Programming with Applications in

An Introduction

Kim-Ahn Le Cao & Peter Bailey
University of Queensland

Contents

At the end of this practical you should be able to:

- Describe the structure of an expressionSet object and the type of data contained with an expressionSet object;
- extract data from and expressionSet object;
- subset an expressionSet object;
- filter genes exhibiting little variation or having low signals across samples;
- standardize and/or normalize gene expression values;
- perform differential expression analysis using expression values obtained from an expressionSet object;
- annotate gene expression values and produce a HTML table of gene annotations; and
- perform multiple testing corrections.

Chapter 1

Bioconductor

1.1 Bioconductor, what is it?

1.2 Installing `Bioconductor`

Bioconductor can be installed by typing the following line in an R command window:

```
> source("http://bioconductor.org/biocLite.R")
```

After installation of `Bioconductor`, individual Bioconductor Packages can be installed by using the `biocLite.R` script:

```
# installing the package 'limma':
> biocLite("limma")
```

Or multiple packages:

```
> biocLite(c("HTqPCR", "Rsamtools"))
```

1.3 Learning Resources

The `Bioconductor` home page can be found at the following address http://www.bioconductor.org/

The students are also directed to the following learning resources:

- Wim P. Krijnen (2009) Applied Statistics for Bioinformatics using R (dowload from blackboard)
- R & Bioconductor Manual at http://manuals.bioinformatics.ucr.edu/home>

Each Bioconductor package contains at least one *vignette*, a document that provides a task-oriented description of package functionality. Vignettes contain executable examples and are intended to be used interactively.

You can access the PDF version of a vignette for any installed package from inside R using the `browseVignettes()` function. For example, to view the vignettes in the Biostrings package enter the following at an R prompt:

```
> browseVignettes(package = "Biostrings")
```

This will open a web browser with links to the vignette PDF as well as a plain-text R file containing the code used in the vignette.

The vignette files, both the PDF and the Rnw sources document, are located in the doc directory of an installed package (inst/doc for an uninstalled package tarball). You can discover the location of an installed package as follows:

```
> system.file(package = "Biostrings")
```

Chapter 2

Introduction to the ExpressionSet class

2.1 ALL data

Before starting this Section, students should read the following paper:

'Sabina Chiaretti, Xiaochun Li, Robert Gentleman, Antonella Vitale, Marco Vignetti, Franco Mandelli, Jerome Ritz, and Robin Foa Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. Blood, 1 April 2004, Vol. 103, No. 7.

Here we use a data set derived from Chiaretti et al., a study relating to Acute Lymphoblastic Leukemia (ALL). The data set consist of microarray data (chip series HG-U95Av2) from 128 individuals, 95 patients with B-cell ALL and 33 patients with T-cell ALL. The data set is provided as an `expressionSet` object called `ALL`. To get an overview of the structure of the `ALL expressionSet` object refer to the "expressionSet-class" help page:

```
> library("Biobase")
> #help("ExpressionSet-class")
> library("ALL")
> data(ALL)
> ALL
```

You can explore the slots available in the `ALL expressionSet` object by using the `slotNames()` function.

```
> slotNames(ALL)
```

```
[1] "experimentData"    "assayData"         "phenoData"
[4] "featureData"       "annotation"        "protocolData"
[7] ".__classVersion__"
```

Accessing the data in each slot is as easy as calling the accompanying accessor function:

- `annotation(ALL)`
- `pData(ALL)` for accessing the phenotypic data in the phenoData slot
- `varLabels(ALL)` for accessing the phenotypic data labels
- `exprs(ALL)` for accessing gene expression values
- `fData(ALL)` for accessing the probe feature data in the featureData slot

- `featureNames(ALL)` for accessing the probe names on the array

Exercise 1 *Call each of the functions above. What type of information is returned by each function? N.B. It is important to understand the type of information contained within each slot of the* `expressionSet` *object.*

The `phenoData` slot contains, among other things, important information about the class of tumour we are studying. How do we obtain this information from the ALL `expressionSet` object?
The `phenoData` slot contains a table with the rows representing the tumour samples and the columns representing the phenotypic data of interest. We can take a peek at the type of phenotypic data contained within the phenoData slot by calling the `varLabels(ALL)` function. This function simply returns the column names of the table contained within the phenoData slot.

The phenotypic data of interest is specified by the column labelled `"mol.biol"`. We can extract the data under the `"mol.biol"` column heading as follows:

```
> pData(ALL)[,"mol.biol"]
```

You can also access the columns of the phenotype data using the $ operand.

```
> ALL$mol.biol[1:5]
```

The names of the features (probe names or gene names) comprising the microarrary can be retrieved using the `featureNames()` function. For many microarray datasets, the feature names are the probeset identifiers.

```
> featureNames(ALL)[1:5]
```

It is important to note that these feature names represent the rownames of the matrix returned by `exprs(ALL)`. Check for yourself:

```
> featureNames(ALL)[1:5]
> data <- exprs(ALL)[1:5,]
> rownames(data)
```

The unique identifiers of the samples in the data set are available via the textttsampleNames() method. Again it is important to remember that the textttvarLabels() function lists the column names of the phenotype data.

```
> sampleNames(ALL)[1:5]
> varLabels(ALL)
```

The expression matrix and the `AnnotatedDataFrame` of sample information can be extracted using the `exprs` and `pData` methods, respectively:

```
> exprs(ALL)[1:5, 1:5]
> pData(ALL)[,"mol.biol"]
```

2.2 Subsetting

Subsetting an `ExpressionSet` is very similar to subsetting an expression matrix, the first argument subsets the features (rows) and the second argument subsets the samples (columns). For example:

```
> subset <- ALL[1:5, 1:3]
> exprs(subset)
```

What if we wanted to select a subset of patients on the basis of sex. We can obtain a subset consisting of males only as follows:

```
> ALL[, ALL$sex=="M"]
```

For the rest of this section we will compare a subset of B-cell tumours that comprise a BCR and ABL1 gene fusion product (as a result of translocation t(9;22)(q34;q11)) with a group of tumours that do not comprise a common cytogenic aberration. We can construct a subset of the original `ALL expressionSet` object comprising only the samples of interest as follows:

```
> bcell <- grep("^B", as.character(ALL$BT))
> moltyp <- which(as.character(ALL$mol.biol) %in% c("NEG", "BCR/ABL"))
> ALL_bcrneg <- ALL[, intersect(bcell, moltyp)]
> ALL_bcrneg$mol.biol <- factor(ALL_bcrneg$mol.biol)
> ALL_bcrneg
```

Exercise 2 *Deconstruct the code above. What is each line of code doing?*

2.3 Non-specific filtering

Filtering features exhibiting little variation or having consistently low signals across samples is usually an important first step in data analysis. Furthermore, one may decide that there is little value in considering features with insufficient annotation.

How do we exclude genes that show little variation across samples?

We might agree that genes showing a standard deviation of less than 0.7 across the tumour samples should not be included for further analysis. We can exclude these genes by using the `sd()` function in combination with the `apply()` function as follows:

```
> data <- exprs(ALL_bcrneg)
> nrow(data)
> gene.sd <- apply(data, 1, sd)
> sum(gene.sd>=0.7)
> selected.data <- data[gene.sd>=0.7,]
> nrow(selected.data)
```

We can also provide the `apply()` function with our own homemade function. The coefficient of variation (cv) is a useful measure of variability in relation to the mean and is defined as the standard deviation divided by the absolute value of the mean. If cv=1 the standard deviation is equal to the mean and we can say that the variability relative to the mean is small. Alternatively, if cv=0.2 we can say that the variability relative to the mean is large and can assume that the experimental effect between samples will be significant.

So let us first make a function called 'func.cv'. Functions are defined by the reserved word "function"" followed by closed brackets. The closed brackets in this case will comprise a variable which we will arbitrarily call x.

```
> func.cv <- function(x){
+   sd(x)/abs(mean(x))
+ }
```

This function will calculate the cv for each row of the matrix holding the expression values. It is supplied to the apply function as follows:

```
> cv <- apply(data, 1, func.cv)
> head(cv)
```

The call to apply above is doing the following:

1. extracting the expression values for each row of matrix "data"" and passing those values onto the function func.cv;
2. the function func.cv accepts those values and assigns them to the variable x;
3. the cv is then calculated for the values in x and is passed back the variable cv.

Finally we can select the genes having a cv less than 0.2 as follows:

```
> sum(cv<0.2)
> data.cv <- data[cv<0.2,]
> nrow(data.cv)
```

2.4 Standardising gene expression values

The recorded gene expression levels across an array are not directly comparable. For some statistical comparisons it is important that all gene expression values have equal weight. Standardization of gene expression values may be achieved by mean or median centering across samples. In the following code segment we will mean centre the gene values across the samples using the apply function in combination with the sweep function:

```
> gene.median<- apply(data, 1, median)
> data.median.centred <- sweep(data, 1, gene.median)
```

The `sweep()` function is used here to subtract the gene.medians from each row of the expression matrix "data". Look at the sweep help page "?sweep" for details.

Exercise 3 *Use the `apply()` and `sweep()` functions to divide the median centred gene values with the standard deviation. Hint: Use the apply function first with `sd()` and then call the sweep function on the `data.median.centred` matrix generated above; you will need to change the argument `FUN`*

Expression values between patients may also be non-normally distributed. In such a case it may be advantageous to subtract the median and divide by the Median Absolute Deviation (MAD). The following code segment shows how this can be achieved by utilising the `sweep()` function. Consult the documentation provided with the `sweep()` function so that you understand what each call is doing.

```
>   mads <- apply(data.cv, 2, mad)
>   median <- apply(data.cv, 2, median)
>   data.m <- sweep(data.cv, 2, median)
>   data.standardised <- sweep(data.m, 2, mads, FUN="/")
```

The following plots show the effects of the standardisation procedures.

```
> par(mfrow=c(2,1))
> boxplot(data.cv[,1:10], main="Filtered  & No standardisation")
> boxplot(data.standardised[,1:10], main="Filtered & Standardised: Column Median/Mad")
```

2.5 Differential Expression

We are now in a position to identify genes that are differentially expressed between patients. Patients that are positive for the BCR/ABL gene fusion will be compared to patients that do not comprise a common cytogenic aberration.

We will use the `apply()` function in combination with the `t.test()` function to identify differentially expressed genes. To categorise the samples on the basis of presence or absence of the BCR/ABL fusion product will use the factor mol.biol contained in the ALL_bcrneg expressionSet object.

```
> bcrneg <- ALL_bcrneg$mol.biol
> bcrneg
> test <- function(x){
+   t.test(x ~ bcrneg)$p.value
+ }
> test.pvalue <- apply(data.cv, 1, test)
```

We can select the genes having a p-value less than 0.05 as follows:

```
> sum(test.pvalue < 0.05)
> de.genes <- test.pvalue[test.pvalue < 0.05]
> head(de.genes)
> names(de.genes)[1:10]
```

We also might like to order the genes on the basis of significance. To do so we can use the function order as follows:

```
> ord <- order(de.genes, decreasing=FALSE)
> de.genes.ordered <- de.genes[ord]
> de.genes.ordered[1:10]
```

By making the argument `decreasing=FALSE` we are forcing the function to place the p-values in order from lowest to highest i.e. from 0 to 1.

2.6 Basic Gene Annotation

After we have obtained a list of probe identifiers we will want to map the identifiers to their corresponding gene names. In the code segment below, we show how to use the **annaffy** bioconductor package to produce an HTML table of gene annotations.

```
> library("annaffy")
> library("hgu95av2.db")
> aaf.handler()
> annotation.col <- aaf.handler(chip="hgu95av2.db")[c(1:3, 8:9, 11:13)]
> annotation.table <- aafTableAnn(names(de.genes.ordered), "hgu95av2.db", annotation.col)
> saveHTML(annotation.table, "ALLs.html")
```

Exercise 4 *What gene is at the top of your list? Does this make sense?*

2.7 Multiple testing correction

2.7.1 Bonferroni correction

The simplest form of multiple testing correction is the Bonferroni correction. This correction says that, if you are aiming for a significance threshold of 0.05 but you repeat your test 1000 times, then you should adjust your threshold to $0.05/1000 = 0.00005$.

This method involves lowering the alpha threshold to ensure that $\alpha_{\text{corrected}} = \alpha/T$, where T is the number of tests.

2.7.2 E-value

The E-value is an alternative method for multiple testing correction.The E-value calculation is essentially the converse of the Bonferroni correction. Rather than dividing the target significance threshold by the number of tests performed, the E-value is the product of the p-value and the number of tests. If the significance

threshold is kept the same, then using E-values is exactly equivalent to the Bonferroni correction.

The E-value is calculated by simply multiplying the p-value by the number of genes on which the tests were performed.

$$\text{Adjusted p-value} = \text{p-value} * T$$

where T is the number of tests.

2.7.3 False Discovery Rate (FDR)

The False Discovery Rate (FDR) is the proportion of false positives among those genes that are selected as positives. The Benjamini-Hochberg correction consists in estimating this FDR, by computing a q-value in the following way. Data are first ordered by increasing p-value and a rank $(1, 2, 3, \ldots, k, \ldots, T)$ is computed for each row of the sorted data table. The q-value can then be computed as follows:

$$\text{q-value} = \frac{\text{p-value} * T}{k}$$

One then identifies the last item of the ranked list having a q-value =< alpha and retains as significant all items up to that item. The Benjamini-Hochberg correction is less stringent than the other multitesting corrections mentioned above and achieves a higher sensitivity.

The following code segment shows how to perform multiple testing correction:

```
> #Bonferroni
> length(test.pvalue)
> sum(test.pvalue < 0.05)
> Bonferroni <- 0.05/length(test.pvalue)
> bonferroni.passed <- test.pvalue[test.pvalue<Bonferroni]
> length(bonferroni.passed)
> #Alternatively
> adjusted.pvalues <- test.pvalue*length(test.pvalue)
> sum(adjusted.pvalues < 0.05)

> #FDR
> ord <- order(test.pvalue, decreasing=FALSE)
> pvalue.ordered <- test.pvalue[ord]
> q.value.factor <- length(pvalue.ordered)/1:length(pvalue.ordered)
> q.value <- pvalue.ordered * q.value.factor
> fdr.passed <- test.pvalue[q.value < 0.05]
> sum(q.value < 0.05)
```

Exercise 5 *We are interested in identifying genes that are differentially expressed between T-cell leukemia tumours identified as either stage T2 or T3. Tumours belonging to these stages are identified in the phenoData slot of the ALL expressionSet object under the column heading 'BT".*

1. *Produce an expressionSet object called ALL_BT comprising tumour samples categorised as stage T2 or T3.*
2. *Use the function "table" to find the frequencies of the patient types and leukemia stages in the ALL_BT expressionSet that you have generated.*
3. *Write a function to identify genes which are normally distributed in the ALL_BT expressionSet. Hint: We covered tests for normality in the earlier practicals.*
4. *How many genes pass the normality test?*
5. *Using this function, produce a matrix called ALL_BT_NORM which only consists of genes that are normally distributed*

6. *Using the matrix ALL_BT_NORM select only those genes with a significant p-value from an appropriate two sample T-test.*

7. *How many genes are significant?*

8. *Apply bonferroni and fdr correction to the pvalues obtained. How many genes now are significant after correcting for multiple tesing?*

9. *Having identifed genes differentially expressed between the two stages produce an annotated HTML table for publication. Use the corrected fdr p-values.*

10. *Produce a heatmap of the genes identified in question 6.*

11. *Perform PCA on the expression values comprising ALL_BT. What do you observe?*