Introduction to Preprocessing: RMA (Robust Multi-Array Average)

Utah State University – Spring 2014

STAT 5570: Statistical Bioinformatics

Notes 1.4

References

- Chapter 2 of Bioconductor Monograph (course text)
- Irizarry et al. (2003) Biostatistics 4(2):249-264.
- Irizarry et al. (2003) Nucleic Acids Research 31(4):e15
- Bolstad et al. (2003) Bioinformatics 19(2):185-193
- Tukey. (1977) Exploratory Data Analysis
- Wu et al. (2004) Journal of the American Statistical Association 99(468):909-917

Three steps to preprocessing

- Background correction
  - Remove local artifacts and "noise"
    - so measurements aren't so affected by neighboring measurements
- Normalization
  - Remove array effects
    - so measurements from different arrays are comparable
- Summarization
  - Combine probe intensities across arrays
    - so final measurement represents gene expression level

Preprocessing – essentials

- Many different methods exist
- Three main steps in most preprocessing methods
- Keep eye on big picture:
  from probe-level intensities to estimate of gene expression on each array
- Choice makes a difference

Spike-in Experiment

- Prepare a single tissue sample for hybridization to a group of arrays
- Select a handful of control genes
- Separately prepare a series of solutions where the control genes' mRNA is spiked-in at known concentrations
- Add these spiked-in solutions to the original solution to be hybridized to the arrays

Why Spike-in?

- What can be done with a spike-in experiment?
  - What changes will be observed?

    The only differences in gene expression should be due to spike-ins
  - What is being measured?

    Gene expression; methods of estimation (RMA, GCRMA, MAS5, PLIER, others) can be calibrated

Motivation for RMA approach

- MM can detect true signal for some probes

  (but others seem to represent "background")
- Difference of PM from "background" increases with concentration - (in spike-in)
- Probe effects exist

Convolution Background Correction

$$PM_{ijk} = \underbrace{bg_{ijk}}_{\text{Background caused by optical noise and non-specific binding}} + \underbrace{s_{ijk}}_{\text{Signal for probe j of probe set k on array i}}$$

$$B(PM_{ijk}) = E[s_{ijk} \mid PM_{ijk}] > 0$$

$$s_{ijk} \sim Exp(\lambda_{ijk}) \qquad bg_{ijk} \sim N(\beta_i, \sigma_i^2)$$

Gives a closed-form transformation B()

(Model could be improved, but works very well in practice.)

Quantile Normalization

- An approach to normalize each array against all others – why?

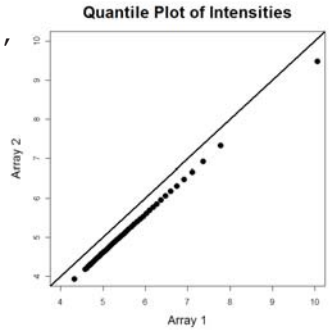
  Need arrays to be comparable

- Consider 2 arrays – how to tell if probe intensities have same distribution?

  Could consider a quantile plot

Quantile Plot for Two Arrays

```
library(affydata); data(Dilution)
int.1 <- c(pm(Dilution[,1]),
           mm(Dilution[,1]))
int.2 <- c(pm(Dilution[,2]),
           mm(Dilution[,2]))
q.1 <- quantile(log(int.1),probs=seq(0,1,0.02))
q.2 <- quantile(log(int.2),probs=seq(0,1,0.02))
par(mar=c(5,5,4,2)+0.1)
plot(q.1,q.2,pch=16,cex=1.5,xlim=c(4,10),
  cex.lab=1.5, cex.main=2,ylim=c(4,10),
  xlab='Array 1', ylab='Array 2',
  main='Quantile Plot of Intensities')
abline(0,1,lwd=3)
```



Can project points onto diagonal;

what about multiple arrays?

Quantile Normalization

- What about multiple arrays?

  If n vectors have the same distribution, plotting quantiles in n dimensions would give the unit vector "diagonal" $d = \left(\frac{1}{\sqrt{n}} \quad \frac{1}{\sqrt{n}} \quad \cdots \quad \frac{1}{\sqrt{n}} \right)$

- Make n vectors have same distribution by

  projecting n-dimensional quantile plot onto the "diagonal"

- Does this eliminate meaningful differences?

  Not if only relatively few genes should change expression value

(see Bolstad paper for details)

Summarization

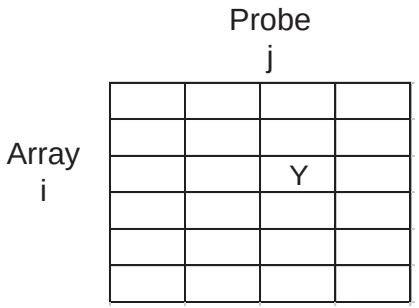
- Use the background-adjusted, quantile-normalized, and log-transformed PM intensities:

$$Y_{ijk} = \underbrace{\mu_{ik}}_{\text{Log-scale expression level}} + \underbrace{\alpha_{jk}}_{\text{Probe affinity effect; for each k, } \sum_j \alpha_{jk} = 0} + \varepsilon_{ijk}$$

  Log-scale expression level for gene k on array i

- Estimate model parameters by use of the Median Polish

Tukey's Median Polish



$$Y_{ij} = \mu + \underbrace{\delta_i}_{\text{row effects}} + \underbrace{\alpha_j}_{\text{column effects}} + \varepsilon_{ij}$$

Alternately remove (subtract) row and column medians
until sum of absolute residuals converges (for one gene k at a time)

What are we interested in here?
The fitted (predicted) row values $\hat{\mu}_i = \hat{\mu} + \hat{\delta}_i$

Properties of Median Polish

- Robust
  - important because of potential for outliers in large data sets
- Exploratory
  - Allows for a "general picture" approach to statistical ideas
  - Important for computational efficiency and complex structures
- Could be "dominated" by column effects
  - here, primarily interested in row effects
    (center expression on array)
  - best if have more arrays than probes
    (authors recommend 10-12 or more arrays)

RMA and Standard Error

- How to calculate SE of RMA median polish estimate?

  There is no way – it's just an exploratory approach
  - but the bootstrap can be applied (G. Nicholas)

- "Naïve nominal estimate"

  Fit an ANOVA model to $Y_{ijk} = \underbrace{\mu_{ik}}_{\text{Use SE of the estimate of this; treat with skepticism}} + \alpha_{jk} + \varepsilon_{ijk}$

GCRMA

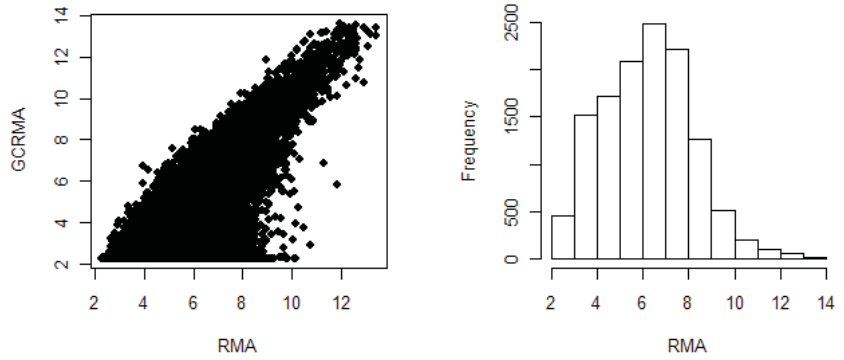
- Similar to RMA, but calculates background differently
- Makes use of MM intensities to correct background
  - Background more directly addresses non-specific binding (appears to be sequence-dependent)
- Not necessarily better than RMA

RMA in Bioconductor

```
print(date())

# data <- ReadAffy(celfile.path="C:\\folder")
## - NOTE: usually will create AffyBatch object this way

data <- Dilution  # Dilution is an AffyBatch object
gn <- geneNames(data)

# RMA - this is part of the affy package
rma.eset <- rma(data)
rma.exprs <- exprs(rma.eset) # a matrix of expression values

# Compare with another preprocessing method: GCRMA
library(gcrma)
gcrma.eset <- gcrma(data)
gcrma.exprs <- exprs(gcrma.eset)

print(date())
```



```
# Compare expression estimates (on just one array)
par(mfrow=c(2,2))
plot(rma.exprs[,1],gcrma.exprs[,1],
  xlab='RMA', ylab='GCRMA', pch=16)
# Identify highest-expressed genes
hist(rma.exprs[,1], xlab='RMA', main=NA)
gn[which.max(rma.exprs[,1])]
# AFFX-hum_alu_at
```



side note: what's lost here?

Comparing Preprocessing Methods

- Big picture:
  - probe level intensities → gene expression estimates
  - background correction, normalization, summarization
- We focused on one (RMA) and mentioned another (GCRMA)
  - There are many others: MAS5, PLIER, dChip (Li-Wong), vsn, … – why just these?
- Which is best?
  - one way – a competition (iteration 3 began in 2011): http://affycomp.biostat.jhsph.edu/
  - another consideration: statistical properties of estimates (independence, bias, SE, robust, etc.)

Numerical Dependence in Gene Expression Summaries - notation

- Let $\hat{\mu}_x$ be a given gene's log-scale expression level estimate for array *x*, after some preprocessing method
- Let $\hat{\mu}_{x(y)}$ be the gene's expression level estimate for array *x* when array *y* is not included in any step of preprocessing
- Use convention $\hat{\mu}_{x(x)} \equiv 0$

(Stevens & Nicholas, PLoS ONE 2012)

Jackknife Expression Difference (JED)

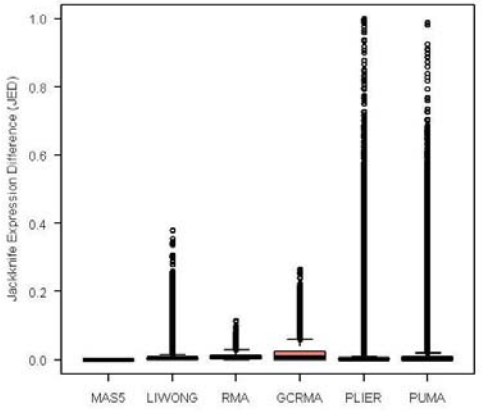
- JED(x,y) between arrays x and y for the gene:

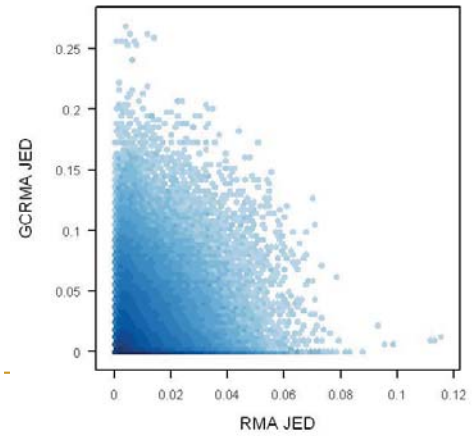
$$\frac{\left|\hat{\mu}_x - \hat{\mu}_{x(y)}\right|}{2 \cdot \max\left\{\hat{\mu}_x, \hat{\mu}_{x(y)}\right\}} + \frac{\left|\hat{\mu}_y - \hat{\mu}_{y(x)}\right|}{2 \cdot \max\left\{\hat{\mu}_y, \hat{\mu}_{y(x)}\right\}}$$

- By definition, JED(x,x)=1 (strict dependence)
- JED(x,y)=0 when strict numerical independence: $\hat{\mu}_{x(y)} = \hat{\mu}_x$ and $\hat{\mu}_{y(x)} = \hat{\mu}_y$

Numerical dependence in most common preprocessing methods

(for 1000 random genes from a public dataset)





Summary

- Preprocessing involves three main steps:
  - Background / Normalization / Summarization
- RMA
  - Convolution Background Correction
  - Quantile Normalization
  - Summarization using Median Polish
- Almost all preprocessing methods return expression levels on log2 scale
  ("the approximately right scale")
- By most reasonable metrics, RMA performs well
  (at least well enough to justify using it without losing too much sleep)