Lab: Using R and Bioconductor

Robert Gentleman
Florian Hahne
Paul Murrell

January 18, 2006

Introduction

In this lab we will cover some basic uses of R and also begin working with some of the Bioconductor data sets and tools. Topics covered include basic use of R, R graphics, working with environments as hash tables.

Some Basic R

First load the Biobase package and then the data set `exSet`.

```
> library("Biobase")
```

```
> data(exSet)
> exSet
```

```
Expression Set (exprSet) with
        500 genes
        26 samples
                 phenoData object with 3 variables and 26 cases
         varLabels
                sex: Female/Male
                type: Case/Control
                score: Testing Score
```

The expression set is an S4 class and `exSet` is an instance of this class. You can get help (a description of the class) by using the `?` operator; Try typing `class ? exprSet`.

```
> class(exSet)
```

```
[1] "exprSet"
attr(,"package")
[1] "Biobase"

> slotNames(exSet)

[1] "exprs"        "se.exprs"     "description"  "annotation"   "notes"
[6] "reporterInfo" "phenoData"

> exSet$cov1

NULL

> exSet[1, ]

Expression Set (exprSet) with
        1 genes
        26 samples
                 phenoData object with 3 variables and 26 cases
         varLabels
                sex: Female/Male
                type: Case/Control
                score: Testing Score

> exSet[, 1]

Expression Set (exprSet) with
        500 genes
        1 samples
                 phenoData object with 3 variables and 1 cases
         varLabels
                sex: Female/Male
                type: Case/Control
                score: Testing Score
```

You can extract the values in the slots using the `@` operator, or in many cases accessor functions are available. The names of the slots can be obtained using *slotNames*, as shown above. Extract the values for some of the named slots.

Exercise 1

- *What happens when we subset* `exSet`*? What kind of an object do we get?*
- *What happened to the phenotypic data? What happened to the expression data?*
- *Subset* `exSet` *by selecting all elements for which* `cov1` *has value* `1`*.*

Environments

In R an `environment` is a set of symbol-value pairs. These are very similar to lists, but there is no natural ordering of the values and so you cannot make use of numeric indices. Otherwise they behave the same way.
We first create an environment and then add, remove, list etc.

```
> e1 = new.env(hash = TRUE)
> e1$a = rnorm(10)
> e1$b = runif(20)
> ls(e1)

[1] "a" "b"

> xx = as.list(e1)
> names(xx)

[1] "a" "b"

> rm(a, envir = e1)
```

Exercise 2

- *Create an environment and put a copy of* `exSet` *into it.*
- *Fit a linear model to the data* `x=1:10`, `y=2*x+rnorm(10, sd=0.25)`, *and also place this into your environment.*
- *Write a function, myExtract, that takes an environment as an argument and returns a list, one element is the variable* `cov2` *from* `exSet` *and the other is the vector of coefficients from the linear model.*

Something Harder

Later we will spend some time discussing machine learning (ML), but here we will just use one simple algorithm, k-nearest neighbors (*knn*) to make predictions. You should read the R manual page for a description of *knn*.

```
> library("class")
> apropos("knn")

[1] "knn"    "knn.cv" "knn1"
```

The *knn* algorithm predicts the class of a given observation (the test case) according to a majority vote of the k nearest neighbors in the training set. We will show how you can use this to predict the class of sample 1, given data on samples 2 through 26.

```
> exprsExSet = exprs(exSet)
> classExSet = exSet$cov2
> esub = exSet[, -1]
> pred1 = knn(t(exprs(esub)), exprs(exSet)[, 1], esub$type)
> classExSet[1]
```

```
NULL
```

Exercise 3

- *Write a function, that takes an exprSet as its input and carries out a leave-one-out set of predictions. Your function should return the vector of predicted values for the given covariate.*
- *Modify your function to allow the user to specify some of the parameters for knn, such as k.*

The `apply` functions

In R a great deal of work is done by applying some function to all elements of a list, matrix or array. There are several functions available for you to use, *apply*, *lapply*, *sapply* are the most commonly used. From the next release of R onwards there will also be an *eapply* for use with environments.
To get some understanding of the apply functions we will attempt to extract some information from the Gene Ontology information that is supplied with each data package.
This next code chunk shows how to use apply to extract all the molecular function GO terms for each Affymetrix probe set.

```
> library("GO")
> library("hgu95av2")
> affyGO = as.list(hgu95av2GO)
> affyMF = lapply(affyGO, function(x) {
+     onts = sapply(x, function(z) z$Ontology)
+     if (is.null(unlist(onts)) || is.na(unlist(onts)))
+         NA
+     else unique(names(onts)[onts == "MF"])
+ })
```

Exercise 4

- *How are the GO terms stored? What information is available for each?*
- *What are the evidence codes and what do they mean?*
- *Turn this code into a function that would allow users to obtain either the MF, BP or CC data.*
- *Extend this function to allow the user to include only given evidence codes. (Or if you think it better - to exclude specific codes).*

Finding help in R

In Section 1 you have already learned about the ? operator and how you can get information about a certain R function or object. In addition there are a lot of other sources for help in and out of R.
Function *apropos* can be used to find objects in the search path partially matching the given character string. *find* also locates objects, yet in a more restrictive manner.

```
> apropos(mean)

[1] "kmeans"          "weighted.mean"   "mean"            "mean.Date"
[5] "mean.POSIXct"    "mean.POSIXlt"    "mean.data.frame" "mean.default"
[9] "mean.difftime"

> find(mean)

[1] "package:base"
```

If you want to get information about a certain topic or concept, try *help.search*. The function searches the help system for documentation matching a given character string in the (file) name, alias, title, concept or keyword entries and names and titles of the matched help entries are displayed.

```
> help.search("mean")
```

Moreover there is a wealth of information just waiting for you out in the web: For many of the usual R-related questions you may most likely find an answer in the R-FAQ at `http://cran.r-project.org/faqs.html`. A more specialised source for help are the R and Bioconductor mailing lists (`http://www.r-project.org/mail.html`, `http://www.bioconductor.org/mailList.html`). You can subscribe to different sublists, regarding your interests and level of expertise and post your questions to the R society. Before doing so, you should by all means read the posting guides. Many questions on the mailing lists will most likely not be answered because major posting rules have been violated. It is also a good idea to search the online mailing archives before posting a question. Most of them have already been asked and answered by someone else.A searchable Bioconductor archive can be found at `http://files.protsuggest.org/cgi-bin/biocond.cgi`, the R archives at `http://maths.newcastle.edu.au/~rking/R`.
All of these links can of course also be found on the Bioconductor and R-Project webpages.

Exercise 5

- *There are a number of different plotting functions available. Can you find them?*
- *Try to find out how to do a Mann-Whitney test.*
- *Take a look at the R posting guide and find out about the most common mistakes when posting a question.*

- *Use the searchable R mail archive and find out how to color tick marks in a plot with the segments function. You may need this information in one of the following exercises! [Hint: Try the keywords 'plot', 'tick' 'labels' and 'colour']*

Working with packages

There are now hundreds of packages available for R and over 150 for Bioconductor. It will be important that you learn how to find, download and install different packages.
There are a number of different methods that can be used and over time we expect them to become more standard. R packages are stored in libraries, you can have multiple libraries on your computer, although most people have only one on their personal machine.
Packages must be downloaded and installed. You need to do this only once. After that, each time you want to use the package you must load it. You do this using either the *library* function or the function *require*.
Downloading packages can be done using the menu on a distribution of R that has a GUI (this is either Windows or OS X). On these platforms you simply select the packages you want and they are downloaded and installed, but they are **not** loaded into your R session, you must do that. By default this mechanism will download the appropriate *binary* packages.
You can use the function *install.packages* to download a specified list of packages. One of the arguments to *install.packages* controls whether package dependencies should also be downloaded and for Bioconductor packages we strongly recommend setting this to `TRUE`.
In Bioconductor we have a developed a second, and somewhat more intricate set of package management tools. For the most part this is because of the complex set of interdependencies that exist in Bioconductor (most packages on CRAN have no dependencies). The package reposTools contains all of the Bioconductor functions for dealing with packages and we strongly recommend that you download and install this package first. The names of most of the functions in reposTools are similar to those for standard R functions, for example *install.packages2* works in much the same way as *install.packages* does. The tools in reposTools can download packages from CRAN (but the converse is not true) so we recommend using reposTools.

```
> .libPaths()
```

```
[1] "/Users/seth/RLIB"
[2] "/Users/seth/proj/builds/R-devel-upstream/library"
```

The Bioconductor project also has a script, called *getBioC* that downloads and installs a minimal set of Bioconductor tools for doing microarray analysis. Please note that the complete set is pretty large and will take some time to download even on a fast ethernet connection.
For this course we will be using the development version of R and also of the Bioconductor packages. You will have some substantial problems and things won't work if you are not using the development version of the packages.

Exercise 6

- *What is the output of function sessionInfo?*

Graphics

In this section you will work through some examples that allow you to create very general plots in R. The function *plot* can be used to produce dot plots. Read through its documentation (`? plot`) and also take a look into the documentation for `par`, which controls most of the parameters for R's base graphics.

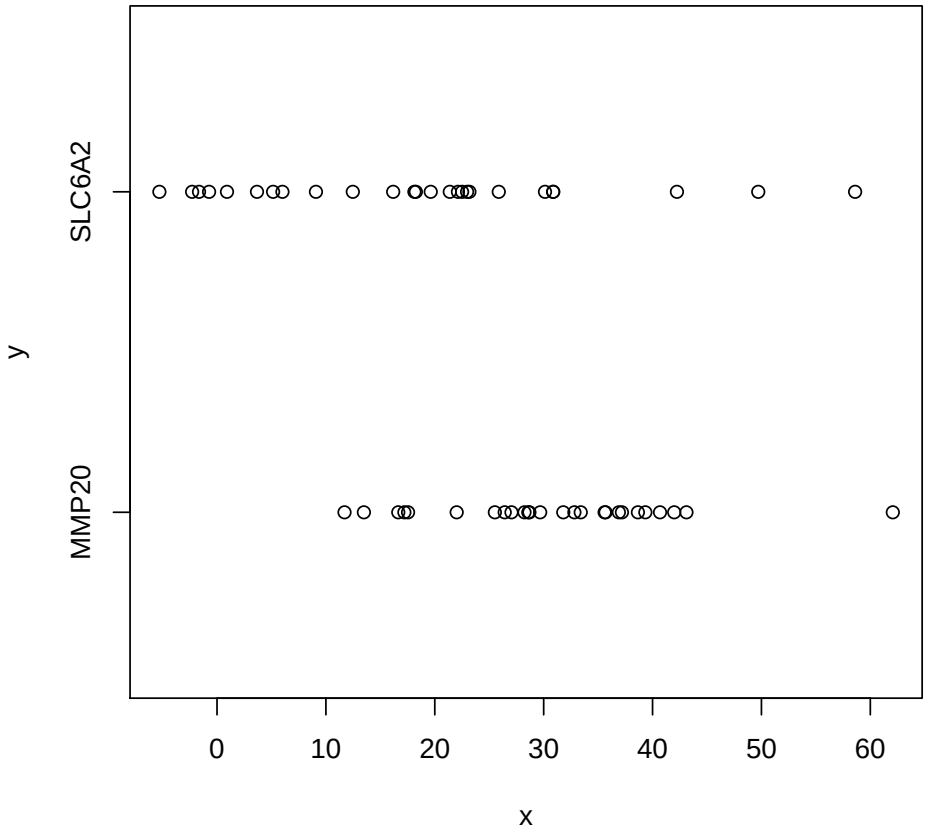

Figure 1: Figure for Graphics Question 1.

Exercise 7

- *Select two probesets from* `exSet` *and use their expression data to produce a plot like the one in Figure 1. The relevant features are the tick marks on the y-axis and the*

vertical positioning of the data symbols. [Hint: You can find the gene symbols in hgu95av2SYMBOL and function axis might be useful.]

Now, let us go one step further and try to plot some details of our example data. We first find out the chromosomal locations of our genes.

```
> whCHR = unlist(mget(geneNames(exSet), hgu95av2CHR))
> chrGenes <- table(whCHR)
> chrGenes

whCHR
 1 10 11 12 13 14 15 16 17 18 19  2 20 21 22  3  4  5  6  7  8  9 Un  X  Y
39 13 20 31 11 14  7 12 19  5 21 32  6  3 14 15 21 12 34 22 20 14  1 15  7
```

This tells us how many genes from each chromosome are included in our data set. We now want to use this data to produce a barplot like the one in Figure 2 using the R function *barplot*. We also want to compute moving average values (which can be used in breakpoint analysis) for our data. For chromosome i, the moving average is the average of chromosomes $i-7, i-6, \ldots, i$).

```
> mean.8 <- rep(0, length(chrGenes) - 7)
> for (i in 1:length(mean.8)) mean.8[i] <- mean(chrGenes[i:(i +
+     7)])
```

Exercise 8

- *Create the boxplot.*
- *Now superimpose the moving averages values over your plot.*

We now want to plot our chromosome data, basically creating plots similar to those in geneplotter, such as *alongChrom*.

Exercise 9

- *Select a chromosome (any one) to produce your plot.*
- *Find out the length of this chromosome (in bases). [Hint: the necessary data is in hgu95av2.]*
- *Find the position for each gene, on your selected chromosome. [Hint: hgu95av2CHRLOC]*
- *Create a plot with a single horizontal line and add a tick mark for each gene (perpendicular to the horizontal line).*
- *Can you color the tick marks according to gene expression?*

The version number of R and packages loaded for generating this document are:

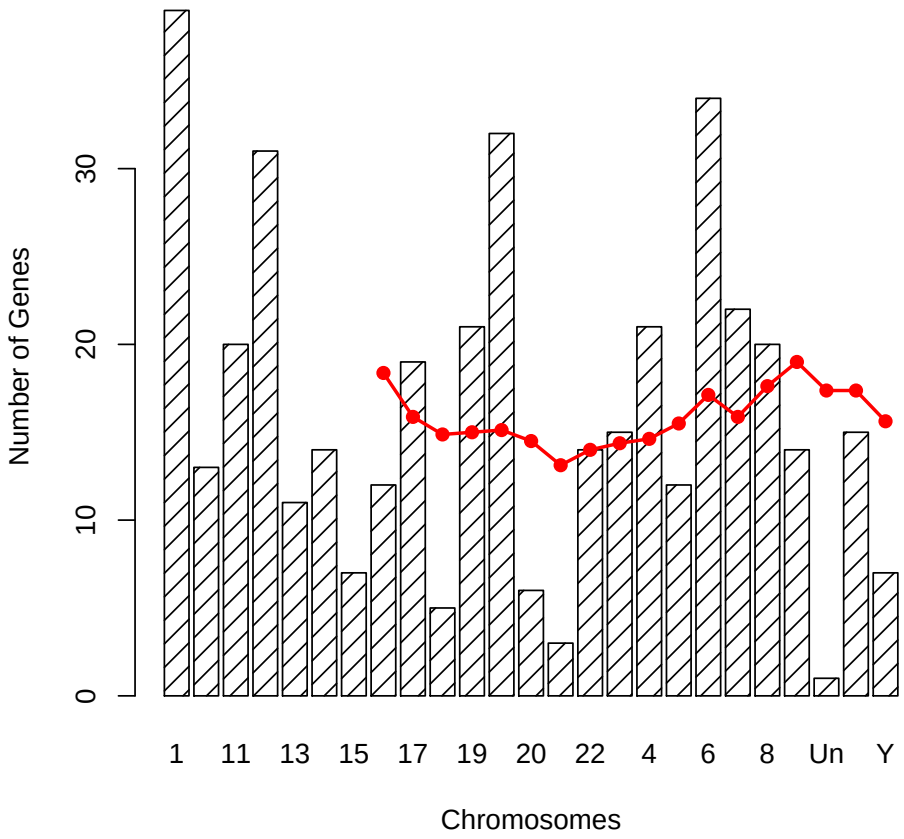

Figure 2: Figure for Graphics Question 2.

```
Version 2.3.0 Under development (unstable) (2006-01-15 r37092)
powerpc-apple-darwin8.4.0

attached base packages:
[1] "tools"     "methods"   "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "base"

other attached packages:
hgu95av2       GO    class  Biobase
"1.11.0" "1.10.0" "7.2-25"  "1.9.2"
```