Julie Garcia
January 15, 2018

Lab 1
Advanced Genomics and Genetics Analyses

This lab introduces concepts for analyzing exon array data. The dataset that we are using has both exon- and gene-level summaries, separated into 2 distinct data matrices. The data was taken from a study that was designed to investigate alternative splicing between pediatric patients that have experienced liver rejection after an organ transplant and pediatric patients that have not experienced such rejection. These are blood samples taken from these patients, so the effect sizes are rather modest. As a result, we will be a little more liberal in our exon probe filtering methods to retain as much information as possible.

1.) Obtain and load in the exon-rma-sketch.summary.txt, gene-rma-sketch.summary.txt, dabg.summary.txt, and HuEx-1_0-st-v2.na24.hg18.probeset_abbr.csv files into R.

```
e = read.table("c:\\temp\\datasets\\exon-rma-sketch.summary.txt", header=T,
row.names=1)
e[1:4,1:4]

g = read.table("c:\\temp\\datasets\\gene-rma-sketch.summary.txt", header=T,
row.names=1)
g[1:4,1:4]

p = read.table("c:\\temp\\datasets\\dabg.summary.txt", header=T, row.names=1)
p[1:4,1:4]

map = read.csv("c:\\temp\\datasets\\HuEx-1_0-st-v2.na24.hg18.probeset_abbr.csv",
header=T, row.names=1)
map[1:4,]
```

2.) Continue to follow code in the lecture, up to the line of:
#you should filter probes based on guidelines…

```
#class membership
r <- c(1,0,1,1,1,0,1,1,1,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1)

#intersect matching probes between annotation file and data matrix
x <- intersect(dimnames(e)[[1]],dimnames(map)[[1]])

#subset the rows in the exon, p-value matrix, and annotation file to the
intersecting probes
e <- e[x,]
p <- p[x,]
map <- map[x,]
```

```
#get unique transcript cluster IDs (gene IDs) from annotation file
u <- unique(as.character(map$transcript_cluster_id))
u <- intersect(u,dimnames(g)[[1]])
```

3.) Filter probes in the exon matrix using the detection above background (DAGB) p-values. If >50% of the subjects in BOTH (total) groups have a p-value > .05 remove this exon probe. Note that this is a little more liberal than the thresholds that we explained in the lecture. The effects are small for this dataset, so this is why we are being a little less conservative with our filtering. **How many exon probes remain after this filtering?** Subset the exon matrix by those probes that pass the filter.

```
count.det <- function(x) { length(which(x>0.05))}
det <- apply(p, 1, count.det)
probes_to_remove <- names(det[det > 13]) # filter these out
e_filtered <- e[-which(rownames(e) %in% probes_to_remove),]
p_filtered <- p[-which(rownames(p) %in% probes_to_remove),]
dim(e_filtered)
```

```
158465 exon probes are left after filtering.
```

4.) Go back to the annotation table (i.e. map) and subset it by the shorter set of exon probes that you will now be using. Get the unique transcript cluster IDs from this new annotation table and intersect them with the gene matrix. You should now have a shorter u variable. **How many unique transcript cluster IDs do you have?**

```
map_filtered <- map[-which(rownames(map) %in% probes_to_remove),]
u_filt <- unique(as.character(map_filtered$transcript_cluster_id))
u_filt <- intersect(u_filt,dimnames(g)[[1]])
length(u_filt)
```

```
There are 16480 unique cluster IDs remaining.
```

5.) Now, using the code in the lecture, write a loop that loops through all values of the variable u (unique transcript cluster ID). In each loop iteration, you need to:

a. Get the correct mapping between the ith transcript cluster ID and the exon probe IDs
b. Subset the gene and exon matrices by the appropriate transcript cluster ID or exon probe IDs
c. Call the exon.ni function
d. Store the output p-values, test statistics, and CIs from this function for each transcript cluster ID (and exon probe IDs)

For this loop, you also need to add an if statement to only conduct this calculation when you have at least 2 rows in the d.exon matrix (i.e. at least 2 exon probe IDs for a single transcript cluster ID). This loop will take up to an hour to run (depending on your computer speed), so plan accordingly.

```
# two t-test called by exon ni function below
t.two <- function(x,sam,v=F) {
  x <- as.numeric(x)
  out <-
    t.test(as.numeric(x[sam]),as.numeric(x[!sam]),alternative="two.sided",var.equal=v)
  o <- as.numeric(c(out$statistic,out$p.value,out$conf.int[1],out$conf.int[2]))
  names(o) <- c("test_statistic","pv","lower_ci","upper_ci")
  return(o)
}

# exon ni function
exon.ni <- function(genex,exonx,rx) {
  ni <- t(t(exonx)-genex)
  ttest <- t(apply(ni,1,t.two,sam=as.logical(rx),v=F))
  return(ttest)
}


pvalues <- data.frame(test_statistic=as.numeric(), pv=as.numeric(),
                      lower_ci=as.numeric(), upper_ci=as.numeric())
for (uniqId in u_filt) {
  ex <- dimnames(map[map$transcript_cluster_id %in% uniqId,])[[1]]
  d.exon <- e[ex,]
  d.gene <- g[uniqId,]
  if(dim(d.exon)[[1]] > 2) {
    ni.out <- exon.ni(genex=as.numeric(d.gene),exonx=d.exon,rx=r)
    pvalues <- rbind(pvalues, ni.out)
  }
}
```

6.) Sort the matrix where you stored all of the summary statistics by p-value and get the transcript cluster ID with the lowest p-value.

```
sig_transcript <- rownames(pvalues[order(pvalues$pv),][1,])
sig_transcript
```

The transcript cluster ID with the lowest p-value is 2426958, with p-value of 9.756647e-07.

7.) Run the exon plot function with this transcript cluster ID and provide the plot in your solutions. You will need to redefine d.exon and d.gene to do this.

```
#boxplot of exons for gene 2426958 (sig_transcript)
plot.exons <- function(exonx,genex,rx,ti) {
  rr <- rx
  rx <- rep(rx,nrow(exonx))
  rx[rx==1] <- "A"
  rx[rx==0] <- "B"
  rx <- as.factor(rx)
  ni <- t(t(exonx)-genex)
  exonx <- as.data.frame(t(ni))
  ex.stack <- stack(exonx)
  d <- data.frame(ex.stack,rx)
  names(d) <- c("exon_values","exon_id","class")

  d$exon_id <- as.factor(d$exon_id)
```

```
  d$class <- as.factor(d$class)
  genex.title <-
as.character(map[match(ti,as.character(map$transcript_cluster_id)),"gene
_assignment"])

plot(c(.5,(ncol(exonx)+.5)),range(d[,1]),type="n",axes=F,xlab="",ylab=""
)

boxplot(exon_values~exon_id,add=T,subset=d$class=="A",d,col="salmon",bor
der='red',cex.axis=.75,las=2,ylab='Log2 normalized
intensity',main=paste("Gene ID:",ti,"\n",genex.title),boxwex=0.4)

boxplot(exon_values~exon_id,subset=d$class=="B",d,add=T,col="green",bord
er='darkgreen',axes=F,boxwex=0.4, at=c(1:ncol(exonx))+0.1)
}

d.exon <- e[sig_exon,]
d.gene <- g[sig_transcript,]
plot.exons(exonx=d.exon,genex=as.numeric(d.gene),rx=r,ti=sig_transcript)
```