Using limma to analyze microarray and RNA-Seq data

Humberto Ortiz-Zuazaga

March 7, 2013

1 Introduction

Bioconductor [3] is a set of R packages for analysis of biological data, with an emphasis on microarray and other high-throughput datasets.

The bioconductor web site has instructions to install R and bioconductor. I also highly recommend installing R Studio, a graphical environment for running R, that groups together a source editor, help, figures, data browsers, and many other tools. R Studio is available on it’s own website.

This example will use standard affy [2] and limma [4] commands to analyze example datasets. Bioconductor has extensive help, which you can access in many ways. One simple way is to type ?foo where you want help on the object called “foo” (or ??foo to search for relevant topics). You can open an interactive browser interface to the help system by typing help.start(). In the browser, you can look at the documentation for the installed packages to find help on limma and affy.

The limma package, in particular, has an extensive 120 page user’s guide, with many examples, including around 8 full case studies of high-throughput data analysis. You can read the guide using the limmaUsersGuide() command.

> library(limma)
> library(affy)

2 Microarrays

2.1 Reading the data

Both limma and affy have many functions to simplify import of microarray data into the system. See the documentation for readTargets and ReadAffy for some examples.

In this document, we will instead use sample data that has already been read into the appropriate R objects.

Affydata is one such example data set.
> require(affydata)

Package
[1,] "affydata"
LibPath
[1,] "/Library/Frameworks/R.framework/Versions/2.15/Resources/library"

Item   Title
[1,] "Dilution" "AffyBatch instance Dilution"

> data(Dilution)
> Dilution

AffyBatch object
size of arrays=640x640 features (35221 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=4
number of genes=12625
annotation=hgu95av2
notes=
The Dilution affybatch contains four samples, two each of 20µg and 10µg liver tissue from human subjects, read on two different scanners (A and B).

> phenoData(Dilution)

An object of class 'AnnotatedDataFrame'
sampleNames: 20A 20B 10A 10B
varLabels: liver sn19 scanner
varMetadata: labelDescription

Examine MA plot for raw data, comparing the 20µg samples on scanner A and B.

> plotMA(exprs(Dilution)[,c(1,2)])

Dilution will contain the AffyBatch, with the raw expression values for each probe in each sample, with additional information on the probes and samples.

2.2 Normalization and pre-processing

We can use the rma command to normalize and summarize the probes for each feature. Prior to the summarization, each feature is represented with four probes. After the normalization and summarization routine, we have a single expression value for each feature in each sample.

> probeNames(Dilution)[1:20]

[1] "100_g_at" "100_g_at" "100_g_at" "100_g_at" "100_g_at" "100_g_at"
[7] "100_g_at" "100_g_at" "100_g_at" "100_g_at" "100_g_at" "100_g_at"
[13] "100_g_at" "100_g_at" "100_g_at" "100_g_at" "1000_at" "1000_at"
[19] "1000_at" "1000_at"
Figure 1: MA plot before normalization
Small part of dilution study

Figure 2: Box plot before normalization

```r
> eset <- rma(Dilution)

Background correcting
Normalizing
Calculating Expression

> featureNames(eset)[1:10]

[1] "100_g_at"  "1000_at"  "1001_at"  "1002_f_at"  "1003_s_at"  "1004_at"
[7] "1005_at"  "1006_at"  "1007_s_at"  "1008_f_at"

A boxplot shows the distribution of expression values before (Figure 2) and after (Figure 3) the normalization.

> boxplot(Dilution)

> boxplot(exprs(eset))
```
Figure 3: Box plot after normalization
2.3 Experimental design

The experiment has a simple design, each sample is labeled in the targets file with the target it was hybridized with. This information can be used to construct a design matrix that identifies each group.

```r
> f <- factor(c("C20", "C20", "C10", "C10"))
> design <- model.matrix(~0+f)
> colnames(design) <- c("C20", "C10")
> design
     C20 C10
 1   1  0  1
 2   0  1  1
 3   1  0  0
 4   0  1  0
attr(,"assign")
 [1] 1 1
attr(,"contrasts")
attr(,"contrasts")$f
 [1] "contr.treatment"
```

We can fit a model that has a mean for each group, and test if the group means are different. The `eBayes` function computes an empirical Bayes factor, pooling the variances from all the genes to estimate significance.

```r
> cont.matrix <- makeContrasts(conc=C20-C10, levels=design)
> cont.matrix
   Contrasts
Levels conc
  C20  1
  C10 -1

> fit <- lmFit(eset, design)
> fit2 <- contrasts.fit(fit, cont.matrix)
> fit.b <- eBayes(fit2)
```

2.4 Reporting the results

We now have a model fit that estimates the log ratios between the positive and negative samples. An MA plot (Figure 4) summarizes the fit. The y axis plots $M$, the log ratio of expression in the positive and negative coefficients. The x axis plots the $A$, or average log intensity of each gene.

```r
> plotMA(fit.b)
```

The fit also has an estimate of the Bayes factor, the log odds of differential expression for each gene. A plot of the B vs log ratios is called a volcanoplot (see Figure 5).
Figure 4: MA plot
Another way to report the results is exporting a table with the most significant features. Estimated p-values using a number of multiple testing corrections can be computed, in this case we use the Benjamini & Hochberg correction.

```
> topTable(fit.b, adjust="BH")
```

<table>
<thead>
<tr>
<th>ID</th>
<th>logFC</th>
<th>AveExpr</th>
<th>t</th>
<th>P.Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>12579</td>
<td>AFFX-DapX-M_at</td>
<td>0.6068933</td>
<td>7.248605</td>
<td>8.046185</td>
</tr>
<tr>
<td>4143</td>
<td>34103_at</td>
<td>-0.4884786</td>
<td>2.945512</td>
<td>-7.624196</td>
</tr>
<tr>
<td>1657</td>
<td>31642_at</td>
<td>-0.5097732</td>
<td>2.954299</td>
<td>-7.472093</td>
</tr>
<tr>
<td>12607</td>
<td>AFFX-M27830_5_at</td>
<td>0.5780179</td>
<td>6.203493</td>
<td>6.982726</td>
</tr>
<tr>
<td>2962</td>
<td>32934_i_at</td>
<td>-0.4774220</td>
<td>2.961046</td>
<td>-6.301995</td>
</tr>
<tr>
<td>12599</td>
<td>AFFX-HUMRGE/M10098_M_at</td>
<td>0.4273471</td>
<td>5.555700</td>
<td>6.301609</td>
</tr>
<tr>
<td>10990</td>
<td>40886_at</td>
<td>0.4393364</td>
<td>11.308830</td>
<td>5.895455</td>
</tr>
<tr>
<td>12561</td>
<td>AFFX-BioB-5_at</td>
<td>0.4799698</td>
<td>9.407338</td>
<td>5.785891</td>
</tr>
<tr>
<td>8993</td>
<td>38907_at</td>
<td>0.3554734</td>
<td>3.165857</td>
<td>5.503947</td>
</tr>
<tr>
<td>9483</td>
<td>39393_r_at</td>
<td>0.4151446</td>
<td>7.560304</td>
<td>5.320512</td>
</tr>
</tbody>
</table>

Figure 5: Volcano plot
3 RNA-Seq

3.1 Introduction

Many gene expression studies are turning to RNA-Seq, a form of second-generation sequencing that sequences products derived from mRNA. RNA-Seq count data can be analyzed with the same tools we have learned to use for microarrays. Obtaining count data (also known as digital gene expression data) from raw RNA-Seq reads is beyond the scope of this guide, but tools such as galaxy [5], bowtie [6], and trinity [7] can be used to map reads to genes and tally the counts.

3.2 RNA-Seq packages

> library(Biobase)
> library(biomaRt)
> library(edgeR)

3.3 Load ReCount data

ReCount [8] is an online database of RNA-seq data from 18 experiments. These experiments have already been read into R and are published on the web. R can load prepared datasets directly from the web:

> gilad <- load(
  + url("http://bowtie-bio.sourceforge.net/recount/ExpressionSets/gilad_eset.RData"))
> gilad

[1] "gilad.eset"

The data is already an Expression Set, with raw counts for each probe, and information on the samples. This data comes from a published study of gene expression in liver in males and females of different species [9]. We will use data from 3 males and 3 females from the human species.

> gilad.eset
ExpressionSet (storageMode: lockedEnvironment)
assayData: 52580 features, 6 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: SRX014818and9 SRX014820and1 ... SRX014828and9 (6 total)
  varLabels: sample.id num.tech.reps gender
  varMetadata: labelDescription
featureData
  featureNames: ENSG00000000003 ENSG00000000005 ... LRG_99 (52580 total)
  fvarLabels: gene
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

> phenoData(gilad.eset)$gender
[1] F F F M M M
Levels: F M

> exprs(gilad.eset)[1:5,]
       SRX014818and9 SRX014820and1 SRX014822and3 SRX014824and5
ENSG00000000003   60    60     16      9
ENSG00000000005     0     0      0      0
ENSG000000000419   25     9     15     15
ENSG000000000457   32    19     21     31
ENSG000000000460    1     3      0      5
       SRX014826and7 SRX014828and9
ENSG000000000003   56    37
ENSG000000000005     0     0
ENSG000000000419   26    11
ENSG000000000457   28    28
ENSG000000000460    1     1

3.4 Removing genes that are not expressed
Many genes are not present in any sample, it is simpler to remove these before continuing.

> isexpr <- rowSums(cpm(exprs(gilad.eset))>1) >= 3
> sum(isexpr)
[1] 8069

> gilad.isexpr <- gilad.eset[isexpr,]
3.5 Normalize the counts

Limma provides a routine, `voom`, designed to normalize digital gene expression data.

```r
> nf <- calcNormFactors(gilad.isexpr)
> groups <- phenoData(gilad.isexpr)$gender
> design <- model.matrix(~ groups)
> y <- voom(exprs(gilad.isexpr),design,
        + lib.size=colSums(exprs(gilad.isexpr))*nf,
        + normalize.method="quantile")
```

3.6 Build a linear model

Once we have log-normalized counts, we can proceed to construct a linear model, using the same tools we used for one-color (Affymetrix) arrays.

```r
> fit <- lmFit(y,design)
> fit <- eBayes(fit)
> topTable(fit, coef=2)
```

<table>
<thead>
<tr>
<th>ID</th>
<th>logFC</th>
<th>AveExpr</th>
<th>t</th>
<th>P.Value</th>
<th>adj.P.Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>286</td>
<td>1.4179955</td>
<td>8.353328</td>
<td>4.932432</td>
<td>0.0006353586</td>
<td>0.9932438</td>
</tr>
<tr>
<td>1774</td>
<td>-2.4011726</td>
<td>6.693654</td>
<td>-4.493731</td>
<td>0.0012221106</td>
<td>0.9932438</td>
</tr>
<tr>
<td>5957</td>
<td>-0.909697</td>
<td>7.477112</td>
<td>-3.614733</td>
<td>0.0049090586</td>
<td>0.9932438</td>
</tr>
<tr>
<td>191</td>
<td>0.9631780</td>
<td>7.737762</td>
<td>3.342743</td>
<td>0.0076923282</td>
<td>0.9932438</td>
</tr>
<tr>
<td>6993</td>
<td>-1.0631848</td>
<td>6.102709</td>
<td>-3.413789</td>
<td>0.0068359089</td>
<td>0.9932438</td>
</tr>
<tr>
<td>7743</td>
<td>1.0644053</td>
<td>6.614080</td>
<td>3.275959</td>
<td>0.0085985800</td>
<td>0.9932438</td>
</tr>
<tr>
<td>4913</td>
<td>-1.0201108</td>
<td>6.587511</td>
<td>-3.254500</td>
<td>0.0089125858</td>
<td>0.9932438</td>
</tr>
<tr>
<td>5654</td>
<td>-1.5657336</td>
<td>5.564722</td>
<td>-3.758463</td>
<td>0.0038844280</td>
<td>0.9932438</td>
</tr>
<tr>
<td>3045</td>
<td>-1.3333333</td>
<td>5.721775</td>
<td>-3.134114</td>
<td>0.0109064995</td>
<td>0.9932438</td>
</tr>
<tr>
<td>4827</td>
<td>-0.7220998</td>
<td>7.084751</td>
<td>-3.117506</td>
<td>0.0112154685</td>
<td>0.9932438</td>
</tr>
</tbody>
</table>

3.7 Volcanoplot model fit

The fit can be summarized using the same reporting tools as before.
> volcanoplot(fit, coef=2)

![Volcano plot](image)

**References**


