Data Mining Lab 3: Tree Detail, Variable Importance and Missing Data

1 Introduction

In this lab we are going to continue looking at the Titanic data set, but try to understand the output a bit better. As such there's less coding to get through in this lab, so don't feel any rush: make sure you understand what's going on.

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 Building the Tree

2.1 Getting Setup

Exercise: Load the Titanic data set into a variable called `data` in your workspace. (Hint: see lab 2, §2.1 if you've forgotten)

```
>
```

Exercise: Attach the variable `data` to your workspace, which will allow us to refer to Survived, Class, Sex and Age directly. (Hint: see lab 2, end of §2.1 or lab 1, §5.2 if you've forgotten)

```
>
```

Exercise: Load the `rpart` package, which contains the tree building functions. (Hint: see lab 2, §3.1 if you've forgotten)

```
>
```

2.2 Constructing the Tree

Once again, we want to build the classification tree. However, now that you have covered the Gini index in lectures, we're going to explicitly request that `R` uses this as the splitting criteria.

```
> fit = rpart(Survived ~ Class + Age + Sex, parms=list(split="gini"))
> fit
```

You should notice that in this instance the tree hasn't changed from the last lab because `R` had used Gini because of the format we presented the data in. However, it will not always do so for different data sets, so the additional argument `parms=list(split="gini")` is good to know to force it.
Exercise: Rerun the last two commands, this time changing the split criterion in the first to `"information"` instead of `"gini"`. Is there a change in the tree?

```
>
>
```

2.3 More Tree Detail

We want to continue to dig deeper into understanding the tree than in the last lab.

Before the detail, we can get a nice summary of the terminal node rules with the following:

```
> library(rattle)
> asRules(fit)
```

This tells us for each terminal node that the prediction for survival is (`yval=`), how many of the observations we fitted which fall into this category (`cover=`) and the corresponding probability of a 'yes' for survival (`prob=`).

Now, we want to see the full detailed guts of the fitted tree:

```
> summary(fit)
```

Exercise: It is really important to understand some of the key portions of this output, so spend 5-10 minutes going over it and noting the following:

CP	=	complexity parameter as calculated in class but divided by R(0), the misclassification for root node. You should know how to reproduce this from the lectures. The default stopping value for cp = .01. We will see later what happens with more complicated data.
nsplit	=	number of terminal nodes-1 at that point.
rel error	=	relative error or misclassification for tree at that stage — to convert to absolute error multiply by root node error.
xerror/xstd	=	refer to the results of a cross classification procedure Ideally we want to pick the tree with the lowest xerror $\pm$ 1 SD. Again these are relative errors: to convert multiply by root node error.
Impurity	=	decrease in impurity$\times n$. This is why it is such a big number.
CLASS	=	RRLL means that the first two classes go 'R'ight and the second two go 'L'eft — this depends on order of categories. You can find this out with `labels()` function, or just use the fact that `R` always orders categories alphabetically.

You also get a description of the competing splits and the surrogates. For each surrogate the agreement is printed (I have to admit I have no idea what the adj means).

2.4 Numerical or Categorical?

You might notice that up to now all the Titanic data have been treated as strictly categorical even though, certainly for some of the variables, there is a natural order (eg for class 1st>2nd>3rd>crew)

Let's rerun the analysis, this time treating everything (appropriately or not) as numerical. First off, create a new data frame with the changes we want and then have a look at it (note that the first two lines are in fact all one line in `R`):

```
> data2 = data.frame(Survived = Survived, Class = as.double(Class),
                         Age=as.double(Age), Sex = as.double(Sex))
> data2
```

Note: we were lucky here that `R` *orders categorical variables alphabetically, so 1st, 2nd, 3rd, Crew is in that order automatically when we convert to a number with* `as.double()`*. Otherwise it would have been more involved.*

Now, here's a slightly different way of running the rpart command. We do the following (inserting the `data2` into the rpart command) so that we *don't* have to attach the data to the workspace, which would overwrite the existing Survived, Class, Sex and Age information we attached at the start of the lab.

```
> fit2 = rpart(Survived ~ Class + Age + Sex, data2, parms=list(split="gini"))
> fit2
```

Exercise: Compare the two trees you now have (`fit` and `fit2`). Is there a difference in how the tree has been constructed when treated as numerical data?

3 Variable Importance

A technique for how to identify which variable is the most important one for successful tree classification was covered in lectures and we see that implemented here. The function to do this is not part of the standard `R` distribution, so you can add it by running the following:

```
> source("http://www.maths.tcd.ie/~louis/DataMining/importance.R")
```

Now, we will use the tree we just constructed and test which variable is most important to classification.

```
> i = importance(fit2)
> i
```

This output gives an indication of the importance of each variable in the data. So, you can see that Sex is, by a factor of over 4 times, considered the most important variable here.

4 Missing Data

The Titanic data set is very nicely behaved in so far as having no missing data at all. However, many real world data sets you are likely to encounter will have missing data.

We are going to intentionally 'mess up' the Titanic data to see what happens. Since we saw that sex is the most important variable we'll be particularly mean and remove some of the values! The following will force about 10% of the sex values to be absent and then show you the new Sex data:

```
> Sex[seq(1,2201,10)] = NA
> Sex
```

As covered in lectures, trees handle 'missingness' by using other non-missing variable values as surrogates to predict the absent value. In `R` we are able to specify how many other variables can be used as surrogates as follows (note that the following is all on one line in `R`):

```
> fit3 = rpart(Survived ~ Class + Age + Sex,
              control=rpart.control(maxsurrogates=1),
              parms=list(split="gini"))
```

Exercise: Has the tree changed much with the missing values?