Data Mining Lab 1: Examining data

1 Introduction

In this lab we are going to look at how to load a data set from outside R and produce some basic statistics.

2 The Data

2.1 Loading Data From Outside R

For the lab today we will be using the Churn data set, which provides information about a group of customers of a telephone company. We are interested in whether we can predict who will churn based on the available information. **Exercise:** Download the SmallChurn dataset from the folder ST4003 and save it on the lab computer. It is also available at

http://www.maths.tcd.ie/∼louis/DataMining/

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:/.../smallchurn.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 variables will be accessible directly by name. This can be done by

```
> attach(data)
```

2.2 Coding of the Dependent variable

In this course we are only dealing with variables with 2 outcomes e.g success/failure, yes/no or alive/dead. In using R it is very important to code this variable or at least understand how R treats the data. It is advisable to code the data to a 0/1 variable where the code of 1 indicates event of interest. This may vary from problem to problem. In R terms these are referred to as positive events in some packages with 0 as negative events. If you do not recode the variable R will order the coding and treat the first category (as defined by the ordering e.g. N comes before Y) as the negative event. There is a function called `recode` in the `car` package which you might find useful.

2.3 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. Start with drawing a histogram of appropriate variables.

```
> hist(day.charge)
```

There are many fabulous facilities in R for graphing data e.g. the lattice package which you may want to explore. We will stick to the simpler graphs here. Categorical data can be explored by using the `table` command

```
> table(churn)
```

Now we would like to examine the relationship between each variable and the **churn** variable looking for differences between the group who churned and those who did not churn. For quantitative data we should start with a boxplot of each variable against **churn** variable.

```
> boxplot(day.charge~churn)
```

We should also generate summary statistics for each group. There are many ways to do this in R. One way is to use the `R tapply` function.

```
> tapply(day.mins,churn,mean)
```

The first argument is the variable of interest, the second the grouping variable and the third the function to be applied to each group. You can only specify one function to be applied but there is nice neat function called `each` in a library called `plyr` which allows you concatenate a number of functions. So load library `plyr` (there may be problems doing this in the lab) and then type

```
> tapply(day.mins,churn,each(mean,sd))
```

We should also examine the relationships between quantitative variables. Hint: use **plot** . There is also a scatter-plot matrix command `splom` available in the library `lattice`. remember you have to load the library first with the library command `library(lattice)`

```
> splom(data[c(x,y,z])
```

where x, y and z are the number of the variables to plot. It is probably best not to use too many variables at a time.

Do you notice anything in the graphs? Why?

2.4 How to compare the two and calculate a t-test?

It is very simple to do a t-test. We may wish to see if there is a significant difference between the average of day minutes in the population in the two groups (churn vs no churn) . Just type the following and make sure you understand the output. Remember also that with large numbers anything will be significant - use the confidence interval to get a feel for the size of the difference.

```
> t.test(day.mins ~ churn)
```

2.5 How to handle categorical data

For categorical variables we can produce tables. In the churn data set we may want to look at **area** by textbfchurn. We use the table command **table** to do this.

```
> table(churn,vmail)
```

This gives the numbers in each cell and can be difficult to read. To get the row proportions or column proportions use the following:

```
> newt=table(churn,vmail)
> prop.table(newt,1)
```

This gives the row proportions. For column proportions change the 1 to a 2.
More elegantly this can be achieved in one command as follows:

```
> prop.table(table(churn,vmail),1)
```

You can also produce a graph of the above using the `barplot` command

```
> barplot(table(churn,vmail),beside=TRUE,legend=levels(churn))
```

Remember that the `barplot` command takes a table as input. Have a look at the help file for `barplot` command to label output. Finally to calculate a chisquare test type

```
> chisq.test(table(churn,vmail))
```

The package `vcd` includes some helpful methods for displaying tables in many different ways. Have a look at it.

2.6 Missing data in R

In `R` missing values are represented by the symbol `NA` (not available) . Impossible values (e.g., dividing by zero) are represented by the symbol `NaN` (not a number).
Here are a few useful commands for working with missing data

```
> is.na(x) # returns TRUE of x is missing

# recode 99 to missing for variable v1
# select rows where v1 is 99 and recode column v1
> mydata[mydata$v1==99,"v1"] <- NA
```

The function complete.cases() returns a logical vector indicating which cases are complete.
Some algorithms in R don't support missing (NA) values. If you have a data.frame with missing values and quickly want the ROWS with any missing data to be removed then try:

```
> myData[rowSums(is.na(myData))==0, ]
```

2.7 Detecting patterns of missing data

The package `VIM` is useful for looking at patterns of missing data. Load the package. It has a GUI interface which loads. We are only going to use one or two commands so it is easier to close the GUI interface. Datasets come with this package which have missing values. We are going to use one called `tao` which records a small subsample of the Tropical Atmosphere Ocean (TAO) project data. To examine the patterns of missing data type

```
>aggr(tao)
>oaggr=aggr(tao)
>summary(oaggr)
```