Data Mining Lab 2: A Basic Tree Classifier

1 Introduction

In this lab we are going to look at the Titanic data set, which provides information on the fate of passengers on the maiden voyage of the ocean liner 'Titanic'. The data is in the folder for this course. Download the file. There are 4 variables `Class Sex, Age and Survived`. We are interested in predicting who survived. Remember to work from a script file. See under `File`.

2 Explore data

You should start by exploring the data e.g. frequencies of each variable separately and some tables. A good alternative when faced with lots of categorical information is to use `table` which puts together a full set of summary tables. Try this on the full data set:

```
> table(data)
```

You should see four tables. Each table is headed by a combination of age (adult/child) and survived (yes/no) . The tables are further subdivided into counts by class (1st/2nd/3rd/crew) and sex (male/female). This is far more informative than looking at the 2201 observations directly, although perhaps still a little overwhelming. We can look at just pairs of our choice as follows:

```
> table(Age, Survived)
```

Exercise: Which variables are related to Survived?

2.1 One way to recode a variable

At the moment the variable `Survived` is coded as Yes, No. You may wish to recode it to 1 and 0 where 1 indicates the event of interest. There is a `recode` command in the library `car`. Explore it - the documentation is not great. The format of the `recode` command is:

```
>recode(variable name, 'character string')
```

where the character string consists of range/pairs of values with = signs separating ranges and values, and ; separating each range/value pair.
There are four possibilities for range/value pairs;

1. single values, e.g. 1='Yes'
2. multiple values, e.g. c(1,5)= 5

3. range of values , e.g. 1:5 ='High'. The value `lo` and `hi` can be used to indicate lowest or highest value.

4. the word `else` to represent other values not covered by the other range/pair values

An example of `recode` is

```
>newgroup=recode(group,'c(1,5)=1;c(2,4)=2;else=3')
```

This recodes a variable called `group`: values 1 and 5 to 1 , 2 and 4 to 2 and other values to 3 into a variable called `newgroup` . As a matter of practice you should always tabulate the recoded variable in this case `newgroup` just to make sure that the recoding went to plan. It is very easy to make mistakes and a little paranoia is good!!

Example of a character string variable

```
rchurn=recode(churn,"'Yes'=0;'No'=1")
```

Note the use of the two different quotes.

2.2 Graphical Output

A graphical output is also useful for conveying information quickly and naturally it is often easier to quickly pick out anything unusual than from the numbers. The following produces a bar plot:

```
> barplot(table(Survived, Class), beside=TRUE, legend=levels(Survived))
```

Exercise: Modify the `barplot()` command to also look at Age and then Sex subdivided by Survived.

In terms of graphics, `R`'s built in capabilities are slightly more limited with respect to categorical data. However, there are add-ons which give nice results. If you are on your home computer where it's easier to download packages you might want to explore the `vcd` package.

3 Building a Classification Tree

3.1 Fitting a Tree

We are now ready to see how well we are able to fit a classification tree in order to predict survival based on passenger class, age and sex. Load the `rpart` package. Dive straight in and run the following:

```
> fit = rpart(Survived ~ Class + Age + Sex)
```

This command is asking `R` to fit a tree where Survived is to be predicted from Class, Age and Sex and then store that tree for later reference in the variable `fit`. You can look at the tree that has been built by just looking at the `fit` variable:

```
> fit
```

This will bring up an initially intimidating looking mountain of text. However, be sure to take a good 5-10 minutes and study it carefully – it should start to become clear. We will cover this output in detail in class so make sure and bring a copy of the output with you.

The `1)` that you see in your output is the root node where you start when trying to classify a new observation. You then choose between the indented `2)` or `3)` lines – one is when sex is male, the other female. Were this a female observation you would then proceed down the tree in the same fashion until you reach a terminal node of the tree, signified by a `*` at the end of the line. The Yes/No before the bracketed numbers then indicates the prediction the tree has returned based on the information you provided.

Do not proceed with the lab until you're happy that this text output makes some sense: at this juncture the detail of the numbers is not so important, just make sure you can see how the classifier is working.

So fitting a tree is remarkably easy in `R`, but: i) the text output is not great; and ii) we don't want to have to manually predict the survival if we have new data.

3.2 Variables Created

Lots of commands create new variables. To see these type

```
>attributes(fit)
```

To have a look at these type `fit$name` where name is what you want to look at e.g.

```
fit$parm
```

We will explain some of this output later. But it is very useful to explore these every time you run a command.

3.3 Graphical Output

This is one of the weaknesses of R with regard to trees. There is a package called Rattle which has a command called DrawTreeNodes which you can investigate. Try the following and see what you think. `fit` is the name of the output from the tree from above

```
>plot(fit, compress=TRUE,uniform=TRUE)
>text(fit,use.n=T,all=T,cex=.7,pretty=0,xpd=TRUE)
```

If you load the library `rattle` you can try the following

```
>drawTreeNodes(fit,cex=.8,pch=11,size=4*.8, col=NULL,nodeinfo=TRUE,
>units="",cases="obs",digits=getOption("digits"),decimals=2,print.levels=TRUE,
>new=TRUE)
```

If anybody finds anything better please, please let me know.

3.4 Automated Prediction

Prediction is in fact relatively easy and most of the work lies in preparing the new data we want to predict.. First, we must have our query data in a new data frame, with the appropriate labels (Class, Age, Sex) to match the labels of the original data on the input/independent variables (i.e no Survived – that's what we want to predict!)

```
> newdata = data.frame(Class=c("2nd"), Age=c("Child"), Sex=c("Male"))
> newdata
```

Then, we call the aptly named `predict()` function and provide it the tree to use (ie the one we stored in `fit`) and the data to predict (ie `newdata`)

```
> predict(fit, newdata)
```

You will see it returns a No/Yes matrix. The number under each represents the probability it assigns to that label being true for the new observation (so here you should see it predicts Yes with apparent certainty).

You should have noticed above that in the data frame command we gave a `c()` to each variable, in that example only length one *and* textual (ie, with `" "` round it). Thus, we can in fact pass in multiple new observations by adding more quoted (`" "`) elements to the vector and predict them all at once.

Exercise: Determine the predictions for 1st class/child/female, 2nd class/adult/male and crew/adult/male all at once by setting up a data frame with multiple observations and passing it to `predict()` along with our fitted tree model

3.5 How Did The Tree Do? (*harder, see how you do*)

A simple way to answer the question of how accurate the tree was is to simply run `predict()` on the original data and compare it to the truth. This is not quite statistically sound in-so-far as you're always going to do as well as possible on the data you fitted from and one should really test on data the fitting algorithm never saw. However, it is sufficient for a quick assessment – after all if it *can't* predict the data which trained it, then it really is doing badly! We will see later on how to get around this problem.

This bit of `R` code is a bit harder than the rest to date and has some new commands, but go through it slowly and try to understand what is happening.

```
> newdata = subset(data, Survived=="No")
> noPredictions = predict(fit, newdata)
> noPredictions
> correct = (noPredictions[,1] > 0.5)
> correct
> table(correct)
```

The first line picks out only those observations who did not survive and stores them in the `newdata` variable (examine `?subset`). Then, on the second line `predict()` is using the tree in `fit` to predict only those "No"'s – so we should get all "No" predictions if it is completely accurate.

On the fourth line, `noPredictions[,1]` picks out only the probabilities of answer "No" (the first column) and then tests if the prediction is over 0.5 (so no is more strongly predicted than yes). As you see, this produces a vector of true/false values – true indicates the prediction for that observation number was correct since the probability of "No" was over 0.5, while false indicates it predicted wrongly.

Finally `table()` arranges the counts of the true/false (ie correct/wrong) result. You should see 'TRUE' has 1470 and 'FALSE' has 20. So there were 1490 who didn't survive and of those the tree only failed to correctly predict that for 20 of them (1.3%).
Exercise (hardish): Repeat this for those who did survive and see what percentage of wrong predictions there were (you'll see it's rather high with this simple first attempt at a tree: 62%). We will look at an easier way to do this in another lab.

4 Comparison with logistic regression

For those who have enough time and want a challenge trying running a logistic regression and comparing the predictions to the tree result above. Hint: Use the function `lm`