Lab: Annotation and meta-data

Robert Gentleman

April 14, 2006

Introduction

In this lab we will see how to use different data packages to provide meaning to our analyses and how to generate hyper-linked output. The basic premise is that we have obtained a list of genes (probes) that are of interest and we will use the available meta-data to better interpret them.

Our data

First load the *Biobase* package and then the data set `ALL`.

```
> library("Biobase")

Welcome to Bioconductor

        Vignettes contain introductory material.
        To view, simply type 'openVignette()' or start with 'help(Biobase)'.
        For details on reading vignettes, see the openVignette help page.

> library("ALL")
> library("hgu95av2")
> library("annotate")
> data(ALL)
```

The `ALL` data set has 128 samples. We will consider only a smaller subset. Our goal will be to compare those with $ALL1/AF4$ to those with BCR/ABL, these two phenotypes arise due to two different translocations. The first between chromosomes 4 and 11 and the second between chromosomes 9 and 22. This leaves us with 47 cases.

```
> ALLs1 = ALL[, ALL$mol.biol == "ALL1/AF4" | ALL$mol.biol == "BCR/ABL"]
```

Next we will do some non-specific filtering to remove genes that do not show much variation in expression levels across samples. We have set a rather arbitrary requirement on the level of the IQR. You might want to experiment with that and see if the output changes very much. You could also filter on other criteria, but we have found that filtering for variability seems to be a reasonable criteria for reducing the set of genes to a reasonable number.

```
> library(genefilter)
```

```
Loading required package: survival
Loading required package: splines
```

```
> f1 <- function(x) (IQR(x) > 0.8)
> ff <- filterfun(f1)
> selected <- genefilter(ALLs1, ff)
> sum(selected)
```

```
[1] 1697
```

```
> ALLs2 <- ALLs1[selected, ]
```

So we will now analyze these data. Our first step is to use *multtest* to carry out a two group comparison. But we note that many other options are available, but our interest here is to get a sensible gene list and to subsequently use that to demonstrate the use of the different meta-data packages. This is also why we set the parameter B in the code below to be so low.

```
> library(multtest)
> cl <- as.numeric(ALLs2$mol == "BCR/ABL")
> resT <- mt.maxT(exprs(ALLs2), classlabel = cl, B = 1000)
> ord <- order(resT$index)
> rawp <- resT$rawp[ord]
> names(rawp) <- geneNames(ALLs2)
> sum(resT$adjp < 0.05)
```

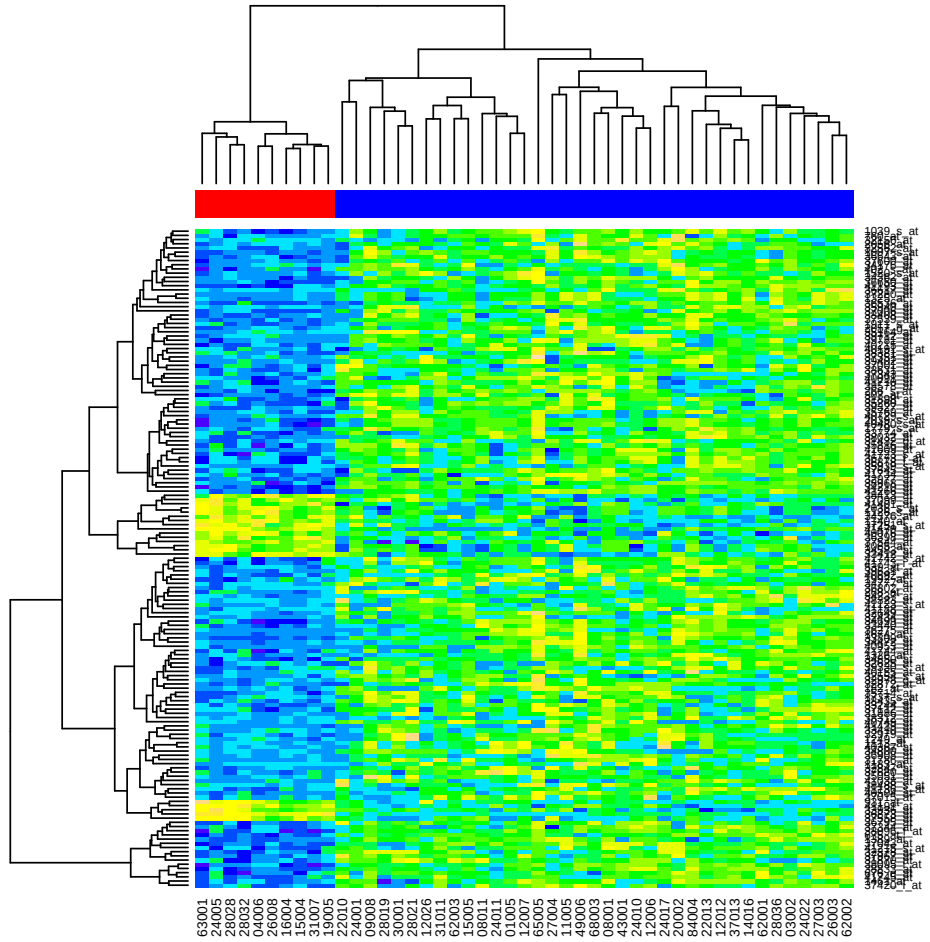
We see that there are a number of genes with different expression values in these two subsets (157). Our goal now is to see if we can provide some interpretation for the genes that were selected.

```
> ALLs3 = ALLs2[resT$index[resT$adjp < 0.05], ]
> myLLs = unlist(mget(geneNames(ALLs3), hgu95av2LOCUSID))
> sum(duplicated(myLLs))
```

```
[1] 20
```

Let's look at a heatmap to get some idea of how well these genes separate the two groups.

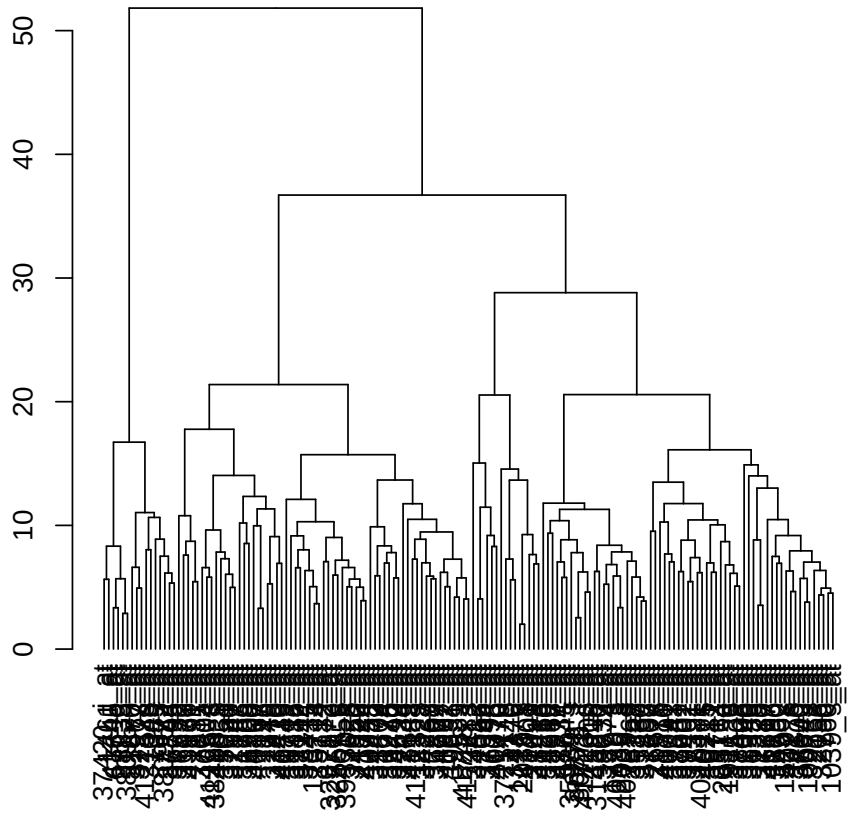
```
> hm1 = heatmap(exprs(ALLs3), ColSide = ifelse(ALLs3$mol == "ALL1/AF4",
+     "red", "blue"), col = topo.colors(15), keep.dendro = TRUE)
```



Now we have asked to keep the row and column dendrograms so that we can try to make some sense out of the genes that we have found. We see that there are roughly five interesting groups of genes.

So we want to cut the dendrogram into five pieces. The easiest way to decide where to cut the dendrogram is to plot it separately.

```
> plot(hm1$Rowv)
```



We then see that it is fairly easy to split this dendrogram into four pieces by cutting it at the level of about 25.

```
> cdend = cut(hm1$Rowv, h = 25)
> dendlabs = lapply(cdend$lower, labels)
> dendlabs[[1]]
```

```
 [1] "37420_i_at" "1461_at"    "41745_at"   "676_g_at"   "38833_at"
 [6] "38095_i_at" "31870_at"   "37967_at"   "32378_at"   "41215_s_at"
[11] "37043_at"   "1389_at"    "675_at"     "38096_f_at" "37344_at"
[16] "36795_at"
```

We have now extracted the dendrogram labels, for these four groups.

Exercise 1

Use some of the basic functionality in the annaffy package to produce HTML output describing the genes in each of these groups separately. What sorts of things can you find out about them? You may want to concentrate on just one of the four lists and do a more in-depth study.

Exercise 2
Optional: Identify the small group of genes that are obvious in the plot, but which we did not extract. Can you say anything about them?

We might also be interested in whether the genes within one of these groups show some sort of correlated expression. You could start by computing pairwise distances (or correlations) between the genes. It might also make some sense to examine pairwise scatter plots (for groups of genes that are relatively small. A last consideration is to examine multi-dimensional scaling (on the genes). Our goal is to see whether or not we can reduce the number of genes in each set.

Multiple probe sets per gene

The annotation package *hgu95av2* provides information about the genes represented on the array, including LocusLink identifiers (http://www.ncbi.nlm.nih.gov/EntrezGene), Unigene cluster identifiers, gene names, chromosomal location, Gene Ontology classification, and pathway associations. While the term *gene* has many aspects and can mean different things to different people, we operationalize it by identifying it with entries in the LocusLink database. One problem that does arise is that some genes are represented by multiple probe sets on the chip. The multiplicities for the HGU95AV2 chip are shown in the following table.

Multiplicity	1	2	3	4	5	6	7	8
No. LocusLink IDs	6872	1553	505	106	21	14	7	6

This leads to a number of complications, as we discuss in the following. Of the 2212 LocusLink IDs that have more than one probe set annotated at them, we found that in 86 cases our nonspecific filtering step of Section **??** selected some, but not all corresponding probe sets.

Exercise 3
Select some pairs of duplicated (or triplicated) probe sets and plot the expression values against each other. Compute the correlations between all duplicated (I suggest only those the `slen1=2`*) probe sets and draw a histogram.*

Categories and over-representation

A bit later in this exercise we will consider using data from the Gene Ontology (GO) to try to make some sense out of the selected genes. In this section we first address a more general question. Suppose that you can divide your data into k groups (for example k might be 24 and represent chromosomes in humans).
We can also consider the approach we took above as dividing the genes into two groups; those that are interesting (have low p-values) and those that are not. Using these two categorizations simultaneously gives us a two-way table. There are many different tests for association in two way tables, Pearson's χ^2 test, *chisq.test* or Fisher's exact test, *fisher.test*,

among them. Before trying the next two exercises read to the end of this section as some help is given on obtaining the different quantities you need.

Exercise 4
Use these two tests, chisq.test and fisher.test, to determine if there is an association between being selected and being on a particular chromosome. If so, explain which chromosomes seem to be involved.
Be careful, what does it mean to be selected? What is your reference population? Make sure that you are clear about what hypothesis you are testing.
How will you deal with Entrez gene IDs that have multiple probe sets mapped to them?

An alternate approach that seems to be widely used, but is generally inappropriate is to make each comparison separately. This is usually done using a Hypergeometric sampling paradigm (genes are either on a specific chromosome or not, and they are either interesting or not), but that is exactly the same thing as using Fisher's test for the corresponding two-way table.

Exercise 5
Use either a Hypergeometric calculation or Fisher's exact test to consider each chromosome separately. How do these results compare with those found above. Report the per chromosome summary statistics. If any p-values are significant, explain whether there are more, or fewer genes than you would expect by chance?
Use the same criteria, as you used for the preceding exercise to determine which comparisons to make.

For any chromosome, the first thing that we need to do is to compute all genes that map to the chromosome. Next we need to count the number in our data set that also mapped to the chromosome. And those two numbers, together with the number of unique LocusLink IDs form the basis for our Hypergeometric calculation. We carry this out for Chromosome 1.

```
> chrs = as.list(hgu95av2CHR)
> table(sapply(chrs, length))

    1
12625

> chr1 = sapply(chrs, function(x) x[1])
> table(chr1)

chr1
   1   10   11   12   13   14   15   16   17   18   19    2   20   21   22    3
1234  444  695  695  224  416  353  503  714  172  745  812  314  145  358  664
   4    5    6    7    8    9   Un    X  X|Y    Y
 453  534  718  583  421  423    1  484   15   25

> onC1 = (chr1 == "1")
> onC1[is.na(onC1)] = FALSE
> sum(onC1)
```

```
[1] 1234
```

```
> lls = unlist(as.list(hgu95av2LOCUSID))
> badll = duplicated(lls)
> badllnames = names(badll)
> onC1unique = onC1 & !badll
> myLLunique = !duplicated(unlist(mget(geneNames(ALLs3), hgu95av2LOCUSID)))
> myCr = unlist(mget(geneNames(ALLs3), hgu95av2CHR))
> myC1 = (myCr == "1")
> myC1[is.na(myC1)] = FALSE
> myC1unique = myC1 & myLLunique
```

So now we have a Hypergeometric distribution with $x = 13$, $m = 945$, $n = 9085$, and $k = 137$. We want to compute the probability that x is as large, or larger than, the observed x.
In most cases our interest will not really focus on whole chromosomes but rather on particular regions of chromosomes. Consider how to make use of the approach discussed here using cytochrome band information.

Working with GO

The package *GOstats* has some of the necessary functionality built in. In particular the function *GOHyperG* will compute the Hypergeometric p-values for over-representation of genes at all GO terms in the induced GO graph.
The induced GO graph is the GO graph that results from taking the union of the most extreme set of GO terms for each selected gene and then including all less specific terms that are joined by an edge to a selected term. This is repeated until the root node is reached.
While one is certainly performing a number of hypothesis tests, the method for adjusting them is not straight forward. The tests are not independent, the p-values are related to the size of the node (number of LLIDs annotated there) and the sampling distribution is not clear - hence the appropriate method of adjustment is also not clear. Despite this many people do use FDR, or similar, adjustments. I tend to use unadjusted p-values.

```
> library("GOstats")
```

```
Loading required package: graph
Loading required package: cluster
Loading required package: Ruuid
Loading required package: GO
Loading required package: RBGL
Loading required package: xtable

Attaching package: 'xtable'
```

```
        The following object(s) are masked from package:graph :

         label
```

```
> library("Rgraphviz")
> mfhyper = GOHyperG(myLLs[myLLunique])
```

We now will set things up to plot this graph.

```
> whGO = resT$index[resT$adjp < 0.05]
> gNsel <- geneNames(ALLs2)[whGO]
> if (as.integer(R.version$major) < 2) {
+     gNLL2 = unique(gNsel)
+     gGO <- makeGOGraph(gNLL2, "MF", "hgu95av2")
+ } else {
+     gNLL = unlist(mget(gNsel, hgu95av2LOCUSID, ifnotfound = "NA"))
+     gNLL2 = as.character(unique(gNLL))
+     gNLL2 = gNLL2[!is.na(gNLL2)]
+     gGO <- makeGOGraph(gNLL2, "MF", TRUE)
+ }
> nL <- rep("", length(nodes(gGO)))
> names(nL) <- nodes(gGO)
> nA <- list()
> gGhyp.pv <- mfhyper$pv[nodes(gGO)]
> gCols <- ifelse(gGhyp.pv < 0.1, "tomato", "lightblue")
> names(gCols) = names(gGhyp.pv)
> lbs = rep("", length(nodes(gGO)))
> names(lbs) = nodes(gGO)
> nA$label = lbs
> nA$fillcolor = gCols
```

Exercise 6
To answer the following questions, you should look at the manual page for GOHyperG to see what structure is returned.

- *How many tests were carried out? How many were significant? [Hint: lapply and sapply will be useful.*
- *Which nodes are significant? Is there a pattern?*

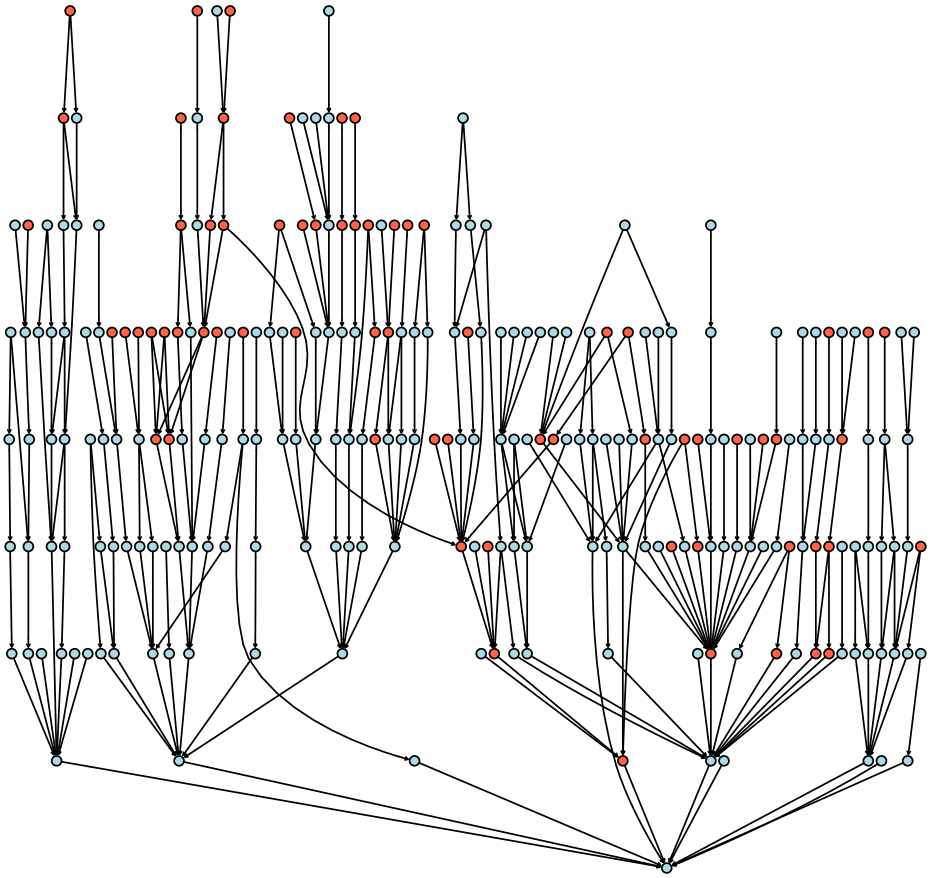


Figure 1: GO graph for ALL1/AF4 - BCR/ABL comparison

Using GO for similarity

In some settings we want to identify sets of genes that have some degree of similarity. GO can be used to define a measure of similarity. For any gene, g, we label the induced GO graph G_g. There will be a different GO graph for each ontology. The GO graph can be computed using *oneGOGraph*.
Some specific GO terms and their meanings (you might find them helpful). You might want to exclude some (or all) from different computations that you are making.

- `GO:0003673` is the GO root.
- `GO:0000004` is biological process unknown
- `GO:0005554` is molecular function unknown
- `GO:0008372` is cellular component unknown

We first get all the MF terms for our Affymetrix data. We do this by first turning the hash table into a list and then extracting from that list the set of GO terms that have an MF label (as mentioned in lecture you might also want to only choose those with a particular evidence code).

```
> affyMF = eapply(hgu95av2GO, function(x) {
+     onts = sapply(x, function(z) z$Ontology)
+     if (is.null(onts) || is.na(onts))
+         NA
+     else unique(names(onts)[onts == "MF"])
+ })
```

Here is a problem: how many of these genes (probes) have multiple GO terms associated with them? What do we do if we want to compare two genes that have multiple GO terms associated with them?
Should we map the Affymetrix identifiers to GO terms or should we map LocusLink identifiers?
Now, for any probe we can construct the GO graph, in this example we only use one Affy ID, and leave it to you to extend this result to accommodate the general case.

```
> gc()

           used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells 10040692 268.2   14466089 386.3 13739133 366.9
Vcells 15640974 119.4   33433498 255.1 33342714 254.4

> affyMF[5]
```

```
$"738_at"
[1] "GO:0008253" "GO:0016787"
```

```
> ggs = lapply(affyMF[5], function(x) {
+     if (is.null(x))
+         return(NULL)
+     ans = NULL
+     for (i in 1:length(x)) {
+         tmp = oneGOGraph(x[i], GOMFPARENTS)
+         ans[[i]] = tmp
+     }
+     a1 = ans[[1]]
+     if (length(x) == 1)
+         return(a1)
+     for (j in 2:length(x)) a1 = join(a1, ans[[j]])
+     return(a1)
+ })
> ggs
```

```
$"738_at"
A graphNEL graph with directed edges
Number of Nodes = 9
Number of Edges = 8
```

Suppose that there are M genes under consideration. For each pair of genes g_i and g_j and for each ontology assign a measure of similarity as follows:

- find the set of common GO terms within an ontology, S_{ij}
- find the depth, D_{ij} of each term in S_{ij}, where depth is distance to the root node (number of edges)
- then the similarity measure is the maximum depth, D_{ij}

The larger the depth the more similar the two genes are. They have a very specific GO term in common, within that ontology.
One might use some sort of threshold based on the quantiles of D_{ij} to identify closely related genes.
Given a chip (a set of assayed genes) one can develop a collections of genes that are likely to be highly related (or possibly interacting; sometimes this is called a *predictome*). It will often make sense, especially if one is considering physical interaction to make use of the MF and BP ontologies to define similarity and to then requires additionally a high similarity in the CC ontology to ensure that the gene products are likely to be in the same place and hence able to interact.

To carry out these different operations we might want to use some of the tools that have been produced in *RBGL*, *graph* and *Rgraphviz*. We will be spending a lot more time on these later, but we introduce them now so that we can make better use of GO.
To find the distances we will use *dijkstra.sp*. To use that we must turn our directed graph (that is what GO graphs are) into an undirected graph using *ugraph*.

```
> library("RBGL")
> dd1 = dijkstra.sp(ugraph(ggs[[1]]), "GO:0003674")
> max(dd1$distance)

[1] 7

> if (require(Rgraphviz) && interactive()) plot(ggs[[1]])
```

Exercise 7
Using the tools described here write a function to implement the gene similarity measure described above.

An alternative way of assigning similarity measures would be to make use of some measure of information content. The work of Lord et al (and others) will be relevant here.
Yet a third measure of GO similarity between two genes, for a specific ontology is to take the cardinality of the terms that they have in common and divide it by the cardinality of the *union* of the two graphs. In this case the word union is being used to mean, take all nodes that appear in at least one of the graphs and all edges that appear in at least one of the graphs.
So if we let G_i denote the induced GO graph for gene i and G_j denote the induced GO graph for gene j, their intersection, $G_i \cap G_j$ is the same as S_{ij} above. Their union is $G_i \cup G_j$.

Exercise 8

- *Using the tools described here implement this version of GO similarity.*
- *Compare the similarity measures obtained using this measure with those obtained using the measure described above.*

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

```
R version 2.2.1, 2006-04-09, powerpc-apple-darwin8.6.0

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

other attached packages:
 Rgraphviz    GOstats     xtable       RBGL         GO      graph      Ruuid
   "1.8.0"    "1.4.0"    "1.3-1"    "1.6.0"   "1.10.0"    "1.8.0"    "1.8.0"
```

```
 cluster   multtest genefilter   survival   annotate   hgu95av2        ALL
"1.10.2"    "1.8.1"    "1.8.0"     "2.20"    "1.8.0"   "1.10.0"    "1.0.2"
 Biobase
 "1.8.0"
```