From CEL Files to Annotated Gene Lists

Jean-Yves Sgro

May 9, 2017

Contents

1 Learning objectives | Acknowledgments

This document is a `knitr` trasnscription of the calculation process presdented in chapter 25 **"From CEL FIles to Annotated Lists of Interesting Genes"** by R.A. Irizarry in the book Bioinformatics and Computational Biology Solutions Using `R` and Bioconductor Released 1 Sept 2005. (Gentleman et al. 2005)

This example uses microarrays, but Next Gen data can also be used later albeit with different `R` packages.

> **The main purpose is to learn the process & to understand how to use R/Bioconductor.**

Some summary of the chapter is included for clarity.

2 Add R / Bioconductor packages

Some packages from CRAN and/or Bioconductor may be necessary to complete your `R` installation. Installation is different depending on which source the pacakge comes from.

Namely these may not be yet intalled, here is a procedure to intall them.

Unless you have "admin" privilege on the computer, a separate, local "library" will be created to add the packages.

2.1 CRAN

```
install.packages("Hmisc")
```

You may be requested to choose a "mirror", scroll down the list and choose a US-based mirror for faster download.

2.2 Bioconductor

```
source("http://bioconductor.org/biocLite.R")
biocLite("affy")
biocLite("annaffy")
biocLite("annotate")
biocLite("SpikeInSubset")
biocLite("hgu95acdf")
biocLite("genefilter")
biocLite("limma")
biocLite("GenomeInfoDb")
biocLite("org.Hs.eg.db")
biocLite("hgu95av2.db")
```

3 Chapter 25 introduction

This is the last chapter of the book. The title of the chapter seemed promising. We do create a list of genes at the end, even in HTML format. However, this chapter does not consitute a "pipeline" for a general analysis.

Since the publication of the book in 2005 some `R` code already needs some adaptation. Therefore the code included in this text is "updated" to work with the current `R` version.

4 Preparation

The data used for illustration is embeded within the `R` package. They are Affymetrix data and the `affy` package needs to be loaded.

Data can be found in **CEL** (probe-level) and **CHP** (expression level: summarized probe level) and the authors assert that probe-level yields better results. Therefore whenever the `.CEL` files are available they should be preferred.

The function `AffyBatch` can be used to read all `.CEL` files contained wihin a directory without the need to specify the name of the files as all files will be read into an `AffyBatch` object.

In this chapter they use an existing `AffyBatch` object made from "spike-in" data where known amounts of mRNA have been added to the mixture. Therefore we have the advantage to "*know the truth*" of at least those mRNAs that have been "spiked in" named `spikein95`. We'll use only a ***subset*** for faster calculations thanks to the package `SpikeInSubset`.

If you need to install this package run the following code:

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

The following code will then activate this package and load the relevant data within `R`.

```
library(SpikeInSubset)
data(spikein95)
```

5 Data info

The data consists of **2 sets of triplicates** for a total of **6** arrays. They were created for a calibration experiment with Affymetrix.

The experiment results and sample information is contained within a data structure called "data frame." This "data frames" can contain information in multiple forms (text, numbers, etc.)

There are other types of information embeded as well. These can be listed with the `slotNames` command:

```
slotNames(spikein95)
```

```
 [1] "cdfName"           "nrow"                "ncol"
 [4] "assayData"         "phenoData"           "featureData"
 [7] "experimentData"    "annotation"          "protocolData"
[10] ".__classVersion__"
```

The name of the samples can be seen with:

```
sampleNames(spikein95)
```

```
[1] "1521a99hpp_av06"  "1532a99hpp_av04"  "2353a99hpp_av08"
[4] "1521b99hpp_av06"  "1532b99hpp_av04"  "2353b99hpp_av08r"
```

The name of the Affymetrix microarray that was used for the experiement can be revealed with the command:

```
annotation(spikein95)
```

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

The Affymetrix web site still offered a PDF file with information about version2 of this array:

GeneChip® Human Genome U95 Set - (archived 2013.)

(Affymetrix has been purchased by Thermofisher and the updated link is GeneChip® Human Genome U95 Set (no archive available.)

Excerpt:

The first array in the set, the Human Genome U95Av2 Array (HG-U95Av2), contains primarily full-length genes. This single array represents ~12,000 sequences previously characterized in terms of function or disease association. This array can help you identify gene associations in a particular cell type or tissue, define key players in signaling pathways, and address many other exciting experimental questions.

6 phenoData

Embeded within the data also is the *phenoData* information about the samples. This is simply a table labeling each sample based on the experimental data: which ones are controls or which treatment they received.

The ***original*** data contains information about the amount of material that was contained within the "spike-in" experiement and can be seen with the following command:

```
pData(spikein95)
```

```
                  37777_at 684_at 1597_at 38734_at 39058_at 36311_at
1521a99hpp_av06       0.00   0.25     0.5        1        2        4
1532a99hpp_av04       0.00   0.25     0.5        1        2        4
2353a99hpp_av08       0.00   0.25     0.5        1        2        4
1521b99hpp_av06       0.25   0.50     1.0        2        4        8
1532b99hpp_av04       0.25   0.50     1.0        2        4        8
2353b99hpp_av08r      0.25   0.50     1.0        2        4        8
                  36889_at 1024_at 36202_at 36085_at 40322_at 407_at
1521a99hpp_av06          8      16       32       64      128   0.00
1532a99hpp_av04          8      16       32       64      128   0.00
2353a99hpp_av08          8      16       32       64      128   0.00
1521b99hpp_av06         16      32       64      128      256   0.25
1532b99hpp_av04         16      32       64      128      256   0.25
2353b99hpp_av08r        16      32       64      128      256   0.25
                  1091_at 1708_at 33818_at 546_at
1521a99hpp_av06       512    1024      256     32
1532a99hpp_av04       512    1024      256     32
2353a99hpp_av08       512    1024      256     32
1521b99hpp_av06      1024       0      512     64
1532b99hpp_av04      1024       0      512     64
2353b99hpp_av08r     1024       0      512     64
```

However, in the book the following code was used to replace/update this information.

The content was altered in order to illustrate an example of 2 populations (2 groups) that we shall name `1` and `2` each with triplicate biological replicates that are numbered from `1` to `3`.

We start by preparing an "R object" called `pd`. The data structure of columns and the contained values are created upon assignment:

```
pd <- data.frame(population = c( 1, 1, 1, 2, 2, 2), replicate = c(1, 2, 3, 1, 2, 3))
```

We can see what this object "looks like" by printing it to screen:

```
pd
```

```
  population replicate
1          1         1
2          1         2
3          1         3
4          2         1
```

```
5          2          2
6          2          3
```

We can then change the row names of `pd` to the name of the original sample names by assigning them to the `rownames` of `pd` as such:

```r
rownames(pd) <- sampleNames(spikein95)
print(pd)
```

```
                 population replicate
1521a99hpp_av06           1         1
1532a99hpp_av04           1         2
2353a99hpp_av08           1         3
1521b99hpp_av06           2         1
1532b99hpp_av04           2         2
2353b99hpp_av08r          2         3
```

Some comments can be created and added as well. These are placed within the `v1` object that will later be combined with the `pd` object:

```r
v1 <- list(population = "1 is control, 2 is treatment", replicate = "arbitrary numbering")
```

Finally update everything together: (***DO NOT RUN this command, is obsolete and WILL NOT WORK! - Hence it is commented out !***)

```r
# do not run
# phenoData(spikein95) <- new("phenoData", pData=pd, varLabels = v1)
```

Until now it worked fine, but **if you try to run the above command you will get an error**.

Since the book was written in 2005 there has been some changes in the data structure and we need to do a little exploration to make the necessary changes.

You can read the current help by typing `?phenoData`. The help page will be titled **Retrieve information on experimental phenotypes recorded in eSet and ExpressionSet-derived classes.** (If you are given multiple help choices choose "Biobase::phenoData".)

The help page provides the name of 3 functions: `phenoData()`, `varLabels()`, `varMetadata()`, and `pData()`.

We can look at the current `phenoData` value:

```r
phenoData(spikein95)
```

```
An object of class 'AnnotatedDataFrame'
  sampleNames: 1521a99hpp_av06 1532a99hpp_av04 ...
    2353b99hpp_av08r (6 total)
  varLabels: 37777_at 684_at ... 546_at (16 total)
  varMetadata: labelDescription
```

Then we can ask what is the `class` of the various objects accessed by the functions presented in the help file, including `phenoData` itself:

```r
class(phenoData(spikein95))
```

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

```r
class(varMetadata(spikein95))
```

```
[1] "data.frame"
```

```
class(varLabels(spikein95))
```

```
[1] "character"
```

```
class(pData(spikein95))
```

```
[1] "data.frame"
```

6.1 pData

The item we need to change is `pData` that we created earlier. Its current content looks like this by default:

```
pData(spikein95)
```

```
                 37777_at 684_at 1597_at 38734_at 39058_at 36311_at
1521a99hpp_av06      0.00   0.25     0.5        1        2        4
1532a99hpp_av04      0.00   0.25     0.5        1        2        4
2353a99hpp_av08      0.00   0.25     0.5        1        2        4
1521b99hpp_av06      0.25   0.50     1.0        2        4        8
1532b99hpp_av04      0.25   0.50     1.0        2        4        8
2353b99hpp_av08r     0.25   0.50     1.0        2        4        8
                 36889_at 1024_at 36202_at 36085_at 40322_at 407_at
1521a99hpp_av06         8      16       32       64      128   0.00
1532a99hpp_av04         8      16       32       64      128   0.00
2353a99hpp_av08         8      16       32       64      128   0.00
1521b99hpp_av06        16      32       64      128      256   0.25
1532b99hpp_av04        16      32       64      128      256   0.25
2353b99hpp_av08r       16      32       64      128      256   0.25
                 1091_at 1708_at 33818_at 546_at
1521a99hpp_av06      512    1024      256     32
1532a99hpp_av04      512    1024      256     32
2353a99hpp_av08      512    1024      256     32
1521b99hpp_av06     1024       0      512     64
1532b99hpp_av04     1024       0      512     64
2353b99hpp_av08r    1024       0      512     64
```

However, according to the book we want it to look like the output of the `pd` object that we created above. We can do that by assigining the value of `pd` into that object:

```
pData(spikein95) <- pd
pData(spikein95)
```

```
                 population replicate
1521a99hpp_av06           1         1
1532a99hpp_av04           1         2
2353a99hpp_av08           1         3
1521b99hpp_av06           2         1
1532b99hpp_av04           2         2
2353b99hpp_av08r          2         3
```

This will automatically change the `varLabels` and`varMetadata` as well:

```
varLabels(spikein95)
```

```
[1] "population" "replicate"
```

```
varMetadata(spikein95)
```

```
           labelDescription
population             <NA>
replicate              <NA>
```

However, the table in `varMetadata(spikein95)` has `NA` for `labelDescription`. We entered these descriptions earlier in object `v1` and we can use that to update the descriptions. We can do it in a simple way, one element at a time since there are only 2 of them. We can specify the column thanks to the `$` sign and we can choose which one with the `[ ]` nomenclarture. With the last command we print the updated content:

```
varMetadata(spikein95)$labelDescription[1] <- v1[1]
varMetadata(spikein95)$labelDescription[2] <- v1[2]
varMetadata(spikein95)
```

```
                       labelDescription
population 1 is control, 2 is treatment
replicate            arbitrary numbering
```

7 Preprocessing

In chapter 2 of that book they go over preprocessing in detail. Nowdays the RMA method or that which also takes base composition into account (gcrma) are commonly used.

RMA provides a measure of the gene expression after 3 preprocessing steps:

1. background correction
2. quantile normalization
3. summarization based on multi-array model fit "robustly" to the median polish algorithm.

The command `rma` is a simple way to calculate these values, resulting in an "expresssion set" of values called `eset` summarizing all probe-level data into expression values in log base 2 scale. The many probe-level observations are now summarized as expression level classes.

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

```
Background correcting
Normalizing
Calculating Expression
```

The `eset` retains the same `slotNames` as the original `spikein95` object but now contains the expression values as log 2 within the `exprs` section of the data frame.

To make calculation easier, it is easier to simply create a new object, `e` containing just the expression values:

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

We can then ask the dimensions and class of these 2 objects. Object `e` is a *matrix* and a subset of `eset` which is an *ExpressionSet*.

```
dim(eset); class(eset)
```

```
Features  Samples
   12626        6
```

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

```
dim(e); class(e)
```

```
[1] 12626     6
```

```
[1] "matrix"
```

As stated in the book the `phenoData` from `spikein95` has been inherited:

```
pData(eset)
```

```
                  population replicate
1521a99hpp_av06            1         1
1532a99hpp_av04            1         2
2353a99hpp_av08            1         3
1521b99hpp_av06            2         1
1532b99hpp_av04            2         2
2353b99hpp_av08r           2         3
```

The samples have a complex name, and it may be easier for some later calculation to simlpy call them samples 1 to 3 representing "population 1" or samples 4 to 6 representing "population 2" which can be more easily accomplished if we create an index number:

```
Index1 <-  which(eset$population == 1)
Index2 <-  which(eset$population == 2)
```

These index are therefore a simple list of integer numbers:

```
print(Index1)
```

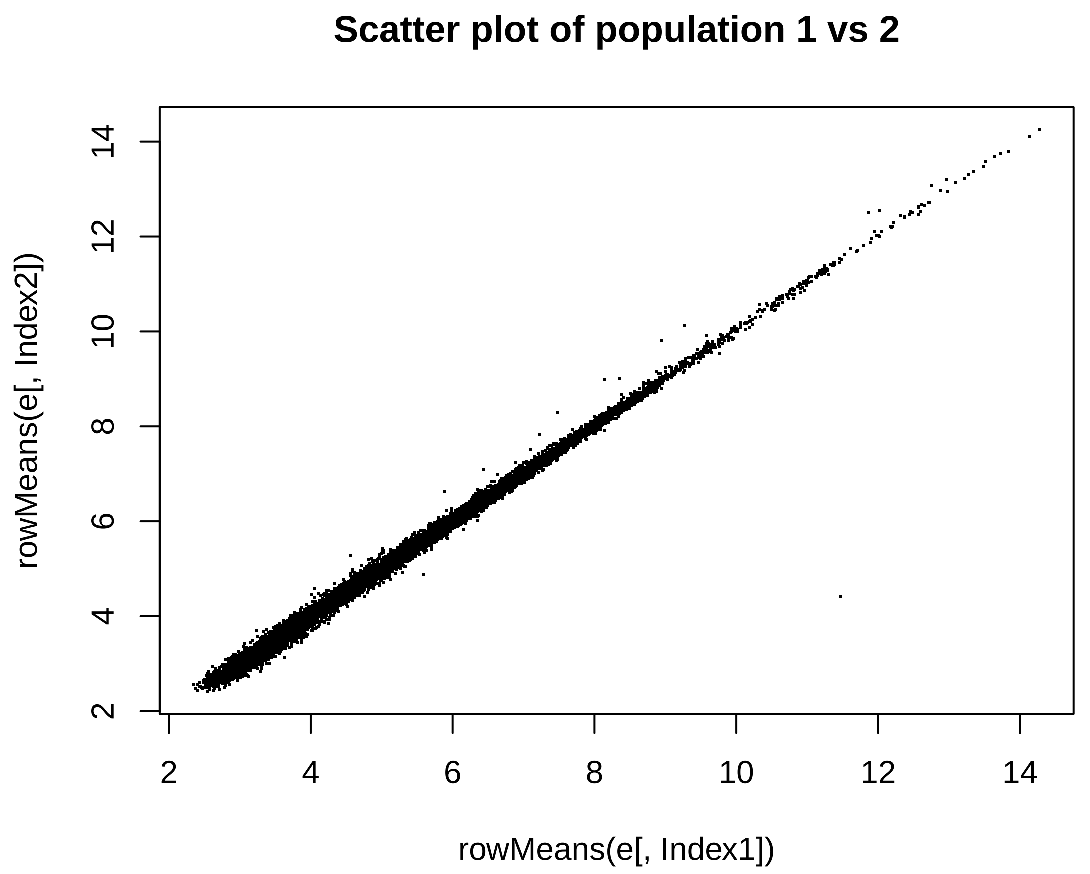
```
[1] 1 2 3
```

```
print(Index2)
```

```
[1] 4 5 6
```

One way to compare a 2 sample set is simply to plot them against each other in a scatter plot. (*Note*: This is not in the book chapter.) We can use the indexes we just created to easily select the 2 populations from within `e`:

```
plot(rowMeans(e[,Index1]), rowMeans(e[,Index2]), pch=".",  main="Scatter plot of population 1 vs 2")
```



Shortly we will see a variation of this plot, called `MAplot` that is simply a 45 degrees rotations of this plot showing the main diagonal of data rotated 45 degrees and appearing as a horizontal line within the `MAPlot`.

8 Ranking and filtering genes

Let's summarize what we have in hand:

- object `e` containing the log base2 of expression for each gene
- each value can be written as x_{ijk} for:
  - each gene j
  - on each array i
  - from a population with 2 values: $k = 1, 2$

To rank genes, it is convenient to calculate the average level of differential expression for each gene. One simple choice is the difference between "average log fold-change" for each gene in each population, *i.e.* the difference of averages:

$$d_j = \bar{x}_{j2} - \bar{x}_{j1}$$

where $\bar{x}_{j2}$ is the average over the 3 arrays for population 2 and can be calculated as $\bar{x}_{j2} = (x_{1j2} + x_{2j2} + x_{3j2})/3$ for each gene j.

Note that this is sometimes written as the *log ratio* M from a time when arrays contained both control and treated samples on the same chip but colored with a different fluorescent marker, either red (`R`) or green (`G`) with the expressed log ratio formula:

$$M = log_2(R/G) = log_2(R) - log_2(G)$$

As part of the `R base` package the function `rowMeans` can be used to calculate the mean and therefore we can calculate the differential expression for all genes at the same time place them in object `d`:

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

A useful plot is called `MA plot` and we already have M and we can easily calculate A which is the average intensity across all samples on the same row:

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

This value A can also be understood as the average intensity:

$$A = \frac{1}{2}log_2(RG) = \frac{1}{2}(log_2(R) + log_2(G))$$

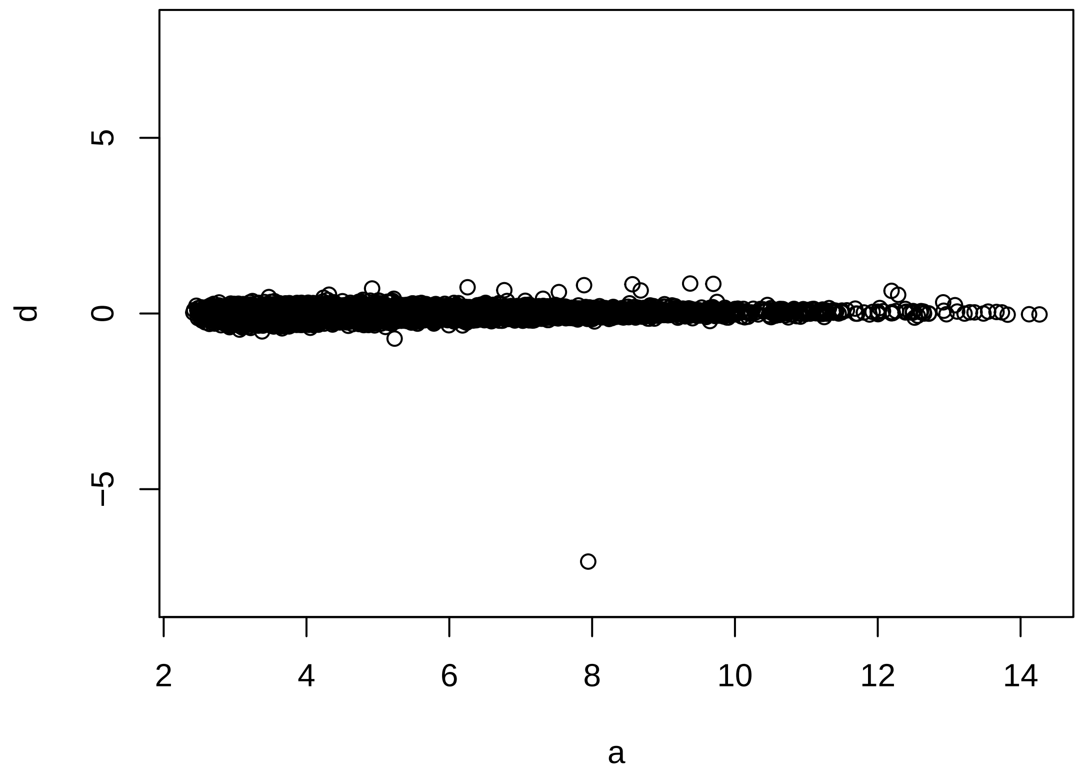
Note: Learn more about MA Plot

We could manually create this plot with this simplest of command based on objects `a` and `d` already created:

```
plot(a,d)
```

but to make it easier to see that this plot is essentially the scatter plot above rotated by 45 degrees it's better to specify the limits on the `y` axis for a better comparison:

```
plot(a,d, ylim=c(-8,8))
```



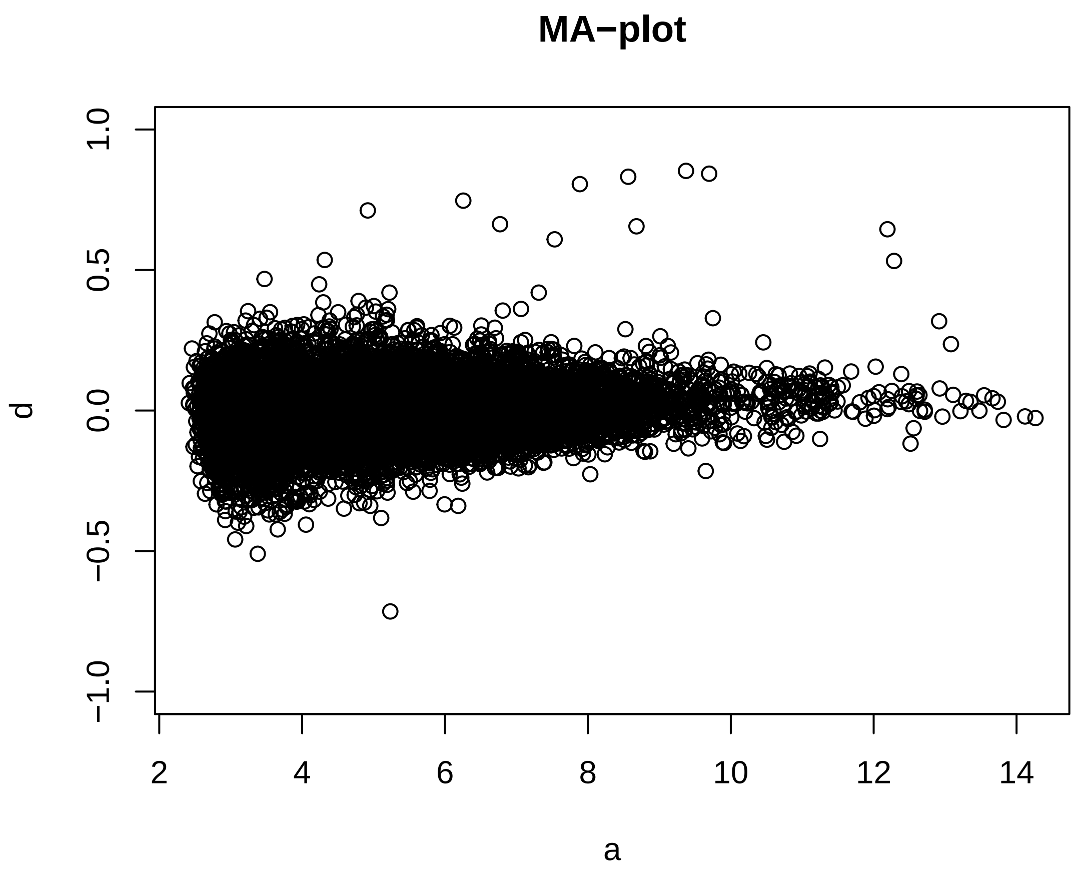
We can check how many genes have a fold change higher than 2-fold by counting absolute values above `1` since $log_2(2) = 1$:

```
sum(abs(d) > 1)
```

```
[1] 1
```

Therefore we can replot, limiting the `y` values to between `-1` and `1`. This will reproduce the plot from the book after we add the title with the `main=` command addition:

```
plot(a,d, ylim=c(-1,1), main="MA-plot")
```



9 Summary statistics and tests for ranking

In the previous MA plot the data varies more for small values of `a` and the "variance" appears smaller for larger values.

In a set of 2 groups, the *t*-statistic is a ratio comparing the difference of the **mean of each group** to that of the **variance**, *estimated as the standard error within the group.*

We can calculate this value for each gene thanks to the `rowttests` function from the package `genefilter`.

If you need to install `genefilter` then run this installation code first:

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

Then load the library and calculate the *t*-statistic and save them as `tt`. `e` is the matrix of values that we created earlier and the population factor serves to segregrate which sample is with which group

```
library("genefilter")
tt <- rowttests(e, factor(eset$population))
```

`tt` is constructed as a simple data frame with 3 columns:

```
class(tt)
```

```
[1] "data.frame"
```

```
dim(tt)
```

```
[1] 12626     3
```

```
# show the first 3 rows
tt[1:3, ]
```

```
          statistic          dm   p.value
100_g_at -0.5758147 -0.05230699 0.5955855
1000_at   0.7014592  0.07438505 0.5216808
1001_at  -0.1383839 -0.01663836 0.8966240
```

9.1 Volcano plot

Genes are routinely ranked by fold change or p-value. Both entities that the volcano plot method can show *simultaneously.* The volcano representation plots the $-log_{10}$ of p-values on the `y` axis versus the difference in mean `d` (that statisticians call "size effect") on the `x` axis.

Another way to understand the volacano plot is that it is a plot of significance ($-log_{10}$ of p-values) versus fold change.

Therefore data points with low p-values (highly significant) appear towards the top of the plot. By using the log of the fold change on the `x` axis the changes appear equidistant from the center. Therefore the 2 most important regions of the plot are:

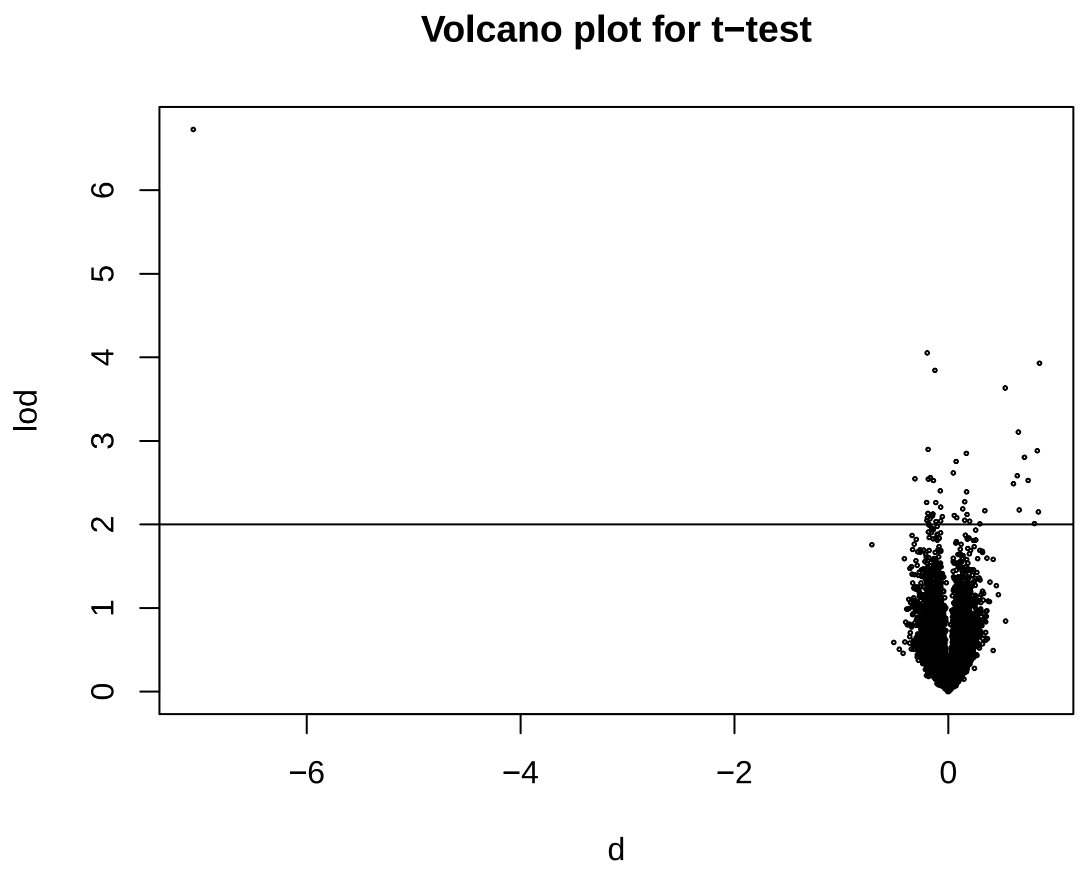
- at the top are data points that are highly significant statistically
- on both edges right and left are the points with the most fold change.

Therefore, points at the top and on the edges represent the data points with the largest magnitude of fold change with the highest statitical significance.

It is assumed that the t-statistic follows at t-distribution (Student.)

The following commands will make a simple volcano plot. The `cex = 0.25` option shows each point as 1/4 size for better clarity. The line at `2` on the `y` axis represents a "confidence" cut-off of 1%.

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



As-is this is not a very useful plot because it is shrunk on one corner. Note that there is one single dot on the top left, that correspond to the one gene we spotted earlier in the `MA plot`.

We can see that there is indeed one outlier:

```
sum(abs(lod) > 6)
```

```
[1] 1
```

The plot can be reconstructed by restricting the x axis to $[-1, 1]$ and adding some color.

First we create a set if indices corresponding to the "best 25" by ranking the t-statistic disregarding positive or negative sign using the absolute value function `abs`.

Then we replot all the dots except those indexed above thanks to the notation `[-o]` that will remove the corresponding points from the plot.

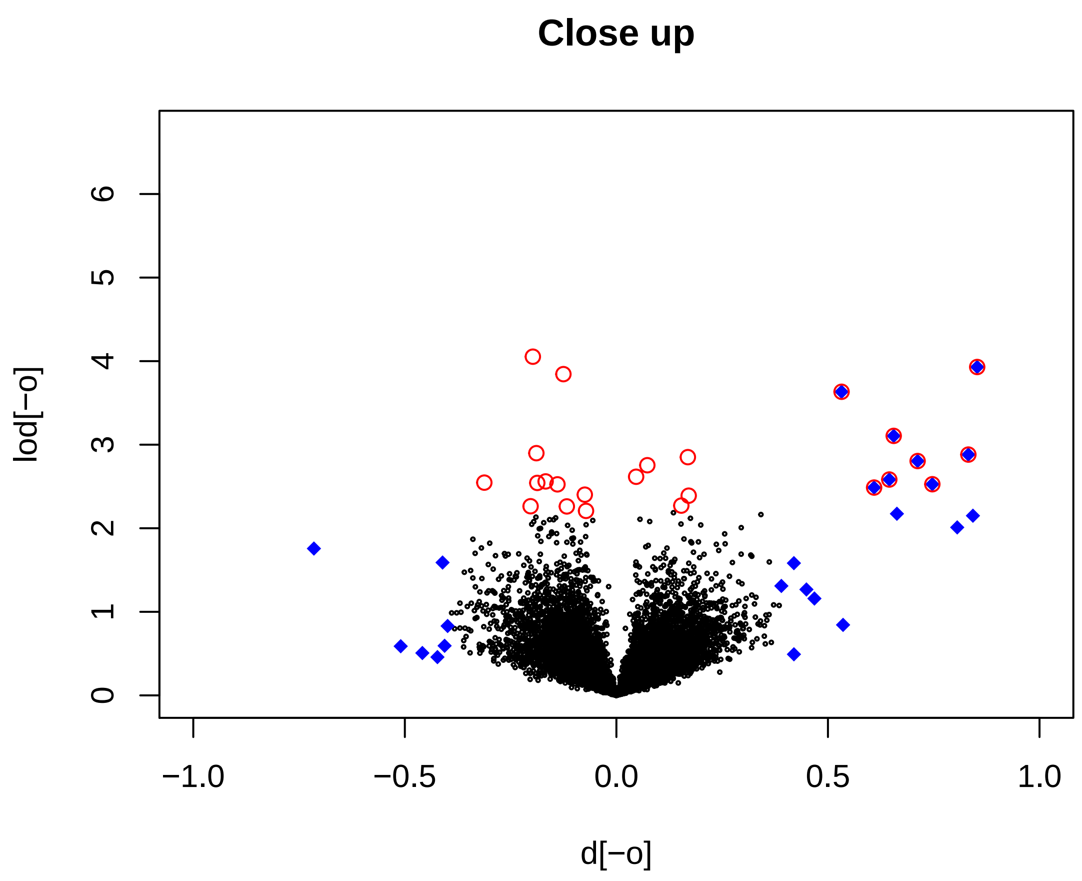
We can then specifically select the genes corresponding to the index and plot them with specific colors and shapes.

The `o1` and `o2` indexes are for the first 25 in the decreasing ordered list of `d` (the distance between means that we calculated and plotted on the `x` axis) - These will represent the 25 with the largest fold change. The selected 25 "best" `tt$statistic` correspond to those with the highest significance.

The object `o` contains all of `o1` and `o2` and serves to remove those points from the main, black and white points plot. Then, the `o1` group for the 25 genes with the largest fold-change are added as blue losenges (`pch = 18`) and the most statistically sgnificant genes are shown as red, unfilled circles (`pch = 1`.)

If the commands are given one at a time it is easier to see the plot being constructed, rather than copy/paste all the commands at once!

```
# create indexes
o1 <- order(abs(d), decreasing = TRUE)[1:25]
o2 <- order(abs(tt$statistic), decreasing = TRUE)[1:25]
o <- union(o1, o2)
# redo the plot
plot(d[-o], lod[-o], cex =  0.25, xlim = c(-1, 1), ylim = range(lod), main = "Close up")
# add points to the plot
points(d[o1], lod[o1], pch = 18, col = "blue")
points(d[o2], lod[o2], pch = 1, col = "red")
```



This volcano plot clearly shows that some of the most statistically significant genes do not have a high fold change associated with them (red circles), while some genes with a high fold-change do not appear within the list of the 25 most significants (blue losanges not circled by red dots.)

Explanations proposed by the author to explain this counter-intuitive finding:

1) Some genes have larger variance than others. Large variance genes that are not differentially expressed have a higher chance of having large log fold changes. Because the *t*-statistics take variance into account, these do not have small *p*-values.
2) With only three measurements per group, the estimate of the standard error of the effect size is not stable and some gens have small *p*-values only by chance, the denominator of the *t*-statistic was very small.

The plot demonstrates that both of these explanations are possible.

9.2 *moderated t*-statistic

The Student t-test is used to compare the means of gene expression values within 2 groups for a given gene and divides that difference `d` by the *estimated* variance `s` calculated from the data available for each gene. Thus we could represent the t-test as:

$$t = \frac{d}{s}$$

The moderated t-test calculates the variance based on information for all genes or exons in the considered group (Smyth 2004) as is also explained in chapter 23 of the book. (Archived July 3, 2016: http://bit.ly/2qnCHWq)

There are many versions of *modified* t-statistics calculated by *borrowing strength accross all genes.*

Thus, rather than estimating a "within-group" variability for the denominator of t-test over and over again for each gene, we pool the information for many similar genes. This has the advantage to eliminate the occurence of accidental large t-statistics to accidentally small "within group"" variance.

The typical approach is:

- Calculate an "overall" *estimate* of the variance, s_0^2
- then for each gene, an *estimate* of the per gene variance, s_g^2 is computed
- the final variance used is a weighted average of s_0^2 and s_g^2

One implementation of the *moderated t*-test is within the `limma` package (Smyth 2004) based on Baysian statistics and fitting of data to a linear model (`lmFit` function.)

The `design` is simply a way to group which samples are together as a group and compared to which other samples in a table.

We encode this experimental design in `R` with two pieces. We start with a formula with the tilde symbol `~`. This means that we want to model the observations using the variables to the right of the tilde. Then we put the name of a variable, which tells us which samples are in which group. (cited from:Expressing design formula in R. Archived March 11, 2016 http://bit.ly/2qOhOne)

The designation of groups are evaluated with the `factor` command.

```
library("limma")
# create a design matrix based on the sample attributes
design <- model.matrix(~factor(eset$population))
# print the design matrix
design
```

```
  (Intercept) factor(eset$population)2
1           1                        0
2           1                        0
3           1                        0
4           1                        1
5           1                        1
6           1                        1
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$`factor(eset$population)`
[1] "contr.treatment"
```

```r
# fit data to the linear model based on the design matrix
fit <- lmFit(eset, design)
# apply baysian statistics
ebayes <- eBayes(fit)
```

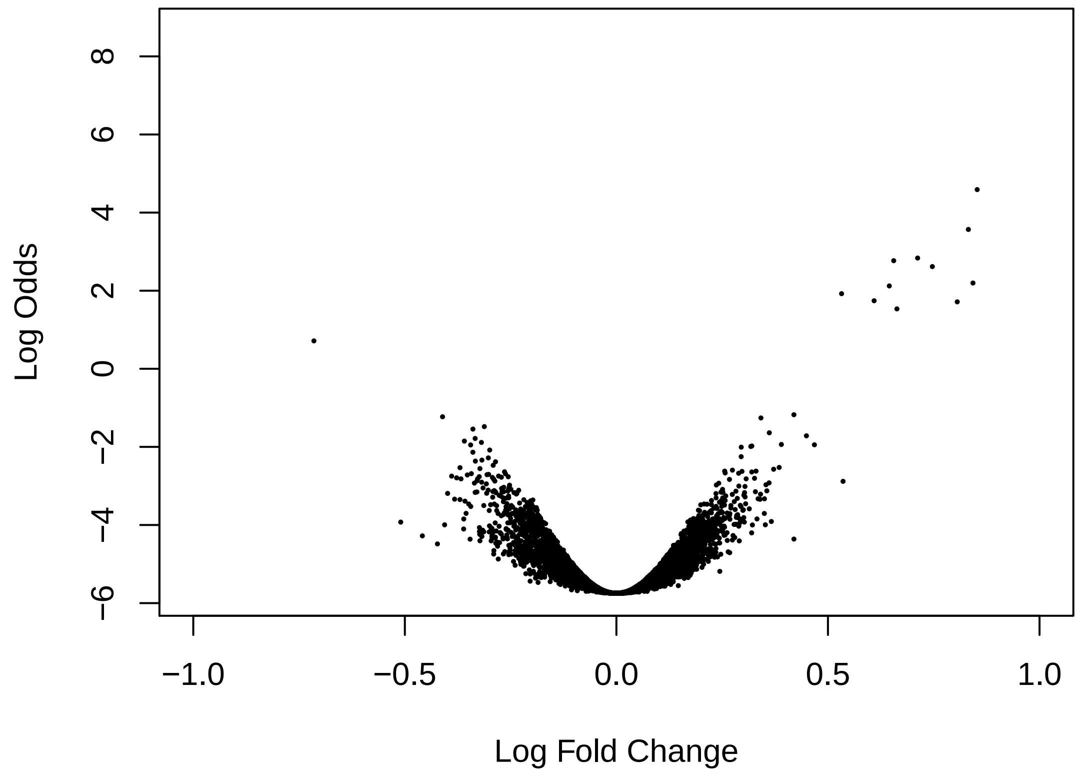
The information contained within the `ebayes` object can be listed:

```r
names(ebayes)
```

```
 [1] "coefficients"     "rank"             "assign"
 [4] "qr"               "df.residual"      "sigma"
 [7] "cov.coefficients" "stdev.unscaled"   "pivot"
[10] "Amean"            "method"           "design"
[13] "df.prior"         "s2.prior"         "var.prior"
[16] "proportion"       "s2.post"          "t"
[19] "df.total"         "p.value"          "lods"
[22] "F"                "F.p.value"
```

A volcano plot can be readily created with the `limma` package command `volcanoplot`

```r
volcanoplot(ebayes, coef=2, xlim =c(-1,1))
```



However, in order to recreate the figure that is shown in the book we need to go through more details.

9.2.1 topTable

The `limma` package command `topTable` will create a list of genes based on the expression values and the `ebayes` statistical and model fitting analysis showing various criteria, including log fold change, which we need as one of the components of a manually created volcano plot. Another column lists the *p*-values that we need as well:

By default the command will only list 10 genes. We can save the data for plotting purposes for all 12,626 genes, saved in an object called `tableTop` (below.)

```
# default run outputs 10 genes - Note the name of the columns:
topTable(ebayes,coef=2)
```

```
              logFC   AveExpr          t      P.Value    adj.P.Val
1708_at  -7.0610986  7.945259 -73.534434 7.836387e-17 9.894222e-13
36202_at  0.8527063  9.373044   9.977745 4.926641e-07 3.110189e-03
36311_at  0.8318614  8.564331   8.363671 3.017747e-06 1.270069e-02
33264_at  0.7119117  4.918949   7.435889 9.659867e-06 2.707993e-02
32660_at  0.6554271  8.680129   7.356267 1.072388e-05 2.707993e-02
38734_at  0.7467434  6.255754   7.184931 1.346297e-05 2.833057e-02
1024_at   0.8426738  9.697298   6.730856 2.504174e-05 4.400625e-02
36085_at  0.6449738 12.193127   6.654136 2.788294e-05 4.400625e-02
33818_at  0.5321975 12.285668   6.455000 3.698776e-05 5.188972e-02
39058_at  0.6090804  7.534582   6.278861 4.770043e-05 5.695439e-02
                B
1708_at  8.645816
36202_at 4.588592
36311_at 3.567609
33264_at 2.836285
32660_at 2.767813
38734_at 2.617210
1024_at  2.195766
36085_at 2.121250
33818_at 1.923223
39058_at 1.742416
```

```
# save all into an object:
tableTop <- topTable(ebayes,coef=2, number = 12626)
```

Now we can use the same method as before to index the 25 "best" genes for fold change and *p*-values. We can create the following indices:

```
# p1 : index of the 25 with most fold change, regardless of up or down
p1 <- order(abs(tableTop$logFC), decreasing= TRUE)[1:25]
length(p1)
```

```
[1] 25
```

```
# p2: index for the 25 with smallest P.Val
p2 <- order(abs(tableTop$P.Val), decreasing = FALSE)[1:25]
length(p2)
```

```
[1] 25
```

```
# union of p1 and p2. Some are not the same hence length is more than 25
p <- union(p1,p2)
length(p)
```

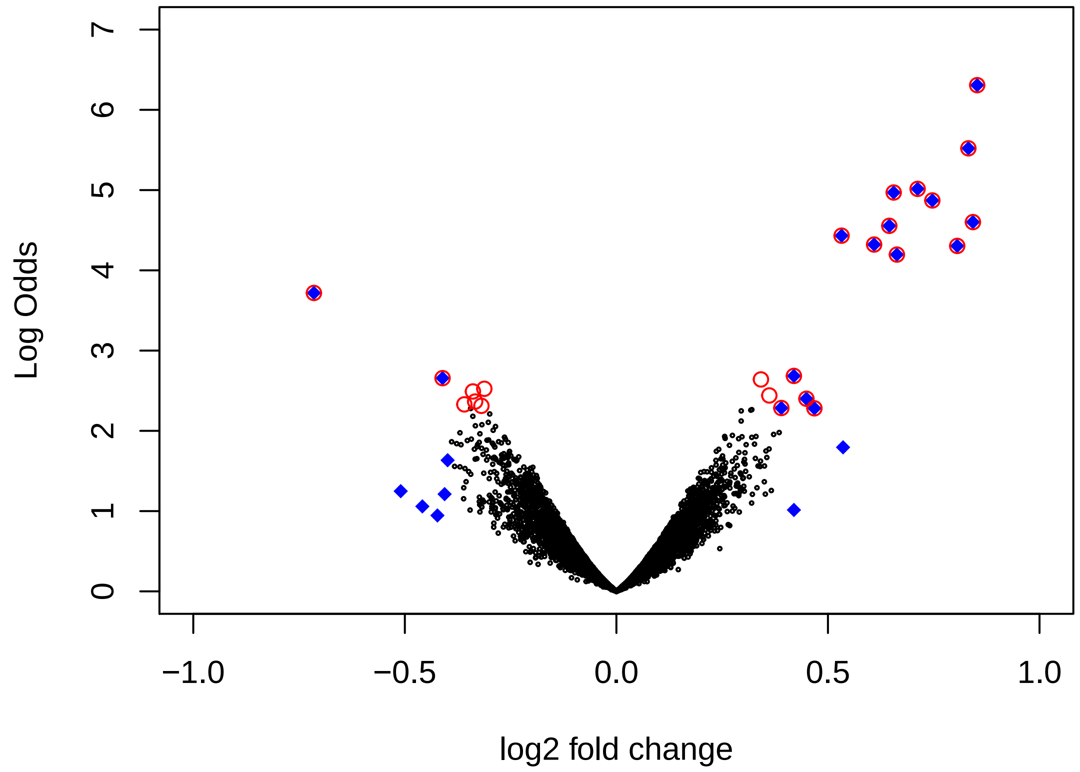
```
[1] 32
```

We could see the values for the "selected" points by using the following code:

```
tableTop[p1,]
tableTop[p2,]
```

We can then "manually" create the volcano plot, by first plotting points except those we want to recolor later, so we remove them with `[-p]`.

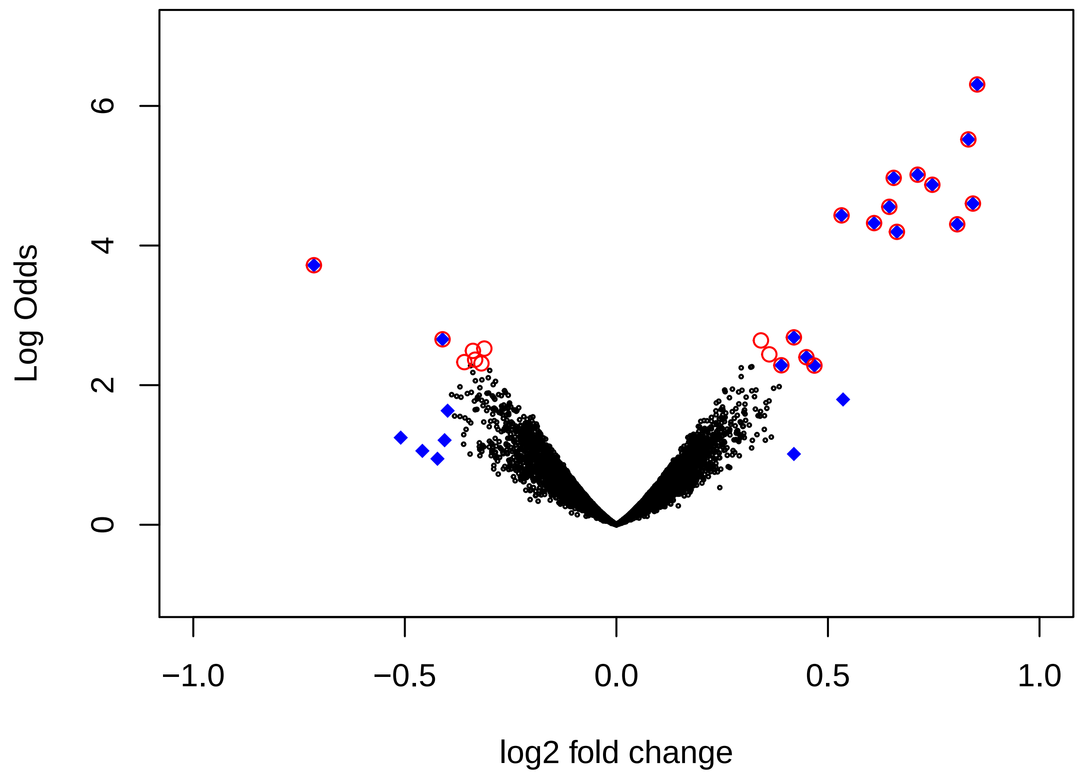
```
plot(tableTop$logFC[-p], -log10(tableTop$P.Val[-p]), cex =0.25, xlim = c(-1, 1), ylim = c(0, 7), xlab =
# Then add the points with the same colors as before:
points(tableTop$logFC[p1], -log10(tableTop$P.Val)[p1], pch = 18, col = "blue")
points(tableTop$logFC[p2], -log10(tableTop$P.Val)[p2], pch = 1, col = "red")
```



The code could be made more readble with the following modifications using alternate variable names:

```
lfc <- tableTop$logFC
lod2 <- -log10(tableTop$P.Val)
# which would make the "plot" and "points" commands easier to read:
plot(lfc[-p], lod2[-p], cex = 0.25, xlim = c(-1, 1), ylim = range(lod2)/2-1, xlab = "log2 fold change",

# add the points
points(lfc[p1], lod2[p1], pch = 18, col = "blue")
points(lfc[p2], lod2[p2], pch = 1, col = "red")
```



10 Selecting cutoffs

The author asserts that now we know how to rank genes based on three statistics.

Thinking back we can realize that these are:

- t-statistic
- moderated t-statistic
- fold change

The question posed now is: how can we find the genes of interest based on the ranking and where do we "draw the line" to cut off that list?

Should we use the "typical" p-value cut-off of `0.05` or `0.01` often used in statistic?

10.1 Multiple testing

One issue with the data generated from microarrays, and Next Gen sequencing, is the fact that we are observing many things at the same time. In other words, when we performed a t-test for a gene, this was in the context of the complete set, and we end-up caclculating 12626 t-tests since there are 12626 genes represented on the array.

In this context the p-values no longer have the typical meaning.

In simple terms, the *null hypothesis* for a gene is that it's *not differentially expressed*, the *alternate hypotheis* is that it *is differentially expressed*. More specific hypotheses could be formulated for an up-regulation or a down-regulation. But to keep it simple we can for now use the simplest idea:

- H_0^g: null hypothese that gene `g` is *not* differentially expressed
- H_1^g: alternate hypothese that gene `g` *is* differentially expressed

If we assume that none of the genes are differentially expressed at a confidence level of `0.01` we would expect the following number of *false positives* in our list: `0.01 x 12626 =  126.26`.

We can see how many genes have a p-value lower than `0.01` in our dataset:

```
sum(tt$p.value <= 0.01)
```

```
[1] 46
```

In other words, we have less than half of the number of just the false positives! Therefore our data may not be statistically significant?

10.2 FDR: False Discovery Rate

Satistic reminder: Type I and Type II Errors

Decision	H_0 is True	H_0 is False
Do not reject H_0	Correct Decision $1-\alpha$	Incorrect Decision: **Type II** Error β
Reject H_0	Incorrect Decision: **Type I Error** α	Correct Decision $1-\beta$

- α is the probability of having Type I Error
- β is the probability of having Type II Error

There are different approaches to control making a Type I Error. The most commonly used now is that of "False Discovery Rate" (FDR) designed to control the proportion of false positives among the rejected hypotheses.

FDR is an option that can be selected when saving a list of gene with the `topTable` command from the `limma` package when we invoke `adjust = "fdr"` as we shall do in the next section.

11 Annotation; generating a report

We have used the `topTable` command above to same the data for all genes. The default adjustment of p-values is that of `fdr` (also called `BH`) and therefore we have already saved this data into `R` object `tableTop` above.

We can review the first 5 genes in the list:

```
tableTop[1:5, ]
```

```
              logFC  AveExpr          t      P.Value    adj.P.Val        B
1708_at  -7.0610986 7.945259 -73.534434 7.836387e-17 9.894222e-13 8.645816
36202_at  0.8527063 9.373044   9.977745 4.926641e-07 3.110189e-03 4.588592
36311_at  0.8318614 8.564331   8.363671 3.017747e-06 1.270069e-02 3.567609
33264_at  0.7119117 4.918949   7.435889 9.659867e-06 2.707993e-02 2.836285
32660_at  0.6554271 8.680129   7.356267 1.072388e-05 2.707993e-02 2.767813
```

Note: the `topTable` command results now have different column headers than were represented in the book but the data remains the same. What was called `M` in the book is now reported as `logFC` and what was reported as `A` in the book is now labeled `AveExpr`.

11.1 Annotation

The name of genes in the table are in annotation form from Affymetrix coding, and it would be usefull to obtain more information about the gene to make the table more informative.

We can create a variable containing the names in this form:

```
genenames <- rownames(tableTop)
length(genenames)
```

```
[1] 12626
```

We have already seen before what array annotation is used, as it is embeded within the dataset:

```
annotation(eset)
```

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

Bioconductor hosts the annotation package for both HG-U95A and HG-U95Av2 in a package that used to be called `hgu95av2` but is now called `hgu95av2.db` that we may need to install before proceeding further.

If you need to install this pacakge use the following commands:

```
source("http://bioconductor.org/biocLite.R")
biocLite("hgu95av2.db")
```

We will also use the `annotate` package

We can obtain the ENTREZ gene ID and the symbol for each gene: (Note: the `getLL` command used in the book now issues the remark: `use getEC`.)

```
library("hgu95av2.db")
library("annotate")
geneID <- getEG(genenames, "hgu95av2.db")
sym <- getSYMBOL(genenames, "hgu95av2.db" )
```

We can peek into the values obtained:

```
geneID[1:5]
```

```
 1708_at 36202_at 36311_at 33264_at 32660_at
  "5602"   "5569"   "5136"  "55556"   "9881"
```

```
sym[1:5]
```

```
 1708_at 36202_at 36311_at 33264_at 32660_at
"MAPK10"   "PKIA"  "PDE1A" "ENOSF1" "TRANK1"
```

We can add these 2 columns as first columns, one at a time to the existing `tableTop` list:

```
T1 <- data.frame(sym, tableTop)
T2 <- data.frame(geneID, T1)
T2[1:5, ]
```

```
         geneID    sym      logFC  AveExpr          t      P.Value
1708_at    5602 MAPK10 -7.0610986 7.945259 -73.534434 7.836387e-17
36202_at   5569   PKIA  0.8527063 9.373044   9.977745 4.926641e-07
```

```
36311_at   5136  PDE1A  0.8318614 8.564331   8.363671 3.017747e-06
33264_at  55556 ENOSF1  0.7119117 4.918949   7.435889 9.659867e-06
32660_at   9881 TRANK1  0.6554271 8.680129   7.356267 1.072388e-05
            adj.P.Val        B
1708_at  9.894222e-13 8.645816
36202_at 3.110189e-03 4.588592
36311_at 1.270069e-02 3.567609
33264_at 2.707993e-02 2.836285
32660_at 2.707993e-02 2.767813
```

11.2 Saving into a text file

We could save the file into a plain, tab- or comma-delimited text file containing all results with the `write.table` or the `write.csv` command:

```
write.csv(T2, file = "T2.csv")
```

Such file can then be manipulated further with a spreadsheet program.

11.3 Saving into an HTML file.

It is also possible to save the table in HTML form. This format is best used when saving a fraction of the genes and not the complete gene list of the array as the file would become too large. However, for small (up to a few 100's) number of genes this may be a practical solution.

The command `htmlpage` from the `annotate` package can write HTML pages with gene results and in addition create clickable links connecting directly to the relevant database.

For that, the command requires a "gene list" for which data will be linked, and another list called `ohternames` for which data will be shown only as plain text.

For *p*-values, the problem with the transfer to HTML is that most low values will only be reported as `0.00`.

As an example let's create an HTML table for the first 25 genes in the `T2` list. We can call this subset *e.g.* `subT2`.

We then need to distinguish between the column that will be linked to the columns that will remain plain text (`othernames`.) The `geneID` column is the one that will be linked. It is the first column of data that we can write as `subT2[1]`. Conveniently we can write all of the other columns to the `othernames` variable with `subT2[-1]` but a more explicit writing could be required.

We also need to supply names for the column headers as part of a required argument.

```
# subset T2:
subT2 <- T2[1:25, ]
htmlpage(subT2[1], filename = "report25.html", title = "HTML report for 25", othernames = subT2[-1], tal
```

The first column of the resulting html page will contain links to the "gene" database at NCBI linked to the "Gene ID" by default.

Given a list of genes with more annotations columns it would be possible to create links to other databases by modifying the command if appropriate columns of data were available and specifying repositories.

Asking help for `htmlpage` reveals:

Argument	description
`repository`	A list of repositories to use for creating the hypertext links. Currently available repositories include 'gb' (GenBank), 'en' (EntrezGene), 'omim' (Online Mendelian Inheritance in Man), 'sp' (SwissProt), 'affy' (Affymetrix), 'ug' (UniGene), 'fb' (FlyBase), 'go' (Gene Ontology), 'ens' (Ensembl).

11.4 Adding more annotation

In fact there are many other annotations that could be added and the choice of what to include may depend on the final need for the table.

To see and access all the annotation available we can ask the help of the `annaffy` package.

If you need to install this package use the following code:

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

The command `keytypes` from the `annaffy` package will list the available annoations within the chip description file.

```
keytypes(hgu95av2.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"
 [5] "ENSEMBLTRANS" "ENTREZID"     "ENZYME"       "EVIDENCE"
 [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"
[13] "IPI"          "MAP"          "OMIM"         "ONTOLOGY"
[17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
[21] "PROBEID"      "PROSITE"      "REFSEQ"       "SYMBOL"
[25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

Therefore it would be possible to augment the number of columns of the gene output, *e.g.* `T2` with more informative columns. Saving this into a plain text file may be more suitable for the complete list of genes, and subset of gene list, e.g. Best 500, may be best saved into an HTML file.

There are specific steps to take for cases where more than one entry code can be found, which is often the case of RefSeq.

11.5 Making "Pretty" output

The following vignette from the `annotate` package can be followed for more elaborate HTML constructions:

HowTo: get pretty HTML output for my gene list

11.6 Opening an HTML file from within R

The question was posed on stackoverflow.com as:

"How to open a local html file from `R` in an operating system independent way?"

The answer provided is to create a simple function, assuming the current directory obtaied by `getwd`:

```
# create a function to open an HTML file"
openHTML <- function(x) browseURL(paste0('file://', file.path(getwd(), x)))
```

```r
# Open the "report25.html" created earlier. THis will open your DEFAULT browser:
openHTML("report25.html")
```

12 Session information

```r
date()
```

```
[1] "Tue May  2 15:36:21 2017"
```

```r
sessionInfo()
```

```
R version 3.3.3 (2017-03-06)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X El Capitan 10.11.6

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] annotate_1.52.1      XML_3.98-1.6         hgu95av2.db_3.2.3
 [4] org.Hs.eg.db_3.4.0   RSQLite_1.1-2        DBI_0.6-1
 [7] AnnotationDbi_1.36.2 IRanges_2.8.2        S4Vectors_0.12.2
[10] limma_3.30.13        genefilter_1.56.0    hgu95acdf_2.18.0
[13] SpikeInSubset_1.14.0 affy_1.52.0          Biobase_2.34.0
[16] BiocGenerics_0.20.0  knitr_1.15.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.10          BiocInstaller_1.24.0  bitops_1.0-6
 [4] tools_3.3.3           zlibbioc_1.20.0       digest_0.6.12
 [7] evaluate_0.10         memoise_1.1.0         preprocessCore_1.36.0
[10] lattice_0.20-35       Matrix_1.2-8          yaml_2.1.14
[13] stringr_1.2.0         rprojroot_1.2         grid_3.3.3
[16] survival_2.41-3       rmarkdown_1.5         magrittr_1.5
[19] codetools_0.2-15      backports_1.0.5       htmltools_0.3.6
[22] splines_3.3.3         xtable_1.8-2          stringi_1.1.5
[25] RCurl_1.95-4.8        affyio_1.44.0
```

References

Gentleman, Robert, Vincent Carey, Wolfgang Huber, Rafael Irizarry, and Sandrine Dudoit. 2005. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor (Statistics for Biology and Health)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Smyth, G. K. 2004. "Linear models and empirical bayes methods for assessing differential expression in microarray experiments." *Stat Appl Genet Mol Biol* 3: Article3.