Data Analytics Lab 5: Model Evaluation II

Learning Objectives

- Setting the parameters cost, priors and number of surrogates for building trees
- Examine in greater depth the `ROCR` package
- Generate more specific model evaluation output using performance objects
- Plot a Receiver Operating Characteristic (ROC) curve
- Compare evaluation output of different data mining models
- Learn how to make adjustments to plots in `R`
- Bootstrapping in R

---

1 New data set

For this lab we are going to use a different dataset. It is a data set about mushrooms. We have a data on sample of mushrooms their various characteristics and whether they are edible or poisonous. We are interested in building a model to predict whether a mushroom is poisonous or not based on 22 given attributes. There is a file called mushroom description which provides information on the attributes. The file **mushroomdata** contains the data.

2 Setting Costs, Priors and Number of Surrogates for trees

2.1 Cost Matrix for growing trees

As discussed in class cost/loss matrix is used to weight misclassifications differently. In other words, different problems may mean that a false positive and false negative are not considered as equally bad. For example, quite often in medical screening a false negative is far worse (we miss that the person has a disease) than a false positive (we suspect they have it when they don't and send for further testing). Our classification tree can take this into consideration by weighting how much to penalise each incorrect classification in a given choice of split.

$$L = \begin{pmatrix} 0 & L_{fp} \\ L_{fn} & 0 \end{pmatrix}$$

Note: due to the quirk of `R` ordering factors alphabetically, you need to be careful about the order of false positive and false negative in the above matrix. Here our target variable is coded as 'e' and 'p'. So 'e' is first as 'e' is before 'p' in the alphabet. The first row of the matrix corresponds to actual/predicted for 'e'. In this case which of the misclassification costs would you expect to be higher?

The format of the commands looks like the following:

```
> fit = rpart(mushroom$target ~ ., data=mushroom,
         parms=list(split="Gini", loss =matrix(c(0,C1,C2,0))))

> fit
```

You have to fill in some values for C1 and C2.
The loss matrix must have 0's on the diagonal. So, this is the loss matrix we used:

$$L = \begin{pmatrix} 0 & C1 \\ C2 & 0 \end{pmatrix}$$

You have got to decide which misclassification is worse i.e. to classify an edible mushroom as a poisonous one or a poisonous mushroom as an edible mushroom. Assign some values for C1 and C2 remembering that it is the ratio of the costs that is important.

2.2 Specifying priors

Sometimes the proportions of classes in a data set do not reflect their true proportions in the population. In the notes the prior probabilities are indicated by π_i or p(+) or p(-) in the discussion on the ROC curves. We can inform `R` of the population proportions, and the resulting model will reflect them. It is essentially another way of weighting data. All probabilities will be modified to reflect the prior probabilities of the classes rather than the actual proportions exhibited in the training dataset. Priors must be positive and sum to 1. It is an addition to the `parms=list(....)` parameter and takes the form:

```
> prior=c(0.2,0.8)
```

3 Exploring `ROCR`

We have seen how to produce a ROC curve last week but we are going to delve into the package further this week. The package `ROCR` allows you calculate various performances measures for the x and y axes. Then the next step is to decide on what performance measures you are interested in - there is a long list. The following are the most common:

1. `acc`: Accuracy
2. `tpr`: True Positive Rate
3. `tnr`: True Negative Rate
4. `fpr`: False Positive Rate
5. `err`: Error Rate
6. `rec`: Recall (same as TPR and % response)
7. `prec`: Precision (same as % captured response)
8. `rpp:`: rate of positive predictions
9. `spec`: Specificity (same as TNR)
10. `sens`: Sensitivity (same as TPR)
11. `lift`: Lift

12. `ecost`: Expected Cost

13. `cost`: Cost of a classifier when class-conditional misclassification costs are explicitly given. Accepts the optional parameters cost.fp and cost.fn, by which the costs for false positives and negatives can be adjusted, respectively. By default, both are set to 1.

4 Before you start

Remember to split the data into a test and training set. Why do we do this? I am assuming that you have created two dataframes **test** and **train**.

5 Creating performance objects

You can see from the above list that `ROCR` is very comprehensive and it is essential in the evaluation of data mining models. It is advisable that you read the associated documentation. In particular pay attention to the `ROCR` function called `performance` which we will be using in this section. Remember before we create the performance objects we have to use the prediction command from the package `ROCR`. It will look something like the following:

```
> predics=prediction(predprob[,2],test$target)
```

where the object `predprob` are the predicted values from a model. (I have to admit I sometimes forget this step)

So, how can we use the list of performance measures above to generate evaluation output in `R`. The first step is to select the performance measures we want and create a performance object. The first measure will be plotted on the y-axis and the second on the x-axis. If only one measure is specified the default for the x-axis is the cutoff value. The following code will give us the error rate at different cutoff values:

```
> perf=performance(predics, "err")
> perf
```

The text based output can be difficult to interpret, so ideally we would like to graph the output. This can be done easily by plotting the performance object we have created:

```
> plot(perf)
```

Exercise: Compare the graph to the textual output and make sure you understand the link between the two.

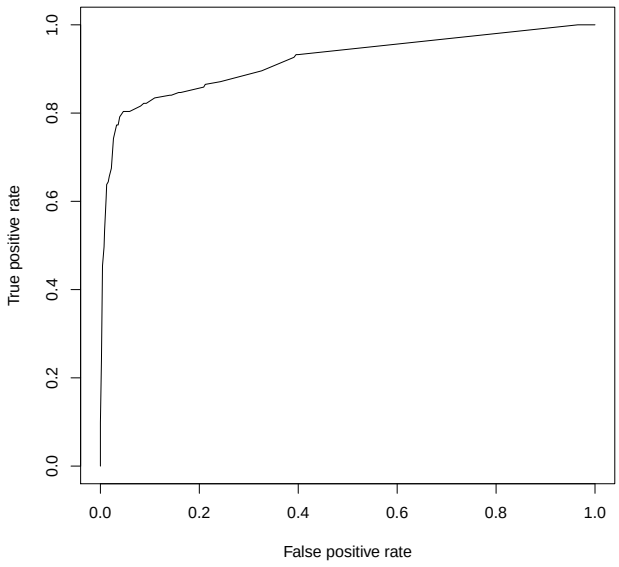
Exercise: Generate output and plot the following using a performance object. What trend do you notice in the graphs?

- False Positive Rate
- False Negative Rate
- True Positive Rate
- True Negative Rate
- Accuracy

We can combine performance measures to extract additional evaluation information about our model. In lectures you will have come across the Receiver Operator Characteristic (ROC) curve. There is no magic command to just generate one of these, but if you notice the components that a ROC curve is made up of (TPR and FPR), you will see that ROCR can store these in performance objects. So with a little adjustment to our code from earlier it is easy to generate the ROC curve. In an ROC curve we want TPR on Y-axis and FPR on x=axis. As mentioned above the default value for the x-axis is the cutoff, so the code below effectively overrides the default.

```
>perf=performance(predics,"tpr","fpr")
>plot(perf)
```

You should get a plot something similar to one on the following page...



To add a diagonal line to your active window simply run `abline(a=0,b=1)`.What does this line represent? For reference the following are some of the main plots which you need to override the defaults for

- ROC curves: takes "tpr" and "fpr".
- Lift charts: takes "lift" and "rpp"
- Expected cost curves - "ecost" and the probability cost measure (see lecture notes). The probability cost measure as defined in class is used for the x-axis.

You can greatly improve the appearance of your graph (adding titles, formatting, using colours) by investigating the documentation for `plot`. You can color each section of the curve by adding colorize=TRUE in the **plot** command. You can also overlay two plots in different colours. This is illustrated later on.

6 Pulling it all together

6.1 Getting value from the output

Really the evaluation output is of limited use when examined on its own. The results need to be contextualised, specifically by comparing them to the evaluation output from other data mining

techniques. As stressed at the beginning of the first lab on model evaluation it is important to recognise that the procedures employed are applicable to any model, not just trees.

We have yet to look at the other data mining techniques prescribed for this course in labs, but for the purposes of illustration we can compare the evaluation output of a linear model to that of the tree model we generated in this lab. Anything can be compared, boxplots, confusion tables, accuracy, error rates, TPR, FNR and so on. In this section the focus will be on comparing ROC curves and overlaying them on the same plot (although this can be done with any plot). Th reason for choosing the ROC curve is that it provides a quick and easy way to visualise the performance of a classification technique.
Exercise: Using the training dataset created above fit a logistic regression and calculate predicted values. To fit a logistic regression use `glm` with `family=binomial`. The command will something like this

```
linreg<-glm(target~.,family=binomial,data=(whatever you want to put in here))
```

Exercise: Calculate the predicted values using a different identifier name such as `predics_lin` so that you can distinguish it from the tree prediction object. Be sure to use your `test` set. This is the `predict` command from package `ROCR`. Before that you have to calculate the predicted values from the model which are used as input to the `predict` command from package `ROCR`. Have a look at what this command generates using the command `str`. Be sure to use different names for each object for each model.

6.2 Demo of ROCR

Have a look at the `demo` contained within `ROCR`. To run just type

```
demo(ROCR)
```

Have a look at `R` code and see can you figure how to draw lines on a `ecost` curve. You may have problems with margins. The following code prints the `R` code in the demo package ROCR.

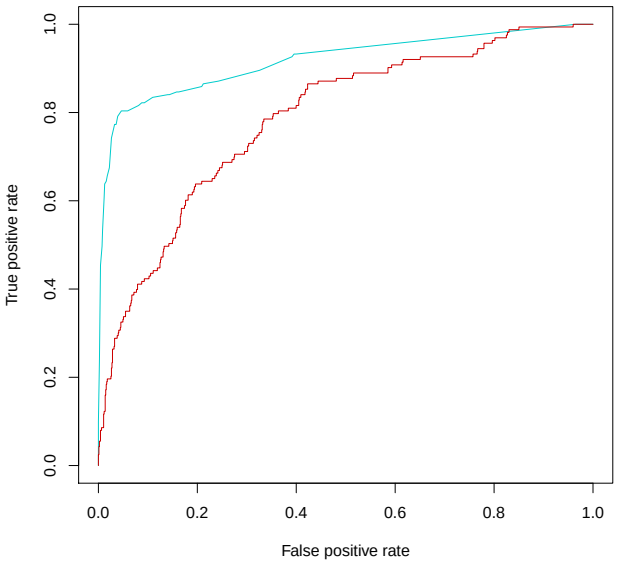
```
print_demo_code <- function(demo, package) {
  demo_file <- system.file("demo", paste0(demo, ".R"), package = package)
  cat(readLines(demo_file), sep = "\n")
}
print_demo_code("ROCR",package="ROCR")
```

6.3 Comparing ROC curves

We will now generate a ROC curve for both models and plot them on the same graph. Once you have completed the exercises above, you can use the following code:

```
> perf_tree = performance(predics,"tpr","fpr")
> plot(perf_tree, col="blue")
> perf_lin = performance(predics_lin, "tpr", "fpr")
> plot(perf_lin, col="red", add = TRUE)
```

You can see in the code that colours for each curve have been specified as well as an additional `add` parameter in the second linear model ROC curve. This is set to `TRUE` and instructs `R` to add the plot to the existing active plot window to create an overlay. You will hopefully get something stunning like this...



You may notice that the linear model (in red) appears to have more lines compared to the tree ROC curve (blue line). This is due to the fact that the linear model performance object specifies more cutpoints. You will see this if you examine the output for the performance objects related to each model.

Exercise: Which model is better? How do you know?

6.4 Improving your plot

The `plot` documentation is an valuable asset when working with `R`. It will be an essential reference point when you are looking to improve the quality of your graphics in `R`

Exercise: Make the following adjustments to the graph by consulting the `plot` documentation

- Add a diagonal line
- Change one of the ROC curves to yellow in colour
- Change the title of the plot. Hint:use `main=Title` in plot command.
- Add a simple legend

6.5 Other Model comparisons

As discussed in class try the following

- Compare predicted probabilities. It is probably better to use the package **ggplot** here. Think how about you might show the values of the **target** variable here.

7 Calculation of margins

The margins for cases give us an idea of the certainty of our estimates. Recall that the margin for a case (when there are only 2 outcomes) is the probability of correctly classifying the case - probability of incorrectly classifying the case. One way to calculate them is to recode the target variable 'e' to -1 'p' to 1. Then calculate the (P(e) - P(p))*Target variable. We can write this as (2*P(e)-1)*Target variable. Hint use the 'ifelse' statement.

8 Bootstrapping

R contains a very good package called **boot** which can be used for bootstrapping. This briefly describes how to use it simply. Again see the pdf file for more documentation. There are some good descriptions on the web as well. The R package **boot** allows a user to generate bootstrap samples easily of virtually any statistic that can be calculated in R. From these samples, you can generate bootstrap confidence intervals, or plots of your bootstrap replicates.

The following gives the basic structure illustrated with a simple example. The **boot** command executes the resampling of your dataset and calculation of your statistic(s) of interest on these samples. Before calling boot, you need to define a function that will return the statistic(s) that you would like to bootstrap. The first argument passed to the function should be your dataset. The second argument can be an index vector of the observations in your dataset to use or a frequency or weight vector that contain the sampling probabilities.

The example below uses the default index vector (called i in this case) and assumes that we wish to use all of our observations and weight them equally. This is the most common way.

We are going to go back to using the **churn** dataset here. The statistic of interest here is the correlation coefficient of two variables `day.calls` and `night.calls` in the churn data set. First the function `corr` is defined

```
corr <- function(d, i){
d2 <- d[i,]
return(cor(d2$day.calls, d2$night.calls))
}
```

With the function `corr` defined, we can use the `boot` command, providing our dataset name (in this case `churndat`), the function, and the number of bootstrap samples to be drawn.

First read the data into an object called `churndata`

```
bootcorr <- boot(churndata, corr, R=500)
```

Essentially what happens is the `boot` command generates a set of random indicies with replacement and these are passed to the function. This is what "i" in the function. The command

```
d2<-d[i,]
```

creates a new dataframe with the rows corresponding to set of cases selected by "boot". Make sure to have a look at what is returned in the object `bootcorr`.

1. Produce a basic summary with

   ```
   bootcorr
   ```

2. You can plot the 500 booted values by

   ```
   plot(bootcorr)
   ```

3. Produce a basic confidence interval with the command `boot.ci`. It can also produce many other type of confidence intervals.

   ```
   boot.ci(bootcorr, type = "basic")
   ```

In the above example only 1 value is returned every time. It is possible to return more than 1 value. Look for examples on web. You should be able to adapt this to calculate the Area under the ROC curve.