Data Mining Lab 3: More trees

Learning Objectives

- Investigate some tuning parameters available for trees
- Examine performance of tree - just a little taster
- How to deal with missing data
- Using terminal node information as input to the logistic regression
- How to build a regression tree

In this lab we are going to look the data set relating to the so-called 'churn' of customers of a telephone company. There is a description of the data on Blackboard. We are interested in whether we can predict which customers are likely to leave (churn) to another company - this could be important to business decisions such as targeting special offers at these customers in an effort to retain them. Since we built a model step by step in the last lab, the focus this week will be on you constructing your own trees. Hence the following...

Exercise: Load the telecom churn data into a dataframe called `churndata` in your workspace. You are welcome to use another name but the notes are written assuming you have called your dataframe **churndata**. The dataset is called **churn.csv** and is stored on Blackboard. Start by getting a summary of the data.

1 Use of the variable `area`

Be careful there is a variable called `area` in the dataset which takes on 3 values 408,415 and 510. Without doing anything this will be treated as a quantitative variable. It should be converted to a categorical variable using the `factor` command. Change to see that it worked by using the `summary` command. How do you know?

Exercise: Load the `rpart` package, which contains the tree building functions.

Important: At this point we would usually divide data up the data into training and test sets. We won't cover that in this lab, but note that you will be required to do it in the future

Exercise: What are the best predictors of **churn** ?

2 Constructing the Tree

We now want to build the classification tree. However, we are not going to leave `R` to choose the complexity parameter this time.

```
churndata$area<- as.factor(churndata$area)
> fit <- rpart(churndata$churn ~ ., data=churndata[,2:18],
         parms=list(split="Gini"), cp=0.00001)
> summary(fit)
```

We use the c(2:18) to select out variables we want to include in model as `churn~.` will use all the variables. Why do you think we have left out the first variable? The **','** in [,2:18] states that you want to select all the rows. You will see that after you run the `summary` command that you get a huge piece of text for the output! If you scroll up to the top you'll see the table listing the complexity parameter along with the various splits and errors. Notice how the xerror column now decreases and then starts to *increase*. Why is the **improvement value** the same for certain splits?

Exercise: What is the best value for the complexity parameter on the basis of this table? Refer back to the previous lab if you need more information about the table, the complexity parameter or graphing the complexity parameter against the relative error. Hint:try using command `plotcp(fit)`. To just print the **cp** table type **fit$cptable**. See the code in the previous lab.

Exercise: Refit the tree, this time using the best value for the complexity parameter (as an input parameter). Store it in the variable `fit`. Examine the output, paying particular intention to the complexity parameter. Try changing the split criterion in the tree to `"Information"` instead of `"Gini"`. Is there a change in the tree?

Exercise Check the parameters of a tree using `fit$parms` and `fit$control`

3 Predicting probabilities

Use the function `predict` to calculate predicted probabilities and store them in an object as follows:

```
predvalues<-predict(fit)
```

We can use the command **str(predvalues)** or **attributes(predvalues)** to see what the object **predvalue** contains. The object contains two columns - 1st predicted probabilities for 'No' category and the second the predicted values for 'Yes' category. The reason that the 'No' predicted are in the first column is that N comes before Y in the alphabet. To print the first ten rows use

```
predvalues[1:10,]
```

Try to see how well the tree did by using graphs, tables or summary statistics. We will be covering this a lot in the next few weeks.

4 Describing the terminal nodes

There is a package called **rattle** which is worth investigating. It has a lovely command for describing the rules for the terminal nodes. After you run **rpart** and assign the result to an object called **fit** in this example or whatever name you chose. Just type the following:

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

An example of the output is

```
 Rule number: 15 [churndata$churn=Yes cover=101 (3%) prob=0.95]
   day.mins>=264.4
   vmail=no
   eve.mins>=187.8

 Rule number: 83 [churndata$churn=Yes cover=16 (0%) prob=0.87]
   day.mins< 264.4
   svc.calls>=3.5
   day.mins>=160.2
   eve.mins>=141.8
   day.mins< 175.8
   eve.mins< 212.2

 Rule number: 11 [churndata$churn=Yes cover=102 (3%) prob=0.87]
   day.mins< 264.4
   svc.calls>=3.5
   day.mins< 160.2
```

The rules are joined by the AND function. See can you understand the output.

5 Missing Data

The Churn 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. I have created another dataset called **churnmiss**. Again you can find it on Blackboard. First of all check the pattern of missing data with the package **VIM**. Hint: just use the **aggr** function from **VIM**. Then create the tree again and examine output carefully.
As covered in lectures, trees handle missing data 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`): We can set the number of surrogates we wish to look for as follows: (Suggest you try three and see what happens)

```
> fit_missing <- rpart(churn ~ ., data=churnmiss[,c(2:18)], parms=list(split="gini"),
              control=rpart.control(cp=0.0001,maxsurrogates=3))
```

I am assuming that the data from the file **churnmiss** was read into a data frame called **churnmiss**. **Exercise:** Has the tree changed much with the missing values? Explain the surrogate results. Do you see any problems here? Any suggested solutions?

For more information on surrogates in `rpart` check the documentation and also this guide at http://www.mayo.edu/hsr/techrpt/61.pdf by Therneau and Atkinson. Not only does the guide have some information on surrogates, but it provides a detailed overview of `rpart` using cancer data.

6 Using the terminal nodes as input to a Logistic regression

Exercise: For those who have enough time and want a challenge, try to run a logistic regression using the terminal nodes as an input variable. You may want to use a small tree here - e.g. a few terminal nodes. Use `cp` to control the size of the tree. It will be easier to understand the output.

The package `rpart` generates the number of the terminal node as part of the output in a variable called fit$where. I am assuming that `fit` is the name of the object containing the output from `rpart`. Type the following code.

.

```
churndata$tnodes<-fit$where
table(churndata$tnodes)
lreg<-glm(churn~factor(tnodes),data=churndata,family=binomial)
summary(lreg)
```

We first add the variable containing the terminal node information to the dataset and tabulate the frequencies. Make sure you understand what the table is telling you. We then carry out a logistic regression with the command `glm` where `churn` is the response variable where the first level (No) denotes failure or 0 and Yes is success or 1 and `tnodes` is the explanatory variable. Why do we use factor(tnodes) instead of just tnodes? See if you can determine the significance of using the node number as an independent variable. Now you can also add other variables here.

A good resource for comparing trees (CART) and logistic regression can be downloaded the web at www.forecastingsolutions.com/downloads/logistic_and_cart.ppt Much of the information may be difficult to comprehend at this stage, but as you get more experience in building tree models it will become clearer.

7 Regression Trees

Up to now we have used trees with a binary outcome variables. We will now use the package **rpart** to grow a regression tree. We will use the **Smallames** data set to predict the Sale Price of a house. First of all we need to create new variables:

- Total area from variables X1st.Flr.SF and X2nd.Flr.SF
- Convert NA in the Garage.Type to "No garage"

The following R code is one way to do the above. The data has been read into a dataframe called Ames. The **forcats** is used to convert the NA to "No garage". We have created a new sales price to make output easier to read.

```
Ames$TotalArea<-Ames$X1st.Flr.SF+Ames$X2nd.Flr.SF
library(forcats)
Ames$Garage.Type<-fct_explicit_na(Ames$Garage.Type,"No Garage")
Ames$SalePrice1<-Ames$SalePrice/1000
```

Now create a regression tree using the following variables

- Lot.Area
- Bldg.Type
- Overall.Cond
- TotalArea
- Garage.Type

Remember you will have to specify these variables in the formula explicitly or create a data frame with just these variables. All of the same theory applies to selecting the tree as before. Do the following:

- Select the best tree
- Generate predictions from the tree
- Compare predicted values to observed values