Data Mining Lab 1: R Refresher

1 Introduction

You have encountered R on previous courses, such as the core course Multivariate Linear Analysis last year. Even so, as the first lab after the summer break most of this is a refresher of the basic parts of R.

If you have a laptop or home computer it would probably be wise to install R so that you can practise and also work on the assignment from home. There is a video tutorial on setting up R and Rattle at home available at:

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

The MSISS labs should hopefully soon have everything setup for use with this course.

2 Manuals and Help Commands

Useful on-line material is available:

- http://cran.r-project.org/doc/manuals/R-intro.pdf
  (Introduction to R).
- http://cran.r-project.org/doc/contrib/Short-refcard.pdf
  (R reference card).
- http://www.rseek.org/
  (Excellent Google powered search engine of the most useful R resources on the web).

In order to use R's own internal help function use either the `?` command or the `help.search` function. Enter the following into R and observe the result (note that you do not need to type the `>` character and that once you have typed a line you will need to hit the return key before R performs your request):

```
> ?exp
> help.search("exponential")
```

The first of these commands brings up the Logarithms and Exponentials help file, and explains how to use the `exp` function. In general, given an R command or function `x`, entering `?x` will bring up its help file.

The second command provides a list of help files in which the term 'exponential' can be found in the concept or title. This can be very useful when searching for the appropriate R command for performing a given task.

3 Basic R Syntax

3.1 Variables

A ‘variable’ is a name that we can give to data, making it easier to refer to in our work. The first thing to remember is that R is case-sensitive, so foo, Foo and FOO are all considered different things.

You can assign data to a variable by using the = operator. Typing a variable name by itself will retrieve the value stored.

```
> x = 3
> X
Error: object 'X' not found
> x
[1] 3
```

So, > indicates a line we input and [1] is indicating R output. The error after the second input line shows R’s case sensitivity.

Anything typed on a line with no = will simply perform the calculation and output the result. Continuing from above, try the following and look at the output:

```
> y = 2
> x + y
> x * y
> z = 2*x + exp(y)
> z
```

3.1.1 Vectors

We can create a vector of values such as $(2, 4, 5, 1)$ by using the function c().

```
> x = c(2, 4, 5, 1)
> x
[1] 2 4 5 1
```

We can also perform the usual operations +, -, *, / on them.

Exercise: Create two vectors, $(1.4, 5, -3)$ and $(-1.4, 2, 9)$ storing them in variables named x and y respectively.

```
>
>
```

Exercise: Add the vectors together, storing the result in a variable named z and check that z then contains $(0, 7, 6)$

```
>
>
```

3.1.2 Matrices

Defining a matrix is slightly more involved and uses the function matrix(). You must supply four pieces of information to the matrix function:

1. a list of values which comprise the matrix entries

2. how many rows the matrix has
3. how many columns the matrix has
4. whether R should take your values and fill the matrix going across rows first, or going down columns first

So, run these two commands, and compare how the matrix looks in each case.

```
> matrix(c(1, 2, 3, 4), nrow=2, ncol=2, byrow=TRUE)
> matrix(c(1, 2, 3, 4), nrow=2, ncol=2, byrow=FALSE)
```

Exercise: store the following matrix in a variable named D

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
>
```

3.1.3 Data Frames

Data frames are just like matricies, but they allow two useful extra features: named columns and non-numerical data (eg male/female, blue eyes/green eyes/etc). Almost all data you will use in the course will be provided in a data frame, or you will need to get into a data frame to use. Details of this will be addressed as we encounter them in the course.

There are lots of built-in data sets in data frames lurking in the background when you run R. Try the following to have a look at one and observe the named columns and non-numerical data that is present.

```
> iris
```

4 Re-using Code

One quick method of re-running previously typed code is to use the ↑ and ↓ keys. Pressing these allows the user to cycle through previous code that is saved in the workspace. This then allows previous code to be either quickly run again as it is, or to first make suitable alteration.

An alternative (and generally better) approach is to write your code as an R script. This allows you to highlight any part of previously written code and to easily run it by pressing the 'Ctrl' and 'R' keys together on the keyboard.

To open an R script select 'File' from the top menu and click 'New script'. R scripts can then be saved and re-used when necessary.

Exercise: create an R script for this lab and copy your code into it, then practise using Ctrl+R to run highlighted commands.

5 Exploring Data

5.1 Simple Summary Statistics

Consider the black cherry trees data which is available immediately for you to use in the `trees` variable. Type the following to have a look at the data:

```
> trees
> ?trees
```

`R` can provide very quick summary information about data sets like this. Try the following:

```
> summary(trees)
> sd(trees)
> var(trees)
```

These three very short commands have given us access to numerical summaries for the girth, height and volume parameters. We can see the min/max values; the quartiles; the means and medians; the standard deviations; and the covariance matrix.

Exercise: square the `sd()` command and notice how, as expected, it gives you the diagonal entries from the result of `var()`

```
>
```

5.2 Getting at the Data

`trees` is an example of a data frame. We can look at just the girth data like so:

```
> trees$Girth
```

Exercise: look at just the height data in the same way and then calculate the mean for just the height data (*Tip: use* ***`help.search()`*** *and/or rseek.org if you don't know how to calculate the mean without getting the min/max/median/quartiles as well*)

```
>
>
```

We can make working with a data frame less cumbersome than this if we 'attach' it to the current workspace: it will mean that we can access the girth, height and volume data simply by their own name (ie without the `trees$` bit above)

```
> attach(trees)
> Girth
```

5.3 Simple Graphical Summaries

`R` can rapidly produce a lot of graphical outputs, a tiny proportion of which we can see here:

```
> boxplot(trees)
> hist(Height)
> plot(Girth, Volume)
```

That last plot looks like there's a strong linear relationship. We can easily ask `R` to calculate a standard linear regression where Volume depends on Girth using `lm` (linear model) and plot the line with `abline`. This is an example of nesting one function call inside another and can lead to powerful yet easy combinations of commands:

```
> abline(lm(Volume ~ Girth))
```

When there are more than two parameters, a nice way to quickly see all the values plotted against each other is to use `pairs()`:

```
> pairs(trees)
```

Exercise: the `hist()` command above produced counts on the vertical axis. Use `?hist` to find out how to turn off representation of frequencies and instead display probability densities.

```
>
```