AbxD01 Analysis

Alyx Schubert

September 29, 2014

# Introduction

This is a digital notebook to accompany the paper titled, “*” that will be published in* . It was written in R markdown and converted to html using the R knitr package. This enables us to embed the results of our analyses directly into the text to allow for a completely reproducible data analysis pipeline. A github repository is available where you can pull down your own version of the notebook to modify our analysis or adapt it to your analysis.

This document was generated using mothur v.1.33 and R.

# Prcessing 16S gene sequence data in mothur

## First, get the required files.

Sequence files can be found here \_.

The following files can be found through the mothur wiki: silva.bacteria.fasta trainset9\_032012.pds.tax trainset9\_032012.pds.fasta

The following files can be found on the github repository for this notebook: abxD01.files

## Now for the curation pipeline.

These sequences were generated using the Illumina MiSeq Platform. For data processing, we followed the MiSeq SOP outlined in the [Kozich et al. manuscript](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3753973/). These commands come from the outline described in the [MiSeq SOP](http://www.mothur.org/wiki/MiSeq_SOP).

make.contigs(file=abxD01.files, processors=14)  
summary.seqs(fasta=abxD01.trim.contigs.fasta, processors=14)  
screen.seqs(fasta=abxD01.trim.contigs.fasta, group=abxD01.contigs.groups, summary=abxD01.trim.contigs.summary, maxambig=0, maxlength=275, processors=14)  
unique.seqs(fasta=abxD01.trim.contigs.good.fasta)  
count.seqs(name=abxD01.trim.contigs.good.names, group=abxD01.contigs.good.groups)  
summary.seqs(count=abxD01.trim.contigs.good.count\_table)  
#Make sure silva.bacteria.fasta file in folder  
pcr.seqs(fasta=silva.bacteria.fasta, start=11894, end=25319, keepdots=F, processors=14)  
system(mv silva.bacteria.pcr.fasta silva.v4.fasta)  
summary.seqs(fasta=silva.v4.fasta)  
align.seqs(fasta=abxD01.trim.contigs.good.unique.fasta, reference=silva.v4.fasta, processors=14)  
summary.seqs(fasta=abxD01.trim.contigs.good.unique.align, count=abxD01.trim.contigs.good.count\_table)  
screen.seqs(fasta=abxD01.trim.contigs.good.unique.align, count=abxD01.trim.contigs.good.count\_table, summary=abxD01.trim.contigs.good.unique.summary, start=1968, end=11550, maxhomop=8)  
summary.seqs(fasta=current, count=current)  
filter.seqs(fasta=abxD01.trim.contigs.good.unique.good.align, vertical=T, trump=.)  
unique.seqs(fasta=abxD01.trim.contigs.good.unique.good.filter.fasta, count=abxD01.trim.contigs.good.good.count\_table)  
pre.cluster(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.fasta, count=abxD01.trim.contigs.good.unique.good.filter.count\_table, diffs=2)  
chimera.uchime(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.count\_table, dereplicate=t, processors=14)  
remove.seqs(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.fasta, accnos=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.accnos)  
summary.seqs(fasta=current, count=current)  
#Make sure the reference and training set files are in the analysis folder  
classify.seqs(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count\_table, reference=trainset9\_032012.pds.fasta, taxonomy=trainset9\_032012.pds.tax, cutoff=80)  
remove.lineage(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count\_table, taxonomy=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.taxonomy, taxon=Chloroplast-Mitochondria-unknown-Archaea-Eukaryota)  
remove.groups(count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.count\_table, fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.fasta, taxonomy=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.taxonomy, groups=mock1-mock2-mock3-mock4-mock5-mock6-mock7-mock8)  
  
#renamed fasta, count\_table, taxonomy to have shorter names  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy abxD01.final.taxonomy)  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count\_table abxD01.final.count\_table)  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta abxD01.final.fasta)  
  
set.current(fasta=abxD01.final.fasta, count=abxD01.final.count\_table, taxonomy=abxD01.final.taxonomy)

## Creating phylotypes

Continuing the pipeline outlined on the mothur wiki's MiSeq SOP.

#PHYLOTYPE analysis:  
#Note: label=1 is the genus level, which is the finest you can do, then label=2 is family and so forth  
phylotype(taxonomy=abxD01.final.taxonomy)  
make.shared(list=abxD01.final.tx.list, count=abxD01.final.count\_table)  
count.groups()  
#The following size 1625 was chosen based on the results of count.groups(). I tried to minimize the number of samples at Day 0 that were dropped at that minimum cutoff.  
sub.sample(shared=abxD01.final.tx.shared, size=1625)  
classify.otu(list=abxD01.final.tx.list, count=abxD01.final.count\_table, taxonomy=abxD01.final.taxonomy)  
#tx.2=family, tx.3=order, etc  
get.relabund(shared=abxD01.final.tx.2.subsample.shared)   
get.relabund(shared=abxD01.final.tx.3.subsample.shared)  
get.relabund(shared=abxD01.final.tx.4.subsample.shared)   
get.relabund(shared=abxD01.final.tx.5.subsample.shared)   
get.oturep(column=abxD01.final.dist, list=abxD01.final.tx.list, label=2-3-4-5, fasta=abxD01.final.fasta, count=abxD01.final.count\_table)

## Creating OTUs

It is worth noting that while phylotype analysis followed the listed protocol exactly, we deviated from the listed SOP for the OTU analysis. This was due to the extensive size of this dataset and limitations in our computing abilities. Thus, in a **separate** folder we copied over the following files from earlier in the protocol: abxD01.trim.contigs.good.unique.good.filter.unique.fasta abxD01.trim.contigs.good.unique.good.filter.count\_table trainset9\_032012.pds.fasta trainset9\_032012.pds.tax

**If this isn't done in a new folder, you will write over all your previous files because they have the same name.**

#Changed diffs from 2 to 3  
pre.cluster(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.fasta, count=abxD01.trim.contigs.good.unique.good.filter.count\_table, diffs=3)  
chimera.uchime(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.count\_table, dereplicate=t, processors=14)  
remove.seqs(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.fasta, accnos=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.accnos)  
summary.seqs(fasta=current, count=current)  
#Make sure the reference and training set files are in the analysis folder  
classify.seqs(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count\_table, reference=trainset9\_032012.pds.fasta, taxonomy=trainset9\_032012.pds.tax, cutoff=80)  
remove.lineage(fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.fasta, count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.count\_table, taxonomy=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.taxonomy, taxon=Chloroplast-Mitochondria-unknown-Archaea-Eukaryota)  
remove.groups(count=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.count\_table, fasta=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.fasta, taxonomy=abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.taxonomy, groups=mock1-mock2-mock3-mock4-mock5-mock6-mock7-mock8)  
  
#renamed fasta, count\_table, taxonomy to have shorter names  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pds.wang.pick.pick.taxonomy abxD01.final.taxonomy)  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.uchime.pick.pick.pick.count\_table abxD01.final.count\_table)  
system(cp abxD01.trim.contigs.good.unique.good.filter.unique.precluster.pick.pick.pick.fasta abxD01.final.fasta)  
  
set.current(fasta=abxD01.final.fasta, count=abxD01.final.count\_table, taxonomy=abxD01.final.taxonomy)  
  
#OTU ANALYSIS:  
#changed SOP taxlevel from 4 to 5  
cluster.split(fasta=abxD01.final.fasta, count=abxD01.final.count\_table, taxonomy=abxD01.final.taxonomy, splitmethod=classify, taxlevel=5, cutoff=0.20)  
make.shared(list=abxD01.final.an.unique\_list.list, count=abxD01.final.count\_table, label=0.03)  
count.groups()  
  
#Note: the size varies based on what you see on count.groups, need to check the distribution of day 0 groups and clindamycin group  
sub.sample(shared=abxD01.final.an.unique\_list.shared, size=1625)  
classify.otu(list=abxD01.final.an.unique\_list.list, count=abxD01.final.count\_table, taxonomy=abxD01.final.taxonomy, label=0.03, cutoff=80)  
get.relabund(shared=abxD01.final.an.unique\_list.0.03.subsample.shared)  
get.oturep(column=abxD01.final.dist, list=abxD01.final.an.unique\_list.list, label=0.03, fasta=abxD01.final.fasta, count=abxD01.final.count\_table)  
collect.single(shared=abxD01.final.an.unique\_list.0.03.subsample.shared, calc=chao-invsimpson-sobs-simpsoneven-shannon-jclass, freq=100)  
rarefaction.single(freq=100)  
#Note summary.single--use the original shared file, and put subsample as same as before, will subsample 1000 times and get avgs  
summary.single(calc=nseqs-coverage-chao-sobs-invsimpson-simpsoneven-shannoneven-shannon, shared=abxD01.final.an.unique\_list.shared, subsample=1625)

# Data Analysis and Paper Figures

## Figure 1

This figure was built using the following R code. The input files are listed in Github.

#################################################  
#Purpose: Convert csv matrix of columns you want converted into barcharts  
#input: csv file following this heading format format: sampleID, expgroup, nextDayCFU, Otu001, Otu002, Otu003, etc  
#ex:file<-read.csv("~/Desktop/mothur/abxD01/barcharts/abxD01.final.tx.5.subsample.relabund.topdose.forlogscale2.csv  
#Note: It is assumed that anything after the expgroup column are vectors containing data to be split by the groups in 'expgroup'   
# and averaged with SD measured.   
#Output: matrices avg[], std[], bp[y] with the barplot coordinates info stored for each subgraph  
#################################################  
# Parameters to change:  
# CSV file: Group expgroup Otu001... (limited by most abund, end with 'Other', OTUs normalized +0.0001, expgroups #'d by graph order & sorted by first graph)  
file<-read.csv("~/Desktop/mothur/abxD01/barcharts/abxD01.final.tx.1.subsample.1.pick.relabund.topdose2.forlogscale.csv", header=T)  
fileIDS<-read.csv("~/Desktop/mothur/abxD01/barcharts/topdose\_tx1\_barchart\_ids.csv", header=T)  
# Y Labels for each graph:   
abx<-c("Untreated", "Ciprofloxacin", "Clindamycin", "Vancomycin", "Streptomycin", "Cefoperazone", "Ampicillin", "Metronidazole")  
  
# If you want each OTU on the Y axis to be sorted by the most abundant phylum and then decreasing abundance within that phylum, change to TRUE  
# The default is false, which means sort be decreasing relative abundance in the top group (untreated/control)  
sortbyphyl<-TRUE  
  
# Highlight all and run!  
#################################################  
  
file<-file[,-1] #delete the first col of group names  
avgs=NULL  
avgs <-data.frame(levels(file$expgroup))  
colnames(avgs) <- c("expgroup")  
stds=NULL  
stds <-data.frame(levels(file$expgroup))  
colnames(stds) <- c("expgroup")  
  
  
#Calculate average cdiff  
avgCD=NULL  
sdCD=NULL  
avgCD<-tapply(file$nextDayCFU, file$expgroup, mean)  
avgCD<-format(avgCD, scientific=TRUE, digits=2)  
sdCD<-tapply(file$nextDayCFU, file$expgroup, sd)  
file<-file[,-2] #delete the nexDayCFU column  
  
  
for(i in 1:ncol(file)){  
   
 if(i==1){  
 ids<-names(file)  
 }  
 else{  
 columni <- data.frame(file[,i])  
 colnames(columni) <- c("columndata")  
   
 #calculates mean for each column in file by expgroup and stores it in avgs data frame  
 poop<- aggregate(columni$columndata~file$expgroup, FUN=mean) #poop stores the aggregated means for each group for this specific column  
 colnames(poop) <- c(ids[1], ids[i])  
   
 avgs<- merge(avgs, poop, by ="expgroup")  
   
 #calculates st dev for each column in file by expgroup and stores it in stds data frame  
 poop2<- aggregate(columni$columndata~file$expgroup, FUN=sd) #poop stores the aggregated means for each group for this specific column  
 colnames(poop2) <- c(ids[1], ids[i])  
   
 stds<- merge(stds, poop2, by ="expgroup")  
 }  
}  
  
ordIDS <- fileIDS[order(fileIDS[,2]), ] #sort by the phyla  
totalOTU <- dim(ordIDS)[1]  
  
########for doing sort by phylum then within phylum (by relabund)  
if(sortbyphyl == TRUE){  
 ordIDS<-cbind(ordIDS, 0)  
   
 for (i in 1:totalOTU){ #loop to fill out the final column for the phyla groups, while phyla is sorted!!  
 if(i==1){  
 ordIDS[i, 4] <- 1  
 }  
 else{  
 if(ordIDS[i, 2]==ordIDS[i-1, 2]){  
 ordIDS[i, 4] <- ordIDS[i-1, 4]  
 }  
 else{  
 ordIDS[i, 4] <- ordIDS[i-1, 4] + 1  
 }  
   
 }  
 }  
   
 names(ordIDS)[4]<-paste("phynum")  
   
 ##add row for phylum number to the avgs?  
 #avgs<-rbind(avgs, 0)[c(9, 1, 2, 3, 4, 5, 6, 7, 8),] #this is hard coded for the top doses... also dont rerun this multiple times!!! might need to change  
 avgs<-rbind(0,avgs) #this is hard coded for the top doses... also dont rerun this multiple times!!! might need to change, will give error  
   
 numphyla<-max(ordIDS[,4])  
   
 #test\_avg <- avgs[, rev(order(avgs[1,2:length(avgs)-1]))] #sort all but other column (last)  
   
 physums <- data.frame(c(1:numphyla))  
 physums <- cbind(physums, 0)  
 colnames(physums) <- c("Num", "Sum")  
   
 leng<-length(avgs)  
   
   
 #jtime<-NULL  
 #itime<-NULL  
 #inside<-FALSE  
 #count<-0  
 j<-1  
 ##This loop is to fill the physums data frame with the sum of the relabund for each phylum represented  
 for(i in 2:(leng-1)){ #searching through avgs, so leng is number of otus, -1 because i dont want to deal with "Other" column  
 found<-FALSE  
 #itime<-rbind(itime, i)  
 j<-1  
 while(found==FALSE){ #searching through ordIDS  
 test<-ordIDS[j, 1] == names(avgs)[i]  
 if(test) {   
 pnum<-ordIDS[j, 4]  
 found<-TRUE  
 inside<-TRUE  
 avgs[1,i]<-pnum  
 }  
 else{  
 j <- j + 1  
 #count<-count+1  
 #jtime<-rbind(jtime, j)  
   
   
 }  
 } #WARNING: could cause an error if dont find the ID, theres no check for the end of the list  
   
 physums[pnum, 2] <- physums[pnum, 2] + avgs[2, i] ##add the relabund of the otu   
 }  
   
   
 y<-rev(order(physums$Sum))  
 x<-avgs[1, 2:(leng-1)]  
   
   
 ord\_physums<-physums[rev(order(physums[,2])),]  
 sort\_avgs<-data.frame(avgs$expgroup)  
 names(sort\_avgs)[1]<-c("expgroup")  
   
 for(i in 1:numphyla){ #loop is to sort by decreasing phyla and perform internal sort within phylum, return the new sorted avgs to go into mavgs  
 val<-NULL  
 phy<-which(x %in% c(ord\_physums[i,1])) #phy returns the indices for where that phyNum is  
 for(j in 1:length(phy)){ #using locations of phylum i to find in avgs  
 val[j]<-avgs[2,phy[j]+1]  
 }  
 phy<-phy[rev(order(val))]  
   
 for(j in 1:length(phy)){ #now loop through the sorted sub cols and add them to the sort\_avgs  
 sort\_avgs<-cbind(sort\_avgs, avgs[,phy[j]+1])  
 names(sort\_avgs)[length(sort\_avgs)]<-names(avgs)[phy[j]+1]  
 }  
 }  
   
 attach(avgs)  
 sort\_avgs<-cbind(sort\_avgs, Other)  
 detach(avgs)  
   
 sort\_avgs <- sort\_avgs[-1,] #get rid of phylum first row from earlier sorting  
   
 #loop for getting the right ids in the right order as sort  
 ids<-data.frame(1:(totalOTU))  
   
 row.names(sort\_avgs)<-sort\_avgs$expgroup  
 sort\_avgs<-sort\_avgs[,-1]  
   
 for (i in 1:(length(sort\_avgs)-1)){  
 found<-FALSE  
 #itime<-rbind(itime, i)  
 j<-1  
 while(found==FALSE){ #searching through ordIDS  
 test<-ordIDS[j, 1] == names(sort\_avgs)[i]  
 if(test) {   
 ids[i,2]<-ordIDS[j, 3]  
 found<-TRUE  
 }  
 else{  
 j <- j + 1  
 #count<-count+1  
 #jtime<-rbind(jtime, j)  
 }  
 }  
   
 }  
   
   
 names(ids)<-c("x", "name")  
 ids<-rbind(ids, data.frame(x=totalOTU+1, name= "Other"))  
   
   
 mavgs<-as.matrix(sort\_avgs)  
   
 leng<-dim(mavgs)[2]  
   
 numgr <- nlevels(file$expgroup)  
} ##end if(sortbyphyl == TRUE){  
  
  
####order the results by the first group alpha numerically  
if(sortbyphyl == FALSE){  
   
 row.names(avgs)<-avgs$expgroup  
 avgs<-avgs[,-1]  
 ordered\_avgs<- avgs[, rev(order(avgs[1,1:length(avgs)-1]))] #sort all but other column (last)  
 attach(avgs)  
 ordered\_avgs <- cbind(ordered\_avgs, Other) #put other back on  
 detach(avgs)  
   
 mavgs<-as.matrix(ordered\_avgs)  
 leng<-dim(mavgs)[2]  
   
 numgr <- nlevels(file$expgroup)  
   
   
 ids<-data.frame(1:(totalOTU))  
   
 for (i in 1:(length(ordered\_avgs)-1)){  
 found<-FALSE  
 #itime<-rbind(itime, i)  
 j<-1  
 while(found==FALSE){ #searching through ordIDS  
 test<-ordIDS[j, 1] == names(ordered\_avgs)[i]  
 if(test) {   
 ids[i,2]<-ordIDS[j, 3]  
 found<-TRUE  
 }  
 else{  
 j <- j + 1  
 #count<-count+1  
 #jtime<-rbind(jtime, j)  
 }  
 }  
   
 }  
   
   
 names(ids)<-c("x", "name")  
 ids<-rbind(ids, data.frame(x=totalOTU+1, name= "Other"))  
   
   
  
} #end if(sortbyphyl == FALSE){  
  
  
###started to do the stds table... didnt complete  
#stds<-stds[,-1]  
#mstds<-as.matrix(stds)  
  
  
###PLOT PARAMETERS  
par(mfrow=c(numgr+1, 1)) #+1 to give extra labeling space  
par(mar=c(0.3, 8, 0.5, 2) +0.1, mgp=c(4.5, 1, 0)) #default is 5.1 4.1 4.1 2.1, bot/left/top/right, also default mgp is c(3,1,0)  
  
  
  
for(j in 1:numgr){  
   
 if(j != numgr){  
 barplot(mavgs[j, 1:leng], ylab=NULL, col="black", yaxt="n", xaxt="n", ylim=c(0.0001, 1), log="y", cex.names=3)  
 #error.bar(bp[k], mavgs[j, 1:leng[2]], mstds[j, 1:leng[2]]) #the bp[k] was for storing the barplot locations to use for these errors  
 # k <- k+1  
 axis(2, las=1, at=c(.0001, .001, .01, .1, 1), labels=c(0, .001, .01, .1, 1), cex.axis=1.1)  
 mtext(abx[j], side=2, line=6, cex=.8)  
 mtext(avgCD[j], side=2, line=4.5, cex=.8)  
 abline(h=c(0.0001, 1), lwd=3)  
 abline(h=c(0.001, 0.01, 0.1), col="black", lty="longdash", lwd=1.5)  
 abline(h=c(.25,.5, .75), col="black", lty="dashed")  
 }  
   
 else{ #the last graph needs different margins   
 label<-barplot(mavgs[j, 1:leng], col="black",ylab=NULL, yaxt="n", xaxt="n", ylim=c(0.0001, 1), log="y", cex.names=3)  
 #error.bar(bp[k], mavgs[j, 1:leng[2]], mstds[j, 1:leng[2]]) #the bp[k] was for storing the barplot locations to use for these errors  
 axis(2, las=1, at=c(.0001, .001, .01, .1, 1), labels=c(0, .001, .01, .1, 1), cex.axis=1.1)  
 mtext(abx[j], side=2, line=6, cex=.8)  
 mtext(avgCD[j], side=2, line=4.5, cex=.8)  
 axis(1, at=(label[,1]), labels=FALSE)  
 text(label[,1]+.13, .00005, label=ids[,2], xpd=NA, pos=2, srt=45, cex=1.2)  
 #text(-3.9,.0001, label=expression(paste(italic("C.d."), " CFU/g Feces:")), xpd=NA, pos=2, srt=90, cex=1.1)  
 abline(h=c(0.0001,1), lwd=3)  
 abline(h=c(0.001, 0.01, 0.1), col="black", lty="longdash", lwd=1.5)  
 abline(h=c(.25,.5, .75), col="black", lty="dashed")  
   
 }   
  
}  
par(mfrow=c(1, 1))  
  
  
##############END CODE###########################  
#################################################

The graph can be sorted 2 ways either by the untreated's decreasing relabund, or by the most abundant within each phylum. For each treatment, the resulting *C. difficile* CFU/g Feces is also labeled on the Y axis below the treatment name. File saved as "topdose2\_tx1\_sorted\_10x20.pdf"

### TO DO STILL

* change the "0" to be min o 0.001 instead
* add the 1/2 dashed lines
* also wanted to try to make this table in the other direction: Phyla/OTUs as each graph with different shadings or colors for each treatment... will be good for titration graphs