Lab: From Affymetrix array CEL files to annotated list of interesting genes

Jean Wu

October 6, 2006

1 Introduction

Identifying a list of differentially expressed genes is one of the major applications of microarray experiments. In this lab we will start with Affymetrix CEL files, which contain the raw probe level (feature level) data and create an annotated report of interesting genes.
This lab is based on Chapter 25, *From CEL files to annotated list of interesting genes* by R. A. Irizarry.

2 Reading CEL files

Load the affy package, which is needed to import the data into Bioconductor.

```
> library("affy")
```

The probe level data provided by Affymetrix is contained in what we call the CEL files, because these usually have extension `.CEL`. The function `ReadAffy` can be used to import these data into `AffyBatch` objects.
One easy way of reading in the CEL files is to have all CEL files in one directory and set the R working directory to that directory. To set the working directory you can use the function `setwd`, or use the menu commands on either Windows or OS X. Then you can simply use the `ReadAffy` function, as shown in the code chunk below.

```
> MyData <- ReadAffy()
```

If instead, you have many CEL files and only want to select some of them, you can supply the names of the ones you would like to use as the *filenames* argument to the function. An example is given in the code chunk below.

```
> ReadAffy(filenames = filenames.of.your.data)
```

The help file for `ReadAffy` gives more details.
For illustration of the rest of this lab we will use an `AffyBatch` named `spikein95` that is already read-in, available through the package SpikeInSubset. You are welcome to try out your own data.

```
> library("SpikeInSubset")
> data(spikein95)
> spikein95@cdfName <- "hgu95av2"
```

2.1 The phenoData

These data are a six array subset (2 sets of triplicate arrays) from a calibration experiment performed by Affymetrix. For the purpose of this lab we generate a simple object, pd of class *phenoData* describing the arrays. The first three arrays are from one population and the next three are from the other population. We will arbitrarily label the first population .

```
> pd <- data.frame(population = factor(c("Pop1", "Pop1", "Pop1",
+     "Pop2", "Pop2", "Pop2")), replicate = c(1, 2, 3, 1, 2, 3))
> rownames(pd) <- sampleNames(spikein95)
> vl <- list(population = "Pop1 is control, Pop2 is treatment",
+     replicate = "1, 2, 3, arbitrary numbering")
> phenoData(spikein95) <- new("phenoData", pData = pd, varLabels = vl)
```

If you have your own data, generate a phenoData object according to your experimental design. The *phenoData* object can also be created using the function read.phenoData. For more details on this object, please refer to Chapter 7.

3 Preprocessing

To convert probe level data to expression measures (preprocessing), we use RMA as example.

```
> eset <- rma(spikein95)
```

If your computer has limited memory this command might fail. In that case you can try to use just.rma, but to use it you must have CEL files. So, if you brought your own CEL files, you can also use something similar to the code below.
just.rma.

```
> eset <- just.rma(filenames = list.celfiles())
```

After this function has run eset is an instance of the class exprSet which contains the expression values and other important experimental information. The expression values calculated by rma are in log base 2 scale. A matrix with the expression information is readily available. The following code extracts the expression data and obtains the dimensions of the matrix containing the data.

```
> e <- exprs(eset)
> dim(e)
```

```
[1] 12625     6
```

Exercise 1
How many probe sets are there in this dataset?

```
> dim(e)
```

```
[1] 12625     6
```

```
> dim(exprs(spikein95))
```

```
[1] 409600      6
```

The phenoData information that was stored with the raw data has been copied to the output of the preprocessing function.

```
> pData(eset)
> pData(spikein95)
```

You can conveniently use `$` to access each column and create indexes denoting which columns represent each population:

```
> Index1 <- which(eset$population == "Pop1")
> Index2 <- which(eset$population == "Pop2")
```

We will use this information in the next section.

Optional exercise: other preprocessing methods

Here we are only demonstrating the use of RMA. Other options are available through the functions `mas5` and `expresso`. If you find `expresso` too slow, the package affyPLM provides an alternative, `threestep`, that is faster due to the use of C code. In the next code chunk we create another expression set object, this time using `gcrma` to normalize the data.

```
> library("gcrma")
> eset2 <- gcrma(spikein95)
```

Then, in this next code chunk we make use of `threestep` from the affyPLM package. There are a number of different options that can be set. In the code chunk below we do not do any background correction, normalization is done using quantile normalization (the default) and a one step Tukey biweight function is used to summarize the data, per probeset.

```
> library("affyPLM")
> eset3 <- threestep(spikein95, background = FALSE, summary.method = "tukey.biweight")
```

4 Ranking and filtering probesets

Now we have, in `e`, a measurement x_{ijk} of log (base 2) expression from each probeset j on each array i for both populations $k = 1, 2$. You can rank the probesets as soon as you select a statistic.

4.1 Summary statistics and tests for ranking

Log fold-change A naive first choice is simply the average log fold-change: the `rowMeans` function works on matrix and provides a much faster alternative to the commonly used function `apply`:

```
> d <- rowMeans(e[, Index2]) - rowMeans(e[, Index1])
```

The variability of fold-change measurements often depends on over-all intensity of the probeset in question. An example is average log expression, which we compute in the code chunk below.

```
> a <- rowMeans(e)
```

Exercise 2
Plot `d` *against* `a` *(see Figure 1) to see if this applies to your data. Also plot* `d` *versus* `rank(a)`*. Does the variability of the* `d`*-values depend on* `a`*?*

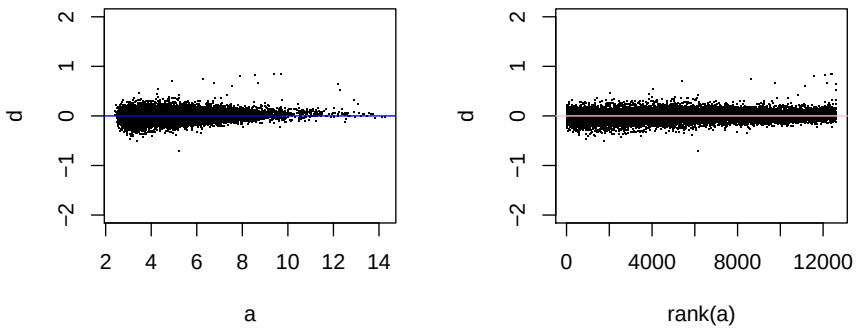

Figure 1: Scatterplot of `d` against `a`.

```
> par(mfrow = c(1, 2))
> plot(a, d, pch = ".", ylim = c(-2, 2))
> abline(h = 0, col = "blue")
> plot(rank(a), d, pch = ".", ylim = c(-2, 2))
> abline(h = 0, col = "pink")
```

Another simple statistic that takes into account the probeset-specific variation is the t-statistic. You can use the function `rowttests` from the package genefilter, which applies the t–test to each row:

```
> library("genefilter")
> tt <- rowttests(eset, "population")
```

The first argument is the *exprSet*, the second indicates the covariate which defines the two groups to be compared.
When there are few replicates the variance is not well estimated and the t-statistic can perform poorly. And alternative statistics that borrow strength across all genes often provide better results. An example of such a modified t-statistic is based on an empirical Bayes approach, implemented in the `eBayes` function in the limma package.

```
> library("limma")
> design <- model.matrix(~eset$population)
> fit <- lmFit(eset, design)
> ebayes <- eBayes(fit)
```

Because the example data set is from a calibration experiment, the truly differentially expressed genes are known. Interested users can refer to the optional exercise to carry out the comparison and confirm the better performance of the moderated versus classical t-statistics in this case. When sample sizes are moderate, say ten in each group there is generally no advantage to using the Bayesian approach.

4.2 Visualization of Differential Expression

The volcano plot is a useful way to see the estimate of the log fold-change and statistic you choose to rank the genes simultaneously. Figure 2 plots p-values (more specifically $-\log_{10}$ versus

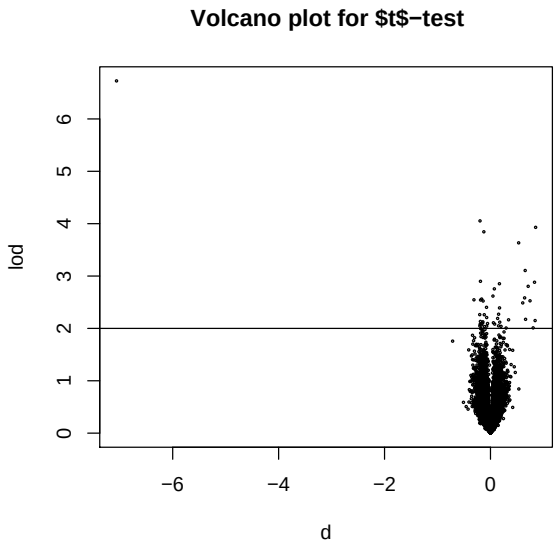

Figure 2: Volcano plot.

the p-value) versus effect size. For simplicity we assume the t-statistic follows a t-distribution to obtain the p-values.
To create a volcano plot you can use the following simple code:

```
> lod <- -log10(tt$p.value)
> plot(d, lod, cex = 0.25, main = "Volcano plot for $t$-test")
> abline(h = 2)
```

Exercise 3
Generate the volcano plot using a moderated t-statistic (Figure 3). Do they look the same?

```
> plot(d, -log10(ebayes$p.value[, 2]), xlim = c(-1, 1), cex = 0.25,
+     main = "Volcano plot for t-test")
> abline(h = 2)
```

You can highlight a subset of genes with the `points` function. For example, you could set the graphical parameters to use blue, (`col="blue"`) diamonds, (`pch=18`), to denote the top 25 genes ranked by average fold change.

4.3 Highlighting interesting genes

Exercise 4
Can you highlight the top 20 genes ranked by smallest p-value or the largest statistic of your choice, with a different color and symbol (as in Figure 4)? (Hint: look at the function `points` *with options* `col="blue"` *and* `pch=18`*.)*

```
> plot(d, lod, cex = 0.25, xlim = c(-1, 1), ylim = range(lod),
+     main = "Volcano plot")
> o1 <- order(abs(d), decreasing = TRUE)[1:25]
> points(d[o1], lod[o1], pch = 18, col = "blue")
```

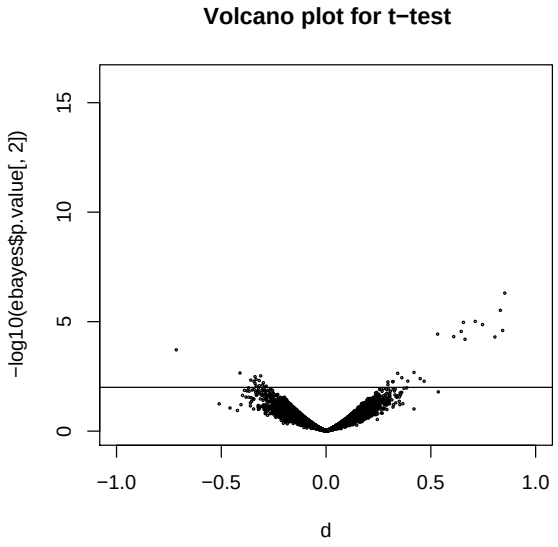

Figure 3: Volcano plot when using moderated t-statistic.

4.4 Selecting cut-offs

When you get to this section, you have generated three statistics that can be used to rank genes. Now we turn our attention to deciding on a cutoff.
A naive approach is to consider genes attaining p-values less than 0.01. However, because we are testing many hypotheses, the p-values no longer have the typical meaning. There are 12625 null hypotheses (this is the number of probesets on the array) in the example data set. If they are all true (no genes are differentially expressed) we expect $0.01 \times 12625 = 126.25$ false positives. How many probesets with $p < 0.01$ do you actually see using the t-test? Or with the moderated t-test?

```
> table(tt$p.value <= 0.01)

FALSE  TRUE
12579    46
```

Various approaches have been suggested for dealing with the multiple testing problem. Chapter 15 gives a detailed discussion and the multtest package provides extensive implementations of the different options. Alternatively, the `topTable` in limma package provides some adjustment to the raw p-values, including Benjamini & Hochberg's False Discovery Rate (FDR), simple Bonferroni correction and several others. For details look at help files for `topTable` and `p.adjust`. The following example lists the top 10 genes and creates a report.

```
> tab <- topTable(ebayes, coef = 2, adjust.method = "BH", n = 10)
> genenames <- as.character(tab$ID)
```

Here, `coef=2` specifies that we care for the second coefficient in the linear mode fit, that is the line slope, as the parameter of interest. The first coefficient is the intercept. The parameter n indicates how many genes should be selected.

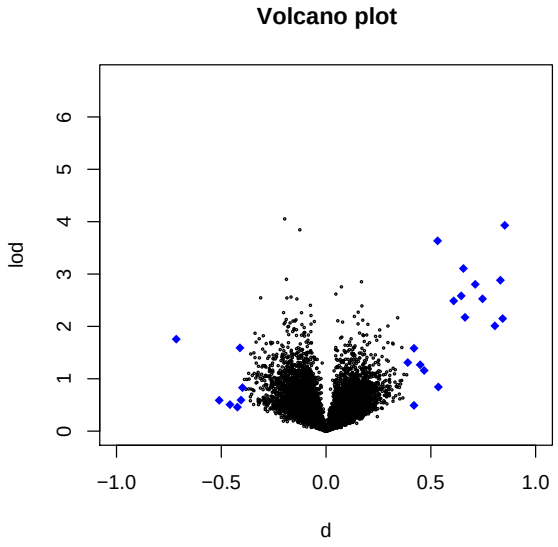

Figure 4: Volcano plot with highlighting of the top 25 probesets.

4.5 Annotation

The following is an optional exercise that you can do when you have internet access. First we load the annotate package

```
> library("annotate")
```

and find out which metadata package we need.

```
> annotation(eset)
```

```
[1] "hgu95a"
```

For most platforms the character returned by `annotate` function is the name of the annotation package you will need. However, Bioconductor maintains annotation information for both HG-U95A and HG-U95Av2 platforms in the same package hgu95av2.
Now load the package hgu95av2.

```
> library("hgu95av2")
```

Now you can add more annotation information about our top 10 interesting genes. For example, the EntrezGene ID and gene symbol,

```
> ll <- getLL(genenames, "hgu95av2")
```

```
 1708_at 36202_at 36311_at 33264_at 32660_at 38734_at  1024_at 36085_at
    5602     5569     5136    55556     9881     5350     1543     2779
33818_at 39058_at
    7415       29
```

```
> sym <- getSYMBOL(genenames, "hgu95av2")
```

```
 1708_at 36202_at 36311_at 33264_at 32660_at 38734_at  1024_at 36085_at
"MAPK10"   "PKIA"  "PDE1A" "ENOSF1"   "LBA1"    "PLN" "CYP1A1"  "GNAT1"
33818_at 39058_at
   "VCP"    "ABR"
```

With these values available, we can use the following code to create an HTML page, useful for instance to share results with collaborators.

```
> tab <- data.frame(sym, signif(tab[, -1], 3))
> htmlpage(ll, filename = "GeneList1.html", title = "HTML report",
+     othernames = tab, table.head = c("Locus ID", colnames(tab)),
+     table.center = TRUE)
```

Look for the resulting file *GeneList1.html* in your R working directory.

```
> browseURL("GeneList1.html")
```

Look for a file named `GeneList1.html` in your R working directory.
The above HTML report only connects with EntrezGene. To create a report with more annotation information, we can use the annaffy package. Below are a few lines of code that create a useful HTML report HTML report with links to various annotation sites.

```
> library("KEGG")
> library("GO")
> library("annaffy")
> atab <- aafTableAnn(genenames, "hgu95av2", aaf.handler())
> saveHTML(atab, file = "GeneList2.html")
```

`aaf.handler()` returns 13 annotation types. You can also select a subset instead of using all of these.

```
> aaf.handler()
> atab <- aafTableAnn(genenames, "hgu95av2", aaf.handler()[c(2,
+     5, 8, 12)])
> saveHTML(atab, file = "GeneList3.html")
```