Working with microarray data in BioConductor

The purpose of this class exercise is to work through the analysis of the bovine microarray data from the last class carefully, to learn how to find information to yourself and to investigate the effects of different normalization or analysis choices on the results you obtain. We will use the analysis pipeline from the last lecture as a starting point (it is on the course website at http://www.maths.nuigalway.ie/~cathal/Teaching/CT560/pauls_array_pipelineI.txt)

```
setwd("/home/nextgen/CEL_files")
library("affy")
rawData <- ReadAffy()
```

As it is used in the pipeline (lines reproduced above) the ReadAffy function reads in all the .CEL* files in the current working directory into an AffyBatch object. Find out how to tell the ReadAffy function to read in only a subset of the CEL files.

(** recall, the CEL files contain the raw probe expression intensities obtained from the array manufacturer's image analysis software*)

Find out about the AffyBatch class of objects. List all the kinds of information stored in an object of class AffyBatch.

The AffyBatch object contains the expression intensities read in from the CEL files. In which slot of the object is this data contained? How do you extract the matrix of probe expression intensities from the AffyBatch object?

For how many probes and how many samples do you have data?

You can plot an image of the raw array intensities as they are laid out on the array with the following command:

```
image(rawData)
```

The MA plot is a further graphical method that can be used for quality control. In the case of an Affymetrix array this compares each array individually against the median behaviour of the remainder of the arrays. On the y axis is M, the deviation in the log intensity ratio for the given array compared to the average of all other arrays and on the x-axis is A, the average log intensity in the other arrays. This allows you to see whether the extent of deviation from the mean for a given array is a function of the expression level (i.e. if higher or lower expressed probes tend to be further from the reference level for a given array).

```
## print a matrix of intensity levels for the probes on the array
exprs(rawData)

## dimensions of the matrix
dim(exprs(rawData))

## first 5 rows and columns of the matrix
```

```
exprs(rawData)[1:5, 1:5]
```

Consider the following line of the pipeline:

```
boxplot(log2(exprs(rawData),las=2)
```

- What does the argument las =2 do?
- What class of object is returned by the function exprs?
- What does log2 do?
- What kind of objects does the function boxplot take as input?
- What do the boxes and whiskers on the boxplot represent (exactly)?

Consider the following line:

pca_proc <- prcomp(t(exprs(rawData)))

- What does the function t do and why is it used here?
- What does the function prcomp? Read the help page for prcomp. Note that you can use as input a formula or a matrix (we have used the latter here).

The following commands were used to plot the results of the principal components analysis

```
mycol <- c(rep(1, 6), rep(2,6))

mypch <- seq(1,12)

plot(pca_proc$x, xlab="Component 1",
ylab="Component 2", col=mycol,
pch=mypch, main="PCA of Preprocessed Expression
set" )

legend("right", sampleNames(rawData), pch = mypch,
col = mycol)
```

Spend some time to work out what these commands are doing. The first two commands are setting up vectors that will later be used to specify colours and point types (circles, triangles etc). These are used in the next two commands. The plot command is plotting the

It might help to strip down the commands above to the simpler:

```
pca_matrix = pca_proc$x
plot(pca_matrix[,1],pca_matrix[,2])
```

Here what is contained in the slot called ‘x’ of the object created above called pca_proc is first stored as pca_matrix. What kind of object is this? Print it out to your screen. The plot function is then simply plotting the first principal component of pca_matrix (i.e. the first column of this matrix) against the second principal component (second matrix column).

Background correction, normalization and summarization

The first command in this section of the pipeline is

```
normalizedData <- rma(rawData)
```

Read the description of the rma function (in the documentation of the affy package here: http://www.bioconductor.org/packages/release/bioc/html/affy.html or else by typing ?rma). The function rma can perform background subtraction, normalization and summarization in one step. Is rma doing all three of these by default? If so, by default, what kind of normalization does rma do?

Try a different normalization function. There are typically many choices that you can make when you perform a microarray analysis. It is useful to be able to experiment with different approaches and to be aware of the impact that these choices have on your results. The difference between rma and gcrma is that gcrma uses probe sequence information to estimate the extent of non-specific binding (the gc stands for the guanine and cytosine nucleotides, but, in fact, the non-specific binding affinity scores are calculated from all bases). These scores were estimated from experiments designed specifically to measure non-specific binding. Use the following command:

```
normalizedData_gcrma <- gcrma(rawData)
```

Visualize the normalized and summarized data:

Use the boxplot command again to see spread of the normalized data

```
boxplot(exprs(normalizedData), las=2)
```

Also try the gcrma data:

```
boxplot(exprs(normalizedData_gcrma), las=2)
```

Perform the pca again (on both sets of summarized data)

```
pca_proc_norm <- prcomp(t(exprs(normalizedData)))

plot(pca_proc_norm$x, xlab="Component 1",
ylab="Component 2", col=mycol, pch=mypch, main="PCA
of Preprocessed Expression set" )
```

```
legend("right", sampleNames(normalizedData), pch =
mypch, col = mycol)
```

and

```
pca_proc_norm <-
prcomp(t(exprs(normalizedData_gcrma)))

X11()

plot(pca_proc_norm$x, xlab="Component 1",
ylab="Component 2", col=mycol, pch=mypch, main="PCA
of Preprocessed Expression set" )

legend("right", sampleNames(normalizedData), pch =
mypch, col = mycol)
```

What was the purpose of the X11( ) function? Did the different normalization strategies make any difference? Did the overall picture remain the same?

It may seem complicated, but everything we have done so far can be recapitulated in the following few commands:

```
setwd("/home/nextgen/CEL_files")
library("affy")
rawData <- ReadAffy()
normalizedData <- rma(rawData)
boxplot(exprs(normalizedData), las=2)
pca_proc_norm <- prcomp(t(exprs(normalizedData)))
mycol <- c(rep(1, 6), rep(2,6))
mypch <- seq(1,12)
plot(pca_proc_norm$x, xlab="Component 1",
ylab="Component 2",
col=mycol, pch=mypch, main="PCA of Preprocessed
Expression set" )
legend("right", sampleNames(normalizedData), pch =
mypch, col = mycol)
```

Make sure you understand these steps.

Inferring differential expression between two groups of samples using the limma package.

The limma (linear models for microarrays) package is frequently used to identify differentially expressed genes. You can find lots of information on the limma package here:
http://bioconductor.org/packages/2.6/bioc/vignettes/limma/inst/doc/usersguide.pdf

The following general R command makes a ‘design matrx’.

```
design <- model.matrix(~ -1+factor(c(rep(1,6),rep(2,6))))
colnames(design) <- c("NEB","Control")
                                # name experimental variables
```

In statistics 'contrasts' are usually differences between group means (note: this is not a precise definition of a statistical contrast). The limma function makeContrasts allows us to specify contrasts. Here, we are specifically interested in the difference in the average expression between our two groups

```
contrasts <- makeContrasts(NEB-Control, levels=design)
                                # define comparisons

# Fitting
fit <- lmFit(exprs(normalizedData),design)
                                # Fit the linear model
fit2 <- contrasts.fit(fit, contrasts)
                                # Fit the comparisons
fit2 <- eBayes(fit2)
```

eBayes gathers information across all genes to try to inform the gene-specific estimates of the standard error. This is important if you have relatively small numbers of samples in each group (typical for microarray experiments). In such cases it is difficult to estimate the standard errors accurately and many of the genes with the greatest apparent differences in mean expression might just be a consequence of low estimates of the standard errors (by chance). eBayes tries to reduce this problem by allowing the standard error estimates for each gene to be informed by the typical standard errors estimated across all genes.

```
# Ranking
top <- topTable(fit2,coef=1, adjust="BH", sort.by="P", number=10000)
        # Rank probes according to BH adjusted p-value
top <- top[top$adj.P.Val <0.05,]
        # define your cutoff
```

dim(top) # find out how many genes are in your 'top table'

Take a look at the genes in 'top'

Recalculate top but this time use bonferroni correction for multiple testing. How many genes are significant following bonferroni correction?

(*see http://folk.uio.no/mortema/PIPELINE_AFFY.txt for more*)

Filtering for non-expressed genes. Firstly – think about why you might want to filter out the non-expressed genes before we test for differential expression?

```
calls=exprs(mas5calls(rawData))
callSums=rowSums(calls=="P")
hist(callSums)
keep=(callSums>3)
```

```
dataNormFilt <- NormalizedData[keep,]
```

In addition to filter out genes that are not expressed in some minimum number of samples, you might also consider filtering out genes that do not show much variability in their expression across samples (not considering which group they come from). Try to find a way to do this (Google is your friend…)

Is there a danger that these filters could bias your results? In general, in a comparison of groups of samples, filters that do not make any use of the group labels are unlikely to bias your results.

Visualizing results:

```
volcanoplot(fit2,main="Volcanoplot")
```

A volcano plot shows a measure of significance of the results (e.g. log (p_value) or log-odds) against the fold-change. The fold-change (on the x-axis) shows the magnitude of the effect and the statistical significance of the effect is shown on the y-axis. You may be particularly interested in highly significant results with a large effect. The limma volcanoplot function shows log-odds of differential expression. A value of zero corresponds to a 50:50 chance of differential expression. Higher values correspond to greater significance. E.g. suppose the B-statistic is 2, then the odds of differential expression is $e^2 = 7$. The probability of differential expression is then 7/(7+1) = 0.875. The B-statistic is 'Bayesian'. It already accounts for multiple testing (by assuming that 1% of genes are differentially expressed – this is used as a 'prior' probability in the Bayesian sense).