Data Mining Lab 8: More Model Evaluation

1 Introduction

In this lab we are going to look at another two methods of model evaluation which we missed out before. We will start off looking at them in the context of trees and then pull together the two types of analyses: trees and neural networks.

You should be getting a bit more comfortable with `R` by now, so most of the preliminary work is left as exercises. See if you can remember how to do each part before looking up the hints.

If you need to refer to previous labs or to download the data set, they will be on the course labs website:

`http://www.maths.tcd.ie/~louis/DataMining/`

2 Classification Trees

2.1 Getting Setup

Exercise: Download the new German credit rating data from the course website. While you are there, have a look at the file which contains the explanation of what the various alphabetic codes for each variable mean.
Exercise: Load the German credit rating data into a variable called `credit` in your workspace. (Hint: see lab 2, §2.1 if you've forgotten)

```
>
```

Exercise: Load the `rpart` package, which contains the functions to build classification trees in `R`. (Hint: see lab 2, §3.1 if you've forgotten)

```
>
```

Exercise: Have a look at the data to get a feel for it using `summary()`. The variable we are interested in correctly predicting is called `good_bad` and indicates whether the customer is a good or bad credit risk for the bank.

```
>
```

Exercise: Split the data in `credit` into two variables, one called `train` and another called `test`. The split should be $\frac{1}{3}$ for the testing data and $\frac{2}{3}$ for the training data. (Hint: see lab 4, §2.3 if you've forgotten)

```
>
>
>
```

Exercise: Confirm that we did not get a freakish random split of the variable. In other words, check that there are a decent number of good and bad credit risks in both `train` and `test`. (Hint: see lab 4, §2.3 if you've forgotten)

```
>
>
```

Exercise: Fit a classification tree to the data. You should specify the Gini splitting criterion, but don't worry about the complexity parameter. (Hint: see lab 4, §2.4 if you've forgotten). Make sure you look at the tree once it's been fitted.

```
>
>
>
```

2.2 Evaluating the Model

We are now going to evaluate how well the tree has done using a battery of tests, some we've seen before and some new. Take this as an opportunity to consolidate your understanding of each of these and to understand the code so that you could do the same model tests on data you encounter elsewhere.

To do this we need a couple of extra libraries ...

Exercise: Load the `ROCR` and `rattle` libraries. (Hint: see lab 2, §3.1 if you've forgotten)

```
>
>
```

2.2.1 Confusion Matrix

Recall that probably the simplest way to see how a classifier is doing is to look at the confusion matrix:

```
> table(predict(fit, test, type="class"), actual=test$good_bad,
        dnn=c("predicted","actual"))
```

2.2.2 ROC

We have seen the ROC curve before, but we used it as a programming exercise and so it would be good to see how to use the built in `R` functions to do it in just a few lines.

The following code will produce the standard ROC curve for the test data:

```
> pred = predict(fit, test)[,2]
> perf = performance(prediction(pred, test$good_bad), "tpr", "fpr")
> plot(perf)
```

The first line (you should recall!) gets us the probabilities of the second factor (ie probability of Good credit rating here, because alphabetically Good is after Bad).

The second line is the meat of the ROC curve calculation: it calculates the true positive and false positive rates based on the predicted values we just calculated and the true values which we know for the test set. The stand-out point here are the arguments `"tpr"` and `"fpr"` which are short for True Positive Rate and False Positive Rate and specify the values to compute for our y and x axes respectively. The `performance()` function is able to calculate *lots* of metrics to measure classification performance besides these two.

The final line just plots the true positive and false positive rates that were calculated, which constitutes the ROC curve.

Exercise: Look at the manual page for the `performance()` function (ie run `?performance`), with particular attention to the details section where there is a long list of measures it can calculate. What would we specify if we wanted to calculate the lift value instead?

2.2.3 Lift Curve

An alternative you have covered in lectures but which is new in the labs is the lift curve. We can again use the built in `R` functions to get at this very quickly.

Exercise: Replace the `???` below with the correct argument for calculating the lift value.

```
> pred = predict(fit, test)[,2]
> perf = performance(prediction(pred, test$good_bad), "???", "rpp")
> plot(perf)
```

2.2.4 Captured Response Curve

The method here is very slightly different:

```
> pred = predict(fit, test)[,2]
> eval <- evaluateRisk(pred, test$good_bad)
> plotRisk(eval$Caseload, eval$Precision, eval$Recall)
```

You saw plots of this in lectures and this is how they can be produced. However, recall the difference in terminology employed by `R`:

- 'Adjustment' = Cumulative % captured response by % caseload
- 'Strike Rate' = Non-cumulative % response by % caseload
- Black line = Cumulative random % captured response

3 Neural Networks

Now we can compare how neural networks compare with the same set of model evaluation tests.

```
> train$good_bad = class.ind(train$good_bad)
> fit = nnet(good_bad ~ ., data=train, size=1, softmax=TRUE)
```

Exercise: Compute the confusion matrix for the neural network fit. What is wrong?
Exercise: Refit the neural network with decay added. Is the situation any better? Remember you might have to play with the decay parameter to get decent convergence. If you are getting many iterations without the final message `converged`, then you may have to add the additional argument `maxit=1000` which allows the algorithm to run longer.
Exercise: Once you have a sensible neural network fit, plot the ROC curve, lift curve and captured response curves. On the basis of all of this, how does the neural network seem to compare to trees for this data set?