Data Science with R
Evaluating Model Performance

Graham.Williams@togaware.com

16th August 2014

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

This module explores the options for evaluating the performance of models. We introduce techniques for evaluating the performance on a testing dataset, as well as ongoing performance evaluation.

The required packages for this module include:

```
library(rattle)          # Weather, riskchart() and psfchart().
library(dplyr)           # Use tbl_df().
library(e1071)           # naiveBayes().
library(rpart)           # Decision tree model rpart().
library(randomForest)    # Impute missing na.roughfix(), and randonmForest().
library(wsrpart)         # Build weighted subspart rpart wsrpart().
library(wsrf)            # Build weighted subspace random forest wsrf().
library(gmodels)         # Generate cross-tabulation using CrossTable().
library(ROCR)            # Plot ROC curves.
library(ggplot2)         # Plot ROC curves.
```

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 Prepare Data for Modelling

```
# Identify the dataset
dsname      <- "weatherAUS"
ds          <- tbl_df(get(dsname))
names(ds)   <- normVarNames(names(ds)) # Lower case variable names.
vars        <- names(ds)
target      <- "rain_tomorrow"
risk        <- "risk_mm"
id          <- c("date", "location")

# Ignore the IDs and the risk variable.
ignore      <- c(id, if (exists("risk")) risk)

# Ignore variables which are completely missing.
mvc         <- sapply(ds[vars], function(x) sum(is.na(x)))
mvn         <- names(which(mvc == nrow(ds)))
ignore      <- union(ignore, mvn)

# Initialise the variables
vars        <- setdiff(vars, ignore)

# Variable roles.
inputs      <- setdiff(vars, target)
numi        <- which(sapply(ds[inputs], is.numeric))
numc        <- names(numi)
cati        <- which(sapply(ds[inputs], is.factor))
catc        <- names(cati)

# Remove all observations with a missing target.
ds          <- ds[!is.na(ds[target]),]

# Impute missing values needed for randomForest().
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])

# Ensure the target is categoric.
ds[target] <- as.factor(ds[[target]])

# Number of observations.
nobs        <- nrow(ds)

# Prepare for model building.
form        <- formula(paste(target, "~ ."))
seed        <- 328058
train       <- sample(nobs, 0.7*nobs)
test        <- setdiff(seq_len(nobs), train)
actual      <- ds[test, target]
risks       <- ds[test, risk]
```

2 Build Models

We build a selection of models, including a decision tree (Therneau and Atkinson, 2014), random forest (Breiman *et al.*, 2012), weighted subspace of rpart decision trees (Zhalama and Williams, 2014) and weighted subspace random forest (Meng *et al.*, 2014).

```
# Naive Bayes
library(e1071)
model      <- m.nb <- naiveBayes(form, ds[train, vars])      #  1s
cl.nb      <- predict(model, ds[test, vars], type="class")   # 20s
pr.nb      <- predict(model, ds[test, vars], type="raw")[,2] # 20s

# Decision tree
library(rpart)
model      <- m.rp <- rpart(form, ds[train, vars])           # 6s
cl.rp      <- predict(model, ds[test, vars], type="class")
pr.rp      <- predict(model, ds[test, vars], type="prob")[,2]

# Random forest
library(randomForest)
model      <- m.rf <- randomForest(form, ds[train, vars], ntree=100) # 20s
cl.rf      <- predict(model, ds[test, vars], type="class")
pr.rf      <- predict(model, ds[test, vars], type="prob")[,2]

# Weighted subspace rpart
library(wsrpart)
model      <- m.wsrp <- wsrpart(form, ds[train, vars], ntree=10) # 30s
cl.wsrp    <- predict(model, ds[test, vars], type="class")
pr.wsrp    <- predict(model, ds[test, vars], type="prob")[,2]

# Weighted subspace random forest
library(wsrf)
model      <- m.wsrf <- wsrf(form, ds[train, vars], ntree=10) # 30s
cl.wsrf    <- predict(model, ds[test, vars], type="class")
pr.wsrf    <- predict(model, ds[test, vars], type="prob")[,2]
```

3 Confusion Matrix

A confusion matrix reports textually the performance of a predictive model's predictions against the actual classes.

We can easily generate a confusion matrix using `table()`. Here we report the actual number of predictions in the four categories of true/false positive/negative.

```
table(actual, cl.nb, dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual    No   Yes
##    No  17913  2954
##    Yes  2301  3886
```

```
table(actual, cl.rp, dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual    No   Yes
##    No  20171   696
##    Yes  3997  2190
```

```
table(actual, cl.rf, dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual    No   Yes
##    No  19775  1092
##    Yes  2866  3321
```

```
table(actual, cl.wsrp, dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual    No   Yes
##    No  20143   724
##    Yes  3928  2259
```

```
table(actual, cl.wsrf, dnn=c("Actual", "Predicted"))
```

```
##       Predicted
## Actual    No   Yes
##    No  19947   920
##    Yes  3328  2859
```

4 Confusion Matrix with Percentage

We can convert to percentages, and include a column to report on the class error rate:

```
pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  tbl <- cbind(round(x/length(actual), 2),
               Error=round(c(x[1,2]/sum(x[1,]), x[2,1]/sum(x[2,])), 2))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  tbl
}
pcme(actual, cl.nb)
```

```
##       Predicted
## Actual   No  Yes Error
##    No  0.66 0.11  0.14
##    Yes 0.09 0.14  0.37
```

```
pcme(actual, cl.rp)
```

```
##       Predicted
## Actual   No  Yes Error
##    No  0.75 0.03  0.03
##    Yes 0.15 0.08  0.65
```

```
pcme(actual, cl.rf)
```

```
##       Predicted
## Actual   No  Yes Error
##    No  0.73 0.04  0.05
##    Yes 0.11 0.12  0.46
```

```
pcme(actual, cl.wsrp)
```

```
##       Predicted
## Actual   No  Yes Error
##    No  0.74 0.03  0.03
##    Yes 0.15 0.08  0.63
```

```
pcme(actual, cl.wsrf)
```

```
##       Predicted
## Actual   No  Yes Error
##    No  0.74 0.03  0.04
##    Yes 0.12 0.11  0.54
```

5 Overall and Average Class Error

The overall error is simply the number of observations mis-classified divided by the total number of observations.

```
overall <- function(x) round((x[1,2] + x[2,1]) / sum(x), 2)
overall(table(actual, cl.nb)/length(actual))
```

```
## [1] 0.19
```

```
overall(table(actual, cl.rp)/length(actual))
```

```
## [1] 0.17
```

```
overall(table(actual, cl.rf)/length(actual))
```

```
## [1] 0.15
```

```
overall(table(actual, cl.wsrp)/length(actual))
```

```
## [1] 0.17
```

```
overall(table(actual, cl.wsrf)/length(actual))
```

```
## [1] 0.16
```

The overall error rate is sometimes quite a blunt measure of the performance of a model, and is particularly misleading when the classes are unbalanced. Consider the case where the majority class has an error rate of 10% and the minority class has an error rate of 30%. Overall the error rate will be closer to the 10% error rate because of the sheer number of observations of this class. That is quite misleading given our usual interest in the minority class.

The averaged class error rate is simply the average of the class errors.

```
avgerr <- function(x) round(mean(c(x[1,2], x[2,1]) / apply(x, 1, sum)), 2)
avgerr(table(actual, cl.nb)/length(actual))
```

```
## [1] 0.26
```

```
avgerr(table(actual, cl.rp)/length(actual))
```

```
## [1] 0.34
```

```
avgerr(table(actual, cl.rf)/length(actual))
```

```
## [1] 0.26
```

```
avgerr(table(actual, cl.wsrp)/length(actual))
```

```
## [1] 0.33
```

```
avgerr(table(actual, cl.wsrf)/length(actual))
```

```
## [1] 0.29
```

6 Cross Tabulation Confusion Matrix

A cross-tabulation can be used to generate a confusion-matrix to present the performance of a model. Here we use `CrossTable()` from `gmodels` (source code and/or documentation contributed by Ben Bolker *et al.*, 2013) to generate the table. This includes the Chi-square test of the independence of all table factors

```
library(gmodels)
CrossTable(actual, cl.nb)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  27054
##
##
##              | cl.nb
##       actual |        No |       Yes | Row Total |
## -------------|-----------|-----------|-----------|
##           No |     17913 |      2954 |     20867 |
##              |   345.742 |  1021.758 |           |
##              |     0.858 |     0.142 |     0.771 |
##              |     0.886 |     0.432 |           |
##              |     0.662 |     0.109 |           |
## -------------|-----------|-----------|-----------|
##          Yes |      2301 |      3886 |      6187 |
##              |  1166.090 |  3446.102 |           |
##              |     0.372 |     0.628 |     0.229 |
##              |     0.114 |     0.568 |           |
##              |     0.085 |     0.144 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     20214 |      6840 |     27054 |
##              |     0.747 |     0.253 |           |
## -------------|-----------|-----------|-----------|
##
##
```

7 Cross Tabulation: Random Forest

```
CrossTable(actual, cl.rf)
```

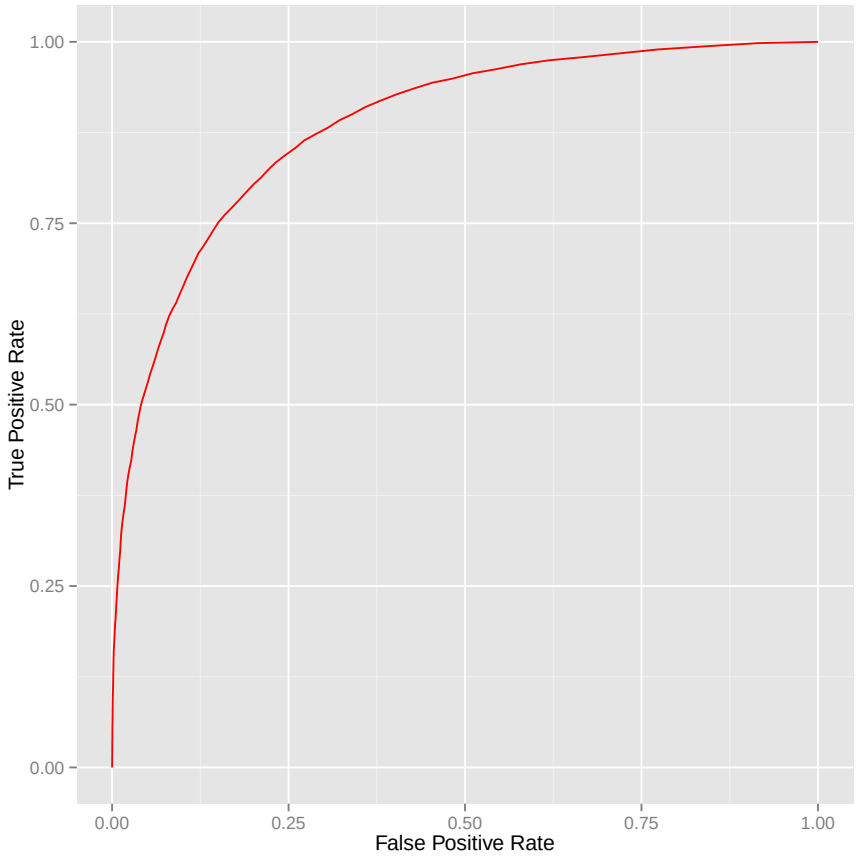
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  27054
##
##
##              | cl.rf
##       actual |        No |       Yes | Row Total |
## -------------|-----------|-----------|-----------|
##           No |     19775 |      1092 |     20867 |
##              |   306.035 |  1570.122 |           |
##              |     0.948 |     0.052 |     0.771 |
##              |     0.873 |     0.247 |           |
##              |     0.731 |     0.040 |           |
## -------------|-----------|-----------|-----------|
##          Yes |      2866 |      3321 |      6187 |
##              |  1032.171 |  5295.577 |           |
##              |     0.463 |     0.537 |     0.229 |
##              |     0.127 |     0.753 |           |
##              |     0.106 |     0.123 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     22641 |      4413 |     27054 |
##              |     0.837 |     0.163 |           |
## -------------|-----------|-----------|-----------|
##
##
```

8 ROC Curves

The Receiver Operating Characteristics (ROC) curve provides a visual guide to performance. The ROCR (Sing *et al.*, 2013) package provides the functionality.

```
library(ROCR)
pr <- prediction(pr.rf, actual)
pe <- performance(pr, "tpr", "fpr")
pd <- data.frame(fpr=unlist(pe@x.values), tpr=unlist(pe@y.values))

p <- ggplot(pd, aes(x=fpr, y=tpr))
p <- p + geom_line(colour="red")
p <- p + xlab("False Positive Rate") + ylab("True Positive Rate")
p
```



The ROC curve plots the true positive rate against the false positive rate. Here we see the curve and note that the more area under the curve (AUC) the better is the performance. To interpret, consider the false positive rate of 0.2 which corresponds to a true positive rate of about 0.8. Using the model to prioritise observations, to identify 80% of the true positives, we falsely include 20% of the false positives.

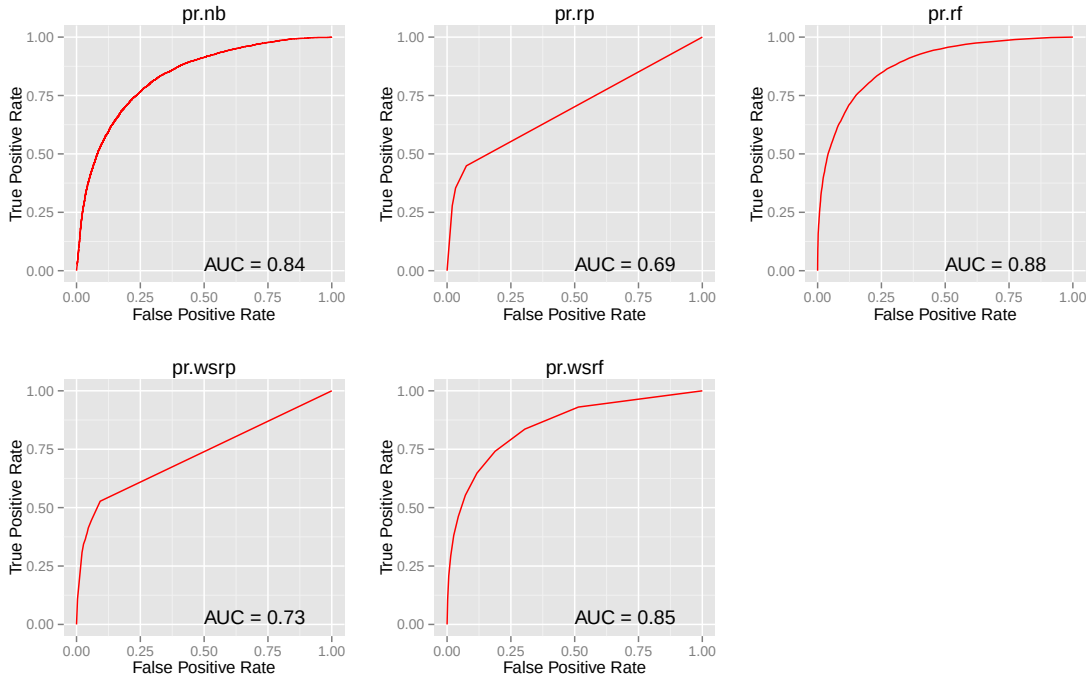
9 ROC Curves—All Models

Here we display all ROC curves. A simple function is used to capture the code that generates the plot from the predictions provided by the model.

```
library(ROCR)
library(gridExtra)

proc <- function(pr.m)
{
  pr <- prediction(pr.m, actual)
  pe <- performance(pr, "tpr", "fpr")
  au <- performance(pr, "auc")@y.values[[1]]
  pd <- data.frame(fpr=unlist(pe@x.values), tpr=unlist(pe@y.values))

  p <- ggplot(pd, aes(x=fpr, y=tpr))
  p <- p + geom_line(colour="red")
  p <- p + xlab("False Positive Rate") + ylab("True Positive Rate")
  p <- p + ggtitle(deparse(substitute(pr.m)))
  p <- p + annotate("text", x=0.50, y=0.00, hjust=0, vjust=0, size=5,
                    label=paste("AUC =", round(au, 2)))
  return(p)
}
grid.arrange(proc(pr.nb), proc(pr.rp), proc(pr.rf),
             proc(pr.wsrp), proc(pr.wsrf), ncol=3)
```

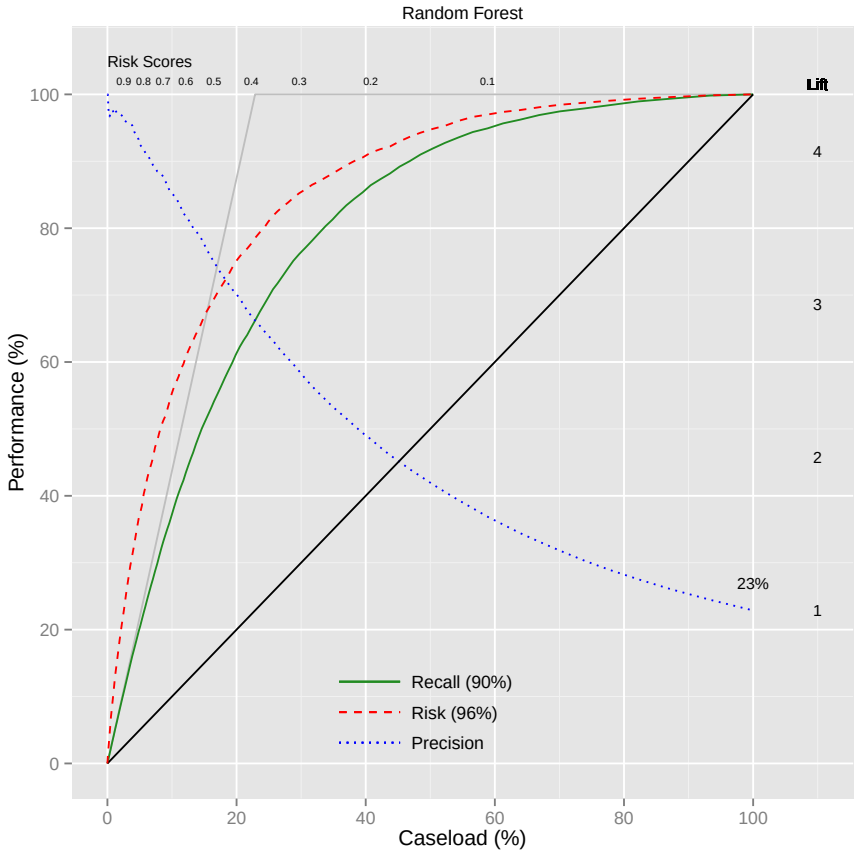


10 ROC Curves—Applied to In-Production Results

ROC curves and the common measure of the area under the curve (AUC) have been shown by Hand (2009) and Fawcett (2003) to have a number of deficiencies. Hari Koesmarno (2013) notes: *Hand (2009) illustrated that the area under the ROC curve (AUC) has a serious deficiency, especially the AUC using different misclassification cost distributions for different classifiers. This means that using the AUC is equivalent to using different metrics to evaluate different classification rules. ROC curves are commonly used in medical decision making and in recent years have been increasingly adopted in machine learning and data mining research communities (Fawcett, 2003). The incoherency of the AUC has been studied by Fawcett (2006) and Hand (2009). However ROC is suited to model comparison and selection with constant cost over a training (rather then an in-production) population.*

Exercise: Include some ROC Curves to illustrate this point.

11 Risk Chart

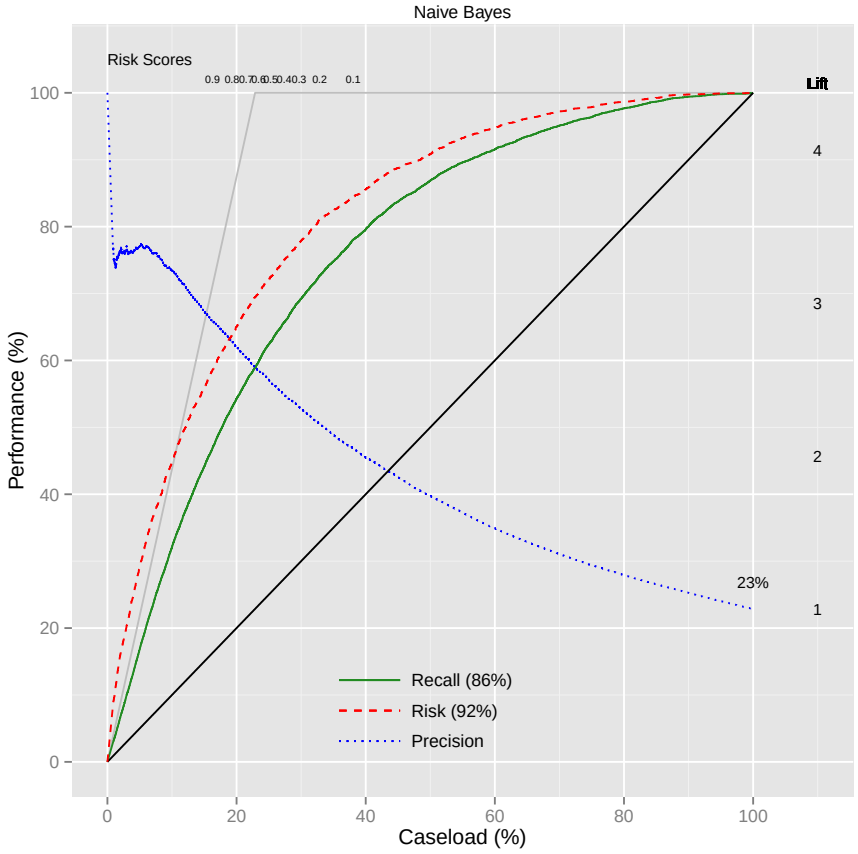


The concept of a risk chart was developed for the Australian Taxation Office, and implemented in rattle (Williams, 2014). It is also similar to a cumulative gain chart. We use the random forest model built earlier to generate the risk chart from the test dataset using `riskchart()` from Rattle.

```
riskchart(pr.rf, actual, risks, "Random Forest")
```

The risk chart plots the performance (the true positive rate) against the caseload (the combined observations).

12 Risk Chart: Naive Bayes




```
riskchart(pr.rf, actual, risks, "Random Forest")
```

13 Risk Chart — Interpreting Based on Financial Risk

Although the previous risk chart is actually based on the weather dataset, we will describe it in terms of predicting the risk that a income tax return is not correct. We might imagine 100 tax returns have been randomly chosen for auditing purposes. Of those just 23 have had some requirement to change their tax return and to pay additional taxes. This is the 23% label on the right hand end of the precision plot.

The x-axis is the case load. It represents the ordering by risk score of the tax payers. Those tax payers with the highest score begin the queue at the left hand end of the axis and those with the lowest scores at the right hand end of the axis. Thinking of the 100 taxpayers that have been risk scored standing in a queue, they might be numbered from 1 to 100, and form the queue from the highest risk score (on the left) to the lowest (on the right).

The distribution of the risk scores generated by the model are shown along the top of the chart. We can see the higher risk scores on the left and the lower scores to the right.

The y-axis is a measure of the performance of the model in identifying the tax payers who required an adjustment to be made to their tax return. A performance of 50%, for example, is then 50% of the 23 tax payers (11 or 12 tax payers) whose tax return required an adjustment.

The black diagonal line is the performance we obtain if we randomly selected tax payers from among the 100 chosen for auditing. If we randomly chose 40 tax payers (a caseload of 40%) then we would expect to have a performance of 40%. That is, we expect to identify 40% of the 23 tax payers requiring adjustments (9 tax payers)

The solid green and the dashed red lines are then the performance lift we obtain by using the model. It is measured as the recall (of the tax payers requiring adjustments) and the risk (which is the additional amount of tax the tax payer needs to pay). We use the model to prioritise the tax payers rather than selecting them at random. If we now select 40% of the tax payers again, but we chose those with the highest risk score, then we now expect to get about 84% of the 23 tax payers requiring adjustments (i.e., 19 of these tax payers). That is quite a lift over the 9 adjustment tax payers identified if we had selected the 40 to audit at random.

As we can see, using the model to select 40 tax payers delivers us twice as many of the tax payers that we are really interested in. This is a lift of 2, and we can read the lift off of the dotted blue precision line against the right hand axis. The precision or strike rate is actually the proportion of the 40 cases selected that are the target tax payers. We can see that it is close the 50%, using the left hand y-axis labels.

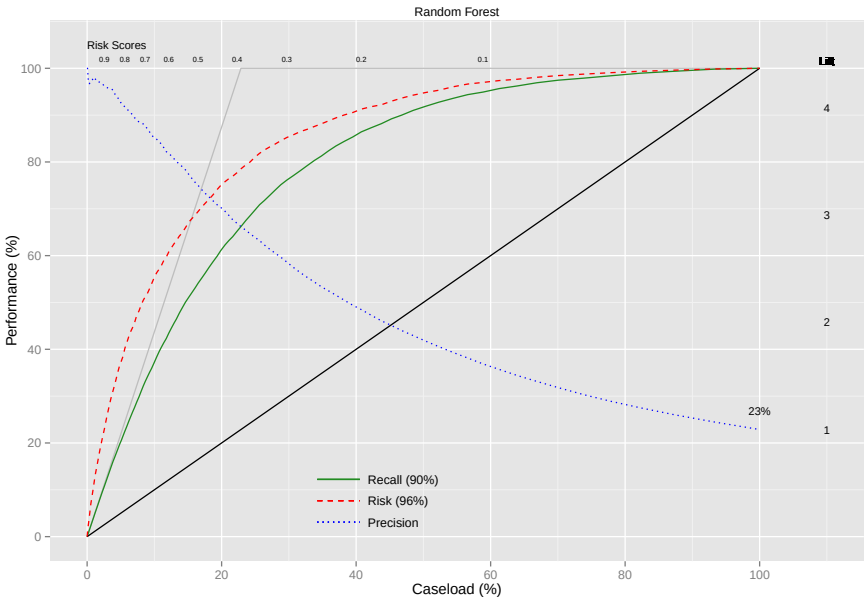
14 Risk Chart—AUC Interpretation

The area under a risk chart curve is calculated relative to a perfect model. The performance of a perfect model is captured as the grey line we see in the risk chart. The grey line will have two segments. The first segment is drawn from the origin to an intercept on the line where y=100. The intercept is at the overall strike rate of the training data. The second segment simply terminates at the top right corner of the plot.

The grey line would be the plot we would obtain for a perfectly accurate model. All and only positive observations are given the higher scores by the model, and then the lower scores are given only to negative observations. If this were the case for a model, then by the time we have processed the strike rate number of highest scored observations, we will have covered all of the positive observations.

In the risk chart below the strike rate is 23%. That is, 23% of the observations (which is of course also 23% of caseload) are positive observations. Thus, the grey line's internal point is at 100% performance after 23% of the caseload—a perfect model.

```
print(riskchart(pr.rf, actual, risks, "Random Forest"))
```



The area under the curve that is reported for a risk chart is then the area under the curve relative to this grey line, rather than relative to the whole chart. For an ROC curve, the perfect model would be one where the first segment of the grey line runs along the y-axis to 100, and with x=0. Note that for an ROC curve, x is the false positive rate, rather than the caseload. Thus the AUC for and ROC is relative to the whole chart (which is the same as being relative to the grey line on the ROC chart since it actually encompasses the whole chart).

15 Risk Chart—Actual Interpretation for Weather

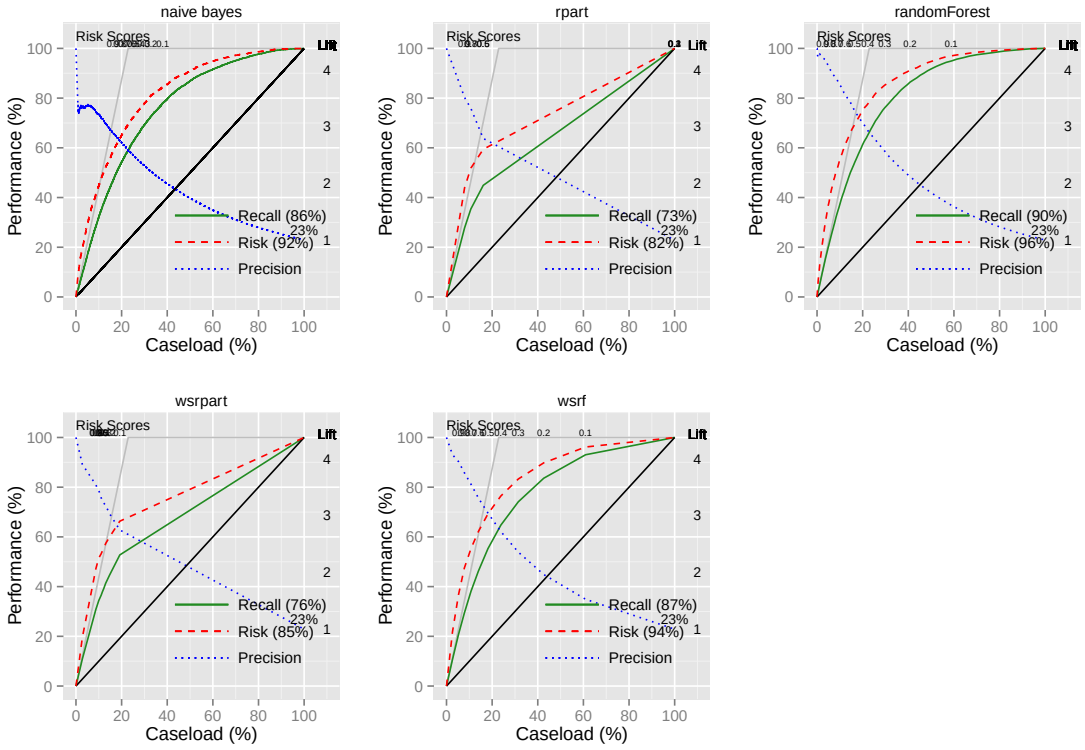
The models we have built are actually based on the weatherAUS dataset.

Exercise: Interpret the model based on the domain of predicting rain_tomorrow.

16 Risk Chart—All Models

Display all Risk Charts as separate plots on a grid.

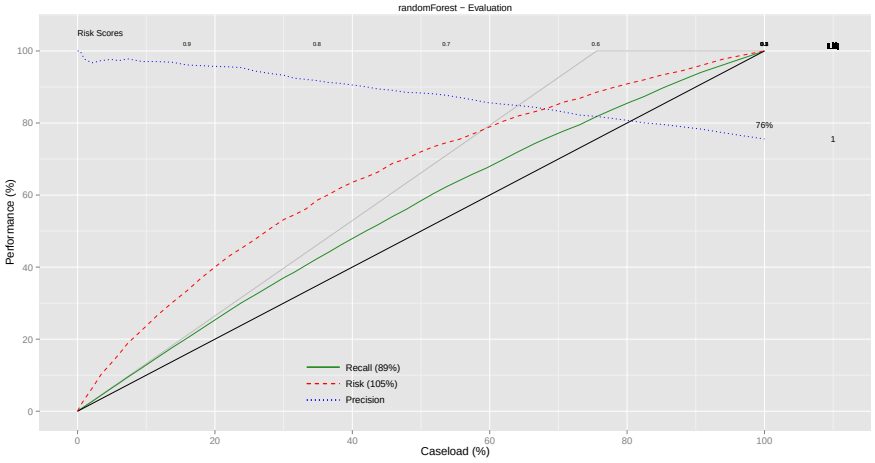
```
library(gridExtra)
rc.nb <- riskchart(pr.nb, actual, risks, "naive bayes")
rc.rp <- riskchart(pr.rp, actual, risks, "rpart")
rc.rf <- riskchart(pr.rf, actual, risks, "randomForest")
rc.wsrp <- riskchart(pr.wsrp, actual, risks, "wsrpart")
rc.wsrf <- riskchart(pr.wsrf, actual, risks, "wsrf")
grid.arrange(rc.nb, rc.rp, rc.rf, rc.wsrp, rc.wsrf, ncol=3)
```



17 Risk Charts—Evaluate Ongoing Model Deployment

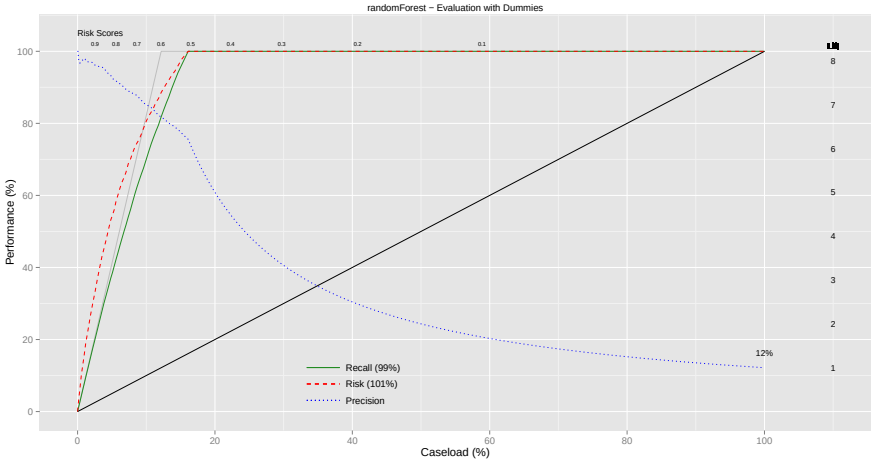
A risk chart evaluates a model based on test data. When deployed the model may be used to filter out observations (with a low risk score) and so they have no outcome recorded (some may be false negatives). A risk chart of just the (high risk) audited cases can be misleading as in this example where only those observations with a risk score greater than 0.5 are selected.

```
pi <- which(pr.rf > 0.5)
riskchart(pr.rf[pi], actual[pi], risks[pi], "randomForest - Evaluation")
```



If we have available all scored observations and treat those not audited as true negatives, we obtain a clearly too optimistic chart.

```
actual0 <- actual; risks0 <- risks; actual0[-pi] <- "No"; risks0[-pi] <- 0
riskchart(pr.rf, actual0, risks0, "randomForest - Evaluation with Dummies")
```



The key is to note the presence of the grey line (the maximal performance line) and the corresponding relative area under the curve calculation. These provide a clearer understanding of the performance of the model on this censored evaluation dataset.

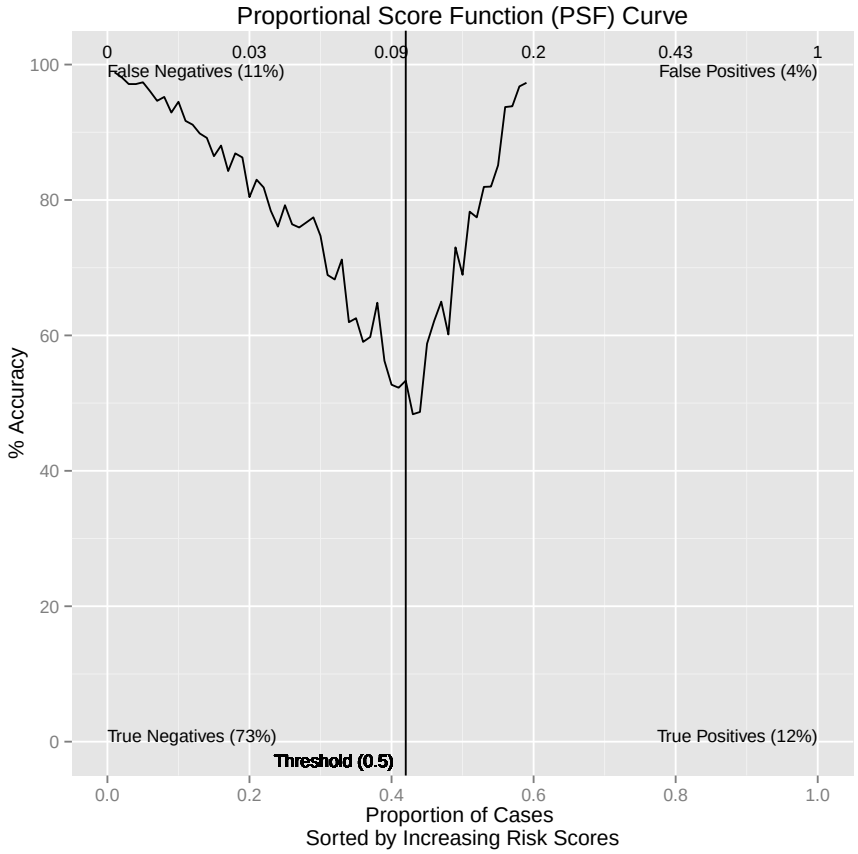
18 PSF Chart

The Proportional Score Function (PSF) chart is a useful tool for evaluating the ongoing performance of a model. We can think of it as a visualisation of a confusion matrix, dividing the plot into 4 regions corresponding to true/false positives/negatives. The idea was developed from the statistical techniques developed by Koesmarno (1996).

We saw previously that a Risk Chart is not appropriate for the visualisation of the ongoing performance of a deployed model. What we see in such a risk chart is the performance on the high risk cases (those selected for action). A PSF Chart is a good alternative for measuring classifier performance.

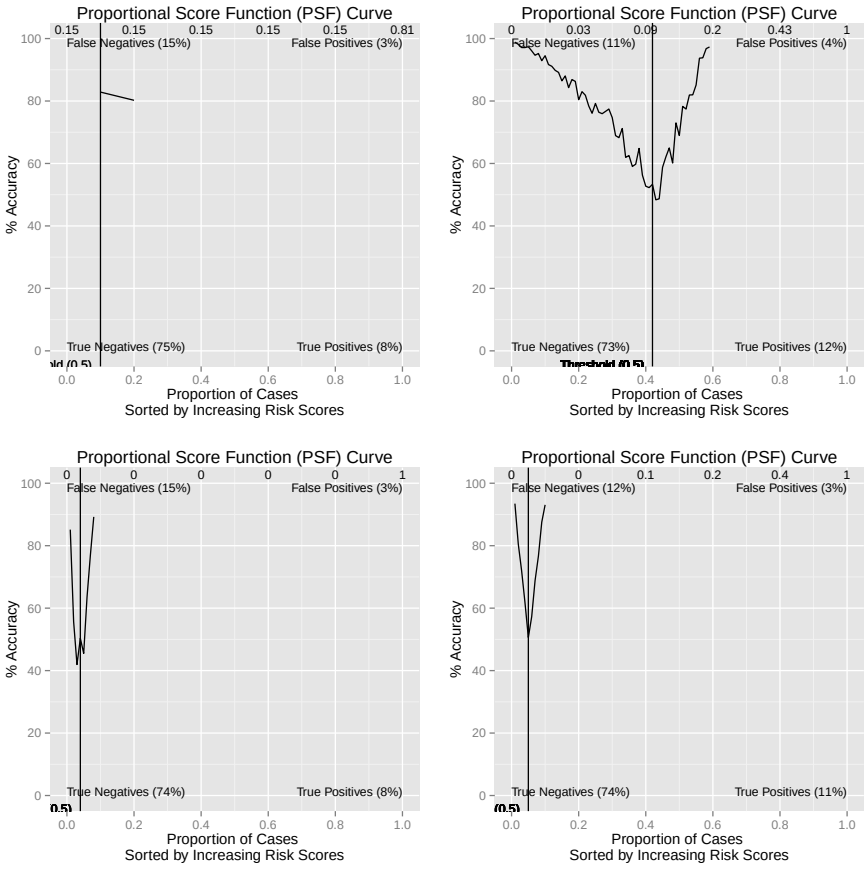
A PSF Chart displays the performance of a model for a chosen cutoff or threshold above which risk scores are regarded to predict the positive class. By default, that the threshold is the traditional 0.5. The curve can provide insights into accuracy and the degradation in the performance of the classifier.

```
print(psfchart(pr.rf, actual))
```



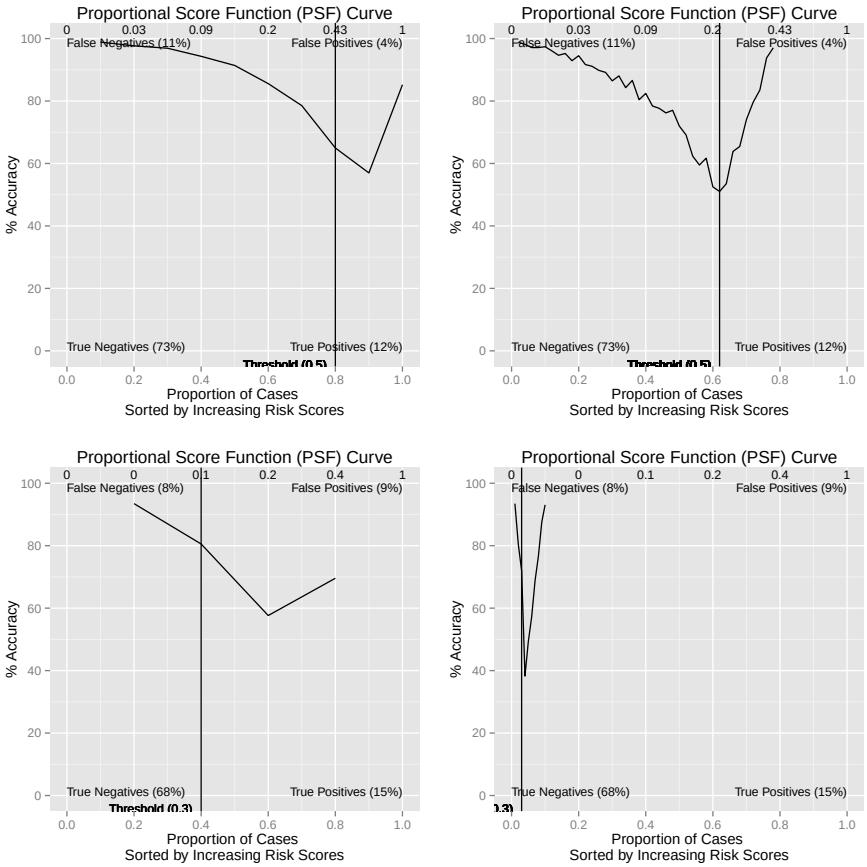
19 PSF Chart—All Models

```
psf.rp   <- psfchart(pr.rp, actual, bins=10)
psf.rf   <- psfchart(pr.rf, actual)
psf.wsrp <- psfchart(pr.wsrp, actual)
psf.wsrf <- psfchart(pr.wsrf, actual)
grid.arrange(psf.rp, psf.rf, psf.wsrp, psf.wsrf)
```

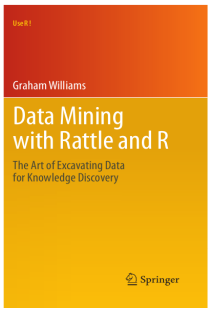


20 PSF Chart—Options

```
psf.rf1   <- psfchart(pr.rf, actual, bins=10)
psf.rf2   <- psfchart(pr.rf, actual, bins=50)
psf.wsrf1 <- psfchart(pr.wsrf, actual, bins=5, threshold=0.3)
psf.wsrf2 <- psfchart(pr.wsrf, actual, bins=100, threshold=0.3)
grid.arrange(psf.rf1, psf.rf2, psf.wsrf1, psf.wsrf2)
```



21 Further Reading



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.

22 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.

Koesmarno HK (1996). "Class-size percentile transformation for reconstructing a distribution function." *Journal of Applied Statistics*, **23**(4), 423–434.

Meng Q, Zhao H, Williams GJ (2014). *wsrf: Weighted Subspace Random Forest.* R package version 1.3.17.

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.

source code and/or documentation contributed by Ben Bolker GRWIR, Lumley T, Johnson RC, are Copyright SAIC-Frederick RCJ, by the Intramural Research Program IF, of the NIH, Institute NC, for Cancer Research under NCI Contract NO1-CO-12400 C (2013). *gmodels: Various R programming tools for model fitting.* R package version 2.15.4.1, URL http://CRAN.R-project.org/package=gmodels.

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/.

Zhalama, Williams GJ (2014). *wsrpart: Build weighted subspace rpart decision trees.* R package version 1.2.151.

This document, sourced from EvaluateO.Rnw revision 484, was processed by KnitR version 1.6 of 2014-05-24 and took 188 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 2014-08-16 11:14:12.