The Process

Madison Hobbs

6/19/2017

Table of Contents

1. Background

The goal of the research of Dr. Stoebel, Dr. Hardin, and their many contributors is to assess how the level of gene expression of each gene in the genome of E. coli is influenced by RpoS, a protein which is known to regulate the stress response of E. coli (you can find a lot more information on the RpoS wikipedia page). See the prior research with three RpoS levels in Wong et al. (2017).

2. RNA-Seq, Breifly

RNA Sequencing (RNA-seq) is a way to measure gene expression in an organism. The RNA is extracted and RNA reads (sequences of A, G, C, and U) are aligned to the organism's genome. The higher the number of reads mapped to a certain gene, the more we can say that gene is expressed. The gene annotations come in different features, CDS (coding sequence), ncRNA, rRNA and tRNA. IGR is another feature which represents intergenic regions, spaces between two genes.

The features rRNA and tRNA are removed as much as possible from the RNA that gets sequenced, but you will likely still find reads of those two features (though hopefully in small proportions). If you're working in a data set where the feature type is not specified, and each gene appears only once, each gene's read count is likely the sum of sens and antisens CDS reads. To keep it simple, in this demonstration we'll work with that type of situation.

3. Start With the Raw Read Counts

```
allCounts <- read.csv("DMS2670_LB_NC_000913.tsv", header = T, sep = "\t")
#collect the geneids
allCounts$GeneidBackup = allCounts$Geneid
# note that the geneids here are really long and contain a lot of information. We are going to parse th
# make a separate column for gene feature (CDS, AS_CDS, IGR, etc).
allCounts <- allCounts %>% separate(GeneidBackup, c("feature", "rest"), sep="[:]")
```

```
## Warning: Too many values at 14108 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
allCounts <- allCounts %>% select(Geneid, feature,
         "0.00_A" = A0, # relabel the columns to reflect RpoS level, not arabinose
         "0.00_B" = B0,
         "0.00_C" = C0,
         "0.35_A" = A10..5,
         "0.35_B" = B10..5,
         "0.35_C" = C10..5,
        "11.59_A" = A5x10.5,
         "11.59_B" = B5x10.5,
        "11.59_C" = C5x10.5,
        "20.40_A" = A10..4,
        "20.40_B" = B10..4,
        "20.40_C" = C10..4,
        "48.37_A" = A10..3,
        "48.37_B" = B10..3,
        "48.37_C" = C10..3,
       "100.00_A" = A2537,
       "100.00_B" = B2537,
       "100.00_C" = C2537,
       "129.96_A" = A2x10..3,
       "129.96_B" = B2x10..3,
       "129.96_C" = C2x10..3,
       "190.38_A" = A5x10..3,
       "190.38_B" = B5x10..3,
       "190.38_C" = C5x10..3)

allCounts %>% group_by(feature) %>% summarise(number_of_genes = n())
```

```
## # A tibble: 10 x 2
##     feature number_of_genes
##       <chr>           <int>
##  1   AS_CDS            4382
##  2   AS_IGR            2496
##  3 AS_ncRNA              65
##  4  AS_rRNA              22
##  5  AS_tRNA              89
##  6      CDS            4382
##  7      IGR            2496
##  8    ncRNA              65
##  9     rRNA              22
## 10     tRNA              89
```

Above, we see the breakdown by feature in our data. The AS before the first five rows simply means "anti-sens" meaning that the read was read backwards, so to speak, but it still belongs to the feature after the "_". For every read in our data, there an antisens read. CDS means coding sequence; these will have bnumbers (bnum). IGR stands for intergenic region; these are regions between genes. ncRNA stands for non-coding RNA. rRNA is ribosomal RNA and tRNA is transfer RNA; these are both removed as much as possible lest the dominate the read counts, but some slip in as we can see.

```
# Now, we must extract the genenames from each Geneid. However, each feature has a slightly different p

# IGR's (this includes AS_IGRSs):
# IGR stands for intergenic region which means a region between coding sequences or different types of
bnum = "b[0-9]{4}" # what do bnumbers look like?
genename = ",[a-z]{3}[A-Z,]." # what does a genename look like? this is regexp lingo
```

```
rna.name = ",rna[0-9].." # what does an RNA name look like?
igr <- allCounts %>% filter(feature %in% c("IGR", "AS_IGR"))
igr$GeneidBackup = igr$Geneid # store the Geneid
igr <- igr %>% separate(GeneidBackup, c("Geneid1", "Geneid2"), sep = "[/]") # separate the first part o
igr$feature1 <- separate(igr, Geneid1, c("feature1", "rest"), sep = "[,]")$feature1
```

```
## Warning: Too many values at 4992 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
igr$feature1 <- separate(igr, feature1, c("rest", "feature1"), sep = "[()]")$feature1 #start feature
igr$feature2 <- separate(igr, Geneid2, c("feature2", "rest"), sep = "[,]")$feature2
```

```
## Warning: Too many values at 4992 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
igr$start.gene <- case_when( #start gene name: many possibilities!
    igr$feature1 == "CDS" ~ str_extract(igr$Geneid1, genename), #if the start feature was CDS, then the
    TRUE ~ str_extract(igr$Geneid1, rna.name)) #otherwise, it's going to have an RNA-style name, so we
igr$end.gene <- case_when( #end gene neame: similar to above!
    igr$feature2 == "CDS" ~ str_extract(igr$Geneid2, genename), # if the end feature was CDS, then we'r
    TRUE ~ str_extract(igr$Geneid2, rna.name)) #otherwise, it must be an RNA-style label of some sort.
igr$start.bnum <- case_when(
    igr$feature1 == "CDS" ~ str_extract(igr$Geneid1, bnum), #bnums only exist for CDS, so we check if t
    TRUE ~ "none") # if not CDS, then no bnum exists so we can put "none"
igr$end.bnum <- case_when(
    igr$feature2 == "CDS" ~ str_extract(igr$Geneid2, bnum), #same thing as above but for end bnum
    TRUE ~ "none")
# now get rid of all those pesky commas that got into our start.gene labels. I could have not included
igr <- igr %>% separate(start.gene, into = c("comma", "start.gene"), sep = "[,]") %>% select(-comma) %>%
```

```
## Warning: Too many values at 4818 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, ...
```

```
## Warning: Too many values at 4800 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 11,
## 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, ...
```

```
allCounts <- full_join(igr, allCounts) #add this new information to allCounts!
```

```
## Joining, by = c("Geneid", "feature", "0.00_A", "0.00_B", "0.00_C", "0.35_A", "0.35_B", "0.35_C", "11
```

What we get back from RNA Sequencing is a data table where each row represents a gene and each column represents an experimental condition and its sample (for instance, column A1 could be condition A, sample 1). Hopefully, your data has more than one sample per condition. These are called "replicates," and they help us know what is really going on at each condition (the more the better!). Each observation in this large data table is the number of reads for that row's gene at that column's sample.

By "raw," I mean unnormalized. See below for a description of normalization.

4. Quality Control

Think about quality control issues and in what ways you might want to clean your data. For us, we decided to remove the 100% condition because the E. coli used in all other conditions had an rssB deletion, and thus would not be comparable. We also decided to remove the columns (samples) for which the median read count across the entire sample was below 10. We also removed rows (genes) for which the maximum read count across all samples was under 5. The logic was that with read counts so low, these samples and genes would

not allow us to reliably distinguish between signal and noise. See more about this and other concerns with the data used in this demonstration in Questions_Regarding_2017_Data_Simplified.

5. Normalize All Counts

```
# keep only genes whose maximum raw count is greater than 5 and exclude the samples `0.00_B`, `0.35_B`,
allCounts.tidy <- allCounts.tidy %>% group_by(Geneid) %>%
  filter(max(rawCount) > 5) %>% filter(cond.samps %in% c("0.00_B", "0.35_B", "20.40_A", "100.00_A", "10
#remomve the conditions we don't want

# create wide, messy version of this which DESeq2 needs. Filter out the unwanted columns but not the un
allCounts <- allCounts %>% select(Geneid, feature, `0.00_A`, `0.00_C`,
         `0.35_A`, `0.35_C`,
         `11.59_A`, `11.59_B`, `11.59_C`,
         `20.40_B`, `20.40_C`,
         `48.37_A`, `48.37_B`, `48.37_C`,
         `129.96_A`, `129.96_B`, `129.96_C`,
         `190.38_A`, `190.38_B`, `190.38_C`) %>%
  filter(Geneid %in% allCounts.tidy$Geneid) #keep only the rows that satisfied the condition above

countsTable.allCond <- allCounts %>% select(`0.00_A`, `0.00_C`,
 `0.35_A`, `0.35_C`,  `11.59_A`, `11.59_B`, `11.59_C`, `20.40_B`, `20.40_C`,
`48.37_A`, `48.37_B`, `48.37_C`, `129.96_A`, `129.96_B`, `129.96_C`,
`190.38_A`, `190.38_B`, `190.38_C`)

# Define the conditions.
conditions.allCond <- as.factor(c("0.00", "0.00", "0.35", "0.35", "11.59", "11.59", "11.59", "20.40", "
coldata.allCond <- as.data.frame(row.names = colnames(countsTable.allCond), conditions.allCond)
# create DESeqDataSet
dds.allCond <- DESeqDataSetFromMatrix(countData = countsTable.allCond, colData = coldata.allCond, desig
# DESeq(dds.allCond) produces the normalized counts, among other things
dds.allCond <- DESeq(dds.allCond)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

```
# the part we really care about: extract the normalized counts
normalizedCounts <- as.data.frame(counts(dds.allCond, normalized = TRUE)) %>%
  mutate(Geneid = allCounts$Geneid) # re-attach the gene names

#make tidy normalized count data
normCounts.tidy <- normalizedCounts %>% gather(cond.samps, normCount, -Geneid)

# add the normalized counts to our big allCounts.tidy data table
allCounts.tidy <- inner_join(allCounts.tidy, normCounts.tidy)
```

```
## Joining, by = c("Geneid", "cond.samps")
```

```
allCounts.tidy <- allCounts.tidy %>%
  separate(cond.samps, c("RpoS%", "replicate"), sep = "[_]") %>%
  mutate(`RpoS%` = as.numeric(`RpoS%`)) %>%
  group_by(Geneid, `RpoS%`) %>%
  mutate(levelMean = mean(normCount)) %>% #create level mean which is the mean normCount between the re
  ungroup() %>%
  group_by(Geneid) %>%
  mutate(scaler = ifelse(levelMean[1] < levelMean[18], levelMean[18], levelMean[1]),
         #scale by the maximum between levelMean at the lowest RpoS condition (levelMean[1]) and levelM
         normCountScaled = normCount/scaler, #scale each normCount by scaler
         meanScaled = levelMean/scaler) %>% #scale all the level means by the scaler
  ungroup()
```

We must normalize the counts because each sample is sequenced at a different depth in RNA Seq. The sequencing depth of each sample refers to the total raw read count (column sum) of that sample. If we simply used the raw read counts from the data table given us by RNASeq, we wouldn't be comparing apples to apples because of that difference in sequencing depth. For instance, if the gene smpB has 3 reads at condition 20% RpoS sample A and has 87 reads at condition 60% RpoS sample A, it might appear that smpB is really differentially expressed across conditions 20% and 60%. But perhaps condition 60% sample A has a much higher sequencing depth than condition 20% sample A. Normalization accounts for problems like this. There are many different techniques, they are really fun to think about, and you should read about them and the theory behind them in Dr. Hardin's, Dr. Stoebel's, and Ciaran Evan's paper (Evans (2016)). You can also read more about the process of DESeq in the appendix of Ciaran Evan's thesis (4).

We use DESeq's normalization technique which is a standard method for this type of analysis. DESeq normalizes in the following way: Start with one sample. For each gene, take the raw count of that gene at that sample over the geometric mean of the raw read count for that gene across all samples. Do this for all genes, and you get a sample-specific vector of numbers. Take the median of that vector, and you have the "size factor" for your sample. Size factors essentially tell us how deeply the sample is sequenced, and the ratio of the size factors of two different samples is an approximation of the ratio of their sequencing depths.

6. Find Your Regulon Using Differential Expression Analysis

```
# Make sure you have loaded the DESeq2 package
# compare 0 to 190.38
countsTable.0.190.38 <- allCounts %>%
  select(`0.00_A`, `0.00_C`, `190.38_A`, `190.38_B`, `190.38_C`)
rownames(countsTable.0.190.38) <- allCounts$Geneid
# define conditions
condition.0.190.38 <- as.factor(c("0.00", "0.00", "190.38", "190.38", "190.38"))
# we only want the relevant column name from countsTable for each condition (we don't want Geneid or ge
rownames <- as.data.frame(colnames(countsTable.0.190.38))
coldata.0.190.38 <- data.frame(row.names = colnames(countsTable.0.190.38), condition.0.190.38)
# create the DESeqDataSet. cds stands for "count data set"
dds.0.190.38 <- DESeqDataSetFromMatrix(countData = countsTable.0.190.38, colData = coldata.0.190.38, de
# estimate size factors, dispersions, etc, and testing differential expression
dds.0.190.38 <- DESeq(dds.0.190.38)
#estimating dispersion by treating samples as replicates
```

We are interested only in the genes that RpoS "regulates," or the genes whose gene expression RpoS affects. Although RpoS, as a primary regulator of E. coli's general stress response, regulates many genes in the E.

coli genome, about two thirds of the genes will be unaffected by RpoS.

As is consistent with previous studies, we define the "regulon" (genes regulated by RpoS) as genes which exhibit differential expression between 0% RpoS (knock-out, where the gene rpoS coding for the protein RpoS is literally knocked out of the specimen's genome) and 100% RpoS (wild-type (WT), or the level of RpoS in the natural, wild E. coli). In the cases where we do not have a 100% RpoS level, as is true in this data, we differentially express between the lowest and highest conditions (in our case, 0% vs 190.38%). Differential expression analysis is a technique we use to assess how differnently each gene is being expressed across the experimental conditions. We use the R package DESeq2 from Bioconductor (Love, Huber, and Anders (2014), see footnote 1).

DESeq first estimates "size factors" for each sample. As mentioned previously, the size factors for, say, sample 0.00_A are calculated in the following way. For imaginary gene uboD, for instance, uboD's raw read count for 0.00_A is divided by the geometric mean of uboD's raw read counts across 0.00_A, 0.00_B, 100.00_A, and 100.00_B We obtain these values for every gene, take the median of those values, and the result is our size factor for the sample 0.00_A. Didn't we already normalize, you may ask? When we remove the other samples to keep only 0% and 190.38%, the size factors are going to change. We want our decision about regulation between 0% and 190.38% RpoS based only on the counts for those two conditions, not affected by different sequencing depths of other samples. It is those normalized counts we produce which shall be compared to teset for significant difference across the conditions.

Next, DESeq estimates "dispersions," which is a measure of how much the variance deviates from the mean. DESEq uses a negative binomial probability model to perform hypothesis testing (Wald test) on differential expression between conditions, and this model has two parameters: mean and dispersion. The negative binomial probability model can be thought of as a generalized Poisson probability model, because in Poisson, dispersion is 1 so mean = variance (Maechler et al. (2017)). Finally, DESeq2 fits a negative binomial probability model to perform hypotheis testing, using a Wald test, to find a p-value for each gene being differentially expressed across the two conditions. Thesse p-values are FDR-adjusted (Benjamini-Hochberg) for multiple comparisons to control the rate of false discoveries (see footnote 2).

Using the results() function from DESeq2 (see below), we extract the p-values and FDR-adjusted p-values. The fuction summary() is very useful in giving you the number of genes differentially expressed between the two conditions. LFC stands for the log of the fold change between two conditions (where fold change is normalized counts at condition 2 divided by normalized counts at condition 1; for instance, if the noramlized counts for gene uboD at 190.38% RpoS was twice that at 0% RpoS, we would have a fold change of 2, and the log fold change would be log(2) = 0.69). Positive LFC's show that a gene's expression increased from condition 1 to 2 (upregulation), and negative LFC's show that a gene's expression decreased from condition 1 to 2 (downregulation). Summary() summarizes the number of genes which, at the specified confidence level, show significant differential expression across the two condtions, outlining how many of these are upregulated, downregulated, outliers, or low counts.

```
#get results
results.0.190.38 <- results(dds.0.190.38, alpha = 0.05)
summary(results.0.190.38)
```

```
##
## out of 10374 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)     : 1056, 10%
## LFC < 0 (down)   : 793, 7.6%
## outliers [1]     : 1, 0.0096%
## low counts [2]   : 2816, 27%
## (mean count < 5)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

The results summary shows the number and percentage of genes up- and down-regulated from 0% to 190.38%;

these will form our "regulon."

```
# extract the information about the regulated genes and store the information with the table of allCoun
Geneids <- rownames(countsTable.0.190.38)
regulated.genes <- as.data.frame(results.0.190.38) %>% mutate(Geneid = Geneids, regulation = ifelse(log
regulated.genes <- filter(regulated.genes, padj < 0.05)

#modify allCounts to include regulation information
allCounts <- left_join(allCounts, regulated.genes, by = "Geneid")
```

```
## Warning: Column `Geneid` joining factor and character vector, coercing into
## character vector
```

```
#modify allCounts.tidy.filterd to include regulation information
allCounts.tidy <- left_join(allCounts.tidy, regulated.genes, by = "Geneid")
```

```
## Warning: Column `Geneid` joining factor and character vector, coercing into
## character vector
```

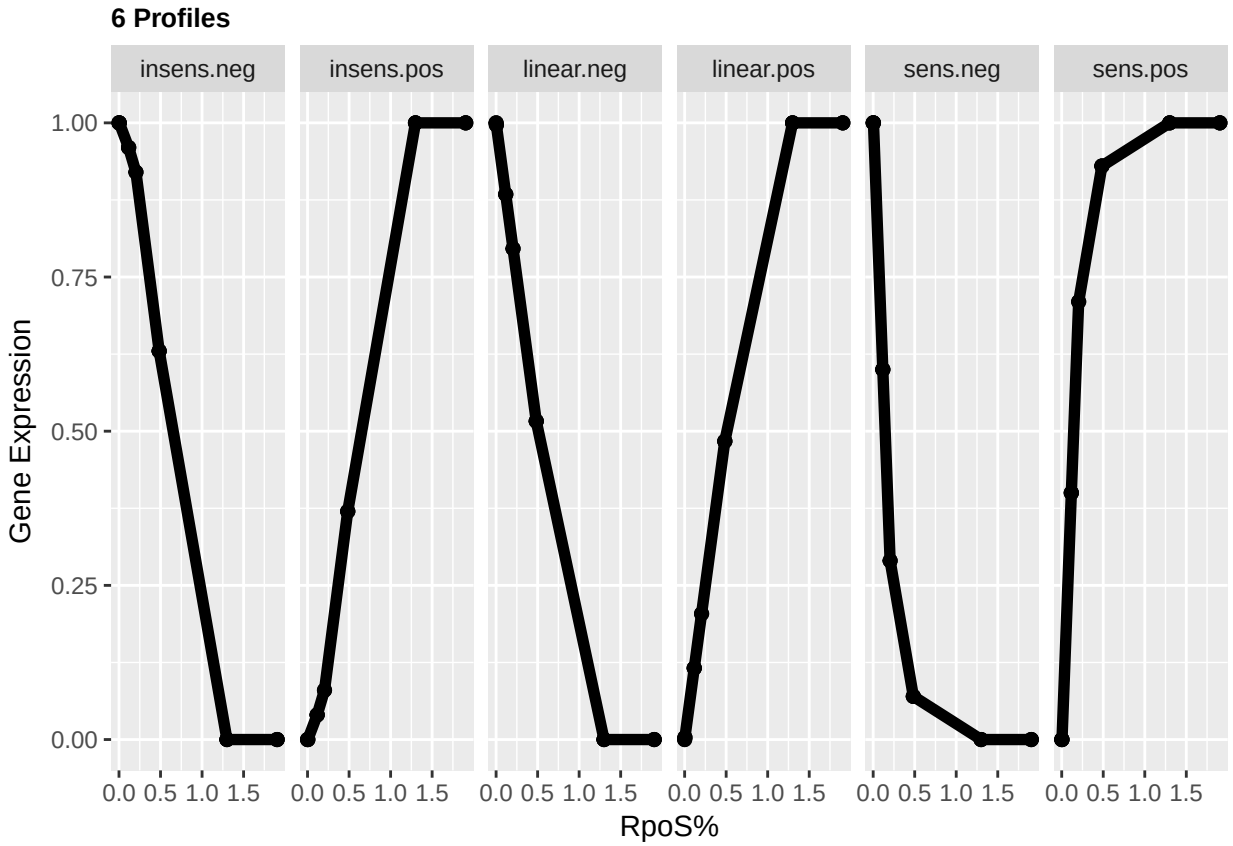
7. Grouping Methods

Profile Assignment

One approach is to create one's one gene expression profiles based on questions one has about the gene expression shapes. For example, the underlying question we have been trying to answer is how sensitive genes are to increased levels of RpoS; is gene expression affected even at low levels of RpoS, or does it take more RpoS to make an impact on gene expression? To these ends, we have designed six ad-hoc profiles (sensitive positive, sensitive negative, insensitive positive, insensitive negative, linear positive, linear negative) to categorize gene expression, shown below. However, if you are interested in a different question, you can design any number of your own profiles.

Note: All of our plots of gene expression represent gene expression as a proportion of the highest mean normalized count between 0% replicates and 190.38% replicates. The y-axis will represents this proportion of gene expression (normCountScaled) and the x-axis represents the percentage of RpoS inputed (RpoS%).

```
#extrapolated from profiles as seen in Madison's reproduction and reanalysis of Garrett's code (Re-Rewr
profile.linear.pos <- c(0,0, 0.0035,0.0035, 0.1159,0.1159,0.1159, 0.2040,0.2040, 0.4837,0.4837,0.4837,
profile.linear.neg <- c(1,1, 0.9965,0.9965, 0.8841,0.8841,0.8841,
0.7960, 0.7960, 0.5163,0.5163,0.5163, 0,0,0, 0,0,0)
profile.sens.pos <- c(0,0, 0,0, 0.4,0.4,0.4, 0.71,0.71, 0.93,0.93,0.93, 1,1,1, 1,1,1)
profile.sens.neg <- c(1,1, 1,1, 0.6,0.6,0.6, 0.29,0.29, 0.07,0.07,0.07, 0,0,0, 0,0,0)
profile.insens.pos <- c(0,0, 0,0, 0.04,0.04,0.04, 0.08,0.08, 0.37,0.37,0.37, 1,1,1, 1,1,1)
profile.insens.neg <- c(1,1, 1,1, 0.96,0.96,0.96, 0.92,0.92, 0.63,0.63,0.63, 0,0,0, 0,0,0)

# data frame of all six profiles side by side
profile.6 <- data.frame(sens.pos = profile.sens.pos, sens.neg = profile.sens.neg, insens.pos = profile.

#make profiles data table tidy
profiles.tidy <- profile.6 %>% mutate(`RpoS%` = c(0,0, 0.0035,0.0035, 0.1159,0.1159,0.1159, 0.2040,0.204
# plot the six profiles
ggplot(profiles.tidy, aes(x = `RpoS%`, y = gene.expression)) + geom_point(size = 2) + geom_line(size = 1
```



Next, we take each gene and correlate its expression (normCount) with each of the profiles above. The gene is then sorted with the profile for which it was most correlated.

```
# gather normalized counts which we will be correlating with the profiles
normCounts.forCorr <- allCounts.tidy %>%
  filter(regulation != "NA") %>%
  select(Geneid, `RpoS%`, replicate, normCount) %>%
  unite(cond.samps, `RpoS%`, replicate) %>%
  reshape2::dcast(cond.samps ~ Geneid, value.var = "normCount") %>%
  select(-1)

# correlate normalized counts with each of the six profiles, and sort each gene with the profile with w
profile.normCounts.forCorr <- as.data.frame(t(cor(profile.6, normCounts.forCorr)))
rownames <- rownames(profile.normCounts.forCorr)
rowMax <- rowMax(as.matrix(profile.normCounts.forCorr))
profile.normCounts.forCorr <- profile.normCounts.forCorr %>%
  mutate(Geneid = rownames,
         maxCorr = rowMax,
         group = ifelse(maxCorr == linear.pos, "linear.pos",
                        ifelse(maxCorr == linear.neg, "linear.neg",
                               ifelse(maxCorr == insens.pos, "insens.pos",
                                      ifelse(maxCorr == insens.neg, "insens.neg",
                                             ifelse(maxCorr == sens.pos, "sens.pos", "sens.neg"))))))

# add info to allCounts.tidy
allCounts.tidy <- left_join(allCounts.tidy, profile.normCounts.forCorr, by = "Geneid")
#how many in each category?
allCounts.tidy %>% filter(regulation != "NA") %>%
```

```
  group_by(group)  %>% summarise(n()/18)
```

```
## # A tibble: 5 x 2
##        group `n()/18`
##        <chr>    <dbl>
## 1 insens.pos        3
## 2 linear.neg        1
## 3 linear.pos       10
## 4   sens.neg      790
## 5   sens.pos     1045
```

```
allCounts.tidy %>% filter(regulation != "NA") %>%
  group_by(regulation, group)  %>% summarise(n()/18)
```

```
## # A tibble: 6 x 3
## # Groups:   regulation [?]
##   regulation      group `n()/18`
##        <chr>      <chr>    <dbl>
## 1   negative insens.pos        2
## 2   negative linear.neg        1
## 3   negative   sens.neg      790
## 4   positive insens.pos        1
## 5   positive linear.pos       10
## 6   positive   sens.pos     1045
```

```
#plot!
ggplot(data = filter(allCounts.tidy, regulation != "NA"), aes(x = `RpoS%`, y = normCountScaled)) + geom_
```

Here are good things to look at: correlation strength and differentiability.

```
summary(allCounts.tidy$maxCorr) # the correlations used to sort each gene
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.11    0.58    0.66    0.65    0.74    0.94  153486
```

```
summary(filter(allCounts.tidy, maxCorr == sens.neg)$maxCorr) #look at sensitive negative correlations i
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1085  0.5913  0.7045  0.6788  0.8062  0.9368
```

```
#check differentiability between sensitive negative and linear negative
summary(filter(allCounts.tidy, maxCorr == sens.neg)$maxCorr - filter(allCounts.tidy, maxCorr == sens.neg
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.004769 0.182500 0.214200 0.210400 0.246700 0.353200
```

Intermediate Differential Expression Testing

If our question is whether a gene is so sensitive that increasing RpoS from 0% to 11.59% has an impact on gene expression, we can use differential expression analysis to answer this question. This alternative is the only that offers p-values and weeds out genes with inter-replicate variability in a way that the previous and following methods do not.

```
# compare 0 to 0.1159
countsTable.0.1159 <- allCounts %>%
  select(`0.00_A`, `0.00_C`,`11.59_A`, `11.59_B`,  `11.59_C`)
# define conditions
condition.0.1159 <- as.factor(c("0.00", "0.00", "11.59",  "11.59", "11.59"))
coldata.0.1159 <- data.frame(row.names = colnames(countsTable.0.1159), condition.0.1159)
rownames(countsTable.0.1159) = allCounts$Geneid
# create the DESeqDataSet. cds stands for "count data set"
cds.0.1159 <- DESeqDataSetFromMatrix(countData = countsTable.0.1159, colData = coldata.0.1159, design =
# estimate size factors, dispersions, etc, and testing differential expression
cds.0.1159 <- DESeq(cds.0.1159)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

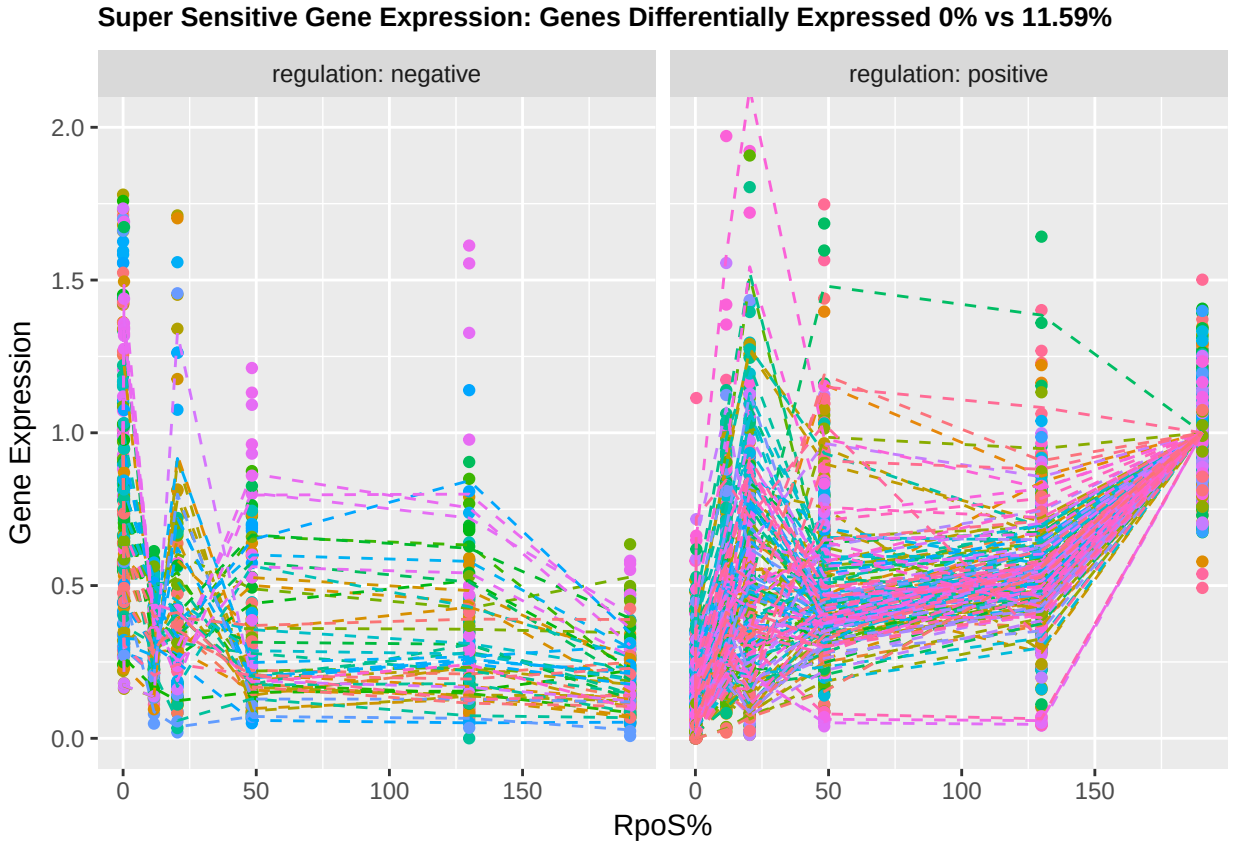
```
## fitting model and testing
```

```
#estimating dispersion by treating samples as replicates
#get results
results.0.1159 <- results(cds.0.1159, alpha = 0.05/2) #bonferroni adjustment because we did the compari
summary(results.0.1159)
```

```
##
## out of 10370 with nonzero total read count
## adjusted p-value < 0.025
## LFC > 0 (up)     : 189, 1.8%
## LFC < 0 (down)   : 53, 0.51%
## outliers [1]     : 0, 0%
```

```
## low counts [2]    : 5830, 56%
## (mean count < 17)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
# extract the information about the genes DE 0% vs 11.59% and store the information with the table of a
Geneids <- rownames(countsTable.0.1159)
superSensitive.genes <- as.data.frame(results.0.1159) %>% mutate(Geneid = Geneids, direction.0.1159 = i
superSensitive.genes <- filter(superSensitive.genes, padj < 0.05/2,
                               Geneid %in% regulated.genes$Geneid) %>% #keep only significantly DE gene
  select(Geneid, direction.0.1159)
#modify allCounts to include regulation information
allCounts <- left_join(allCounts, superSensitive.genes, by = "Geneid")
#modify allCounts.tidy.filterd to include regulation information
allCounts.tidy <- left_join(allCounts.tidy, superSensitive.genes, by = "Geneid")

DE.0.1159.tidy <- filter(allCounts.tidy, direction.0.1159 != "NA")
#plot
ggplot(data = DE.0.1159.tidy, aes(x = `RpoS%`, y = normCountScaled, col=Geneid)) + geom_point() + geom_l
```




```
# I noted that the genes positive between 0% and 11.59% are the same genes positive between 0% and 190.
```

Note that we have to adjust for multiple comparisons with this approach (I use Bonferroni above). One could continue this method to produce all possible combinations and patterns to categorise the genes based on thier differential expression across different RpoS levels.

PAM Clustering (Partitioning Around Medoids)

Another approach is to let an algorithm like PAM (Kaufman and Rousseeuw (1990)) find the most typical gene expression patterns for us, and group genes based on these. This answers a slightly different question. Instead of asking "how many genes look most like this particular shape I'm interested in" as above, now we ask in an unsupervised way "what are the most common shapes in my data?"

PAM works by finding the k observations (genes) which make the tightest and most differentiable clusters. These k observations are called "medoids," and all genes in the data are assigned to the closest of the k medoids. There are a variety ways we can measure "closest," but we choose to use correlation because we are more interested in differences between the shapes of gene expression than in the euclidean distance, for instance, between plotted points.

```
# make wide format that the PAM algorithm likes
countsToCluster <- select(allCounts.tidy, Geneid, `RpoS%`, replicate, normCount) %>%
  unite(cond.samp, `RpoS%`, replicate, sep = "_") %>% spread(cond.samp, normCount)
# only keep the genes that are in the regulon
countsToCluster <- countsToCluster %>%  filter(Geneid %in% regulated.genes$Geneid)
# save the Geneids and make these the names of the rows, keeping only the columns with counts.
Geneids <- countsToCluster$Geneid
countsToCluster <- countsToCluster %>% select(-Geneid)
rownames(countsToCluster) <- Geneids
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```
#generate dissmilarity matrix based on spearman correlation
dissimilarity.matrix <- 1 - cor(t(countsToCluster), method = "spearman")

#one iteration of PAM with k = 6 (6 clusters)
pam.6 <- pam(dissimilarity.matrix, 6)
pam.6.medoids <- data.frame(rownames(pam.6$medoids),  pam.6$silinfo$clus.avg.widths)
# contains the names of the genes which are the medoids for each cluster (around which each cluster is
pam.6.clusters <- data.frame(pam.6$clustering)
Geneids <- rownames(pam.6.clusters) # extract Geneids
pam.6.clusters <- pam.6.clusters %>% mutate(Geneid = Geneids)
allCounts.tidy <- left_join(allCounts.tidy, pam.6.clusters) #store information with allCounts.tidy
```

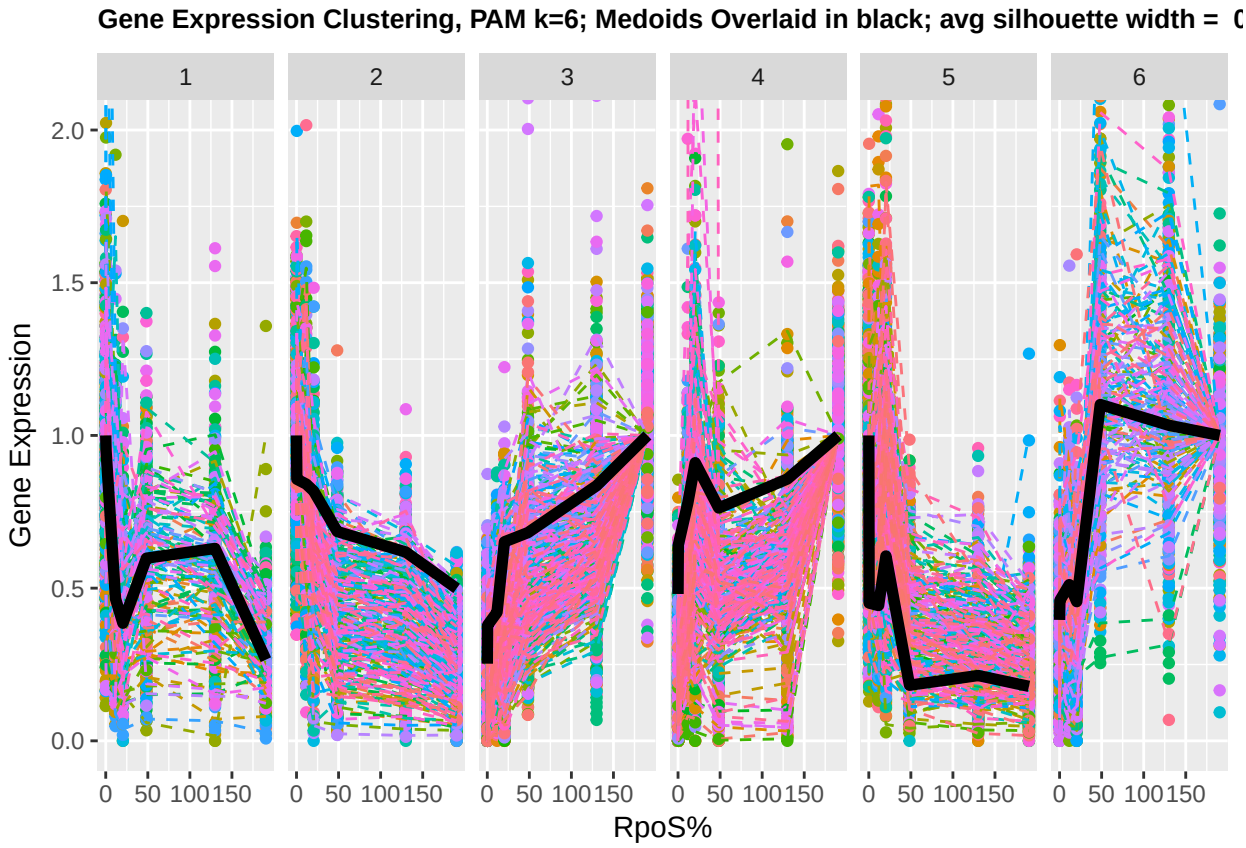
```
## Joining, by = "Geneid"
```

```
# just the regulated genes to plot
regulated.tidy <- filter(allCounts.tidy, Geneid %in% regulated.genes$Geneid)

regulated.tidy %>% group_by(regulation, pam.6.clustering) %>% summarise(numGenes = n()/18) %>% arrange(
```

```
## # A tibble: 8 x 3
## # Groups:   regulation [2]
##   regulation pam.6.clustering numGenes
##        <chr>            <int>    <dbl>
## 1   negative                1      139
## 2   positive                1        1
## 3   negative                2      335
## 4   positive                3      488
## 5   positive                4      380
## 6   negative                5      319
## 7   positive                5        1
## 8   positive                6      186
```

After letting PAM cluster the genes into 6 groups, we see the break-down above of the number of genes in each cluster and how many were found positively and negatively regulated by differential expression analysis on 0% versus 190.38%. The clustering agrees almost perfectly with the separation pertaining to up- and down-regulation found by differential expression.

```
# find medoids normCounts to plot
medoids.toplot <- filter(regulated.tidy, Geneid %in% pam.6.medoids$rownames.pam.6.medoids.)

#plot
ggplot(data = regulated.tidy, aes(x = `RpoS%`, y = normCountScaled, col=Geneid)) + geom_point() + geom_
```




```
#Compare this plot to the plot with k=6 in Grouping_Genes_by_Patterns_of_Gene_Expression - the medoids
```

In the plot above, we see the six clusters PAM produced plotted in terms of their Rpos level and gene expression. The medoid gene with which they were clustered plotted us over each cluster in black.

Average silhouette width, measured between -1 and 1, is a metric of how well each gene is classified into each cluster. The closer this value is to 1, the more differentiable the clusters which indicates stronger evidence that there are distinct k distinct clusters in the data. Low and negative average silhouette width suggests you have chosen too many or too few clusters. The average silhouette width for k = 6 is 0.320, which is not too low, but also not as high as we might like.

A next step would be to let PAM cluster on a range of values of k, examining silhouette width as well as the plots themselves. Even if we found higher silhouette width with another k value, this is just telling us the number of clusters that gives the most prominent difference between clusters. We may still want to think of our data in terms six clusters because we are interested in the six most typical gene expression shapes exhibited.

We can also compare grouping methods using the Adjusted Rand Index, or ARI which measures how similarly two clustering methods classified objects. An ARI of -1 would suggest that the two methods are in perfect

disagreement, and an ARI of 1 would suggest that the methods are in perfect agreement. An ARI of 0 would suggest that the two methods are no better matched than they would be by random chance (based on the hypergeometric probability distribution).

```
paste("ARI 6-profile assignment vs PAM k = 6: ", round(adjustedRandIndex(allCounts.tidy$pam.6.clusterin
```

```
## [1] "ARI 6-profile assignment vs PAM k = 6:  0.368"
```

Taking the ARI between the profile assignment method and the clustering method, we see it is low which makes sense given that profile assignment categorized the bulk of the genes into one of two groups (sensitive positive and sensitive negative).

Footnotes

1. DESeq2 from Bioconductor

- Download the package here: https://bioconductor.org/packages/release/bioc/html/DESeq2.html

- Read this vignette; it will be super useful! https://www.bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

2. When we perform multiple hypothesis tests, our type I error rate (false positives) increases. You may be familiar with the Bonferroni adjustment for multiple comparisons, which simply multiplies the p-value resulting from each test by the number of tests we performed total. However, when we perform many tests as is done in differential expression hypothesis testing, Bonferroni would give a far too large of p-values to be significant at a 0.05 confidence level. Benjamini and Hochberg developped a different, less conservative, adjustment to control the false discovery rate (FDR), or type I error rate, directly. This method takes the p-value generated from each hypothesis test and multiplies it by the total number of observational units on which the tests are performed (in our case, genes) divided by the number of observational units (genes) whose unadjusted p-value is below the confidence level (i.e.: alpha = 0.05).

References

Evans, Ciaran. 2016. "Normalization of Rna-Seq Data in the Case of Asymmetric Differential Expression." http://pages.pomona.edu/~jsh04747/Student%20Theses/CiaranEvans16.pdf.

Kaufman, Leonard, and Peter Rousseeuw. 1990. *Finding Groups in Data*. John Wiley & Sons, Inc.

Love, Michael I, Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for Rna-Seq Data with Deseq2." *Genome Biology* 15 (12): 550. doi:10.1186/s13059-014-0550-8.

Maechler, Martin, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. 2017. *Cluster: Cluster Analysis Basics and Extensions*.

Wong, Garrett T, Richard P Bonocora, Alicia N Schep, Suzannah M Beeler, Anna J Lee Fong, Lauren M Shull, Lakshmi E Batachari, et al. 2017. "Genome-Wide Transcriptional Response to Varying Rpos Levels in Escherichia Coli K-12." *Journal of Bacteriology* 199 (7). Am Soc Microbiol: e00755–16.