type of the model for later use.
```

We can also draw the model.

```
fancyRpartPlot(model)
```



See the Decision Tree module for an explanation of decision tree models.

# 5 Step 6: Evaluate—Training Accuracy and AUC

We could now evaluate the model performance on the training dataset. As we noted above, this will give us a biased (optimistic) estimate of the true performance of the model.

First we use the model to predict the class of the observations of the training dataset.

```
head(cl <- predict(model, ds[train, vars], type="class"))

## 106 288 149 321 341  17
##  No  No  No  No Yes Yes
## Levels: No Yes
```

We are going to compare this to the actual class for these observations from the training dataset.

```
head(actual.train)

## [1] No  No  No  No  Yes Yes
## Levels: No Yes
```

We can calculate the overall accuracy over the training dataset, as simply the sum of the number of times the prediction agrees with the actual class, divided by the size of the training dataset (which is the same as the number of actual values of the class).

```
(acc <- sum(cl == actual.train, na.rm=TRUE)/length(actual.train))

## [1] 0.9023
```

The model has an overall accuracy of 90%.

We could alternatively calculate the overall error rate in a similar fashion:

```
(err <- sum(cl != actual.train, na.rm=TRUE)/length(actual.train))

## [1] 0.09766
```

The model has an overall error rate of 10%.

The area under the so-called ROC curve can also be calculated using ROCR (Sing *et al.*, 2013).

```
pr <- predict(model, ds[train, vars], type="prob")[,2]
pred <- prediction(pr, ds[train, target])
(atr <- attr(performance(pred, "auc"), "y.values")[[1]])

## [1] 0.7882
```

The area under the curve (AUC) is 79% of the total error.

# 6 Step 6: Evaluate—Training Accuracy and AUC

As we have noted though, performing any evaluation on the training dataset provides a biased estimate of the actual performance. We must instead evaluate the performance of our models on a previously unseen dataset (at least unseen by the algorithm building the model).

So we now evaluate the model performance on the testing dataset.

```
cl <- predict(model, ds[test, vars], type="class")
(acc <- sum(cl == actual, na.rm=TRUE)/length(actual))

## [1] 0.8364
```

The overall accuracy is 84%.

```
(err <- sum(cl != actual, na.rm=TRUE)/length(actual))

## [1] 0.1636
```

The overall error rate is 16%.

```
pr <- predict(model, ds[test, vars], type="prob")[,2]
pred <- prediction(pr, ds[test, target])
(ate <- attr(performance(pred, "auc"), "y.values")[[1]])

## [1] 0.6827
```

The AUC is 68%.

All of these performance measures are less than what we found on the trianing dataset, as we should expect.

# 7 Step 6: Evaluate—Confusion Matrix

Exersice: In one or two paragraphs, explain the concept of the confusion matrix and define true/false positive/negative.

Firstly for the training dataset.

```
cl <- predict(model, ds[train, vars], type="class")
round(100*table(actual.train, cl, dnn=c("Actual", "Predicted"))/length(actual.train))

##       Predicted
## Actual No Yes
##    No  80   3
##    Yes  7  10
....
```

Now for the testing dataset.

```
cl <- predict(model, ds[test, vars], type="class")
round(100*table(actual, cl, dnn=c("Actual", "Predicted"))/length(actual))

##       Predicted
## Actual No Yes
##    No  74   5
##    Yes 11  10
....
```

Once again notice that the performance on the testing dataset is lesser than the performance on the training dataset.

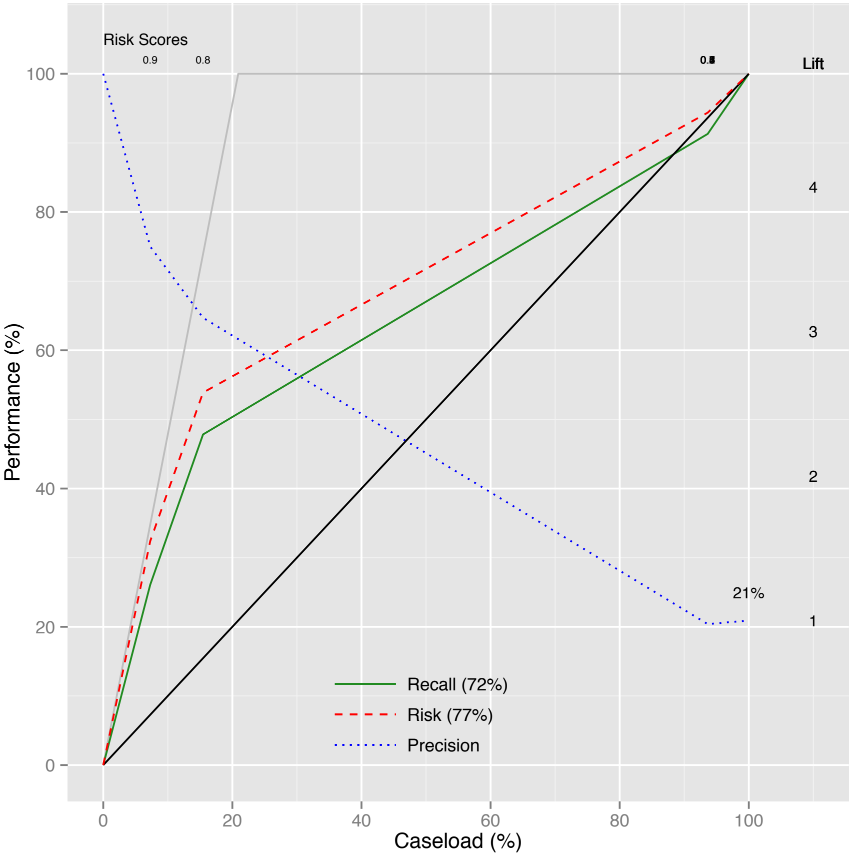
A **confusion matrix** is also called an **error matrix** or **contingency table**. The problem with calling them an error matrix is that not every cell in the table reports an error. The diagonals record the accuracy. Thus a quick look by a new data scientist might lead them to think the errors are high, based on the diagonals. The potential for this confusion is a good reason to call them something other than an error matrix.

Exercise: In the context of predicting whether it will rain tomorrow, and my decision to take an umbrella, explain the different consequences of a false positive and a false negative.

# 8 Step 6: Evaluate—Risk Chart

A risk chart is also known as an accumulative performance plot.

```
riskchart(pr, ds[test, target], ds[test, risk])
```



# 9 Step 7: Experiment—Framework

We can repeat the modelling multiple times, randomly selecting different datasets for training, to get an estimate of the actual expected performance and variation we see in the performance. The helper function `experi()` can be used to assist us here. It is available as http://onepager.togaware.com/experi.R and we show some of the coding of `experi()` below.

```
experi <- function(form, ds, dsname, target, modeller, details="",
                   n=100, control=NULL,
                   keep=FALSE, # Keep the last model built.
                   prob="prob",
                   class="class",
                   log="experi.log")
{
  suppressPackageStartupMessages(require(pROC))

  user <- Sys.getenv("LOGNAME")
  node <- Sys.info()[["nodename"]]

  wsrpart.model <- modeller=="wsrpart"

  numclass <- length(levels(ds[,target]))

  start.time <- proc.time()

  seeds <- cors <- strs <- aucs <- accs <- NULL
  for (i in seq_len(n))
  {
    loop.time <- proc.time()

    seeds  <- c(seeds, seed <- sample(1:1000000, 1))
    set.seed(seed)


....

    result[-c(1:7)]   <- round(result[-c(1:7)], 2)

  row.names(result) <- NULL

  if (keep)
  {
    if (numclass==2)
    {
      attr(result, "pr") <- pr
      attr(result, "test") <- test
    }
    attr(result, "model") <- model
  }
  return(result)
}
```

## 10 Step 7: Experiment—Results

Run the experiments using the algorihtms rpart (Therneau and Atkinson, 2014), randomForest (Breiman *et al.*, 2012), ada (Culp *et al.*, 2012), ctree() and cforest() from party (Hothorn *et al.*, 2013).

```
source("http://onepager.togaware.com/experi.R")
```

```
n <- 10
```

```
ex.rp <- experi(form, ds[vars], dsname, target, "rpart", "1", n=n, keep=TRUE)
```

```
ex.rf <- experi(form, ds[vars], dsname, target, "randomForest", "500", n=n, keep=TRUE,
                control=list(na.action=na.omit))
```

```
ex.ad <- experi(form, ds[vars], dsname, target, "ada", "50", n=n, keep=TRUE)
```

```
ex.ct <- experi(form, ds[vars], dsname, target, "ctree", "1", n=n, keep=TRUE)
```

```
# Generates: error code 1 from Lapack routine 'dgesdd'
ex.cf <- experi(form, ds[vars], dsname, target, "cforest", "500", n=n, keep=TRUE)
```

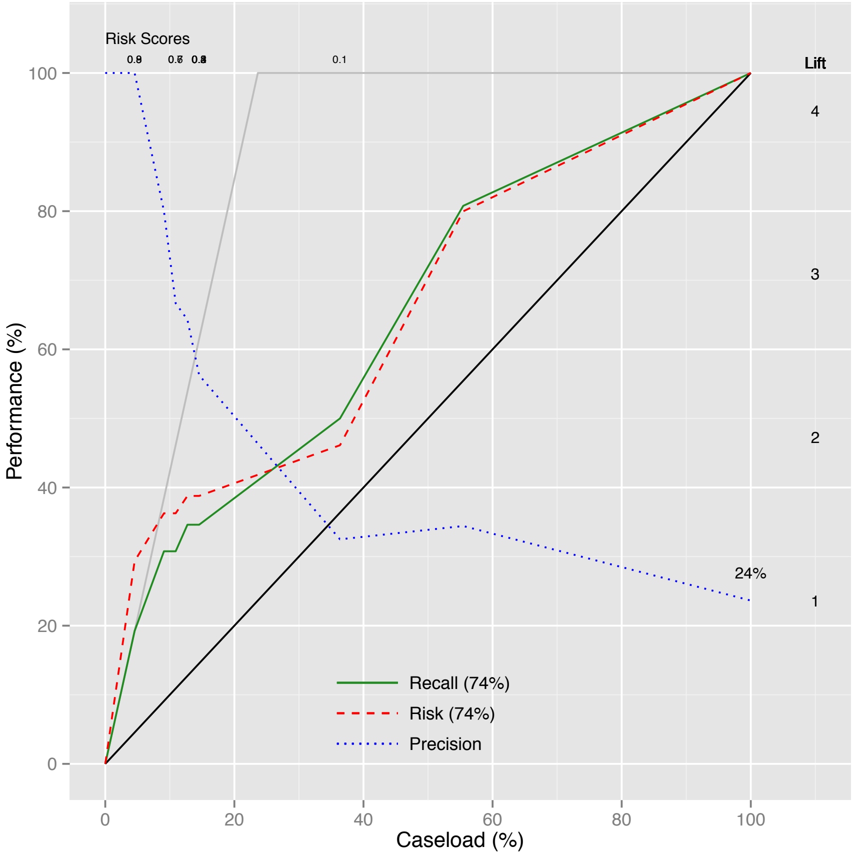
```
results <- rbind(ex.rp, ex.rf, ex.ad, ex.ct)
rownames(results) <- results$modeller
results$modeller <- NULL
results

##                    auc auc.sd cor cor.sd str str.sd  acc acc.sd  n  user
## rpart_1           0.72   0.08  NA     NA  NA     NA 0.81   0.02 10  0.94
## randomForest_500 0.84   0.08  NA     NA  NA     NA 0.77   0.04 10  2.38
## ada_50            0.82   0.02  NA     NA  NA     NA 0.84   0.03 10 14.07
## ctree_1           0.73   0.04  NA     NA  NA     NA 0.82   0.03 10  0.91
##                  elapsed
## rpart_1             0.95
## randomForest_500    2.39
## ada_50             14.11
## ctree_1             0.92
```

Exercise: Repeat these experiments using other model builders including C5.0, J48, lm, svm, nnet.

# 11 Step 7: Experiment—Riskchart Decision Tree

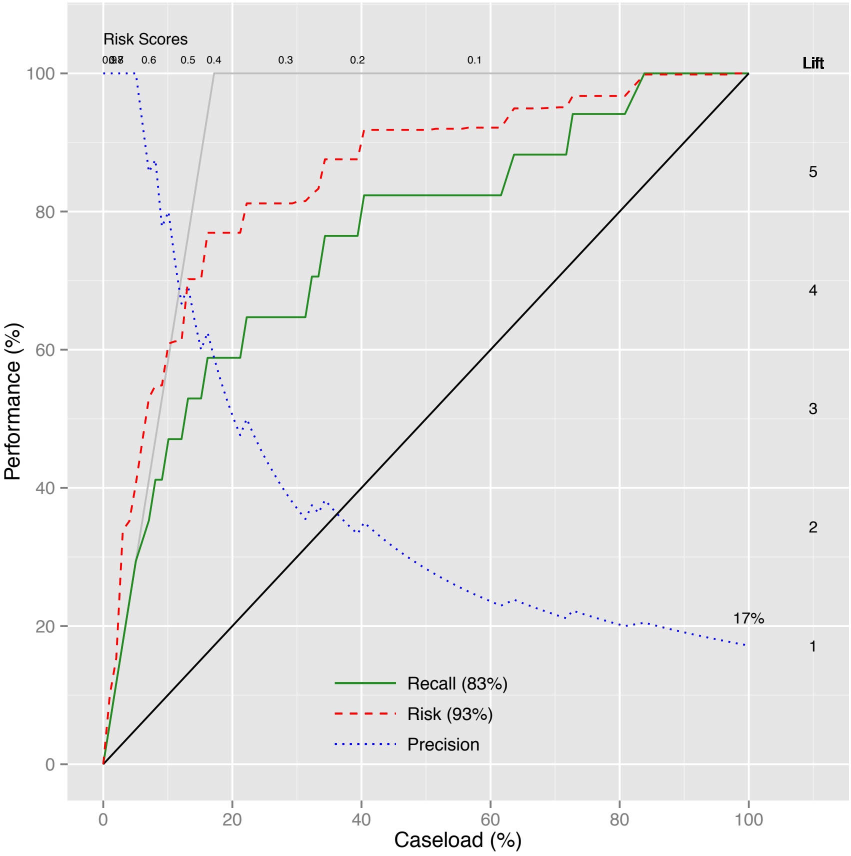
The result of the call to `experi()` includes the attributes *pr* and *test* if the number of classes is two and `keep=TRUE`. We can thus use this to generate a risk chart for the last of the models built in the experiment.

```
ex <- ex.rp
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



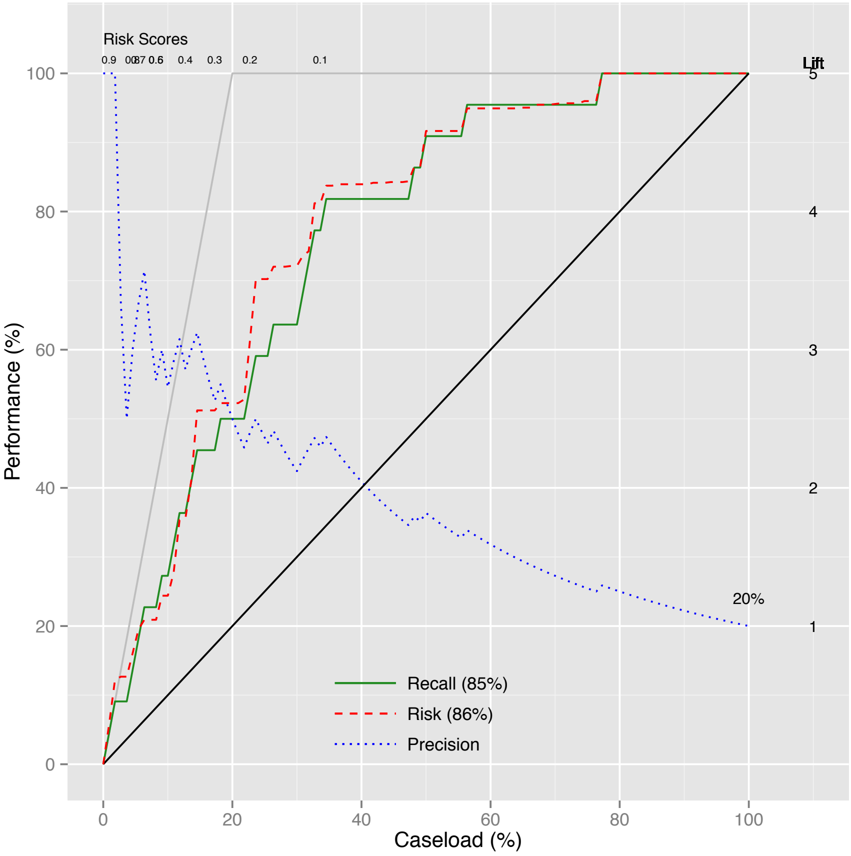
## 12 Step 7: Experiment—Riskchart Random Forest

```
ex <- ex.rf
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



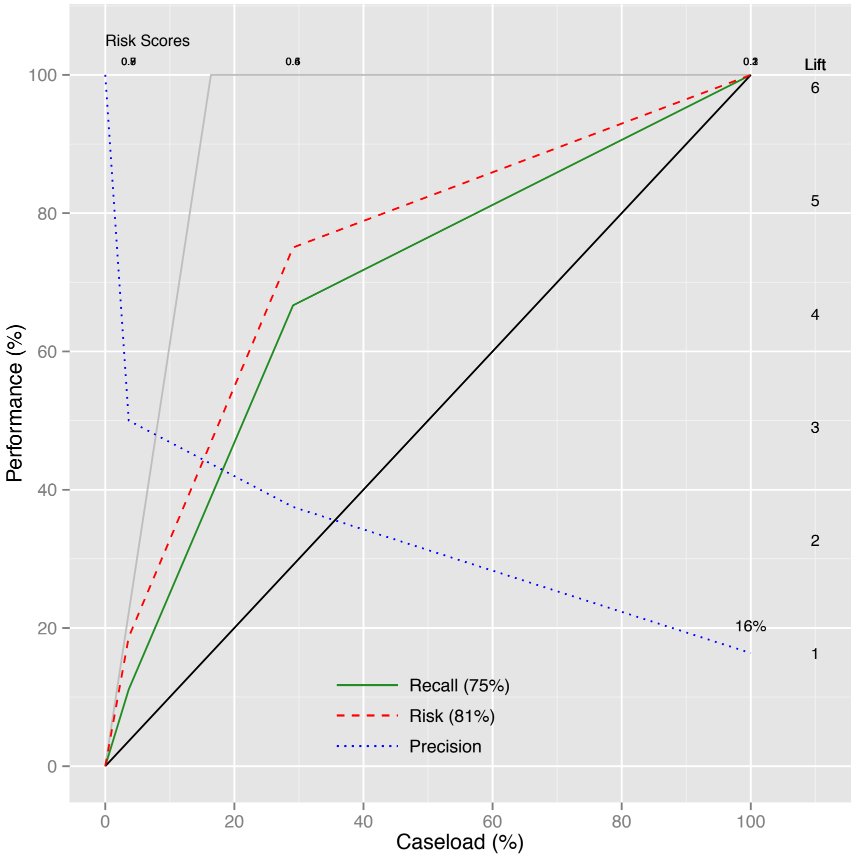
# 13 Step 7: Experiement—Riskchart Ada Boost

```
ex <- ex.ad
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



# 14 Step 7: Experiement—Riskchart Conditional Tree

```
ex <- ex.ct
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



## 15 Step 8: Finish Up—Save Model

We save the model, together with the dataset and other variables, into a binary R file.

```
dname <- "models"
if (! file.exists(dname)) dir.create(dname)
time.stamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
fstem <- paste(dsname, mtype, time.stamp, sep="_")
(fname <- file.path(dname, sprintf("%s.RData", fstem)))

## [1] "models/weather_rpart_20140820_213740.RData"

save(ds, dsname, vars, target, risk, ignore,
     form, nobs, seed, train, test, model, mtype, pr,
     file=fname)
```

We can then load this later and replicate the process.

```
(load(fname))

##  [1] "ds"     "dsname" "vars"   "target" "risk"   "ignore" "form"
##  [8] "nobs"   "seed"   "train"  "test"   "model"  "mtype"  "pr"
```

Note that by using generic variable names we can load different model files and perform common operations on them without changing the names within a script. However, do note that each time we load such a saved model file we overwrite any other variables of the same name.

## 16 Other Models—Random Forest

```
ctrl <- rpart.control(maxdepth=3)
system.time(model <- m.rf <- rpart(form, ds[train, vars], control=ctrl))
```

```
##    user  system elapsed
##   0.014   0.000   0.014
```

```
model
```

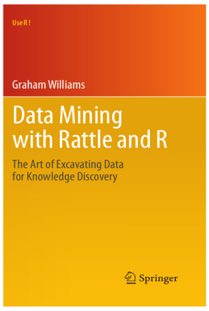
```
## n= 256
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 256 43 No (0.83203 0.16797)
##    2) humidity_3pm< 73.5 243 32 No (0.86831 0.13169)
##      4) pressure_3pm>=1010 206 16 No (0.92233 0.07767) *
##      5) pressure_3pm< 1010 37 16 No (0.56757 0.43243)
##       10) sunshine>=8.85 16  1 No (0.93750 0.06250) *
##       11) sunshine< 8.85 21  6 Yes (0.28571 0.71429) *
##    3) humidity_3pm>=73.5 13  2 Yes (0.15385 0.84615) *
```

# 17 Review—Model Process

Here in one block is the code to perform model building.

```
# Required packages
library(rpart)    # Model builder
library(rattle)   # riskchart()
library(ROCR)     # prediction()

# Load dataset.
load("weather_130704.RData")

# Formula for modelling.
form       <- formula(paste(target, "~ ."))

# Training and test datasets.
seed       <- sample(1:1000000, 1)
set.seed(seed)
train      <- sample(nobs, 0.7*nobs)
test       <- setdiff(seq_len(nobs), train)
actual     <- ds[test, target]
risks      <- ds[test, risk]

# Build model.
ctrl       <- rpart.control(maxdepth=3)
m.rp       <- rpart(form, data=ds[train, vars], control=ctrl)
mtype      <- "rpart"
model      <- m.rp

# Review model.
fancyRpartPlot(model)

# Evaluate the model.
classes    <- predict(model, ds[test, vars], type="class")
acc        <- sum(classes == actual, na.rm=TRUE)/length(actual)
err        <- sum(classes != actual, na.rm=TRUE)/length(actual)
predicted  <- predict(model, ds[test, vars], type="prob")[,2]
pred       <- prediction(predicted, ds[test, target])
ate        <- attr(performance(pred, "auc"), "y.values")[[1]]
riskchart(predicted, actual, risks)
psfchart(predicted, actual)
round(table(actual, classes, dnn=c("Actual", "Predicted"))/length(actual), 2)
```

## 18 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.

The process we have resented here for modelling from data has been tuned over many years of delivering analytics and data mining projects. An early influence was CRISP-DM the CRoss Industry Standard Process for Data Mining.

## 19 References

Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression.* R package version 4.6-7, URL http://CRAN.R-project.org/package=randomForest.

Culp M, Johnson K, Michailidis G (2012). *ada: ada: an R package for stochastic boosting.* R package version 2.0-3, URL http://CRAN.R-project.org/package=ada.

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.

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/.

Sing T, Sander O, Beerenwinkel N, Lengauer T (2013). *ROCR: Visualizing the performance of scoring classifiers.* R package version 1.0-5, URL http://CRAN.R-project.org/package=ROCR.

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 3.1.4, URL http://rattle.togaware.com/.

*This document, sourced from ModelsO.Rnw revision 502, was processed by KnitR version 1.6 of 2014-05-24 and took 24.3 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.1 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-08-20 21:37:41.*