Tutorial: analysing Microarray data using BioConductor

Guiyuan Lei

Centre for Integrated Systems Biology of Ageing and Nutrition (CISBAN)
School of Mathematics & Statistics
Newcastle University
http://www.mas.ncl.ac.uk/~ngl9/

4 Feb, 2008
Outline of Tutorial

Tutorial on CISBAN Internal wiki, under ‘Tools & Resources’ page

http://bioinf.ncl.ac.uk/cisban/doku.php?id=resources:resourceshome

- Introduction to Bioconductor
  - Fibroblast data set (Chapter 2) and Yeast time course data (Chapter 3)
    - Pre-process of data
    - Model Fitting for Identifying Differential Expression
    - Network Inference
Tutorial on CISBAN Internal wiki, under ‘Tools & Resources’ page http://bioinf.ncl.ac.uk/cisban/doku.php?id=resources:resourceshome

- Introduction to Bioconductor
- Fibroblast data set (Chapter 2) and Yeast time course data (Chapter 3)
  - Pre-process of data
  - Model Fitting for Identifying Differential Expression
  - Network Inference
Bioconductor:
- open source software for bioinformatics
- provide innovative methodology for analyzing genomic data
- using R statistical computing environment

R: Powerful grapphic feature and cut-edge statistical techniques, around 800 packages available, around 60 basic packages (like affy, limma) in Bioconductor

Published Papers using Bioconductor
http://www.bioconductor.org/pub
- Google Scholar Beta, PubMed, BEPress (Berkeley Electronic Press), Biostatistics, BioMed Central Bioinformatics and Ingenta
- For example, in Bioinformatics, 161 papers found with ‘Bioconductor’ in title
Why Bioconductor

Bioconductor:
- open source software for bioinformatics
- provide innovative methodology for analyzing genomic data
- using R statistical computing environment

R: Powerful graphic feature and cut-edge statistical techniques, around 800 packages available, around 60 basic packages (like affy, limma) in Bioconductor

Published Papers using Bioconductor
http://www.bioconductor.org/pub
- Google Scholar Beta, PubMed, BEPress (Berkeley Electronic Press), Biostatistics, BioMed Central Bioinformatics and Ingenta
- For example, in Bioinformatics, 161 papers found with ‘Bioconductor’ in title
Bioconductor:
- open source software for bioinformatics
- provide innovative methodology for analyzing genomic data
- using R statistical computing environment

R: Powerful graphic feature and cut-edge statistical techniques, around 800 packages available, around 60 basic packages (like affy, limma) in Bioconductor

Published Papers using Bioconductor
http://www.bioconductor.org/pub
- Google Scholar Beta, PubMed, BEPress (Berkeley Electronic Press), Biostatistics, BioMed Central Bioinformatics and Ingenta
- For example, in Bioinformatics, 161 papers found with ‘Bioconductor’ in title
Getting Bioconductor and associated packages

- Install base packages, such as *affy* and *limma*
  ```r
  source("http://bioconductor.org/biocLite.R")
  biocLite()
  ```
- Install specific packages, such as *yeast2probe*
  ```r
  source("http://bioconductor.org/biocLite.R")
  biocLite("yeast2probe")
  ```
- Set search directories in the `.Renviron` file, e.g.
  ```
  R_LIBS=/data/Rpackages/
  ```
- Update Bioconductor
  ```r
  source("http://bioconductor.org/biocLite.R")
  update.packages(repos=biocinstallRepos(), ask=FALSE)
  ```
Pre-process of data

- Entering data into Bioconductor
- Extraction of Cerevisiae probesets
- Exploratory data analysis
- Normalising Microarray data
- Probeset level expression to gene level expression
- Principal Component Analysis
library(affy)
fns2 = list.celfiles(path="data2", full.names=TRUE)
rawdata = ReadAffy(filenames=fns2)
print(rawdata)

<table>
<thead>
<tr>
<th>Strain</th>
<th>0 hours</th>
<th>1 hour</th>
<th>2 hours</th>
<th>3 hours</th>
<th>4 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutant 1</td>
<td>yeast01.cel</td>
<td>yeast02.cel</td>
<td>yeast03.cel</td>
<td>yeast04.cel</td>
<td>yeast05.cel</td>
</tr>
<tr>
<td>Wild type 1</td>
<td>yeast06.cel</td>
<td>yeast07.cel</td>
<td>yeast08.cel</td>
<td>yeast09.cel</td>
<td>yeast10.cel</td>
</tr>
<tr>
<td>Mutant 2</td>
<td>yeast11.cel</td>
<td>yeast12.cel</td>
<td>yeast13.cel</td>
<td>yeast14.cel</td>
<td>yeast15.cel</td>
</tr>
<tr>
<td>Wild type 2</td>
<td>yeast16.cel</td>
<td>yeast17.cel</td>
<td>yeast18.cel</td>
<td>yeast19.cel</td>
<td>yeast20.cel</td>
</tr>
<tr>
<td>Mutant 3</td>
<td>yeast21.cel</td>
<td>yeast22.cel</td>
<td>yeast23.cel</td>
<td>yeast24.cel</td>
<td>yeast25.cel</td>
</tr>
<tr>
<td>Wild type 3</td>
<td>yeast26.cel</td>
<td>yeast27.cel</td>
<td>yeast28.cel</td>
<td>yeast29.cel</td>
<td>yeast30.cel</td>
</tr>
</tbody>
</table>
Mask file to filter out pombe probesets
http://www.affymetrix.com/Auth/support/downloads/mask_files/s_cerevisiae.zip

s_cerevisiae<-scan("s_cerevisiae.msk", skip=2, list("", ""))
pombe_filter_out<-s_cerevisiae[[1]]

RemoveProbes ¹

source("RemoveProbes.r")
library(yeast2probe)
cleancdf = cleancdfname("yeast2")
RemoveProbes(listOutProbes=NULL, pombe_filter_out, "yeast2cdf", "yeast2probe")

Cerevisiae probesets IDs

- **Yeast probeset IDs**
  ```r
  library(yeast2)
  genenames = as.list(yeast2GENENAME)
  YeastProbeID <- names(genenames)
  ```

- **Cerevisiae probeset IDs**
  ```r
  CerevisiaeProbeID <-
  YeastProbeID[-match(pombe_filter_out,YeastProbeID)]
  ```
Yeast Transcript IDs from annotation file
Yeast_2_na24.annot.csv.zip

```r
yeast2annotation=read.csv(file="yeast2annotation.csv", header=TRUE, stringsAsFactors=FALSE)
YeastTranscriptID<-yeast2annotation[,3]
yeast2annotationProbesetID<-yeast2annotation[,1]
YeastTranscriptID<-YeastTranscriptID[match(YeastProbeID, yeast2annotationProbesetID)]
```

Yeast gene names

```r
YeastGeneName<-character()
for(i in 1:length(YeastProbeID)){
  YeastGeneName[i]=genenames[i][[1]]
  if(is.na(YeastGeneName[i])){
    YeastGeneName[i]=YeastTranscriptID[i]
  }
}
```

Cerevisiae gene names

```r
CerevisiaeGeneName<-
  YeastGeneName[-match(pombe_filter_out, YeastProbeID)]
```
Exploratory data analysis: examining raw images

```
png(filename = "cerevisiaeimage.png", width = 960, height = 480)
par(mfrow = c(1, 2))
image(rawdata[, 1])
image(rawdata[, 27])
dev.off()
```
Exploratory data analysis: probe intensities

```r
png(filename="yeastintensities.png", width=960, height=960)
hist(rawdata, lty=1:30, lwd=2)
legend(14, 0.60, legend=sampleNames(rawdata), lty=1:30, lwd=2)
dev.off()
```

![Graph showing density against log intensity for different samples.](attachment:yeastintensities.png)
Exploratory data analysis: MA plots

```
png(filename="cerevisiaemaplot.png",width=960, height=480)
par(mfrow=c(1,2))
MAplot(rawdata,which=1)
MAplot(rawdata,which=27)
dev.off()
```
Exploratory data analysis: RNA degradation

```r
RNAdeg <- AffyRNAdeg(rawdata)
plotAffyRNAdeg(RNAdeg)
```
Pre-process of data: Normalisation

```r
eset.rma = rma(rawdata)
library(affyPLM)
par(mfrow=c(1,2))
boxplot(rawdata, col="red", main="Cerevisiae Probe intensities")
boxplot(eset.rma, col="blue", main="Cerevisiae RMA expression values")
```

Guiyuan Lei

Tutorial: analysing Microarray data using BioConductor
There are usually several probesets map to one gene in Affymetrix.

```r
CerevisiaeGeneNameLevels <- factor(CerevisiaeGeneName)
# Function to average the expression of probesets
# which map to same gene
probeset2genelevel <- function(onesample) {
  return(tapply(onesample, CerevisiaeGeneNameLevels, mean))
}
# Do the average for each column/array
CerevisiaeGeneData <- apply(CerevisiaeProbeData, 2, probeset2genelevel)
```
library(smida)
cluster.samples(t(CerevisiaeProbeData), method="pca")
Limma model
- Construct design matrix
- Construct constraints

Plot time course for top differential expression

Heatmap
library(limma)
levels = c("m0", "m1", "m2", "m3", "m4", "w0", "w1", "w2", "w3", "w4")
X = rep(levels, 3)
TS <- factor(X, levels = levels)
design <- model.matrix(~0+TS)
colnames(design) <- levels(TS)

E =
\[
\begin{pmatrix}
y_{g_1} \\
y_{g_2} \\
\vdots \\
y_{g_{10}} \\
\vdots \\
y_{g_{26}} \\
y_{g_{27}} \\
y_{g_{28}} \\
y_{g_{29}} \\
y_{g_{30}}
\end{pmatrix}
\times
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
m_0 \\
m_1 \\
m_2 \\
m_3 \\
m_4 \\
w_0 \\
w_1 \\
w_2 \\
w_3 \\
w_4
\end{pmatrix}
\]
For identifying differential expression, combine the contrasts by comparing mutant type and wild type at time point 1, 2, 3 and 4.

```r
# Model Fitting
fit <- lmFit(CerevisiaeProbeData, design)
mc <- makeContrasts('m1-w1', 'm2-w2', 'm3-w3', 'm4-w4', levels=design)
fit2 <- contrasts.fit(fit, mc)
eb <- eBayes(fit2)

Different ways to rank the differentially expressed probesets:

topTable(eb, sort.by = 'logFC') # log-fold change

topTableF(eb) # F-statistics
```
modFpvalue<-eb$F.p.value #$F-test p value
selectedgenesindx<-p.adjust(eb$F.p.value,method="bonferroni")<0.05
Sig<-modFpvalue[selectedgenesindx]
nsiggenes<-length(Sig) #number of differential expression
resultsl<-decideTests(eb, method="global")
modF<-eb$F #$F-test value
modFordered<-order(modF, decreasing = TRUE)
CerevisiaeRankProbe<-CerevisiaeProbeID[modFordered[1:nsiggenes]]
CerevisiaeRankGeneName<-CerevisiaeGeneName[modFordered[1:nsiggenes]]
updown<-results1[modFordered[1:nsiggenes],]

<table>
<thead>
<tr>
<th>Probeset ID</th>
<th>Gene Symbol</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbesetID 1</td>
<td>Gene 1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>ProbesetID 2</td>
<td>Gene 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ProbesetID 3</td>
<td>Gene 3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Table:** Up and down regulated list

\(^2\)Not use real gene names here
# Rank the i+1’th differential expression

```r
indx <- rank(modF) == nrow(CerevisiaeProbeData) - i
```

---

**Figure:** Time course expression for top 3 differentially expressed Yeast genes³

³Not use real gene names here
modF <- eb$F #$F is to get F-statistic
which.M <- c(seq(1,5), seq(11,15), seq(21,25))
which.W <- c(seq(6,10), seq(16,20), seq(26,30))
par(mfrow=c(1,3),ask=T,cex=0.5)#cex:font size
for(i in 0:2){
  indx <- rank(modF) == nrow(CerevisiaeProbeData)-i
  row1 = CerevisiaeProbeData[indx, which.M]
  row2 = CerevisiaeProbeData[indx, which.W]
  id=CerevisiaeProbeID[indx]
  name = CerevisiaeGeneName[indx]
  genetitle<-paste(sprintf("%.30s",id)," ",sprintf("%.30s",name)," Rank=", i+1)
  time=c(0,1,2,3,4)
  plot(time, row1[1:5],ylim=range(min(row1,row2), max(row1,row2)),
       ylab="Expression", main=genetitle,pch='M',type='b',col=2)
  lines(time, row1[6:10] , pch='M', type='b', col=2)
  lines(time, row1[11:15], pch='M', type='b', col=2)
  lines(time, row2[1:5] , pch='W', type='b', col=1)
  lines(time, row2[6:10] , pch='W', type='b', col=1)
  lines(time, row2[11:15], pch='W', type='b', col=1)
}

Guiyuan Lei
```r
ngen = 100
m = matrix(nrow=ngen, ncol=30)
rnames = vector("list", length(1))
for(i in 0:(ngen-1)){
  indx <- rank(modF) == nrow(CerevisiaeProbeData) - i
  m[i+1,] = CerevisiaeProbeData[indx,]
  rnames[i+1]=CerevisiaeGeneName[indx]
}
heatmap(m)
```
Network Inference

- GeneNet
- Strimmer’s VAR model
Figure: Inferred network by GeneNet package for top 100 differentially expressed Yeast genes
# Need to transpose data matrix, rows to be arrays
m = t(m)
# Need to rearrange the rows so that the rows are ordered by time points
# using the property of design matrix
# the entry design[i,j] with value 1 means array i is for time point j!!
mnew = t(matrix(nrow=ngenes,ncol=30))
# Get the index of array ordered by time point
arrayindx<-numeric(0)
ntime=5
for(j in 1:ntime)
{arrayindx<-c(arrayindx,grep(1,design[,j]),grep(1,design[,j+ntime]))}
mnew<-m[arrayindx,]
library("GeneNet")
# step 1: create longitudinal object #
mlong = as.longitudinal(mnew,repeats=c(6,6,6,6),time=c(0,1,2,3,4))
# step 2: compute partial correlations#
pcor.dyn <- ggm.estimate.pcor(mlong, method = "dynamic")

# step 3: assign (local) fdr values to all possible edges#
m.edges <- network.test.edges(pcor.dyn, direct=TRUE)
dim(m.edges)

# step 4: construct graph containing the 150 top edges#
m.net <- extract.network(m.edges, method.ggm="number", cutoff.ggm=150)

# step 5: plot graph using graphviz#
If rnames has no ", Need for Graphviz
for(i in 1:ngenes){
    rnames[i] = paste('"',rnames[i], '"', sep='')
}
colnames(m) = rnames
node.labels <- colnames(m)

network.make.dot(filename="net.dot", m.net, node.labels, main="Yeast Network")