Data Mining Lab 2: A Basic Tree Classifier

1 Introduction

In this lab we are going to look at how to load a data set from outside R and then build a basic tree classifier for it.

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 The Data

2.1 Loading Data From Outside R

For the lab today we will be using the Titanic data set, which provides information on the fate of passengers on the maiden voyage of the ocean liner 'Titanic'.

Exercise: Download the Titanic survivor data from the course labs website and save it on the lab computer.

This is a so-called comma delimited file. R already has built-in functions which can easily read and understand such file formats. This is done by the following command:

> *First, note:*
>
> *(i) a backslash is a special character in R, so it is best to use / instead when specifying the path to a file.*
>
> *(ii) obviously you need to replace the ... below with the directory you saved the file in. If you do not know the path of where you saved the file, simply right-click on it and choose Properties which will display a dialogue with a 'Location' field showing the path – copy and paste it into R and change the backslashes.*

```
> data = read.csv("C:/.../Titanic.csv")
> names(data)
> data
```

The above two commands will (i) read the file into a variable (a data frame to be precise) named `data` and then (ii) prints out the names of the variables that were imported, then finally (iii) prints the contents of the data frame to the screen.

Exercise: To make life easier later in the lab, attach the `data` variable to the current workspace, so that Class, Age, Sex and Survived will be accessible directly by name (refer back to lab 1, §5.2 if you can't remember how).

```
>
```

2.2 Exploring the Data

With any new data set it is a good idea to first explore it and try to get a feel for it. One of the commands which carries over from lab 1 (§5.1) is the function `summary()`.
Exercise: Use `summary` to get a quick overview of what you loaded into the variable `data`.

```
>
```

Because this is categorical data, many of the other summary statistics are not so relevant. 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) and the tables themselves further subdivide 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: Look at Class and then Sex against Survived too. Do you see even from just these single variables alone what comparisons our tree fitting later might identify as useful?

```
>
>
```

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 Loading Packages

Packages are bundles of `R` code which provide additional functionality on top of the basic abilities which come built-in. Loading a package is as simple as running the `library` function and passing the name of the package you want to load (eg `library(mypackage)`).
Exercise: Load the `rpart` package.

```
>
```

3.2 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. 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.

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.

3.2.1 Graphical Output

Again a graphical output is nicer if possible. The labs are not yet setup with Rattle (next week hopefully!), so we can't do the nicest possible plots just yet, but we can do ok interim ones:

```
> post(fit, file="")
```

Now it is more intuitive to follow the tree structure, selecting the next node by the arc labels. The elliptical nodes are not terminal and show the best prediction available at that point, whilst rectangular nodes are terminal and show the final prediction.

Exercise: If you were asked to predict whether a child who was travelling in second class and was male survived, what would the tree say?

Answer:

3.2.2 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 *non-class* variables (ie 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 (ie we gave a vector (lab 1 §3.1.1), 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

```
>
>
```

3.3 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!

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%).

```
>
>
>
>
>
>
```