Seminar III: R/Bioconductor

Leonardo Collado Torres
lcollado@lcg.unam.mx
Bachelor in Genomic Sciences
www.lcg.unam.mx/~lcollado/

August - December, 2009
A Case Study with GOs

Intro

Data and filtering

Complications

Some statistics

GO

More on GO
A Case Study with GOs

Credits

 Homework
Packages we’ll use today

» You’ll probably need to install a few using biocLite.

```r
> library("ALL")
> library("Biobase")
> library("annotate")
> library("hgu95av2.db")
> library("genefilter")
> library("annaffy")
> library("GO.db")
> library("GOstats")
> library("biomaRt")
> library("hgu133a.db")
> library("lattice")
```
To start off

- Similar to the 2nd homework, let's create a subset from the ALL dataset.
- Remember that we are working with leukemia samples and the molecular types BCR/ABL and ALL/AF4 are different translocations.

```
> library("ALL")
> data("ALL")
> types <- c("ALL1/AF4", "BCR/ABL")
> bcell <- grep("^B", as.character(ALL$BT))
> ALL_af4bcr <- ALL[, intersect(bcell, + which(ALL$mol.biol %in% types))]
> ALL_af4bcr$mol.biol <- factor(ALL_af4bcr$mol.biol)

- How many features does our subset have?
```
To start off

▶ Samples?
Filtering

- We can make a table to check how many samples we have:
  ```r
  > table(ALL_af4bcr$mol.biol)
  ALL1/AF4  BCR/ABL
         10       37
  ```

- Our groups are rather different in size, so the outliers of BCR/ABL will dominate the variance.

- There are several options on how to filter the data, but we’ll use the 10% and 90% quantiles.

- How do you find that range?
Filtering II

▶ Lets take advantage of the **quantile** and **diff** functions:

```r
> qrange <- function(x) diff(quantile(x, +    c(0.1, 0.9)))
```

▶ Now we can use the `nsFilter` function from the `genefilter` package:

```r
> suppressWarnings(library("genefilter"))
> library("hgu95av2.db")
> filt_af4bcr <- nsFilter(ALL_af4bcr,
>                          require.entrez = TRUE, require.GOBP = TRUE,
>                          var.fun = qrange, var.cutoff = 0.5)
> ALLfilt_af4bcr <- filt_af4bcr$eset
```

▶ Previously we had used the IQR function instead of our homemade `qrange`. 
Top 100

▶ Now lets find the top 100 genes by carrying out a two group comparison.

▶ We’ll need to load some packages first:
  > library("Biobase")
  > library("annotate")

▶ Now we can use the rowttests function:
  > rt <- rowttests(ALLfilt_af4bcr, + "mol.biol")
  > names(rt)

  [1] "statistic" "dm" "p.value"
Quick exercises

Create a histogram of

- the statistic
- the p values
Solution 1

```r
> hist(rt$statistic, breaks = 100,
+       col = "skyblue")
```
Solution I

Histogram of rt$statistic

rt$statistic

Frequency

-10 0 10 20
0 100 300 500

rt$statistic
Solution II

> hist(rt$p.value, breaks = 100,
+   col = "mistyrose")
Solution II

Histogram of rt$p.value
Lowest 400 p values

- Lets create the `ALLsub ExpressionSet` with the 400 probe sets with the lowest p values.
- Any ideas?
Solution

Here is one way:

```r
> sel <- order(rt$p.value)[1:400]
> ALLsub <- ALLfilt_af4bcr[sel, ]
```

Next, let's find how many probe sets in `ALL` and how many in `ALLsub` map to the same EntrezGene ID.
A trick

- First let's get the IDs into two separate vectors:
  ```
  EG <- as.character(hgu95av2ENTREZID[featureNames(ALL)])
  EGsub <- as.character(hgu95av2ENTREZID[featureNames(ALLsub)])
  ```

- Next, let's use a little trick: using two table functions!
  ```
  head(table(EG))
  ```

  EG
  ```
  10 100 1000 10000 10001 10002
  1  2  2  1  3  2
  ```

  ```
  table(table(EG))
  ```
A trick

```r
> table(table(EGsub))

     1 2 3 4 5 6 7 8
6891 1495 468 97 25 13 5 5
9
1

> table(table(EGsub))

     1 400
400
```

Why do all the probe sets in ALLsub map to a unique EntrezGene ID?
Now let's look at the expression profile of a given gene, for example, CD44.

First, let's find out which features belong to our gene:

```r
> syms <- as.character(hgu95av2SYMBOL[featureNames(ALLsub)])
> whFeat <- names(which(syms == "CD44"))
```

Now let's create a subset of ALLsub with the info we want:

```r
> ordSamp <- order(ALLsub$mol.biol)
> CD44 <- ALLsub[whFeat, ordSamp]
```

What kind of plot should we make to visualize the expression profile of CD44?
Simple plot

A simple plot is enough:

```r
> plot(as.vector(exprs(CD44)), main = whFeat,
+     col = c("sienna", "tomato")[CD44$mol.biol],
+     pch = c(15, 16)[CD44$mol.biol],
+     ylab = "expression")
```
Simple plot

2036_s_at

Index

expression
Now a barplot

We used some mapping tricks to distinguish the two molecular types. Looks like ALL1/AF4 have higher values than BCR/ABL. Now let's make a barplot to group the values per chromosome:

```r
> z <- toTable(hgu95av2CHR[featureNames(ALLsub)])
> chrtab <- table(z$chromosome)
> chrtab

  1  10  11  12  13  14  15  16  17  18  19   2  20
43  23  23  20   9  20   5  12  17   6  14  26   9
21  22   3   4   5   6   7   8   9  X  Y
  7  13  18  14  11  39  22  14  20  15   1
```
Now a barplot

```r
> chridx <- sub("X", "23", names(chrtab))
> chridx <- sub("Y", "24", chridx)
> barplot(chrtab[order(as.integer(chridx))],
  +    cex.names = 0.5, col = rainbow(24))
```
Now a barplot
Checking

- Why did I use the `sub` commands?
- Why did I use `order` inside the `barplot` call?
A sweet html table

- Now lets assume that you want to show a table for the 400 genes in ALLsub to someone.
- Lets use the annaffy package to create an html table:
  ```r
  > library("annaffy")
  > anncols <- aaf.handler(chip = "hgu95av2.db")[c(1:3, +
  8:9, 11:13)]
  > anntable <- aafTableAnn(featureNames(ALLsub), +
  "hgu95av2.db", anncols)
  > saveHTML(anntable, "ALLsub.html", +
  title = "The Features in ALLsub")
  ```
- We can open the html file directly from R using:
  ```r
  > localURL = file.path(getwd(), "ALLsub.html")
  > browseURL(localURL)
  ```
A sweet html table

- Open the html file :)

Multiple measurements

- A big problem is that multiple probe sets can match to the same gene, which means that for some you have more measurements than for others. Also, alternative splicing can give you headaches.
- These R packages follow the ENCODE Project Consortium.
- Let's look at an example:
  ```r
  > probeSetsPerGene <- split(names(EG),
  +     EG)
  > j <- probeSetsPerGene$"7013"
  > j
  ```
Multiple measurements

[1] "1329_s_at" "1342_g_at"
[3] "1361_at" "32255_i_at"
[5] "32256_r_at" "32257_f_at"
[7] "32258_r_at"

- We found 7 probes matching to the same gene (EntrezGene ID 7013).
Example complication

Let's look at the expression values from 2 of them:

```r
> plot(t(exprs(ALL_af4bcr)[j[c(1, 7)], ], asp = 1, pch = 16,
+       col = ifelse(ALL_af4bcr$mol.biol ==
+                      "ALL1/AF4", "black", "grey"))
```
Example complication
A complicated plot

We now used a different trick to map the colors: the `ifelse` function. A better plot in this case is the heatmap using the `lattice` function `levelplot`. Let's make one for the our gene 7013.

```r
> library("lattice")
> mat <- exprs(ALL_af4bcr)[j, ]
> mat <- mat - rowMedians(mat)
> ro <- order.dendrogram(as.dendrogram(hclust(dist(mat))))
> co <- order.dendrogram(as.dendrogram(hclust(dist(t(mat)))))
> at <- seq(-1, 1, length = 21) * max(abs(mat))
> lp <- levelplot(t(mat[ro, co]),
+     aspect = "fill", at = at, scales = list(x = list(rot = 90),
+     colorkey = list(space = "left")))
> print(lp)
```
A complicated plot
One of the tests we can make now is to check for every chromosome, the low and high p values.

To do so we can use `chisq.test` and `fisher.test`.

First we need to create a data frame to map for every EntrezGene ID to which chromosome it belongs:

```r
> ps_chr <- toTable(hgu95av2CHR)
> ps_eg <- toTable(hgu95av2ENTREZID)
> chr <- merge(ps_chr, ps_eg)
> dim(chr)
[1] 11972 3
```

We don’t need the first column, so let’s take it out:
chr

> chr <- unique(chr[, colnames(chr) != "probe_id"])
> dim(chr)
[1] 9009 2
> head(chr)

    chromosome gene_id
   1        14   5875
   2        16   5595
   3         1   7075
   4        10   1557
   5        11    643
   7         5  1843
What problem do you notice? You might need to explore chr in full.
Duplications

- Look at this table:
  ```
  > table(table(chr$gene_id))
  
  1   2
  8985 12
  ```

- Lets take out those complicated genes that have duplicated entries.
  ```
  > chr <- chr[!duplicated(chr$gene_id),
  +]
  ```
Checking for association

- Now we can do the contingency table for the association between the EntrezGene ID with their chromosome mapping and with being differently expressed.

- Let's re-use our EGsub object which had those differently expressed.

```r
> isdiff <- chr$gene_id %in% EGsub
> tab <- table(isdiff, chr$chromosome)
> tab
```

38 / 63
### Checking for association

<table>
<thead>
<tr>
<th>isdiff</th>
<th>1</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>898</td>
<td>304</td>
<td>498</td>
<td>474</td>
<td>150</td>
<td>271</td>
<td>256</td>
<td>366</td>
</tr>
<tr>
<td>TRUE</td>
<td>43</td>
<td>23</td>
<td>23</td>
<td>20</td>
<td>9</td>
<td>20</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>isdiff</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>2</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>512</td>
<td>122</td>
<td>543</td>
<td>547</td>
<td>221</td>
<td>93</td>
<td>249</td>
<td>461</td>
</tr>
<tr>
<td>TRUE</td>
<td>17</td>
<td>6</td>
<td>14</td>
<td>26</td>
<td>9</td>
<td>7</td>
<td>13</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>isdiff</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>Un</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>326</td>
<td>390</td>
<td>490</td>
<td>406</td>
<td>297</td>
<td>311</td>
<td>4</td>
<td>384</td>
</tr>
<tr>
<td>TRUE</td>
<td>14</td>
<td>11</td>
<td>39</td>
<td>22</td>
<td>14</td>
<td>20</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>isdiff</th>
<th>Y</th>
</tr>
</thead>
</table>
Checking for association

<table>
<thead>
<tr>
<th></th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>24</td>
<td>0</td>
</tr>
</tbody>
</table>

Once we have this table, we can do a Fisher’s exact test:

```r
> fisher.test(tab, simulate.p.value = TRUE)
```

Fisher's Exact Test for Count Data
with simulated p-value (based on 2000 replicates)

data:  tab
p-value = 0.01499
alternative hypothesis: two.sided

And a Chi squared test:

```r
> chisq.test(tab)
```
Checking for association

Pearson's Chi-squared test

data:  tab
X-squared = 42.2405, df = 24,
p-value = 0.01213

What can we conclude?
Strand bias

- We can also check for where the genes are located, what other genes are nearby, grouping genes by location before another test, ...
- Lets check if our differentially expressed genes are on the same strand:
  ```r
  > chrloc <- toTable(hgu95av2CHRLOC[featureNames(ALLsub)]
  > head(chrloc)
  ```
Strand bias

<table>
<thead>
<tr>
<th>probe_id</th>
<th>start_location</th>
<th>Chromosome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1635_at</td>
<td>132579088</td>
<td>9</td>
</tr>
<tr>
<td>2 1635_at</td>
<td>132700651</td>
<td>9</td>
</tr>
<tr>
<td>3 39329_at</td>
<td>-68410592</td>
<td>14</td>
</tr>
<tr>
<td>4 40797_at</td>
<td>-56675801</td>
<td>15</td>
</tr>
<tr>
<td>5 33800_at</td>
<td>-3952652</td>
<td>16</td>
</tr>
<tr>
<td>6 34777_at</td>
<td>10283217</td>
<td>11</td>
</tr>
</tbody>
</table>

Alternative splicing will give us some problems:

\[
\text{table(table(chrloc$probe_id))}
\]

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 9 \\
285 & 66 & 33 & 9 & 3 & 3 & 1 \\
\end{array}
\]
Strand bias

- Lets collapse the information so that we only record the strand, which should be the same even if there is alternative splicing:

```r
strds <- with(chrloc, unique(cbind(probe_id, + sign(start_location))))
> table(strds[, 2])
       -1   1
  194  206
```

- What do we conclude?
Quick review

- GO, short for Gene Ontology, classifies genes products according to
  1. Molecular function
  2. Biological process
  3. Cellular component

- GO terms are represented in a graph where there are two types of relationships:
  1. is as
  2. part of

- To facilitate the mapping, GO terms are identified in 7 numbers.

- All the descendants of a given GO term are called offspring. The immediate ones are called children.

- All the parental GO terms are called ancestor.
In R, the package `GO.db` enables us to browse the GO tree:

```r
> library("GO.db")
> as.list(GOMFCHILDREN["GO:0008094"])

```

```
"GO:0008094" isa isa isa
 "GO:0004003" "GO:0015616" "GO:0033170"
 isa isa isa
 "GO:0033676" "GO:0033680" "GO:0043142"
```

```r
> as.list(GOMFOFFSPRING["GO:0008094"])
```

GO.db

`$`GO:0008094`

[1] "GO:0003689"  "GO:0004003"
[3] "GO:0015616"  "GO:0017116"
[5] "GO:0033170"  "GO:0033676"
[7] "GO:0033680"  "GO:0033681"
[9] "GO:0033682"  "GO:0043140"
Hyper Geometric GO test

- The packages `annotate` and `GOstats` are the basic ones to carry out GO analysis.
- Other related packages are `topGO` and `goTools`.
- Let's make the basic GO test. We want to compare the frequency of a GO term on a subset versus the frequency of the same GO term on the overall universe.
- Things get complicated because some GO terms have more offspring than others...
- Let's do the test (actually, lots of tests) for our data:
Hyper Geometric GO test

```r
> library("GOstats")
> affyUniverse <- featureNames(ALLfilt_af4bcr)
> uniId <- hgu95av2ENTREZID[affyUniverse]
> entrezUniverse <- unique(as.character(uniId))
> params <- new("GOHyperGParams",
+   geneIds = EGsub, universeGeneIds = entrezUniverse,
+   annotation = "hgu95av2", ontology = "BP",
+   pvalueCutoff = 0.001, conditional = FALSE,
+   testDirection = "over")

After building up all the parameters we can now make the actual test:

> myhyper <- hyperGTest(params)
```
P values

We didn’t adjust our p values as it can complicated. Instead, let’s visualize the histogram:

```r
> hist(pvalues(myhyper), breaks = 50,
+     col = "mistyrose")
```
P values

Histogram of pvalues(myhyper)

Frequency

pvalues(myhyper)
Summary for myhyper

- As you can notice, we have a peak on the left side. Meaning that we have several low p values.
- Lets look deeper into the results from our test:

```r
> sum <- summary(myhyper, p = 0.001)
> head(sum)

   GOBPID Pvalue  OddsRatio
1    GO:0007154 3.683084e-09   1.903807
2    GO:0007165 9.991034e-09   1.883483
3    GO:0006955 3.396946e-07   2.416384
4    GO:0019882 8.991223e-07   6.479221
5    GO:0002376 5.214422e-06   1.998862
6    GO:0006687 9.127024e-06  50.939086

   ExpCount Count Size
1  116.390817  168  1090
2   109.663641  159  1027
3   27.442605   54  257
4    3.737320   15   35
```
## Summary for myhyper

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>39.829151</td>
<td>67</td>
<td>373</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.747464</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Term**
- 1. cell communication
- 2. signal transduction
- 3. immune response
- 4. antigen processing and presentation
- 5. immune system process
- 6. glycosphingolipid metabolic process

▶ What do you notice? What can you conclude?
Longer definitions

- Even though the GO term definition is better than the GO ID, it is not sufficient.
- So let's take a look at the actual definitions using the GO.db package:

```r
> GOTERM[["GO:0032945"]]
```

**GOID:** GO:0032945  
**Term:** negative regulation of mononuclear cell proliferation  
**Ontology:** BP  
**Definition:** Any process that stops, prevents or reduces the frequency, rate or extent of mononuclear cell proliferation.
Longer definitions

Synonym: negative regulation of PBMC proliferation
Synonym: negative regulation of peripheral blood mononuclear cell proliferation
biomaRt

- Remember that you can use biomaRt to get GO IDs or to use them as a query and get more information on your genes / proteins.
- For instance, take a look at the `getGo` function.
- You can find GO IDs from biomaRt in PFAM, Prosite, and InterPro besides the usual, ENSEMBL.
SQL based packages

- Several packages, for example hgu133a and hgu95av2 were changed from being *environment* based to SQL based packages.
- They did this change to facilitate mapping between different identifiers.
- This was specially useful in cases where you have incomplete data.
- Plus it made everything faster :)

Seminar III: R/Bioconductor
More on GO
An example:

- **Old way:**
  ```r
  goCats <- unlist(eapply(GOTERM, + Ontology))
  old <- table(goCats)[c("BP", "CC", + "MF")]
  ```

- **New way** *WAY faster:*
  ```r
  query <- "select ontology from go_term"
  goCats <- dbGetQuery(GO_dbconn(), + query)
  new <- table(goCats)[c("BP", "CC", + "MF")]
  ```

- **Comparing:**

  ```r
  58 / 63
  ```
An example:

> identical(old, new)

[1] TRUE
Credits

- **Bioconductor Case Studies** by Florian Hahne, Wolfgang Huber, Robert Gentleman and Seth Falcon.
- Specially chapter 8.
Homework

- Choose a different EntrezGene ID (not 7013) that has different probes.
- Make a scatterplot comparing the expression values from two probe sets.
- Make the heatmap showing all the probe sets.
- Add your conclusions.
SessionInfo

> sessionInfo()

R version 2.9.0 (2009-04-17)
i386-pc-mingw32

locale:
LC_COLLATE=English_United States.1252;LC_CTYPE=English_United States.1252;LC_MONETARY=English_United States.1252;LC_NUMERIC=C;LC_TIME=English_United States.1252

attached base packages:
[1] stats graphics grDevices
[4] utils datasets methods
[7] base

other attached packages:
[1] GOstats_2.10.0
[2] graph_1.22.2
[3] Category_2.10.1
[4] lattice_0.17-22
[5] annaffy_1.16.0
[6] KEGG.db_2.2.11
SessionInfo

[7] GO.db_2.2.11
[8] annotate_1.22.0
[9] hgu95av2.db_2.2.12
[10] RSQLite_0.7-1
[11] DBI_0.2-4
[12] AnnotationDbi_1.6.0
[13] genefilter_1.24.3
[14] ALL_1.4.5
[15] Biobase_2.4.1

loaded via a namespace (and not attached):
[1] grid_2.9.0       GSEABase_1.6.1
[3] RBGL_1.20.0      splines_2.9.0
[5] survival_2.35-4  tools_2.9.0
[7] XML_2.5-1        xtable_1.5-5