Lab: Annotation and meta-data

Robert Gentleman

June 9, 2004

Introduction

In this lab we will see how to generate hyperlinked output, how to use different data packages to provide meaning to our analyses. 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()
         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 or which have low levels of expression. Since the ALL1/AF4 group only has 10 samples we will set our threshold to 8 samples.
The 75th percentiles of expression values in our subset is about 6.8 (which is close to 100 on the normal scale), so we will use that.

```
> library(genefilter)
> f1 <- kOverA(8, 7)
> f2 <- function(x) (IQR(x) > 0.8)
> ff <- filterfun(f1, f2)
> selected <- genefilter(ALLs1, ff)
> sum(selected)

[1] 1079

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

So we will now analyse 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 (118). Our goal now is to see if we can make some sense out of the data.

```
> ALLs3 = ALLs2[resT$index[resT$adjp < 0.05], ]
> if (interactive()) {
+     heatmap(exprs(ALLs3), ColSide = ifelse(ALLs3$mol == "ALL1/AF4",
+         "red", "blue"), col = topo.colors(15))
+ }
> myLLs = unlist(mget(geneNames(ALLs3), hgu95av2LOCUSID))
> sum(duplicated(myLLs))

[1] 13
```

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/LocusLink), 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    9
No. LocusLink IDs  6756 1581  498  117   30   17   11    8    1
```

```
Loading required package: Brixen
```

This leads to a number of complications, as we discuss in the following. Of the 2263 LocusLink IDs that have more than one probe set annotated at them, we found that in 39 cases our nonspecific filtering step of Section **??** selected some, but not all corresponding probe sets.

Exercise 1
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.*

Chromosomes

Now, we would like to ask whether, for our gene list there is an overabundance of genes from one specific chromosome. To answer this question we will use the Hypergeometric distribution. (The *right* way to do this is to use a multiway table, but for now we will ask the question on a per chromosome basis).
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     2
12610    15
```

```
> 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
1220  447  703  697  221  410  342  498  731  168  752  793  315  146  360  660
   4    5    6    7    8    9    X    Y
 446  536  722  581  428  425  519   25
```

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

```
[1] 1220
```

```
> 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 $x = 10$, $m = 928$, $n = 9020$, and $k = 105$. We want to compute the probability that x is as large, or larger than, the observed x.

Exercise 2

- *Find out which genes have two chromosomes. If you have an internet connection you can try to understand why this happens.*
- *Use the code above to carry out similar calculations for the other chromosomes (be careful with X and Y).*
- *If you have a statistics background, you should be able to easily carry out a χ^2 test for the real problem.*

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, they 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: GO

> mfhyper = GOHyperG(myLLs[myLLunique])
```

We now will set things up to plot this graph, if you are using Windows you won't be able to plot the graph (yet), since Rgraphviz does not currently work on Windows.

```
> whGO = resT$index[resT$adjp < 0.05]
> gNsel <- geneNames(ALLs2)[whGO]
> gGO <- makeGOGraph(gNsel, "MF", "hgu95av2")
> 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 3
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?*

```
Loading required package: Rgraphviz
Creating a new generic function for "lines" in ".GlobalEnv"
Creating a new generic function for "plot" in ".GlobalEnv"
```

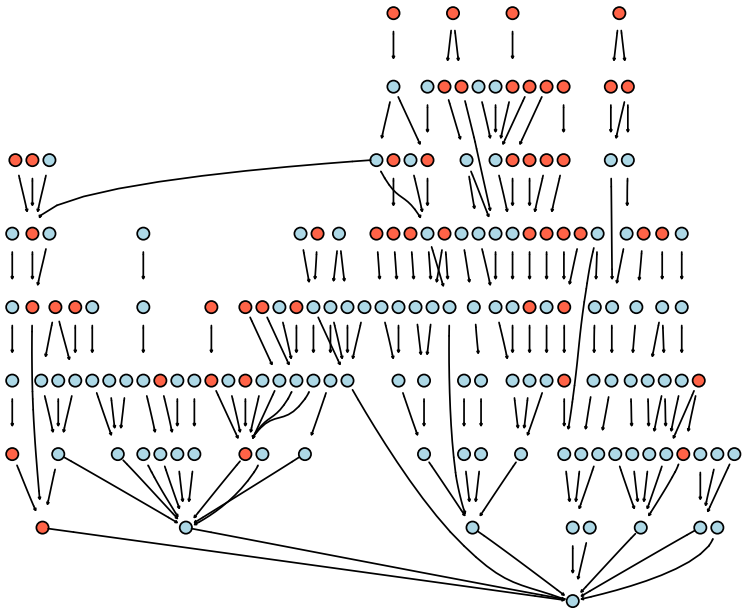

Figure 1: GO graph for ALL1/AF4 - BCR/ABL comparison

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).

```
> affyGO = as.list(hgu95av2GO)
> affyMF = lapply(affyGO, 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.

```
> affyMF[5]

$"2050_s_at"
[1] "GO:0003925" "GO:0005525"
```

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

$"2050_s_at"
A graph with  directed  edges
Number of Nodes = 13
Number of Edges = 13
```

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] 5

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

Exercise 4
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 5

- *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.*