

# str\_analysis\_032517.R

*maximilianpress*

*Tue Apr 25 11:41:12 2017*

```
# analyze new STR genotype data
# refactored 3/25/17
# updated substantially 12/01/16
# first pass 8/11/16
library(vegan)

## Loading required package: permute
## Loading required package: lattice
## This is vegan 2.4-1
library(kernlab)

##
## Attaching package: 'kernlab'
## The following object is masked from 'package:permute':
##
##      how
library(stringr)
library(DAAG)
library(beeswarm)
set.seed(311) # remember the 90s?

# to make rmarkdown work
if(isTRUE(getOption('knitr.in.progress'))){
  setwd('..')} else {
  setwd('~/.Dropbox/Ath_STRs/')
}

num_strs = 2050
pseudocount = .5
boot_divisor = 4 # controls size of bootstrapped samples for SVR selection inference

countnas = function(matrix) {
  nanum = c()
  for (i in rownames(matrix)) {
    nanum[i] = length(which(is.na(matrix[i,])))
  }
  return(nanum)
}

gc_content = function(str) {
  gc = str_count(str, 'G') + str_count(str, 'C')
  n=nchar(str)
  return(gc/n)
}
```

```

genos = read.table('problem_mip_genotypes_081016.txt',header=T)
info = read.csv('mip_design_troubleshoot/mip_lib_info_081116.csv',header=T,stringsAsFactors = FALSE)
#annots = read.csv('~/.Dropbox/Ath_STRs/araport_annots/str_annots.csv',header=T)
annotation = read.delim('araport_annot/Ath_STRs_full_annotations_031317.tsv',header=T)
rownames(annotation) = annotation$ID
lib_names = gsub('_R1_001.fastq.gz',' ',info[1:96,'spikein_file'])
lib_names = gsub('-', '.',lib_names)
info = info[1:96,]
rownames(info) = lib_names
info = info[colnames(genos),]
colnames(genos) = info$Strain
geography = info[, 'Region.1001genomes. ']
regions = unique(geography)
geog_colors = c('red','blue','black','dark green','purple','orange','hot pink')
names(geog_colors) = regions

# work around known crappiness
# probably not expansions because i checked a few of these and they are largely just garbage
genos[genos<0] = NA # this happens when 'impossible' lengths are found- expansions?

all_genos = genos

# really basic analysis... how many calls?
count_calls = function(gen_tab) {
  return(length(which(!(is.na(genos)))))
}

count_alleles = function(gen_tab,numcols,sum=FALSE) {
  tablature = apply(gen_tab[,1:numcols,drop=FALSE],1,table)
  lengthed = simplify2array(lapply(tablature,length))
  if (sum) {
    return(sum(lengthed))
  } else {return(lengthed)}
}

rarefy_count_alleles = function(gen_tab) {
  gen_tab=gen_tab[,sample(colnames(gen_tab))]
  rared = sapply(1:96,count_alleles,gen_tab = gen_tab, sum=TRUE)
  return(rared)
}

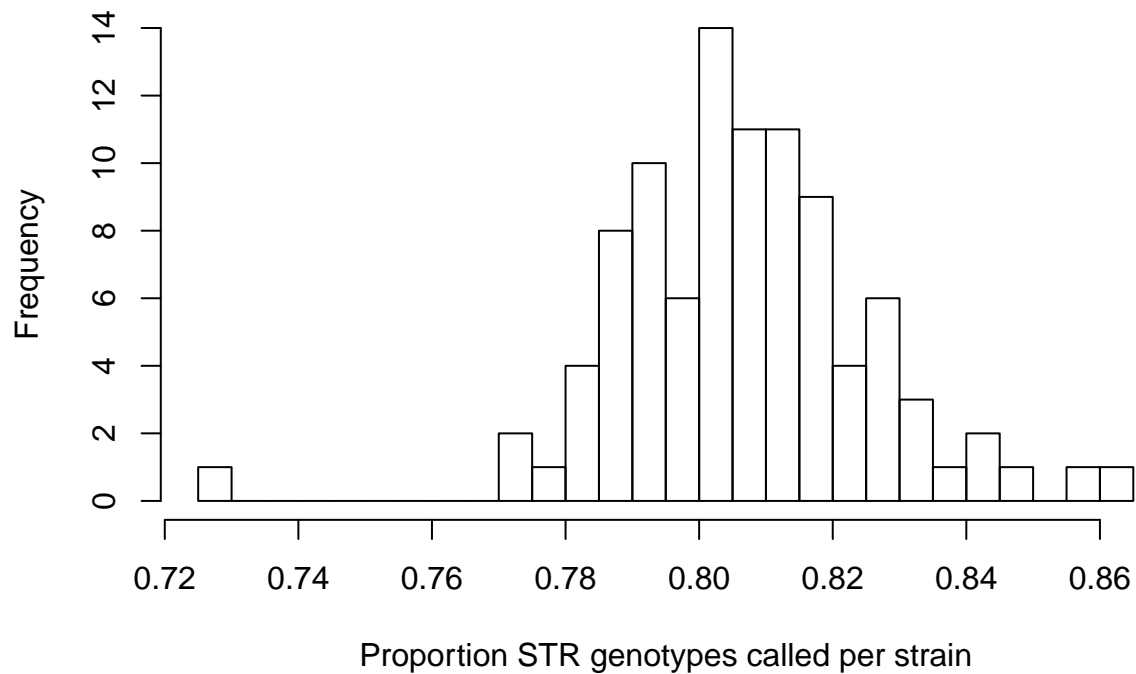
all_calls = count_calls(genos)

cat('num genotype calls total:',all_calls,'\n','proportion called:',all_calls/(num_strs*96),'\n')

## num genotype calls total: 158348
## proportion called: 0.8046138

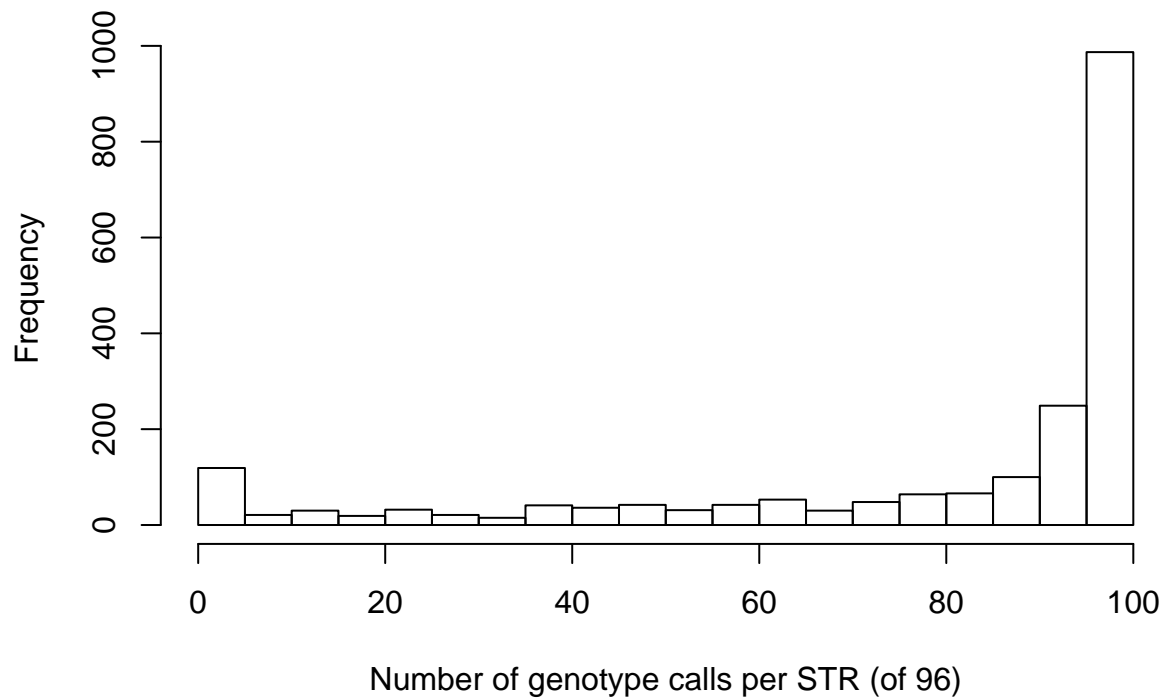
# estimate proportion of calls overall by strain
strain_ascertain = 1 - (countnas(t(genos)) / num_strs)
hist(strain_ascertain,20,xlab='Proportion STR genotypes called per strain',main='')

```



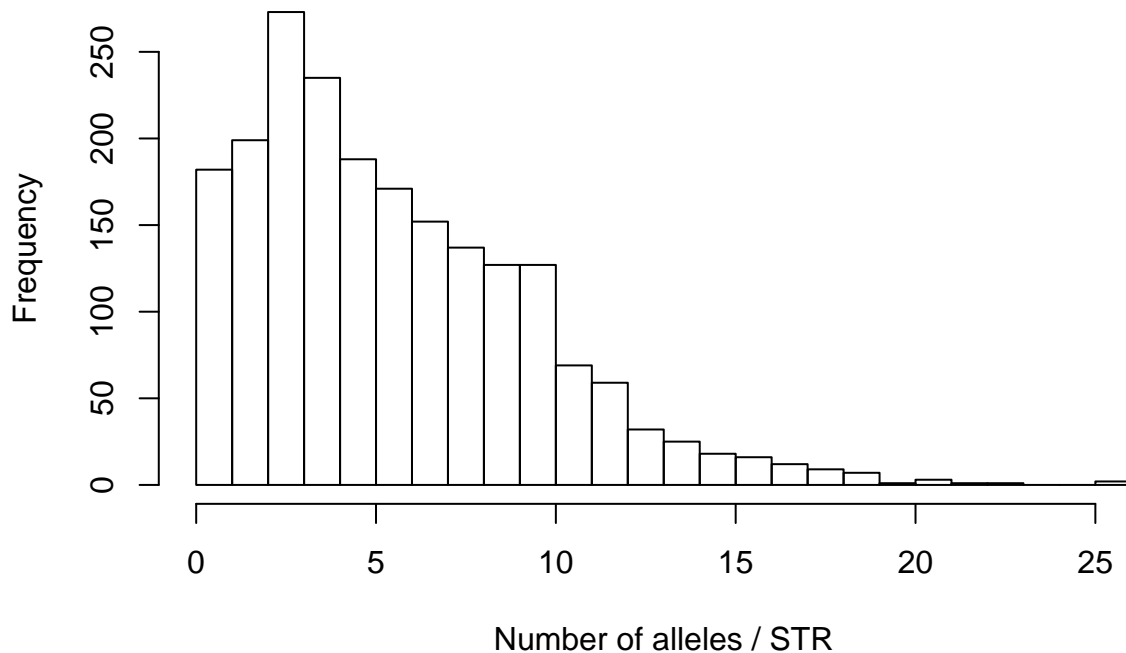
```
# by str
nacount = countnas(genos)
hist(96-as.numeric(nacount),20,xlab='Number of genotype calls per STR (of 96)',main='')

```



```
# how many alleles per repeat??
lengthed = count_alleles(genos,96)
hist(lengthed,25,xlab='Number of alleles / STR',main='')

```



```
cat('average number of alleles:',mean(lengthed),'median:',median(lengthed),'\n')
```

```
## average number of alleles: 5.981427 median: 5
```

```
cat(length(which(lengthed==0)),'strs have no genotype calls in any strain','\n')
```

```
## 78 strs have no genotype calls in any strain
```

```
# by functional annotation:
```

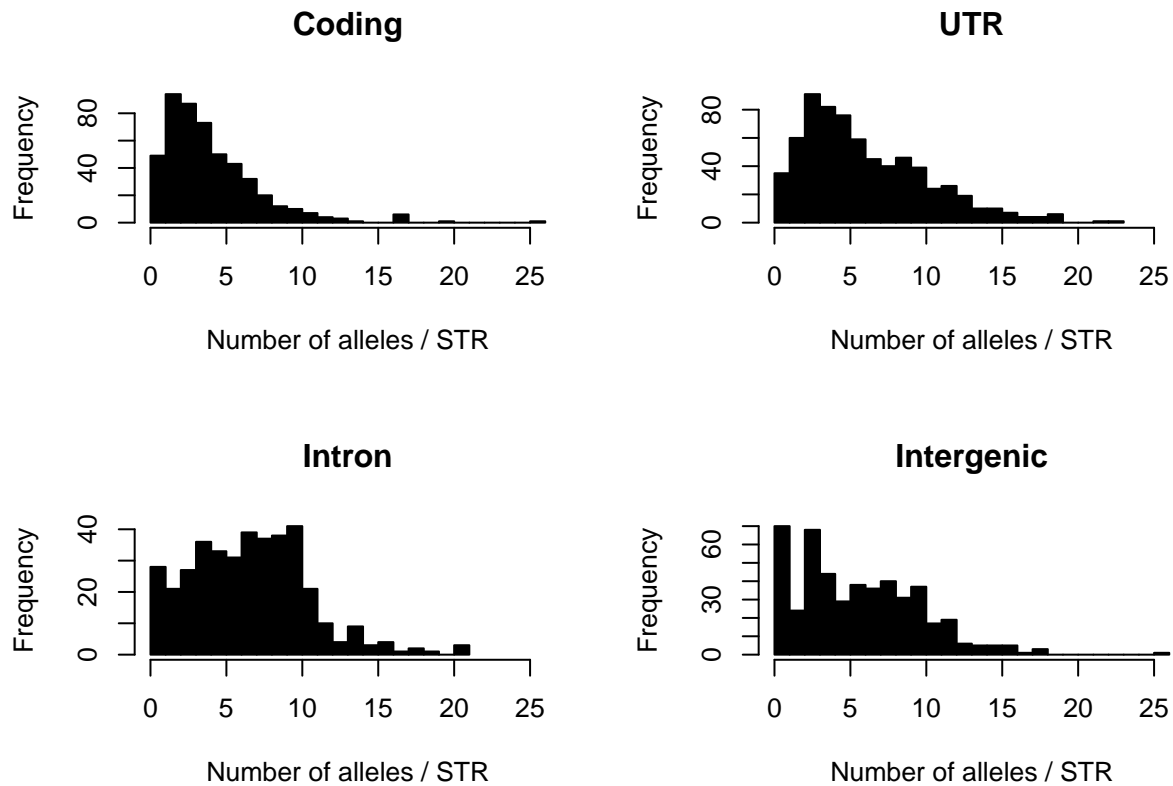
```
par(mfrow=c(2,2))
```

```
hist(lengthed[annotation[names(lengthed),'annotation'] == 'coding'], 25,  
      main='Coding', xlab = 'Number of alleles / STR', xlim = c(0,26), col='black')
```

```
hist(lengthed[annotation[names(lengthed),'annotation'] == 'UTR'], 25,  
      main='UTR', xlab = 'Number of alleles / STR', xlim = c(0,26), col='black')
```

```
hist(lengthed[annotation[names(lengthed),'annotation'] == 'intron'], 25,  
      main='Intron', xlab = 'Number of alleles / STR', xlim = c(0,26), col='black')
```

```
hist(lengthed[annotation[names(lengthed),'annotation'] == 'Intergenic'], 25,  
      main='Intergenic', xlab = 'Number of alleles / STR', xlim = c(0,26), col='black')
```



```
# composition of very badly ascertained STRs
unascertained = annotation[names(which(nacount>90)),]
cat('Motif counts among STRs with low ascertainment (5 or fewer genotypes):\n')
```

```
## Motif counts among STRs with low ascertainment (5 or fewer genotypes):
motified = sort(table(unascertained$Consensus))
print(motified[motified>0])
```

```
##
## ACCACG AGAAAA AGAGAG AGTGGA ATAT ATC TCT TCTATGT TCTT
## 1 1 1 1 1 1 1 1 1
## TTG AGA ATT CTT GAA GAAAAA TTC AAG CT
## 1 2 2 2 2 2 2 3 10
## GA TA AG AT TC
## 16 16 17 17 18
```

```
## rarefaction of alleles??
## takes a while- uncomment if desire to rerun
# num_rars = 10
# num_strains = ncol(genos)
# rar_mat = matrix(rep(NA,num_rars*num_strains),num_rars)
# plot(1,type='n',xlim=c(0,100),ylim=c(1000,sum(lengthed)), las=1, ylab='Number of alleles', xlab='Num
# #
# for (i in 1:num_rars) {
#   cat('rarefaction subsample',i,'\n')
#   rar_mat[i,] = rarefy_count_alleles(genos)
#   lines(1:96,rar_mat[i,])
# }
```

```

# plot distribution of most common allele frequency
major_allele_freq = function(x) {
  x = unlist(x)
  if ( length(which(is.na(x))) > 86 ) {return(NA)}
  commonest = sort(table(na.omit(x)),decreasing=TRUE)[1]
  freq = commonest / length(na.omit(x))
  return(freq)
}

# look at allele freq spectrum (roughly... for lots of info see Haasl and Payseur 2010 MBE)
major_allele_freqs = apply(genos,1,major_allele_freq)
plot(density(na.omit(major_allele_freqs),bw=.01),xlab='Major STR allele frequency',main='',xlim=c(0,1))
lessthan50percent = length(which(na.omit(major_allele_freqs)<=.5))
monoallelic = length(which(major_allele_freqs==1))
total = length(na.omit(major_allele_freqs))
cat(lessthan50percent/total*100,'% of STRs have major allele frequencies <=50%\n',sep='')

## 45.49738% of STRs have major allele frequencies <=50%

cat(monoallelic/total*100,'% of STRs are monoallelic\n',sep='')

## 4.502618% of STRs are monoallelic

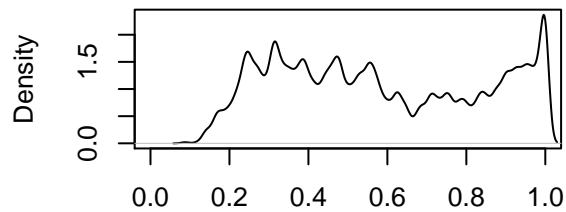
# Haasl and Payseur 2010 state that an estimate of theta (scaled  $\mu = 4N_e\mu$ ) is
#  $\theta = (1 / (8 * \text{mean}(x)^2)) - 1/2$  (Eq. 11), where  $\text{mean}(x)$  is the mean frequency of alleles at a given
# pretty straightforward i think? though the constants may change w/ n (pop sample size)... should email
# esp. because my sample size itself varies a bit!!!
theta_hat = function(x) {
  x = unlist(x)
  if ( length(which(is.na(x))) > 71 ) {return(NA)}
  allele_counts = sort(table(na.omit(x)),decreasing=TRUE)
  if (length(allele_counts) < 2) {return(NA)}
  freqs = allele_counts / length(na.omit(x))
  xbar2 = mean(freqs)^2
  thetahat = (1 / (8*xbar2)) - 1/2
  return(thetahat)
}
thetaed = apply(genos,1,theta_hat)

# plot the estimator... some really high values there!!
plot(density(na.omit(thetaed)),xlab='Theta_hat_xbar',main='')

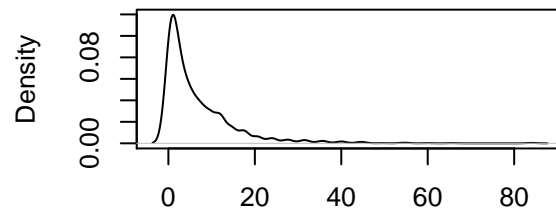
# see if weirdness is introduced by sample size. a few values are below zero (!!!) but no obvious sample
annots = annotation[rownames(genos),]
cored = cor.test(96-nacount, thetaed,na.rm=TRUE,xlim=c(0,40),xlab='# units in Col',ylab='Theta_hat')$est
plot(96-nacount,thetaed,pch='.',xlab='n',ylab='theta_hat_xbar',main=cored)

# look for correlations with other things that would predict var
# make sure that you've run the Ath_MIPs_FigS1_design.R script or version thereof to populate
# env with annots and other stuff!!
cored = cor.test(annots$X.of.Units, thetaed,na.rm=TRUE,xlim=c(0,40),xlab='# units in Col',ylab='Theta_hat')$est
plot(annots$X.of.Units, thetaed,xlim=c(5,40),xlab='# units in Col',ylab='Theta_hat',pch='.',main=cored)

```

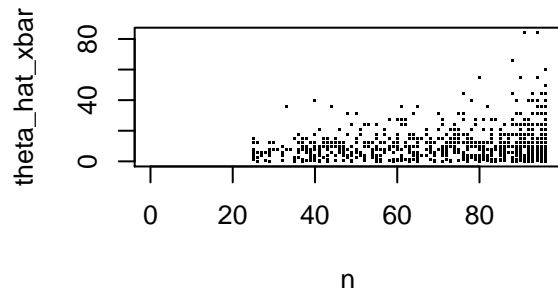


Major STR allele frequency

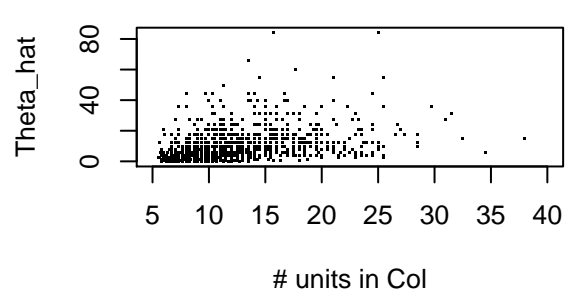


Theta\_hat\_xbar

**-0.154245248143157**



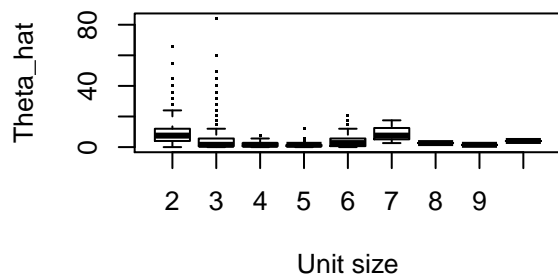
**0.491146441162808**



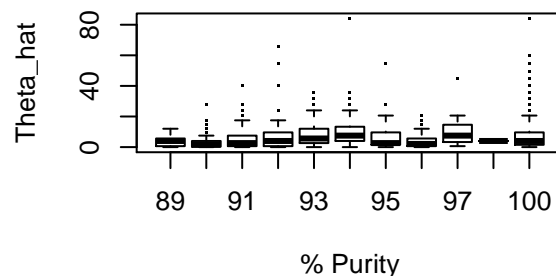
```
cored = cor.test(annots$Unit.Size, thetaed,na.rm=TRUE,xlim=c(0,40),xlab='# units in Col',ylab='Theta_hat')
boxplot(thetaed~annots$Unit.Size,xlab='Unit size',ylab='Theta_hat',pch='.',main=cored)
cored = cor.test(annots$Purity, thetaed,na.rm=TRUE,xlim=c(0,40),xlab='# units in Col',ylab='Theta_hat')
boxplot(thetaed~annots$Purity,xlab='% Purity',ylab='Theta_hat',pch='.',main=cored)

par(mfrow=c(2,3))
```

**-0.243335511105021**



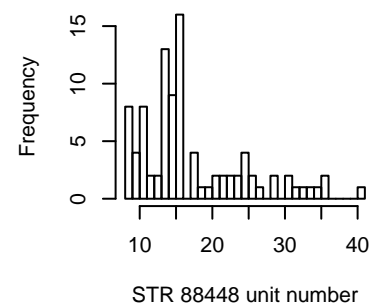
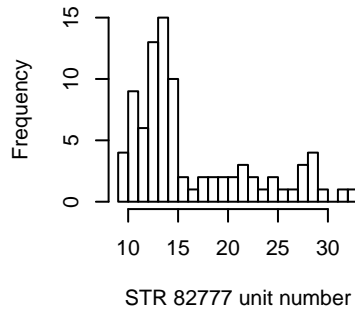
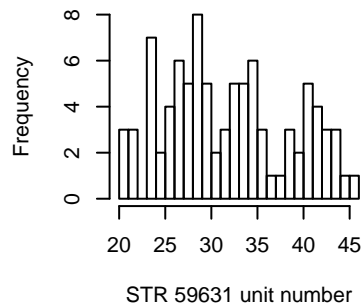
**0.116534346113293**



```
high_theta = names(na.omit(thetaed[thetaed>60]))

for (thet in high_theta) {
  hist(as.numeric(genos[thet,]),30,xlab=paste('STR',thet,'unit number'),main='')
}

par(mfrow=c(1,1))
```



```
# various ordination approaches
# nmds- from vegan package. scaling to avoid weirdness from highly var reps- necessary?
scale_dists = dist(scale(t(genos)))
dists = dist(t(genos))
nmds_scale = metaMDS(scale_dists) # that looks really weird... no convergence # now fixed?
```

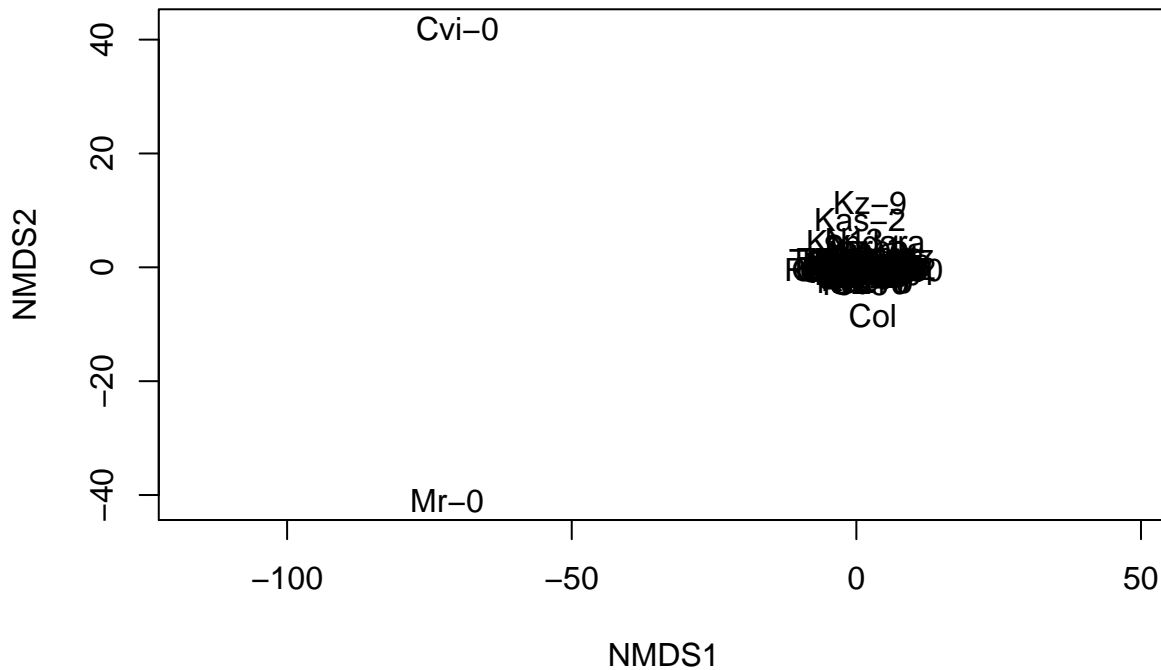
```
## Run 0 stress 0.09896567
## Run 1 stress 0.1002757
## Run 2 stress 0.1000403
## Run 3 stress 0.09956331
## Run 4 stress 0.09961784
## Run 5 stress 0.099456
## ... Procrustes: rmse 0.00851312 max resid 0.03499677
## Run 6 stress 0.09927708
## ... Procrustes: rmse 0.03548751 max resid 0.1815381
## Run 7 stress 0.09999948
## Run 8 stress 0.09988542
## Run 9 stress 0.1019252
## Run 10 stress 0.1041713
## Run 11 stress 0.4115814
## Run 12 stress 0.1005767
## Run 13 stress 0.09926888
## ... Procrustes: rmse 0.03813348 max resid 0.1800915
## Run 14 stress 0.09997857
## Run 15 stress 0.09969494
## Run 16 stress 0.1047144
## Run 17 stress 0.09931245
## ... Procrustes: rmse 0.03520996 max resid 0.1814083
## Run 18 stress 0.1008952
## Run 19 stress 0.0991431
## ... Procrustes: rmse 0.002858077 max resid 0.02579384
## Run 20 stress 0.09893238
## ... New best solution
## ... Procrustes: rmse 0.003566816 max resid 0.03243004
## *** No convergence -- monoMDS stopping criteria:
## 9: no. of iterations >= maxit
## 11: stress ratio > sratmax
```

```
plot(nmds_scale,type='n')
```

```
## Warning in ordiplot(x, choices = choices, type = type, display = display, :
## Species scores not available
```



```
text(nmds_scale, labels=rownames(nmds_scale$points))
```



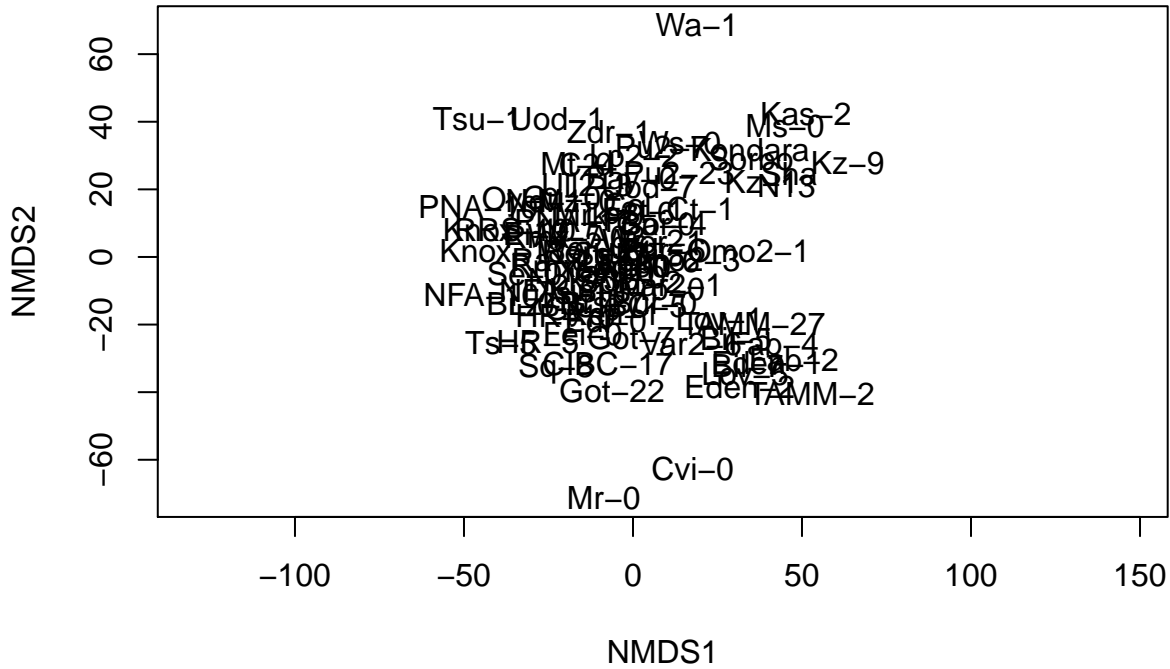
```
nmds = metaMDS(dists)
```

```
## Run 0 stress 0.2690853
## Run 1 stress 0.2797227
## Run 2 stress 0.2976633
## Run 3 stress 0.2759913
## Run 4 stress 0.2711844
## Run 5 stress 0.2724104
## Run 6 stress 0.2773354
## Run 7 stress 0.266248
## ... New best solution
## ... Procrustes: rmse 0.04115021 max resid 0.2639413
## Run 8 stress 0.2771859
## Run 9 stress 0.270384
## Run 10 stress 0.2674382
## Run 11 stress 0.4117872
## Run 12 stress 0.2943031
## Run 13 stress 0.2778138
## Run 14 stress 0.2731318
## Run 15 stress 0.281007
## Run 16 stress 0.4104651
## Run 17 stress 0.282106
## Run 18 stress 0.2663626
## ... Procrustes: rmse 0.03194254 max resid 0.1406816
## Run 19 stress 0.2785611
## Run 20 stress 0.2750292
## *** No convergence -- monoMDS stopping criteria:
##      20: stress ratio > sratmax
```

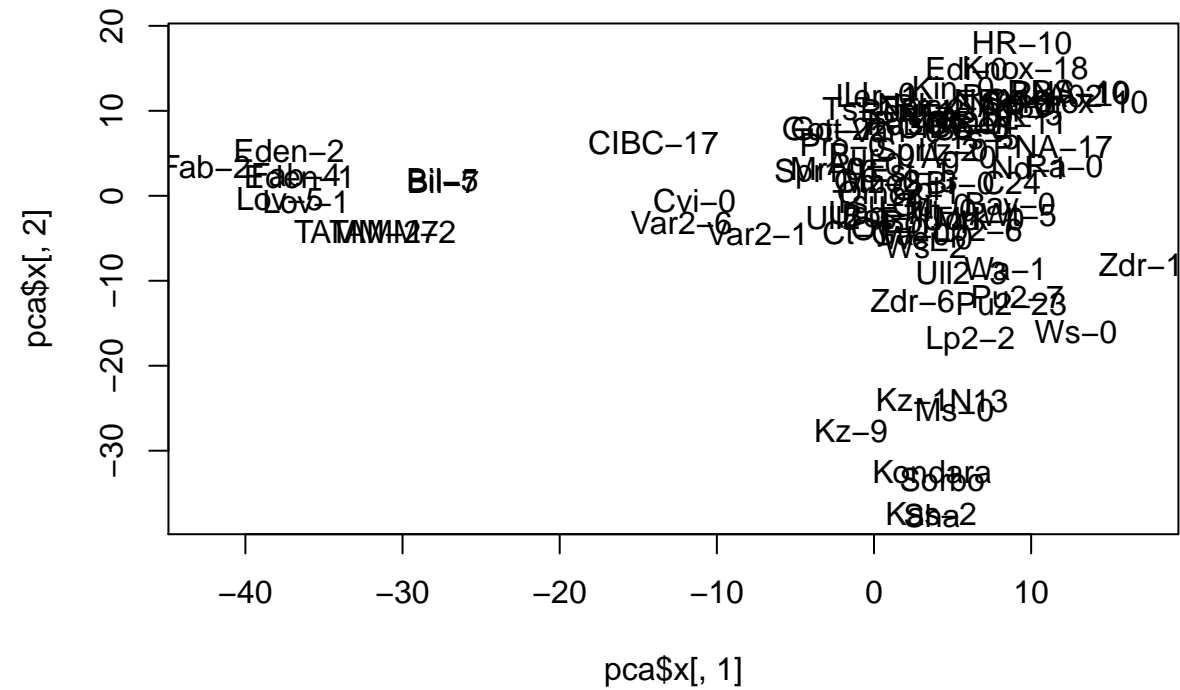
```
plot(nmds,type='n')    # VERY DIFFERENT!!
```

```
## Warning in ordiplot(x, choices = choices, type = type, display = display, :  
## Species scores not available
```

```
text(nmds, labels=row.names(nmds$points)) # different, better, but poor sep
```



```
pca = prcomp(t(na.omit(genos)))
plot(pca$x[,1],pca$x[,2],type='n') # not bad, but not great
text(pca$x[,1],pca$x[,2],labels=colnames(genos))
```



```

var_genos = t(na.omit(genos))
vars = apply(var_genos,2,var)
var_genos = var_genos[,vars>0]
scale_pca = prcomp(var_genos,scale=T)
cat('first PCA eigs:',scale_pca$sdev[1:10]^2/sum(scale_pca$sdev^2),'\n')

```

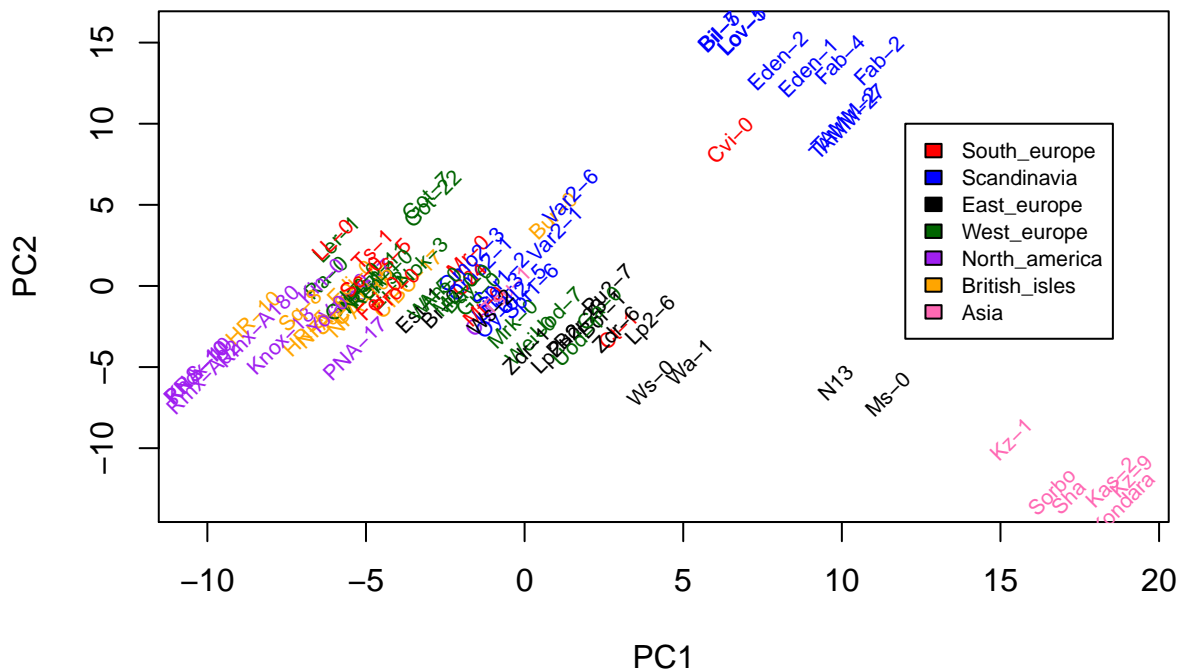
```
## first PCA eigs: 0.05158416 0.04007782 0.0354485 0.03080908 0.02937176 0.02608153 0.02455905 0.022467
```

```

plot(scale_pca$x[,1],scale_pca$x[,2],type='n',xlab='PC1',ylab='PC2',main='PCA')
text(scale_pca$x[,1],scale_pca$x[,2],labels=info[1:96,'Strain'],cex=.7,col = geog_colors[geography],srt=
legend(12,10,legend=regions,fill=geog_colors,cex=.7)

```

## PCA

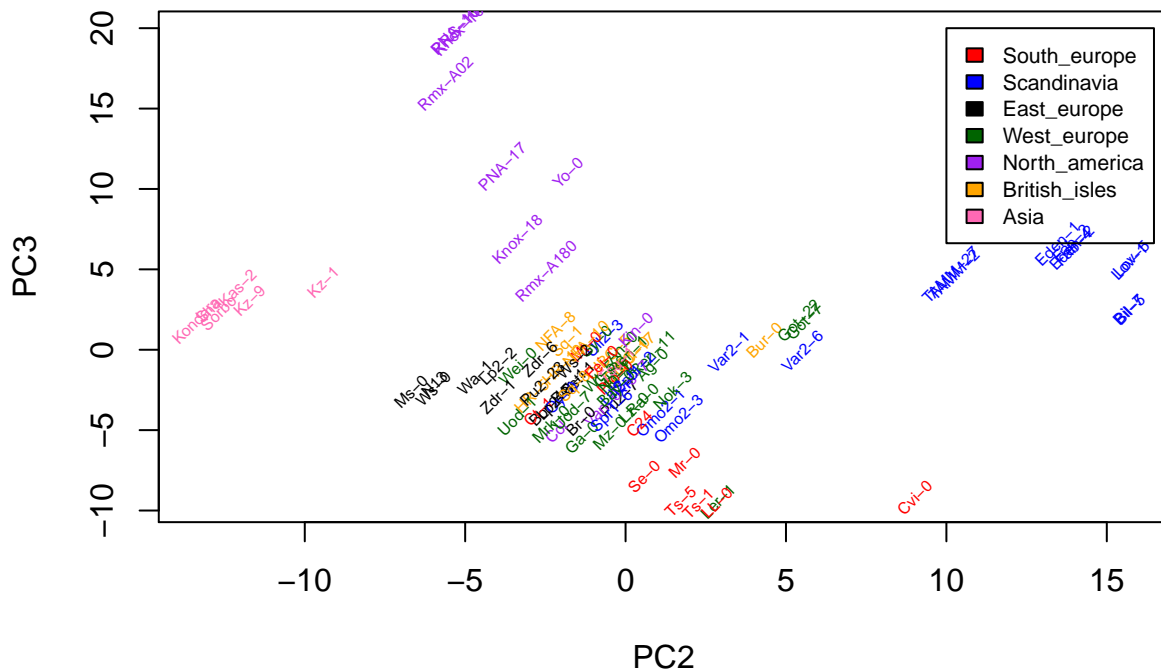


```

plot(scale_pca$x[,2],scale_pca$x[,3],type='n',xlab='PC2',ylab='PC3',main='PCA')
text(scale_pca$x[,2],scale_pca$x[,3],labels=info[1:96,'Strain'],cex=.5,col = geog_colors[geography],srt=
legend(10,20,legend=regions,fill=geog_colors,cex=.7)

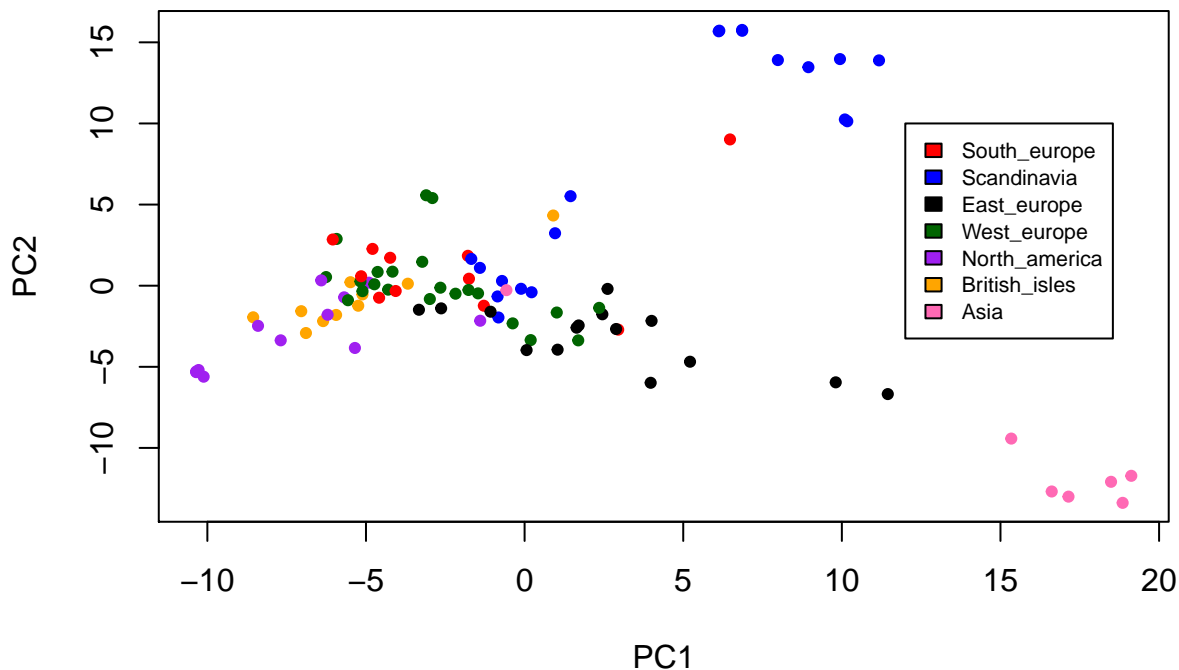
```

## PCA



```
plot(scale_pca$x[,1],scale_pca$x[,2],cex=.7,col = geog_colors[geography],xlab='PC1',ylab='PC2',main='PCA')
#text(scale_pca$x[,1],scale_pca$x[,2],labels=colnames(genos),cex=.5)
legend(12,10,legend=regions,fill=geog_colors,cex=.7)
```

## PCA



```
# PC3 shows cvi/mr-0 craziness well.
```

```

# ok! now selection analysis.
strs = read.csv('~/.Dropbox/Ath_STRs/str_var/str_masterdata.csv',header=T,stringsAsFactors=FALSE)
strs = cbind(strs,gc_content(strs$Consensus))
colnames(strs)[colnames(strs)=="gc_content(strs$Consensus)"] = 'GC_content'
#strs = strs[strs$Keisha_moniker!='' & strs$Keisha_moniker!='1',]
#pat = '([A-Z]+_[0-9]+)_ [A-Za-z_0-9 \\\*]+'
rownames(strs) = strs$ID

# add MIP arm info
mips = read.csv('mip_design_troubleshoot/2100_mips_designed.csv',header=T,stringsAsFactors = FALSE)
# have to reparse these...
mip_ids = gsub('([0-9]+)_.+', '\\1', mips$mip_name)
rownames(mips) = mip_ids
ta_gc = rep(NA,nrow(strs))
la_gc = rep(NA,nrow(strs))
scan_gc = rep(NA,nrow(strs))
names(ta_gc) = rownames(strs)
names(la_gc) = rownames(strs)

# get GC contents
ta_gc[mip_ids] = gc_content(mips$ext_probe_sequence)
la_gc[mip_ids] = gc_content(mips$lig_probe_sequence)
scan_gc[mip_ids] = gc_content(mips$scan_target_sequence)

strs = cbind(strs,ta_gc = ta_gc[rownames(strs)],
             la_gc = la_gc[rownames(strs)],
             scan_gc = scan_gc[rownames(strs)])

# check against ref, report numbers
bound = na.omit(cbind(as.numeric(genos[as.character(strs$ID),'Col']),strs$X..of.Units))
accuracy = length(which(bound[,1]==bound[,2])) / nrow(bound)
cat('in Col, typed',length(na.omit(genos[, 'Col'])), 'of 2050 strs, among which', accuracy, 'accuracy\n')

## in Col, typed 1722 of 2050 strs, among which 0.9518282 accuracy

# some measures of spread
genos[nacount>=70,] = rep(NA,96)

logged = na.omit(log10(apply(genos,1,sd,na.rm=TRUE)+pseudocount))
sd_strs = names(logged)
strs = strs[sd_strs,]

annotation = read.delim('~/.Dropbox/Ath_STRs/araport_annot/Ath_STRs_full_annotations_031317.tsv',header=
rownames(annotation) = annotation$ID
annotation = annotation[sd_strs,]
sd_genos = genos[sd_strs,]

novar = rep('variable',length(sd_strs))
novar[logged == log10(pseudocount)] = 'invariant'
novar = cbind(novar,as.character(annotation$annotation))
print('distribution of invariant STRs across annotations')

```

```
## [1] "distribution of invariant STRs across annotations"
```

```
print(table(as.data.frame(novar)))
```

```
##           V2
## novar      coding Intergenic intron UTR
## invariant   47      10      8  19
## variable  436     350    324 627
```

```
novar[novar[,2]!='coding',2] = 'noncoding'
print('test association of coding with invariant STRs')
```

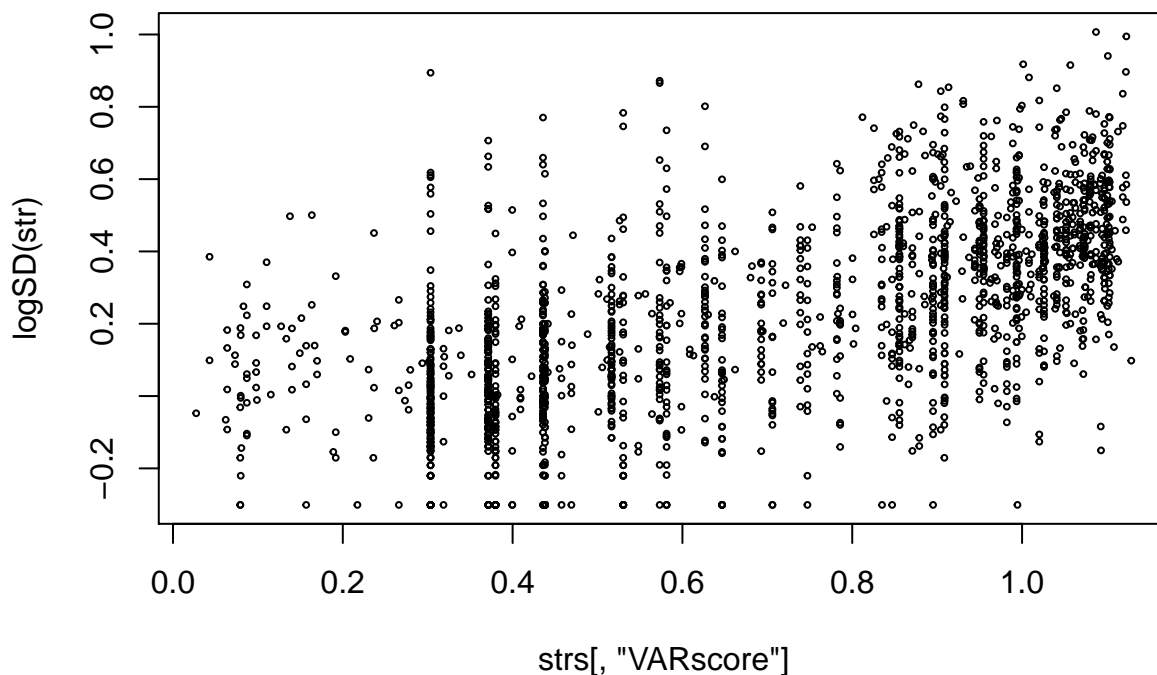
```
## [1] "test association of coding with invariant STRs"
```

```
print(fisher.test(table(as.data.frame(novar))))
```

```
##
## Fisher's Exact Test for Count Data
##
## data:  table(as.data.frame(novar))
## p-value = 6.269e-09
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  2.374222 6.079623
## sample estimates:
## odds ratio
##  3.786941
```

```
# split by annotation
nc = sd_genos[annotation$annotation == 'Intergenic',]
cod = sd_genos[annotation$annotation == 'coding',]
utr = sd_genos[annotation$annotation == 'UTR',]
intr = sd_genos[annotation$annotation == 'intron',]

plot(strs[, 'VARscore'], logged, ylab='logSD(str)', cex=.4)
```



```
require(infotheo)
```

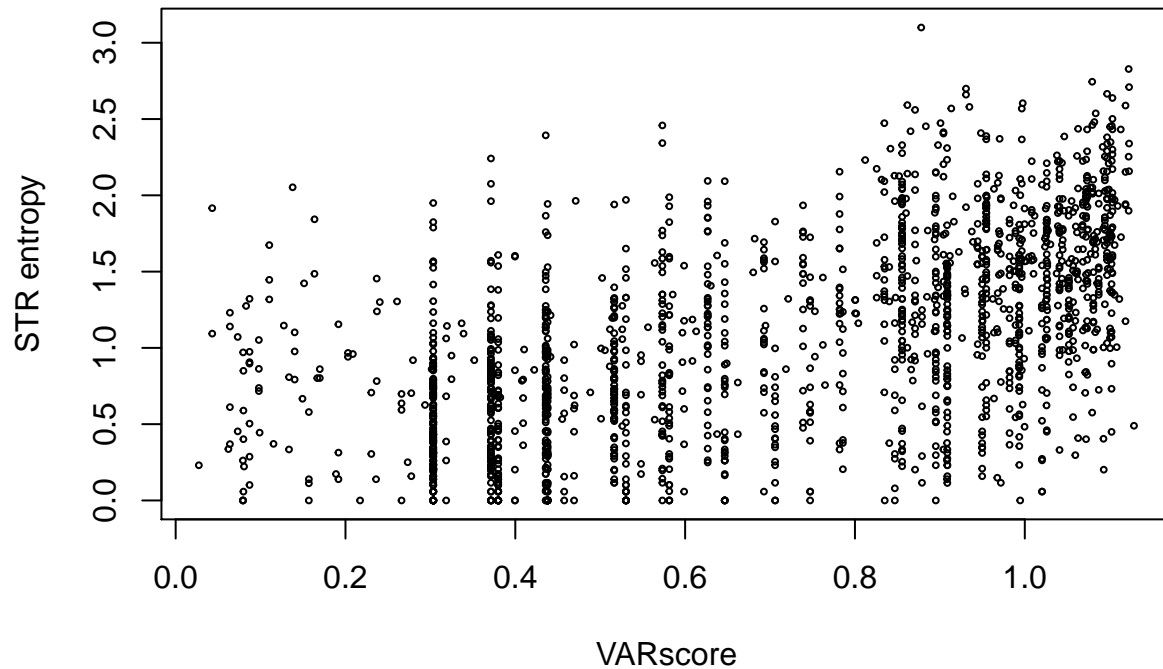
```
## Loading required package: infotheo
```

```
entrop = apply(round(sd_genos[sd_strs,]),1,entropy)
```

```
varcor = cor(strs[, 'VARscore'],entrop)
```

```
plot(strs[, 'VARscore'],entrop,main=paste('VARscore r=',varcor),cex=.4,xlab='VARscore',ylab='STR entropy
```

**VARscore r= 0.590456511820769**



```
# distributions of responses
```

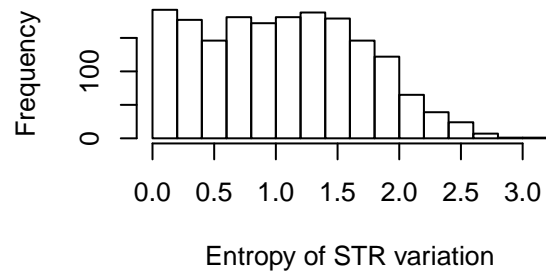
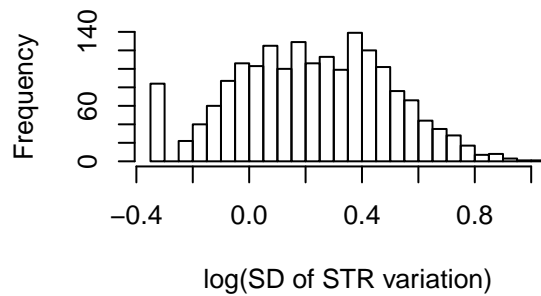
```
par(mfrow=c(2,2))
```

```
hist(logged,20,xlab='log(SD of STR variation)',main='')
```

```
hist(entrop,20,xlab='Entropy of STR variation',main='')
```

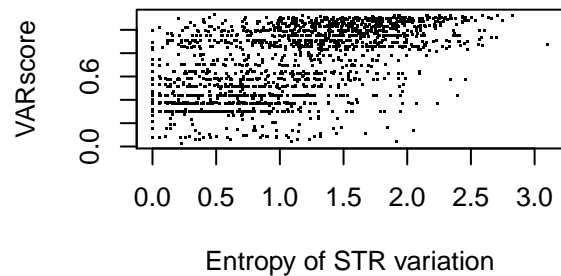
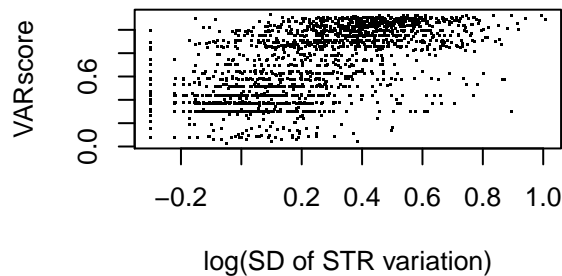
```
plot(logged,strs[, 'VARscore'],xlab='log(SD of STR variation)',ylab='VARscore',pch='.',main=cor.test(logged
```

```
plot(entrop,strs[, 'VARscore'],xlab='Entropy of STR variation',ylab='VARscore',pch='.',main=cor.test(ent
```



**0.623321065681364**

**0.590456511820769**



```
par(mfrow=c(1,1))
```

```
# try a SVM?? (like legendre) (SVR (regression)- using epsilon regression,
# which is apparently better in situations where you're not too worried about
# the complexity of the solution:
# http://stats.stackexchange.com/questions/94118/difference-between-ep-svr-and-nu-svr-and-least-squares
# i have n>>p so i am ok i think
```

```
# mean squared error to evaluate model performance
```

```
MSE = function(predicted,actual) {
  sq_err = (predicted-actual)^2
  return(mean(sq_err))
}
```

```
svr_fit = function(data,str_ids,response) {
# fit=ksvm(response[str_ids]~.,as.matrix(data[str_ids,]),cross=5,scaled=FALSE)
  fit=ksvm(response[str_ids]~.,as.matrix(data[str_ids,]),scaled=FALSE)
  return(fit)
}
```

```
svr_pred_eval = function(svr, newdata, str_ids,response) {
  pred = predict(svr,newdata=newdata[str_ids,])
  core = cor(response[str_ids], pred)
  mse = MSE(response[str_ids], pred)
  out = list(pred,core,mse)
  names(pred) = str_ids
  names(out)=c('predict','cor','mse')
  return(out)
}
```

```
# restrict to numeric predictors, trying various... added multiple gc contents for mips too
```



```

#svm_strs=strs[,c("TAIR10start", "Unit.Size", "Purity", "X..of.Units", "GC_content", "ta_gc", "la_gc", "scan_g
#svm_strs=strs[,c("Unit.Size", "Purity", "X..of.Units", "GC_content", "ta_gc", "la_gc", "scan_gc")]
svm_strs=strs[,c("Unit.Size", "Purity", "X..of.Units", "GC_content", "scan_gc")]
# worked ok
#svm_strs=strs[,c("TAIR10start", "Chromosome", "Unit.Size", "Purity", "X..of.Units", "GC_content")]
# also ok
#svm_strs=strs[,c("Chromosome", "Unit.Size", "Purity", "X..of.Units", "GC_content")]
# a bit narrower, overall somewhat better fit? (with SVR)
#svm_strs=strs[,c("Unit.Size", "Purity", "X..of.Units", "GC_content")]

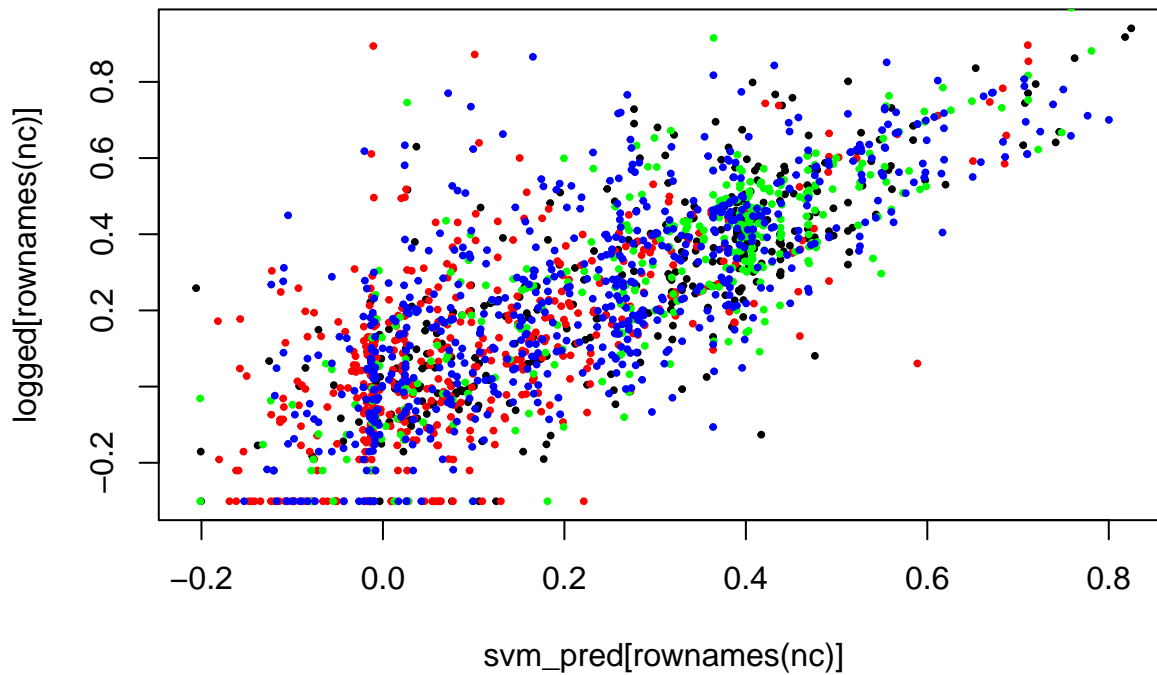
#svm_strs$TAIR10start = svm_strs$TAIR10start / chr_lens[strs$Chromosome]

rownames(svm_strs) = names(entrop)
# replace reference copy number with median of observed if present
for (i in names(entrop)) {
  names(svm_strs$X..of.Units) = names(entrop)
  med = median(as.numeric(genos[i,]), na.rm=TRUE)
  if (is.na(med)) { continue }
  else {
    svm_strs[i, 'X..of.Units'] = med
  }
}

# first do all data to see how it is
tryk = ksvm(logged~, as.matrix(svm_strs), cross=5, scaled=FALSE)
svm_pred = predict(tryk, svm_strs)
names(svm_pred) = rownames(svm_strs)
svm_cor = cor(svm_pred, logged)
svm_mse = MSE(svm_pred, logged)
ncs = rownames(nc)
plot(svm_pred[rownames(nc)], logged[rownames(nc)], main=paste('SVM r=', svm_cor, 'MSE =', svm_mse), cex=.4, pch=19, col='red')
points(svm_pred[rownames(cod)], logged[rownames(cod)], cex=.4, pch=19, col='red')
points(svm_pred[rownames(intr)], logged[rownames(intr)], cex=.4, pch=19, col='green')
points(svm_pred[rownames(utr)], logged[rownames(utr)], cex=.4, pch=19, col='blue')

```

**SVM r= 0.777553670318653 MSE = 0.0280527078268691**



```
nc_mse = MSE(svm_pred[rownames(nc)],logged[rownames(nc)])
cod_mse = MSE(svm_pred[rownames(cod)],logged[rownames(cod)])
intr_mse = MSE(svm_pred[rownames(intr)],logged[rownames(intr)])
utr_mse = MSE(svm_pred[rownames(utr)],logged[rownames(utr)])

print(cor.test(svm_pred[ncs],logged[ncs]))
```

```
##
## Pearson's product-moment correlation
##
## data: svm_pred[ncs] and logged[ncs]
## t = 24.848, df = 358, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.754270 0.830657
## sample estimates:
##      cor
## 0.7956044
```

```
print(cor.test(svm_pred[rownames(utr)],logged[rownames(utr)]))
```

```
##
## Pearson's product-moment correlation
##
## data: svm_pred[rownames(utr)] and logged[rownames(utr)]
## t = 28.56, df = 644, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7114109 0.7797066
## sample estimates:
```

```

##          cor
## 0.7475279

print(cor.test(svm_pred[rownames(intr)],logged[rownames(intr)]))

##
## Pearson's product-moment correlation
##
## data:  svm_pred[rownames(intr)] and logged[rownames(intr)]
## t = 25.763, df = 330, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7780755 0.8501206
## sample estimates:
##          cor
## 0.8172669

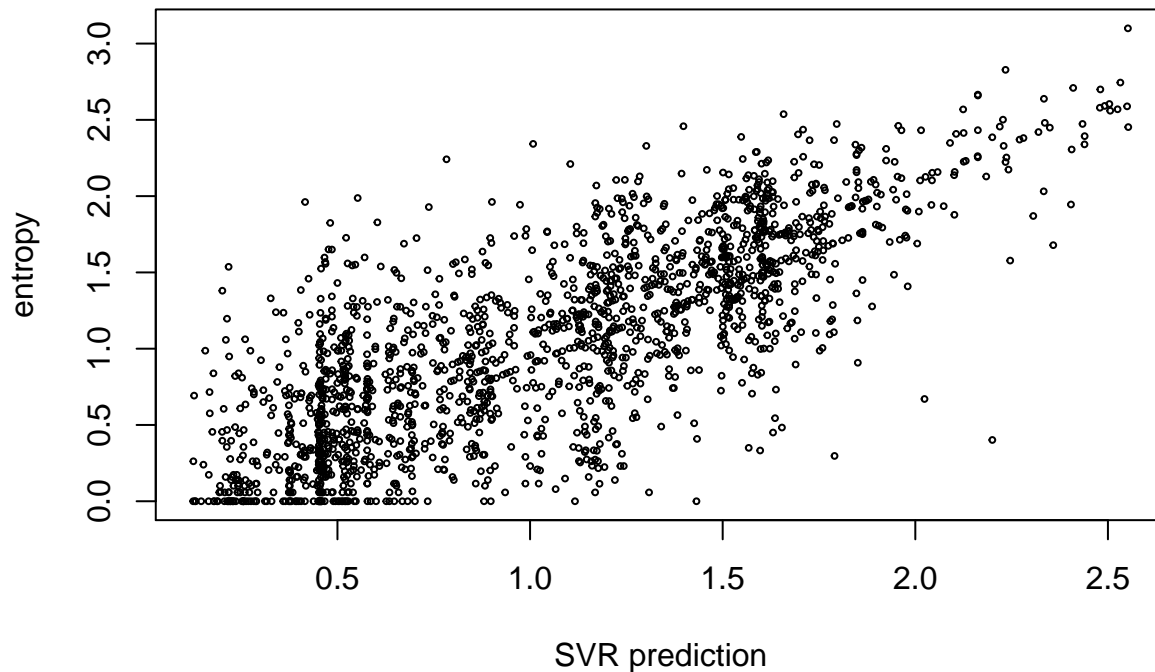
print(cor.test(svm_pred[rownames(cod)],logged[rownames(cod)]))

##
## Pearson's product-moment correlation
##
## data:  svm_pred[rownames(cod)] and logged[rownames(cod)]
## t = 19.96, df = 481, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6211673 0.7191213
## sample estimates:
##          cor
## 0.6730856

# entropy better?
entrok = ksvm(entrop~.,as.matrix(svm_strs),cross=5,scaled=FALSE)
entro_pred = predict(entrok,svm_strs)
names(entro_pred) = names(entrop)
entro_cor = cor(entro_pred,entrop)
entro_mse = MSE(entro_pred,entrop)
plot(entro_pred,entrop,xlab='SVR prediction',ylab='entropy',main=paste('SVM r=',entro_cor,'MSE =',entro_mse))

```

**SVM  $r = 0.775343846977295$  MSE = 0.169509660700635**

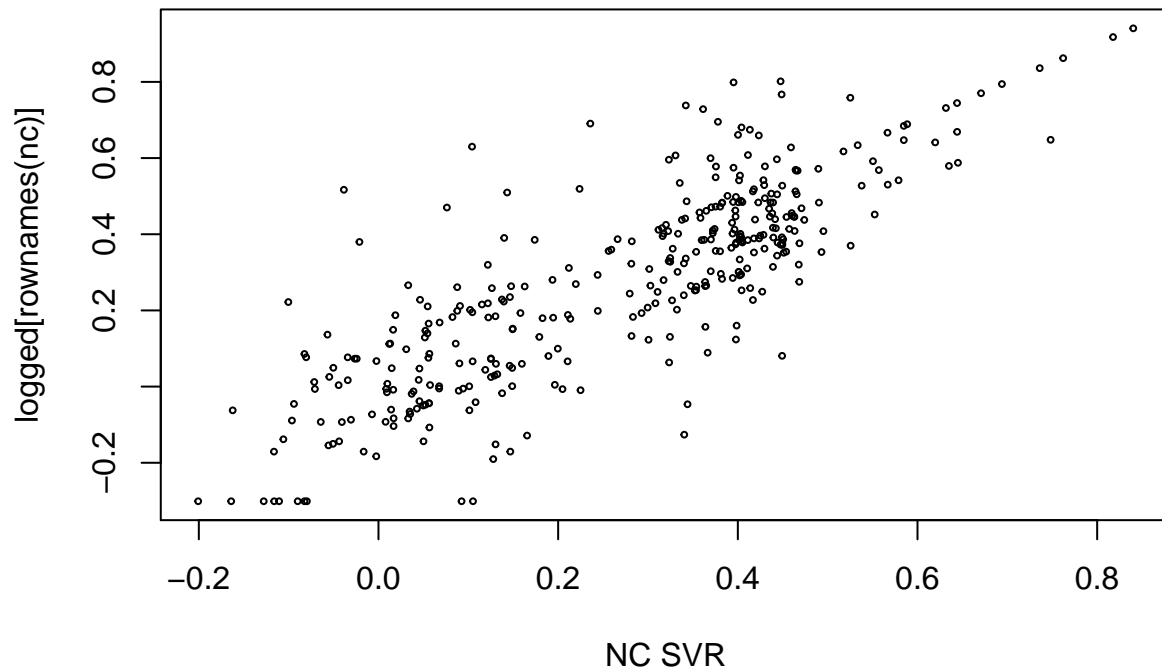


```
# now try on nc data
ncs = rownames(nc)
nck = ksvm(logged[rownames(nc)]~.,as.matrix(svm_strs[rownames(nc),]),cross=5,scaled=FALSE)
nck_entro = ksvm(entrop[rownames(nc)]~.,as.matrix(svm_strs[rownames(nc),]),cross=5,scaled=FALSE)
nc_svm_pred = predict(nck,svm_strs[rownames(nc),])
nc_svm_pred_entro = predict(nck_entro,svm_strs[rownames(nc),])

nc_svm_cor = cor(nc_svm_pred,logged[rownames(nc)])
nc_svm_entro_cor = cor(nc_svm_pred_entro,entrop[rownames(nc)])
nc_svm_mse = cor(nc_svm_pred,logged[rownames(nc)])
nc_svm_entro_mse = MSE(nc_svm_pred_entro,entrop[rownames(nc)])

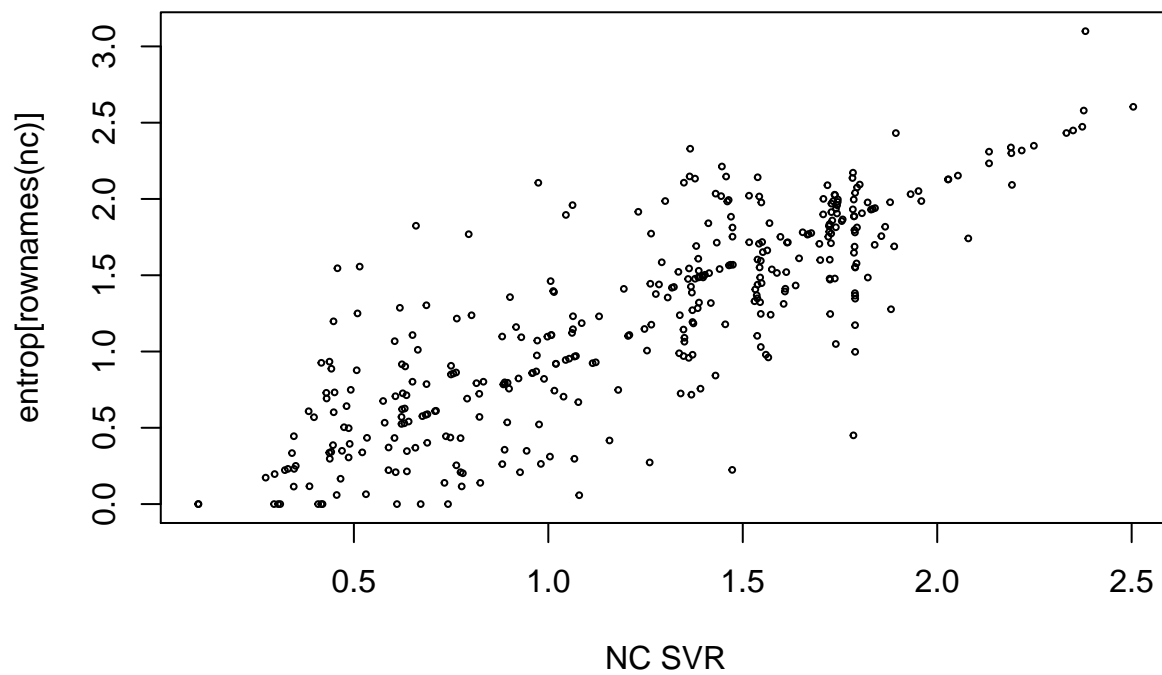
plot(nc_svm_pred,logged[rownames(nc)],main=paste('NC SVM r=',nc_svm_cor),cex=.4,xlab='NC SVR') # better
```

**NC SVM r= 0.826715277543429**



```
plot(nc_svm_pred_entrop,entrop[rownames(nc)],main=paste('NC SVM r=',nc_svm_entrop_cor),cex=.4,xlab='NC SVM r= 0.826715277543429')
```

**NC SVM r= 0.828733527577695**



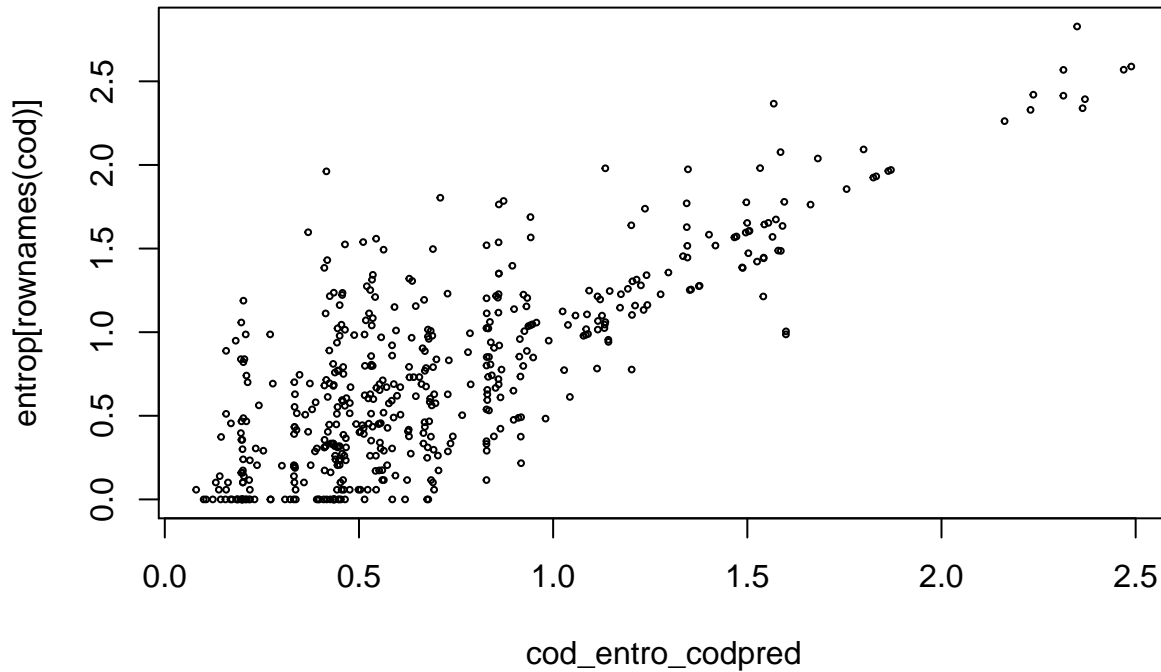
```
# moving forward with entropy, somewhat better justification...
cod_entrop_ncpred = predict(nck_entrop,svm_strs[rownames(cod),])
```

```

cod_entro_svm = ksvm(entrop[rownames(cod)]~., as.matrix(svm_strs[rownames(cod),]),cross=5,scaled=FALSE)
cod_entro_codpred = predict(cod_entro_svm,svm_strs[rownames(cod),])
cod_entro_codpred_cor = cor(cod_entro_codpred,entrop[rownames(cod)])
plot(cod_entro_codpred,entrop[rownames(cod)],cex=.4,main=cod_entro_codpred_cor)

```

**0.786036398919096**

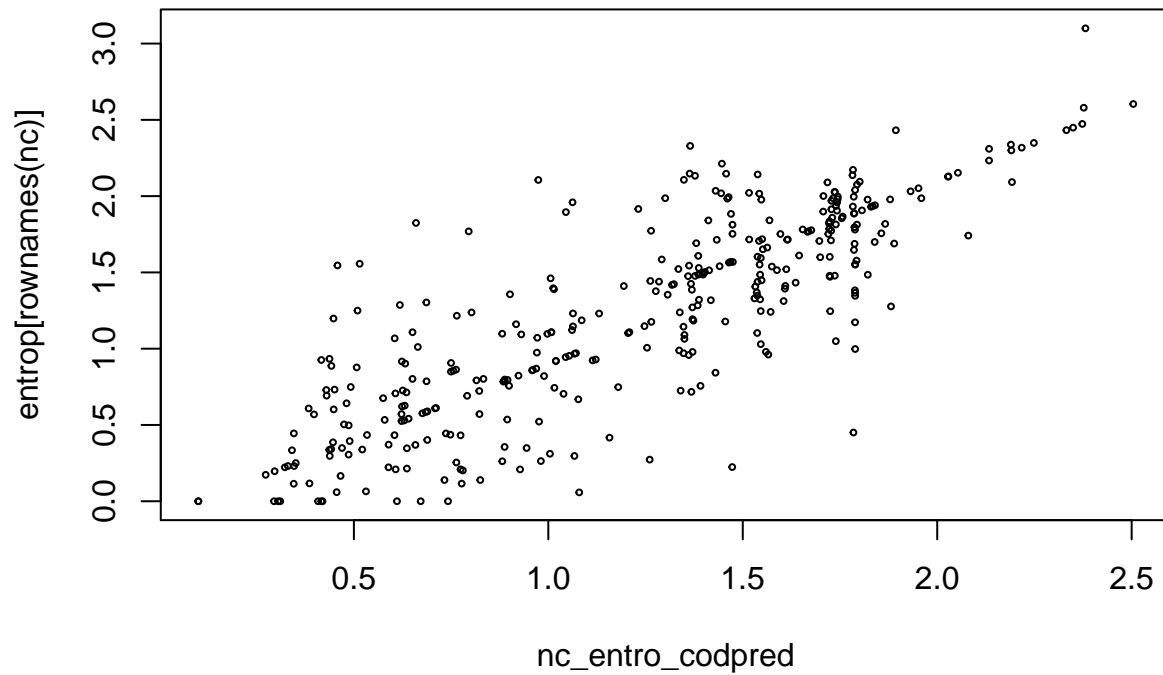


```

nc_entro_codpred = predict(nck_entro,svm_strs[rownames(nc),])
plot(nc_entro_codpred,entrop[rownames(nc)],cex=.4,main=cor(nc_entro_codpred,entrop[rownames(nc)]))

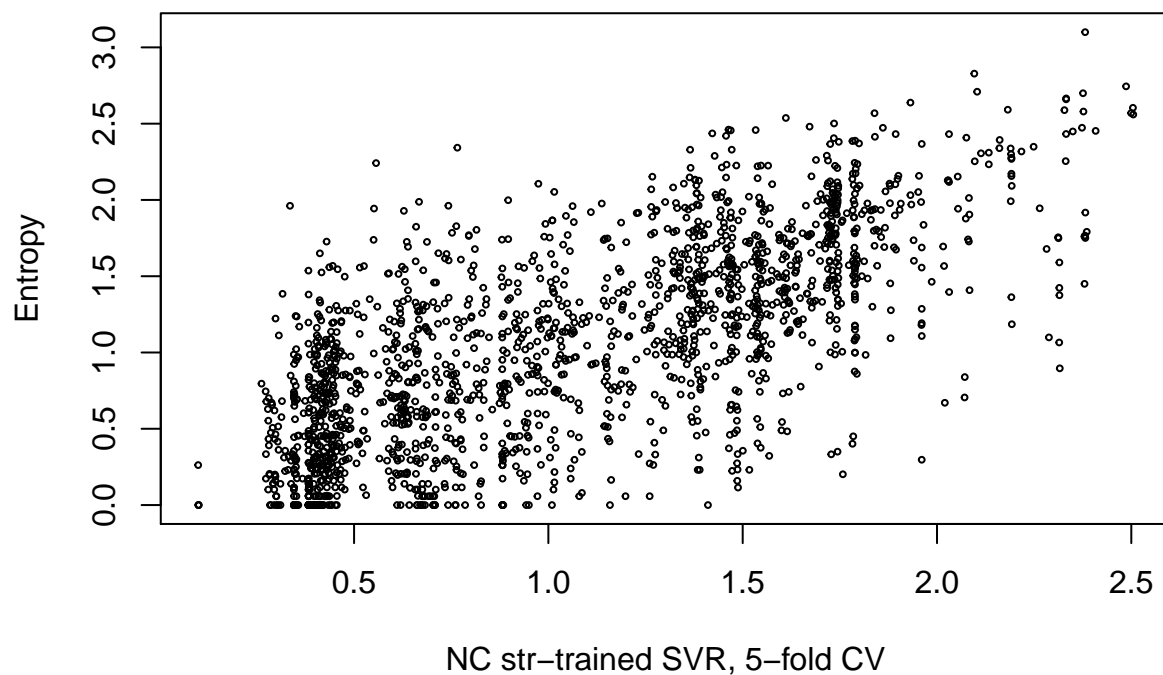
```

**0.828733527577695**



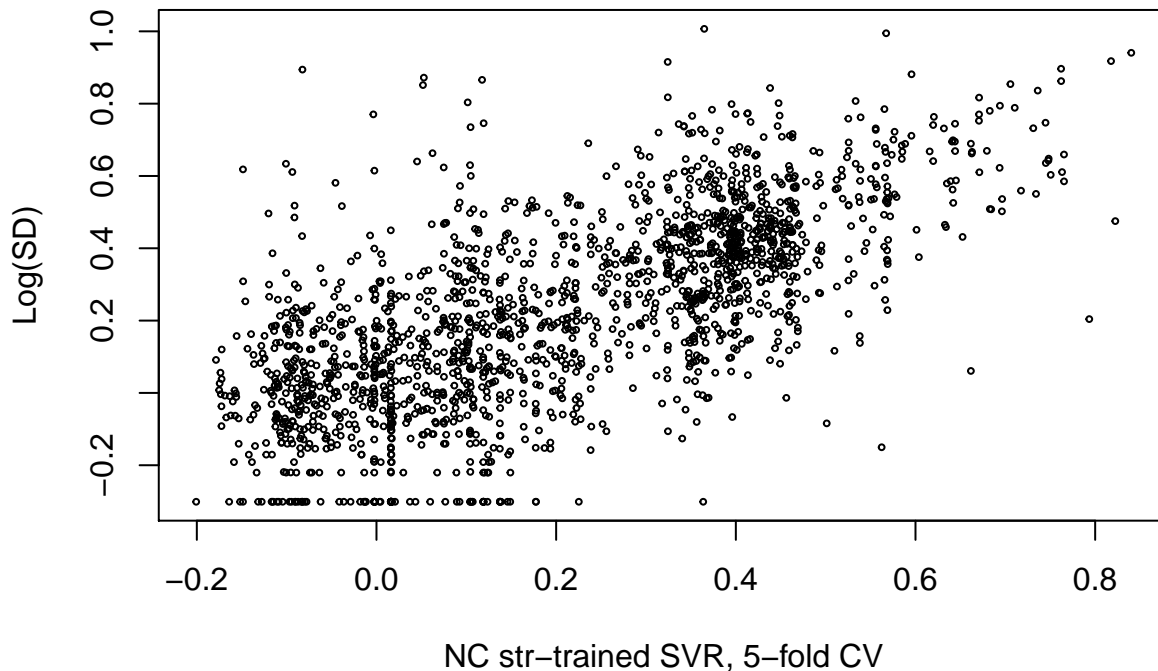
```
nc_entrop_allpred = predict(nck_entrop,svm_strs)
nc_log_allpred = predict(nck,svm_strs)
names(nc_log_allpred) = rownames(svm_strs)
plot(nc_entrop_allpred,entrop,cex=.4,main=cor(nc_entrop_allpred,entrop),xlab='NC str-trained SVR, 5-fold CV',ylab='Entropy')
```

**0.702917400821269**



```
plot(nc_log_allpred,logged,cex=.4,main=cor(nc_log_allpred,logged),xlab='NC str-trained SVR, 5-fold CV',
```

**0.684907740387053**

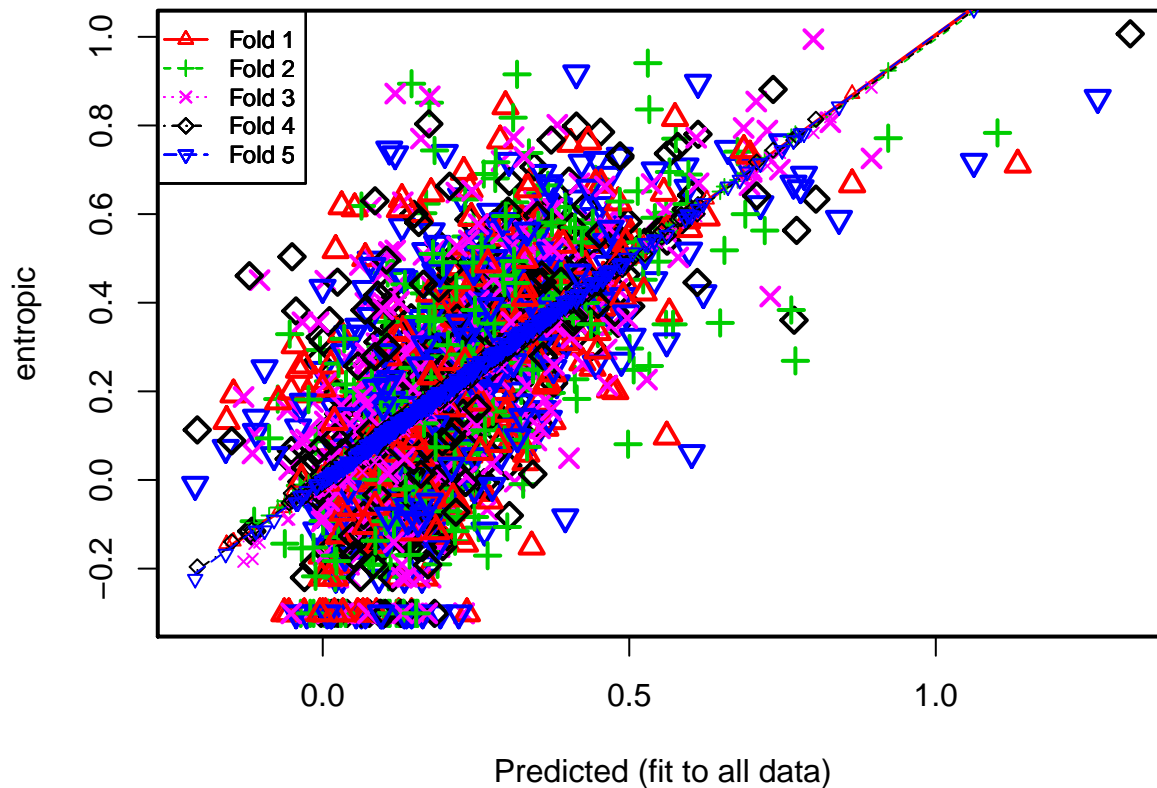


```
# cv.lm() from DAAG package
# compare to SVR fit just to see
entrop = logged
lm.data = cbind(entrop[rownames(nc)],svm_strs[rownames(nc),])
all_lm.data = cbind(entrop,svm_strs)
colnames(lm.data)[1] = 'entropic'
colnames(all_lm.data)[1] = 'entropic'
rownames(all_lm.data) = rownames(svm_strs)
rownames(lm.data) = rownames(nc)
fitlm = lm(entropic ~ ., data=lm.data)
fitalllm = lm(entropic ~., data=all_lm.data)
# commented because they make big irritating plots
#lmallcv= cv.lm(data=all_lm.data,form.lm=fitlm,m=5)
# not printing or plotting breaks fn? weird. should be fixed.
# plotting is less disruptive (one ugly plot rather than pages of printout)
lmcv = cv.lm(data=all_lm.data,form.lm=fitlm,m=5,plot=TRUE, printit=FALSE) #, printit=FALSE)

## Warning in cv.lm(data = all_lm.data, form.lm = fitlm, m = 5, plot = TRUE, :
##
## As there is >1 explanatory variable, cross-validation
## predicted values for a fold are not a linear function
## of corresponding overall predicted values. Lines that
## are shown for the different folds are approximate
```



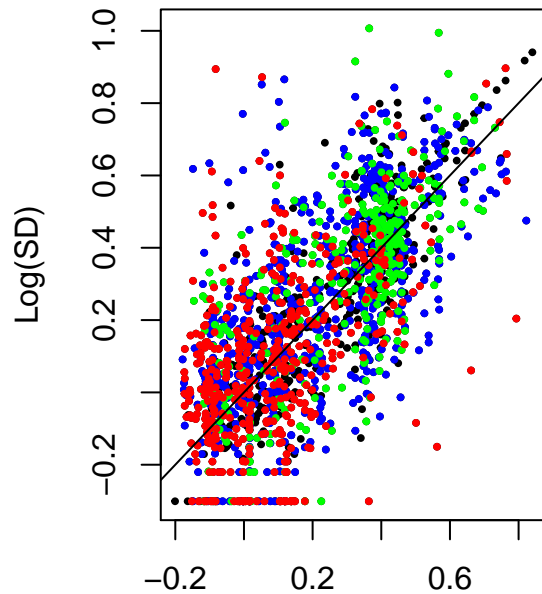
### Small symbols show cross-validation predicted values



```
# plot the results, PCCs
par(mfrow=c(1,2))
plot(nc_log_allpred,logged,cex=.4,pch=19,col='black',main=cor(nc_log_allpred,logged),xlab='NC str-trained LM',ylab='Log(SD)',main=cor(nc_log_allpred,logged))
points(nc_log_allpred[rownames(utr)],logged[rownames(utr)],cex=.4,col='blue',pch=19)
points(nc_log_allpred[rownames(intr)],logged[rownames(intr)],cex=.4,col='green',pch=19)
points(nc_log_allpred[rownames(cod)],logged[rownames(cod)],cex=.4,col='red',pch=19)
abline(0,1)

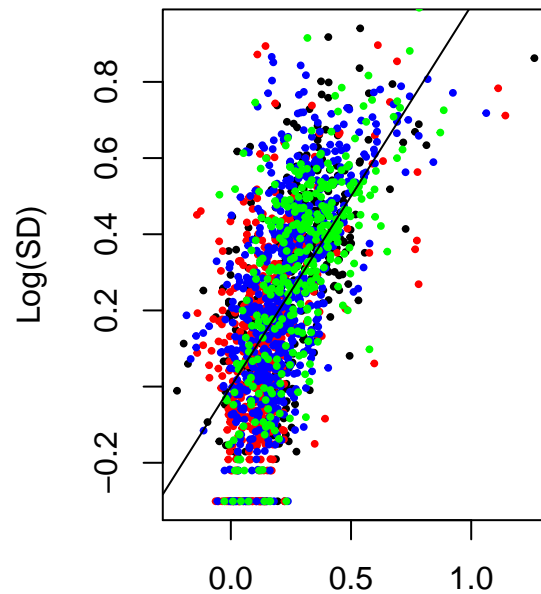
plot(lmcv[ncs,'cvpred'],logged[ncs],pch=19,xlab='NC str-trained LM, 5-fold CV',ylab='Log(SD)',main=cor(lmcv[ncs,'cvpred'],logged[ncs]))
points(lmcv[rownames(cod),'cvpred'],logged[rownames(cod)],cex=.4,col='red',pch=19)
points(lmcv[rownames(utr),'cvpred'],logged[rownames(utr)],cex=.4,col='blue',pch=19)
points(lmcv[rownames(intr),'cvpred'],logged[rownames(intr)],cex=.4,col='green',pch=19)
abline(0,1)
```

**0.684907740387053**



NC str-trained SVR, 5-fold CV

**0.632776707998593**



NC str-trained LM, 5-fold CV

```
print('PCCs for CV SVR mod by genomic location')
```

```
## [1] "PCCs for CV SVR mod by genomic location"
```

```
print(cor.test(nc_log_allpred[ncs],logged[ncs]))
```

```
##  
## Pearson's product-moment correlation  
##  
## data: nc_log_allpred[ncs] and logged[ncs]  
## t = 27.802, df = 358, p-value < 2.2e-16  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## 0.7909399 0.8568579  
## sample estimates:  
## cor  
## 0.8267153
```

```
print(cor.test(nc_log_allpred[rownames(utr)],logged[rownames(utr)]))
```

```
##  
## Pearson's product-moment correlation  
##  
## data: nc_log_allpred[rownames(utr)] and logged[rownames(utr)]  
## t = 20.985, df = 644, p-value < 2.2e-16  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## 0.5890917 0.6809379  
## sample estimates:  
## cor  
## 0.6372723
```

```
print(cor.test(nc_log_allpred[rownames(intr)],logged[rownames(intr)]))
```

```
##
## Pearson's product-moment correlation
##
## data: nc_log_allpred[rownames(intr)] and logged[rownames(intr)]
## t = 16.587, df = 330, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.6109826 0.7290049
## sample estimates:
## cor
## 0.6742767
```

```
print(cor.test(nc_log_allpred[rownames(cod)],logged[rownames(cod)]))
```

```
##
## Pearson's product-moment correlation
##
## data: nc_log_allpred[rownames(cod)] and logged[rownames(cod)]
## t = 12.764, df = 481, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.4332305 0.5667937
## sample estimates:
## cor
## 0.5030092
```

```
# lm pccs
print('PCCs for CV LM mod by genomic location')
```

```
## [1] "PCCs for CV LM mod by genomic location"
```

```
print(cor.test(lmcv[ncs,'cvpred'],logged[ncs]))
```

```
##
## Pearson's product-moment correlation
##
## data: lmcv[ncs, "cvpred"] and logged[ncs]
## t = 14.618, df = 358, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.5422861 0.6722587
## sample estimates:
## cor
## 0.6113791
```

```
print(cor.test(lmcv[rownames(cod),'cvpred'],logged[rownames(cod)]))
```

```
##
## Pearson's product-moment correlation
##
## data: lmcv[rownames(cod), "cvpred"] and logged[rownames(cod)]
## t = 12.027, df = 481, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.4091554 0.5465972
```

```

## sample estimates:
##      cor
## 0.4808244

print(cor.test(lmcv[rownames(intr)], 'cvpred'], logged[rownames(intr)]))

##
## Pearson's product-moment correlation
##
## data:  lmcv[rownames(intr), "cvpred"] and logged[rownames(intr)]
## t = 15.646, df = 330, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5861329 0.7103388
## sample estimates:
##      cor
## 0.6525982

print(cor.test(lmcv[rownames(utr)], 'cvpred'], logged[rownames(utr)]))

##
## Pearson's product-moment correlation
##
## data:  lmcv[rownames(utr), "cvpred"] and logged[rownames(utr)]
## t = 20.526, df = 644, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5798573 0.6733433
## sample estimates:
##      cor
## 0.6288678

par(mfrow=c(1,1))

# evaluate training vs test - loop through and see what distributions are for nc strs
mses = c()
cors = c()
nc_predic_mat = c()
cod_mses = c()
cod_predic_mat = c()
utr_predic_mat = c()
intr_predic_mat = c()
train_size = round(nrow(nc) / boot_divisor)

##### THIS IS THE SELECTION ANALYSIS THAT I ACTUALLY USED
# I KNOW THAT IT LOOKS TERRIBLE DON'T JUDGE ME
# in any case, USING LOG(SD) from now on. because i coded it as entropy everywhere, just doing this jan.
entrop = logged

for (i in 1:1000) {
  if (i%%64 == 0) {print(i)}
  rand_strs = sample(rownames(nc), replace=TRUE)
  train = rand_strs[1:train_size]
  test = rand_strs[(train_size+1):length(rand_strs)]
  fit = svr_fit(data = svm_strs, response = entrop, str_ids = train)
  predic = svr_pred_eval(svr=fit, newdata=svm_strs, str_ids=ncs, response=entrop)
}

```

```

msep[i] = predic$mse
cors[i] = predic$cor
rownames(predic$predict) = ncs # apparently this is a matrix...
nc_predic_mat = cbind(nc_predic_mat, predic$predict[rownames(nc),])
# coding predictions - should have just wrapped in a fn
predic = svr_pred_eval(svr=fit,newdata=svm_strs,str_ids=rownames(cod),response=entrop)
cod_msep[i] = predic$mse
rownames(predic$predict) = rownames(cod) # apparently this is a matrix...
cod_predic_mat = cbind(cod_predic_mat,predic$predict[rownames(cod),])
# utr predictions
predic = svr_pred_eval(svr=fit,newdata=svm_strs,str_ids=rownames(utr),response=entrop)
rownames(predic$predict) = rownames(utr)
utr_predic_mat = cbind(utr_predic_mat,predic$predict[rownames(utr),])
utr_mse[i] = predic$mse
# intron predictions
predic = svr_pred_eval(svr=fit,newdata=svm_strs,str_ids=rownames(intr),response=entrop)
rownames(predic$predict) = rownames(intr)
intr_predic_mat = cbind(intr_predic_mat,predic$predict[rownames(intr),])
intr_mse[i] = predic$mse
}

```

```

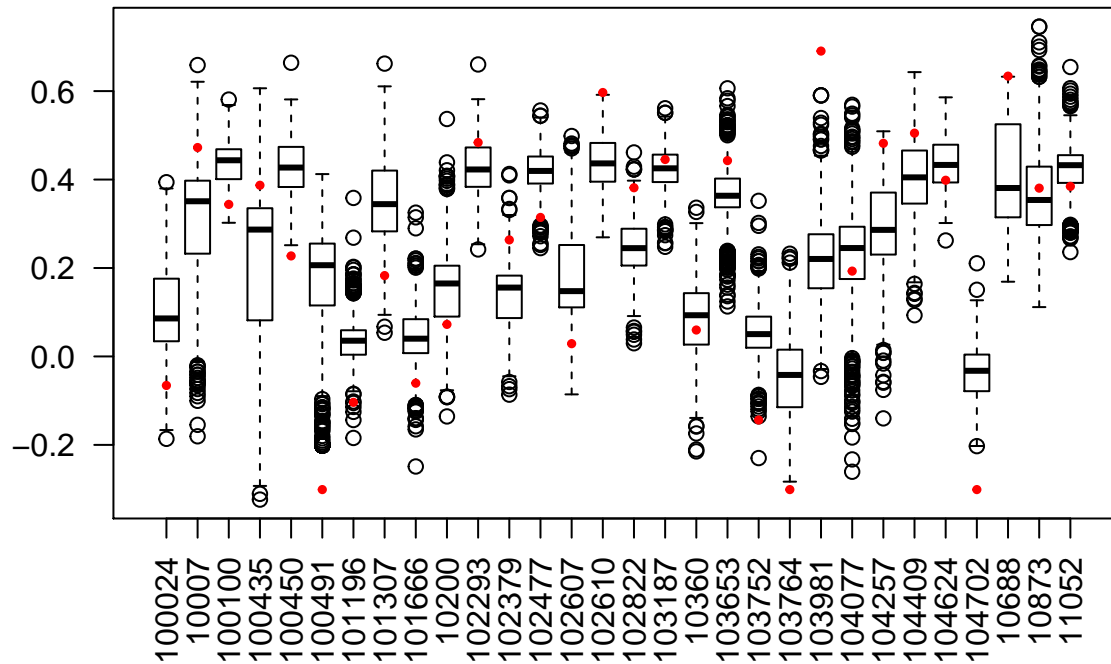
## [1] 64
## [1] 128
## [1] 192
## [1] 256
## [1] 320
## [1] 384
## [1] 448
## [1] 512
## [1] 576
## [1] 640
## [1] 704
## [1] 768
## [1] 832
## [1] 896
## [1] 960

```

```

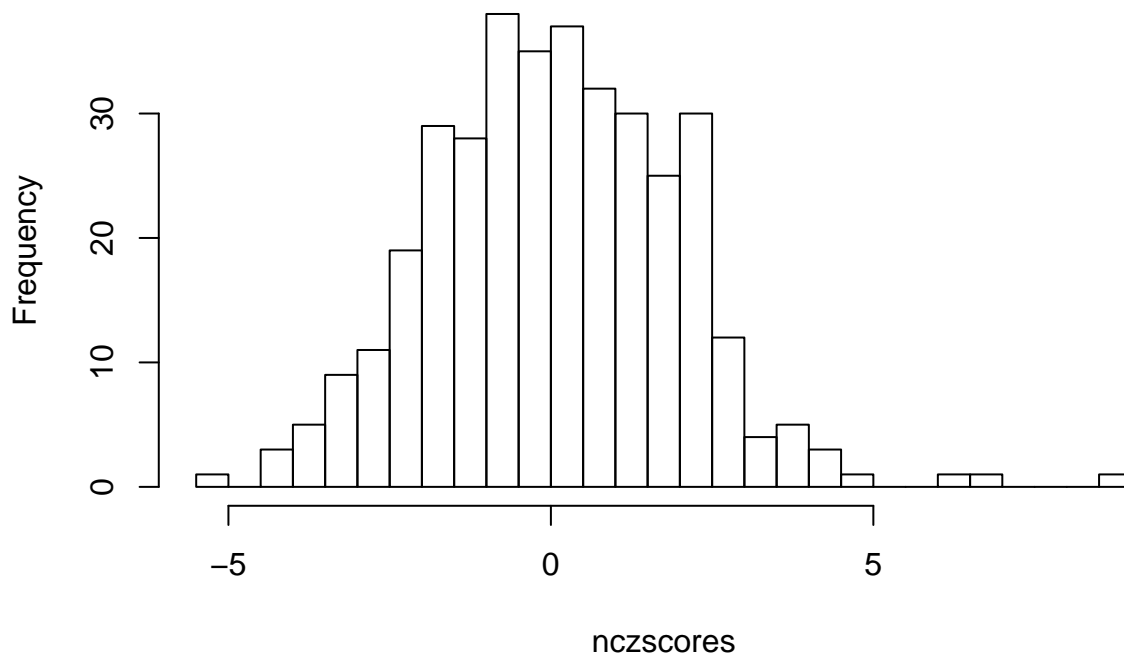
# diagnostics on the predictions...
boxplot.matrix(t(nc_predic_mat[1:30,]),las=2)
points(1:30,entrop[rownames(nc)][1:30],pch=19,col='red',cex=.5)

```



```
# for nc
ncmeaned = rowMeans(nc_predic_mat)
ncsded = apply(nc_predic_mat,1,sd)
nczscores = (entrop[rownames(nc)]-ncmeaned) / ncsded
hist(nczscores,20)
```

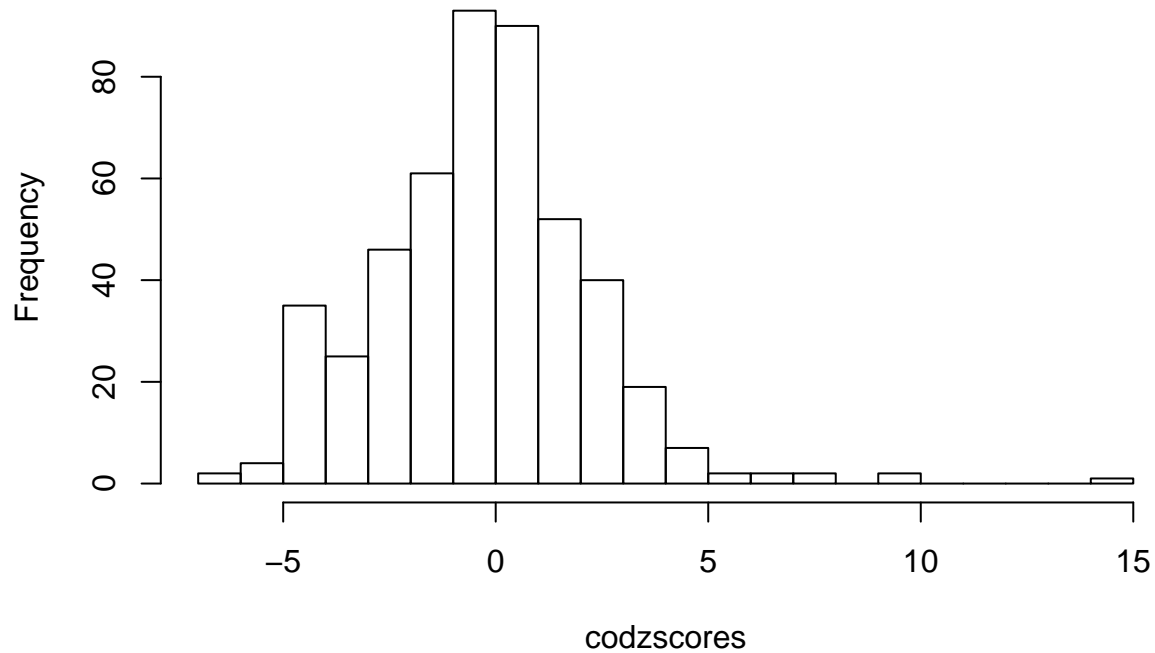
**Histogram of nczscores**



```
codmeaned = rowMeans(cod_predic_mat)
codsded = apply(cod_predic_mat,1,sd)
codzscores = (entrop[rownames(cod)]-codmeaned) / codsded
```

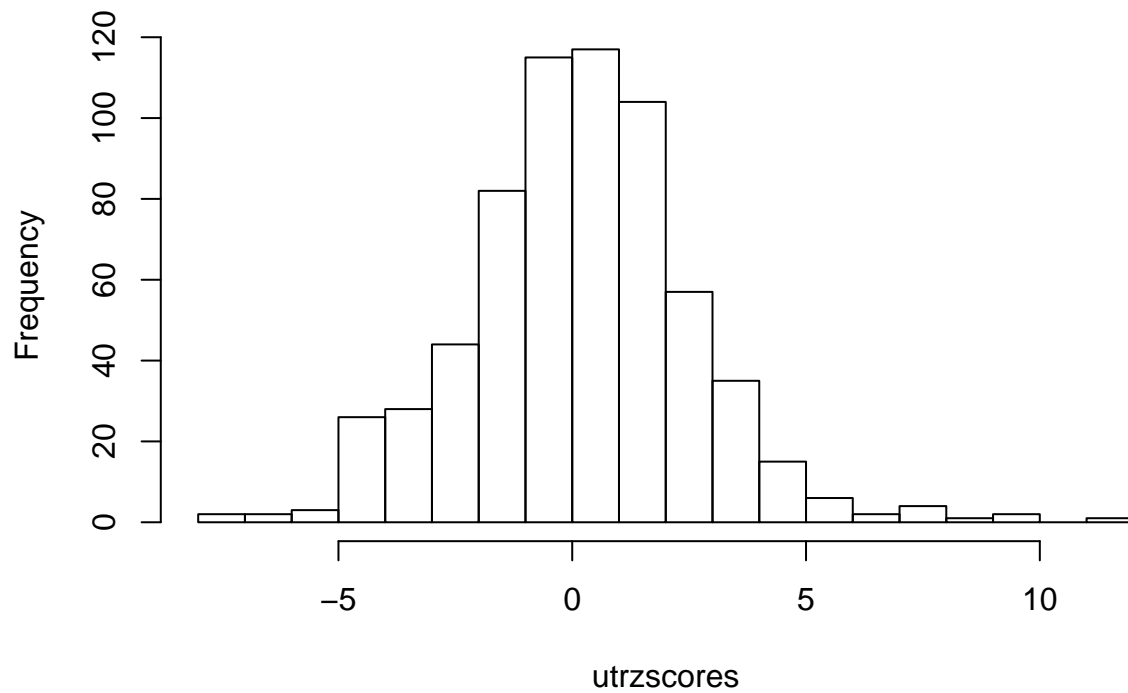
```
hist(codzscores,20)
```

## Histogram of codzscores



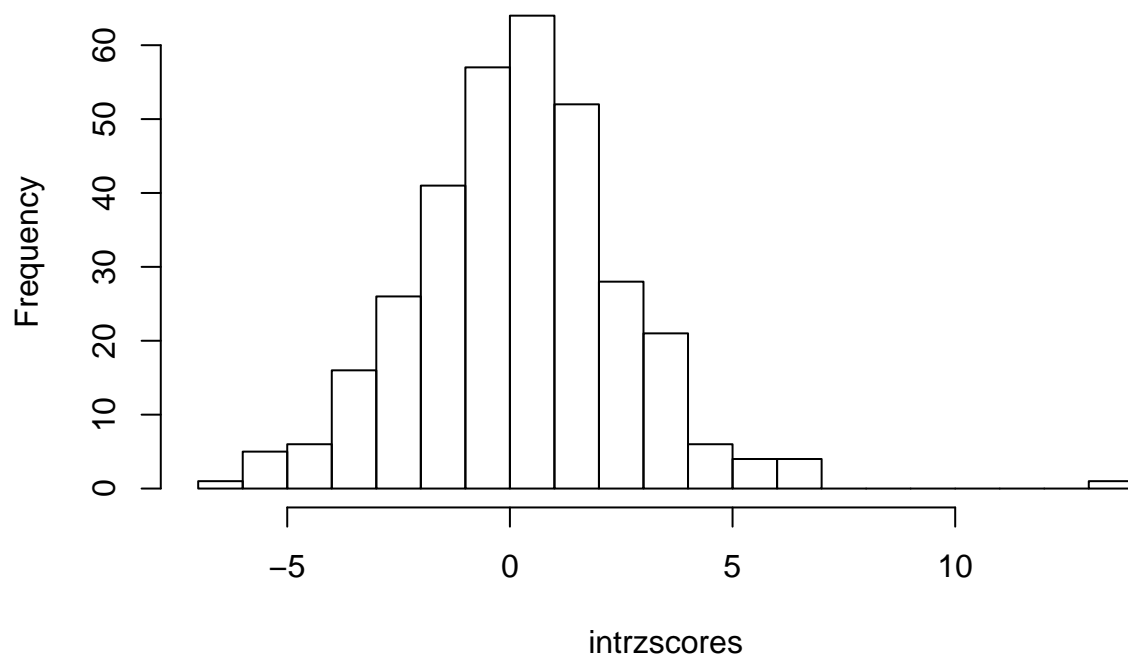
```
utrmeaned = rowMeans(utr_predic_mat)
utrstded = apply(utr_predic_mat,1,sd)
utrzscores = (entrop[rownames(utr)]-utrmeaned) / utrstded
hist(utrzscores,20)
```

### Histogram of utrzscores



```
intrmeaned = rowMeans(intr_predic_mat)
intrsded = apply(intr_predic_mat,1,sd)
intrzscores = (entrop[rownames(intr)]-intrmeaned) / intrsded
hist(intrzscores,20)
```

### Histogram of intrzscores

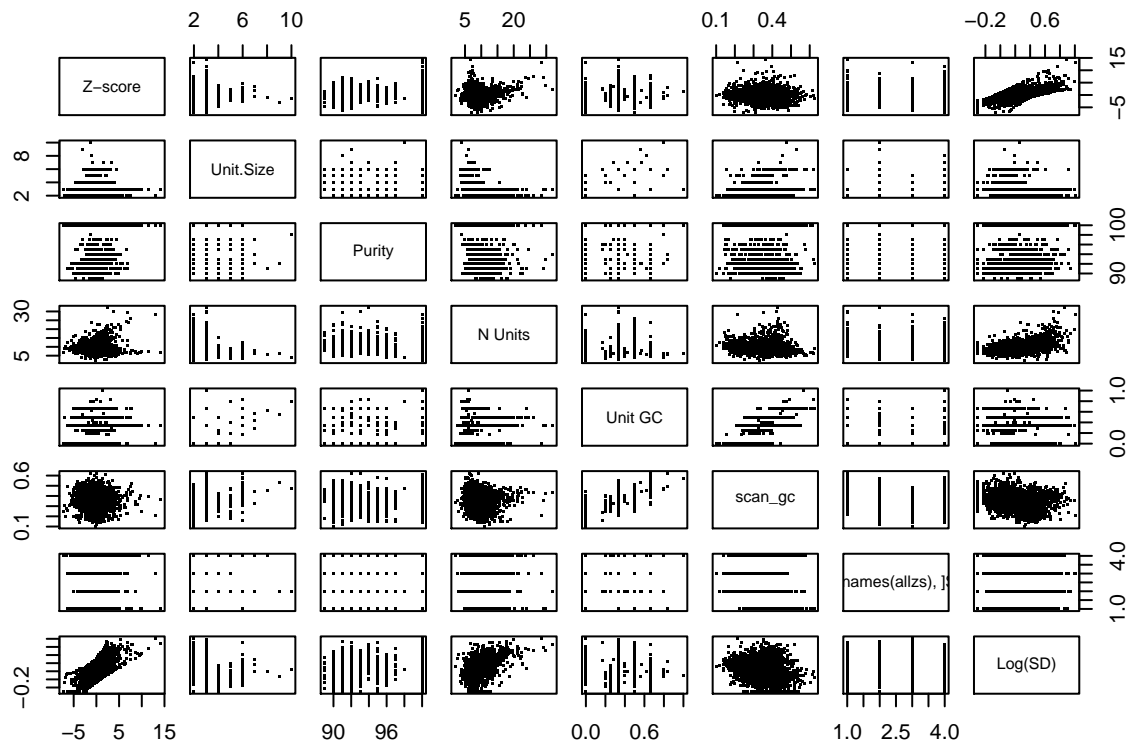




```

allzs = append(nczscores, codzscores)
allzs = append(allzs, intrzscores)
allzs = append(allzs, utrzscores)
zs_w_preds = cbind(allzs, svm_strs[names(allzs),], annotation[names(allzs),]$annotation,logged[names(allzs),])
colnames(zs_w_preds)[1] = 'Z-score'
colnames(zs_w_preds)[4] = 'N Units'
colnames(zs_w_preds)[5] = 'Unit GC'
#colnames(zs_w_preds)[10] = 'Log(SD)'
colnames(zs_w_preds)[8] = 'Log(SD)'
plot(zs_w_preds[, -9], pch='.', cex.main=.6)

```



```

# nothing has anything to do with DHSs
par(mfrow=c(1,2))
logged = logged[as.character(annotation$ID)]

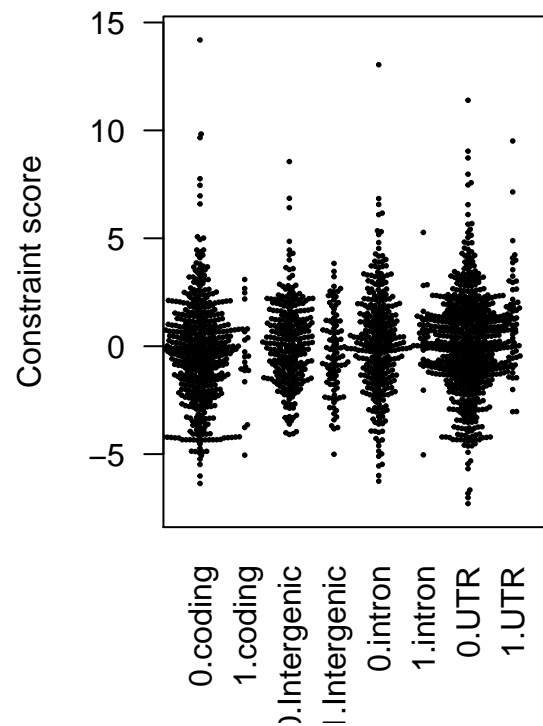
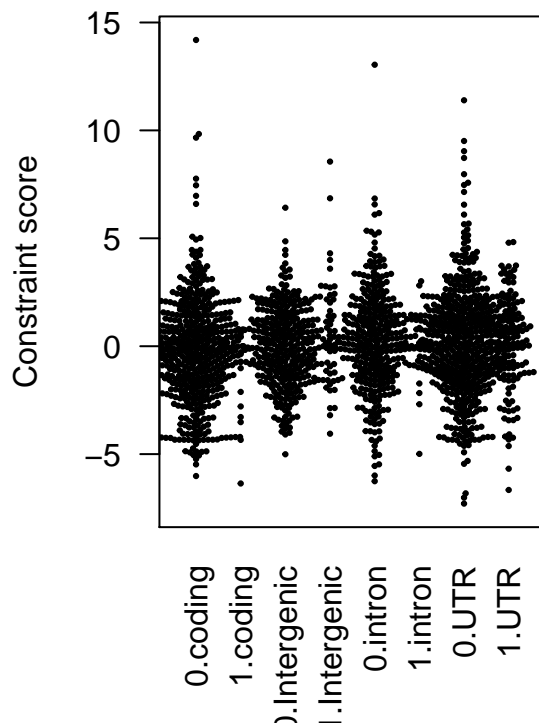
annotation$DHS[annotation$DHS>1] = 1

beeswarm(
  zs_w_preds$`Z-score` ~ annotation[rownames(zs_w_preds), 'DHS'] * annotation[rownames(zs_w_preds), 'annotation']
  pch=19, ylab='Constraint score', las=2, cex = .3, xlab=''
)

#TEs?
beeswarm(
  zs_w_preds$`Z-score` ~ annotation[rownames(zs_w_preds), 'transposon'] * annotation[rownames(zs_w_preds), 'annotation']
  pch=19, ylab='Constraint score', las=2, cex = .25, main='1=associated with TE, 0=not associated with TE'
)

```

associated with TE, 0=not associated



```
par(mfrow=c(1,1))
```

```
# arbitrary thresholds for stabilizing/diversifying selection
```

```
div_thresh = quantile(nczscores,.975)
```

```
stab_thresh = quantile(nczscores,.025)
```

```
codselneg = cbind(codzscores[codzscores<(stab_thresh)],strs[names(codzscores[codzscores<(stab_thresh)])])
```

```
codselpos = cbind(codzscores[codzscores>div_thresh],strs[names(codzscores[codzscores>div_thresh]),])
```

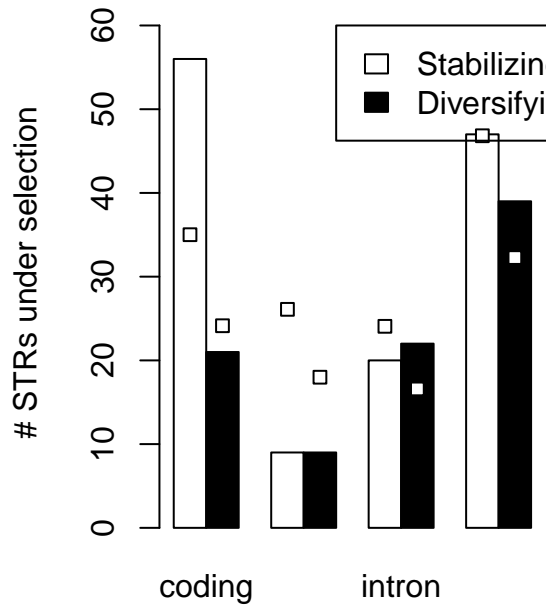
```
var_strs = cbind(strs[names(allzs),], allzs)
```

```
tabled_var_strs = table(var_strs$Consensus)
```

```
var_strs = var_strs[var_strs$Consensus %in% names(which(tabled_var_strs>4)),]
```

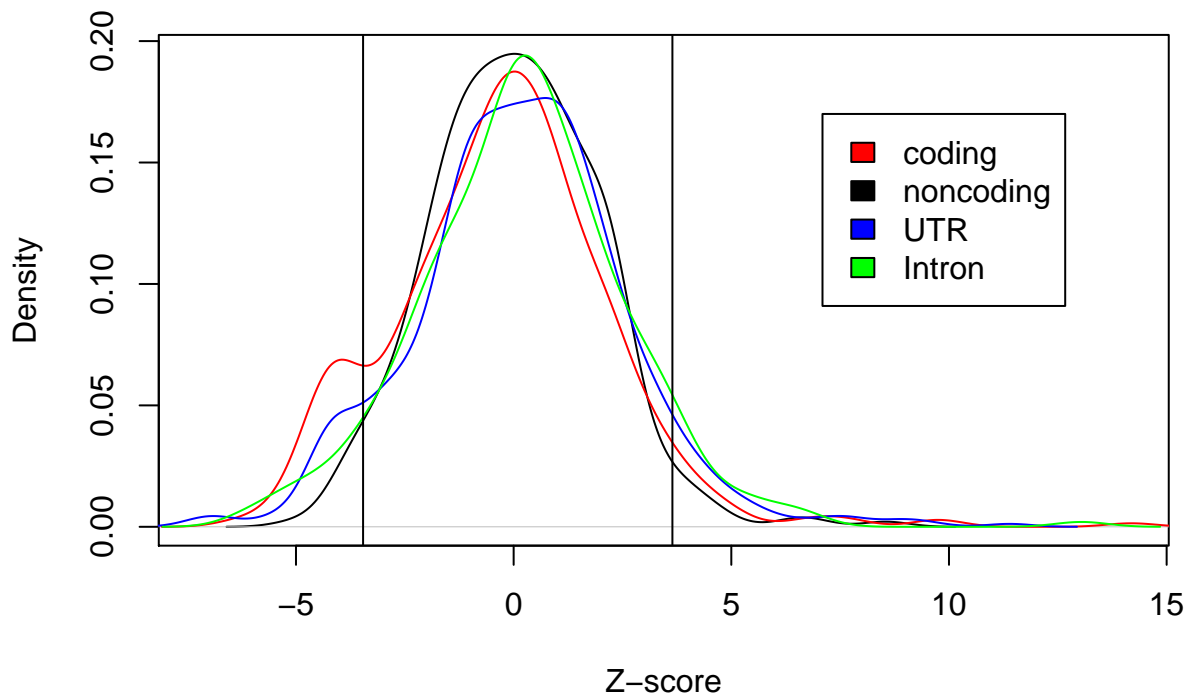
```
boxplot(var_strs$allzs~var_strs$Consensus, las=2, ylab='Z-scores')
```





```
plot(density(nczscores),xlab='Z-score',main='based on ensemble prediction',xlim=c(min(allzs),max(allzs)),
lines(density(codzscores),col='red')
lines(density(utrzscores),col='blue')
lines(density(intrzscores),col='green')
abline(v=div_thresh) #cutoff
abline(v=stab_thresh) #cutoff
legend(max(allzs)*.5,.17,legend=c('coding','noncoding','UTR','Intron'),fill=c('red','black','blue','green'))
```

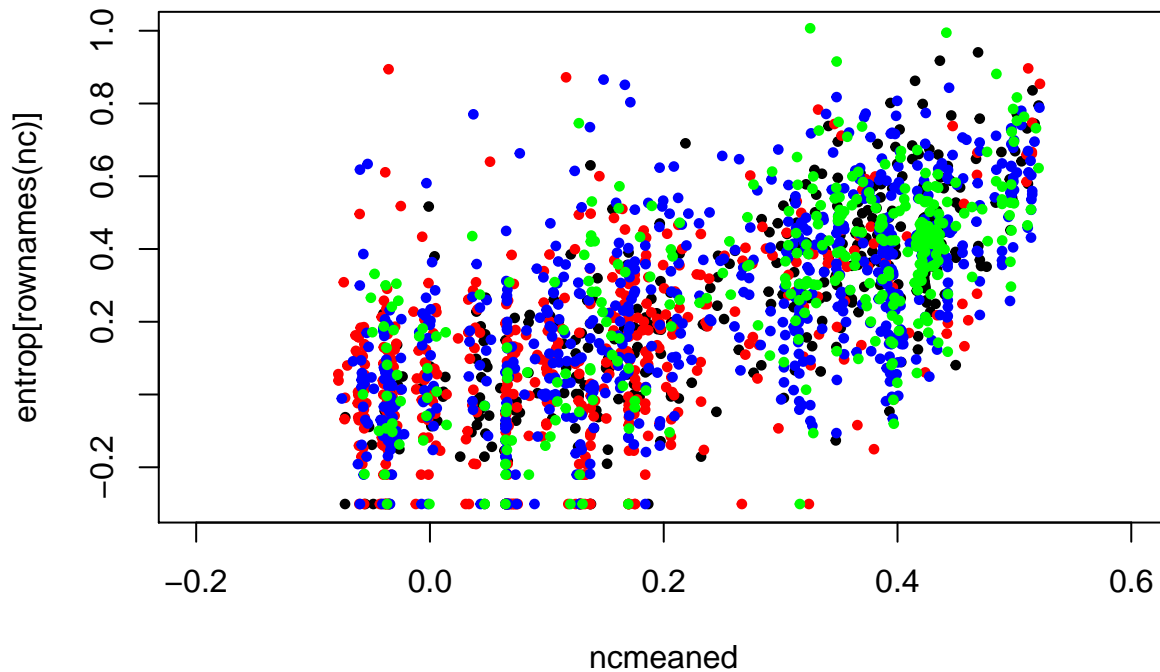
### based on ensemble prediction



```
# make color plot for bagged predictions
plot(ncmeaned,entrop[rownames(nc)],main=cor(ncmeaned,entrop[rownames(nc)]), pch=19, xlim=c(-.2,.6), ylim=c(0,1))
points(codmeaned,entrop[rownames(cod)],main=cor(codmeaned,entrop[rownames(cod)]), pch=19, col='red', cex=1.5)
```

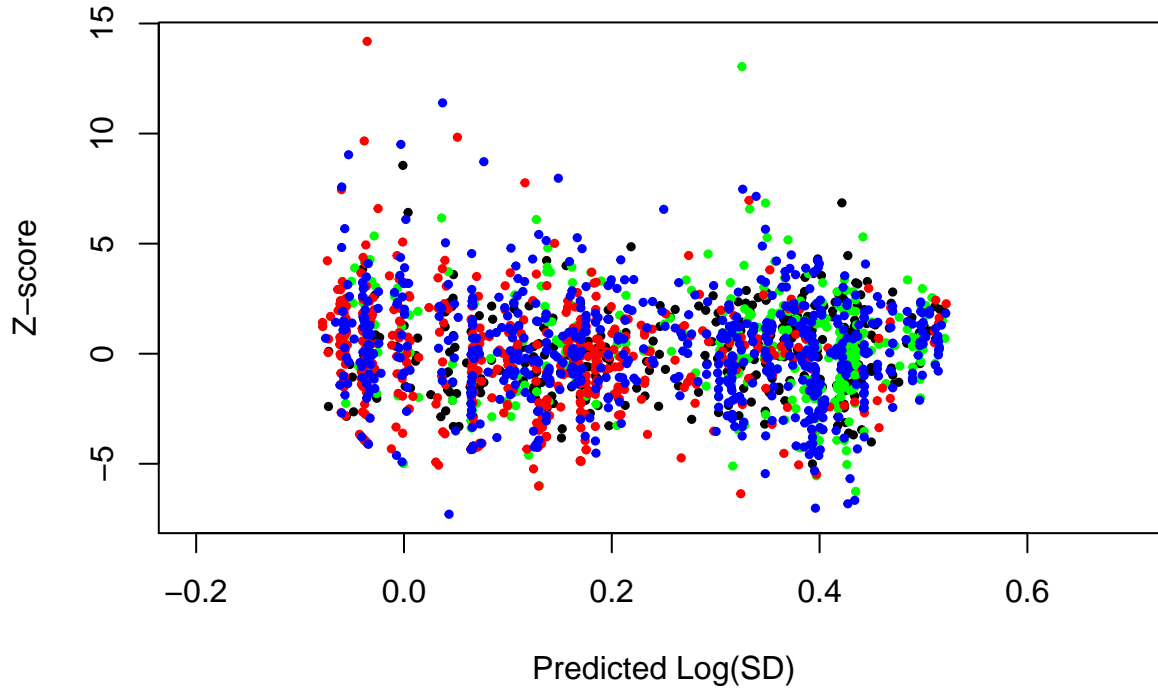
```
points(utrmeaned,entrop[rownames(utr)],main=cor(utrmeaned,entrop[rownames(utr)]), pch=19, col='blue',cex=19)
points(intrmeaned,entrop[rownames(intr)],main=cor(intrmeaned,entrop[rownames(intr)]), pch=19, col='green',cex=19)
```

0.758368635160983



```
# are deviations systematic??? not really...?
#plot(ncmeaned,nczscores,main=cor(nczscores,ncmeaned),xlim=c(-.5,.5),ylim=c(-12,12),cex=.2)
plot(ncmeaned,nczscores,main=cor(nczscores,ncmeaned),xlim=c(-.2,.7),
      ylim=c(min(allzs),max(allzs)),cex=.5, pch=19, xlab='Predicted Log(SD)',ylab='Z-score')
points(intrmeaned,intrzscores,main=cor(intrzscores,intrmeaned),col='green',xlim=c(-.5,.5), pch=19, ylim=c(-10,10))
points(codmeaned,codzscores,main=cor(codzscores,codmeaned),col='red',xlim=c(-.5,.5), pch=19, ylim=c(-10,10))
points(utrmeaned,utrzscores,main=cor(utrzscores,utrmeaned),col='blue',xlim=c(-.5,.5), pch=19, ylim=c(-10,10))
```

0.148192858385493



```
#predic = svr_pred_eval(svr=fit,newdata=sum_strs,str_ids = nc_test,response = entrop)
```

```
# pull out datasets representing each class of str/selxn regime
```

```
# these are all written to spreadsheets
```

```
nc_div_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`>div_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
utr_div_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`>div_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
intr_div_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`>div_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
intr_stab_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`<stab_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
utr_stab_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`<stab_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
nc_stab_sel = cbind(zs_w_preds[zs_w_preds$`Z-score`<stab_thresh & zs_w_preds$`annotation[names(allzs), ]$`
```

```
# give all identical consensus units for a consensus unit by frameshifting
```

```
enum_frames = function(string) {
```

```
  id = c(string)
```

```
  for (i in 1:(str_length(string)-1)) {
```

```
    string = paste(substr(string,2,str_length(string)), substr(string,1,1), sep='')
```

```
    id = append(id,string)
```

```
  }
```

```
  return(sort(id))
```

```
}
```

```
# take vector of units, count while accounting for degeneracy/RC
```

```
summarize_dedup_consensus = function(unit_vec) {
```

```
  require(seqinr)
```

```
  counts = c()
```

```
  maps = c()
```

```
  for (unit in unit_vec) {
```

```
    if (!(unit %in% names(maps))) {
```

```
      rt_unit = str_to_upper(paste(rev(comp(str_split(unit,'')[[1]])),collapse=''))
```

```

    degens = sort(append( enum_frames(unit), enum_frames(rt_unit) ))
    for ( degen in degens ) {
      maps[degen] = degens[1]
    }
    counts[maps[degen]] = 1
  } else {counts[maps[unit]] = counts[maps[unit]] + 1}
}
return(counts[sort(names(counts))])
}

# named vector x, names bkgrd, add 0 for missing names in x
expand_w_zeros = function(x,bkgrd) {
  x[bkgrd[!(bkgrd %in% names(x))]] = 0
  return(x)
}

# specifically for motif abundances
plot_deduped_consensus_freqs = function(div,stab,bkgrd,thresh=1,main) {
  bkgrd = summarize_dedup_consensus(bkgrd$Consensus)
  profs= rbind(
    expand_w_zeros(summarize_dedup_consensus(stab$Consensus),names(bkgrd))[names(bkgrd)],
    expand_w_zeros(summarize_dedup_consensus(div$Consensus),names(bkgrd))[names(bkgrd)]
  )
  to_plot = profs[,colSums(profs)>thresh]
  x_coords = seq(from = 1.5, to = 3*ncol(to_plot)+.5, by=3) # add one for beside col
  expected = sum(profs[1,]) * bkgrd[colnames(to_plot)] / sum(bkgrd)
  expected = rbind(expected, sum(profs[2,]) * bkgrd[colnames(to_plot)] / sum(bkgrd))
  barplot(to_plot, beside=TRUE, las=2, col=c('white','black'), ylim=c(0,max(expected)+2), ylab='Number of',
  points(append(x_coords,x_coords+1), append(expected[1,],expected[2,]),pch=22,bg='white',col='black',col.lab='Number of',col.pch='white'))
}

# plot out unit consensus distributions for all classes selected STRs
# white is diversifying, black is stabilizing
par(mfrow=c(2,2))
plot_deduped_consensus_freqs(codselpos,codselneg,annotation[annotation$annotation=='coding',],thresh=0,

## Loading required package: seqinr

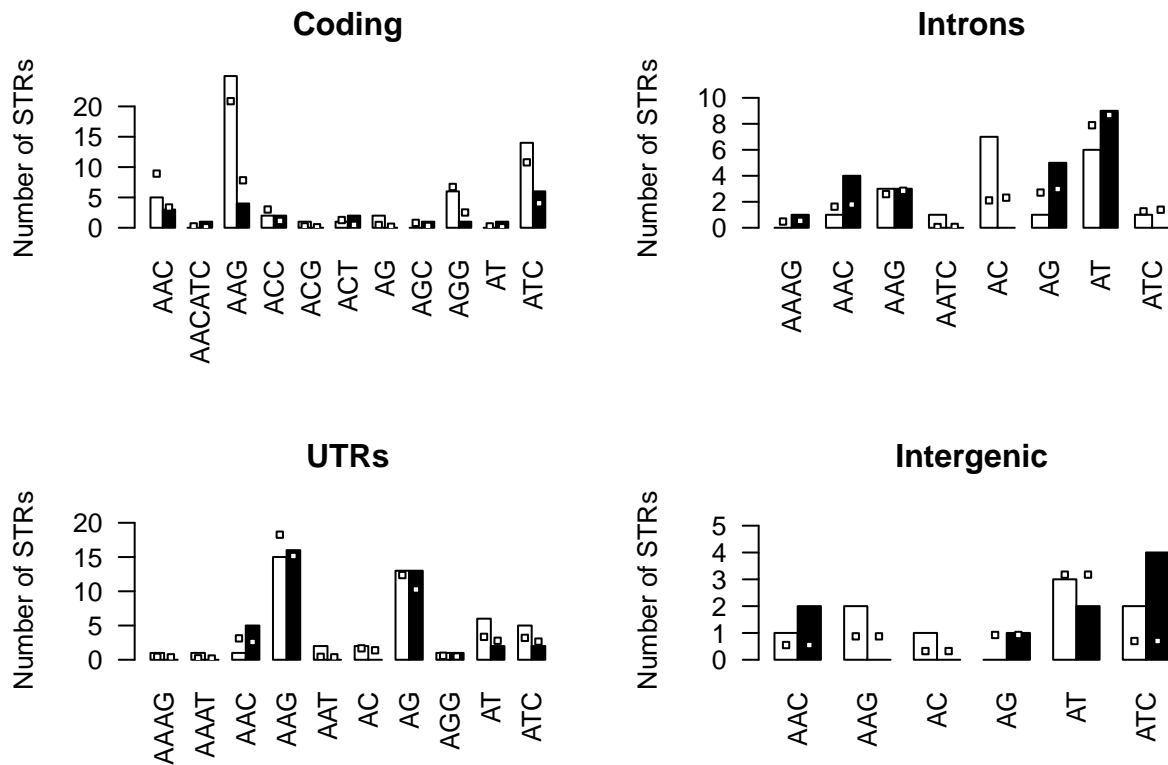
##
## Attaching package: 'seqinr'

## The following object is masked from 'package:kernlab':
##
##     rho

## The following object is masked from 'package:permute':
##
##     getType

plot_deduped_consensus_freqs(intr_div_sel,intr_stab_sel,annotation[annotation$annotation=='intron',],thresh=0,
plot_deduped_consensus_freqs(utr_div_sel,utr_stab_sel,annotation[annotation$annotation=='UTR',],thresh=0,
plot_deduped_consensus_freqs(nc_div_sel,nc_stab_sel,annotation[annotation$annotation=='Intergenic',],thresh=0,

```



```
par(mfrow=c(1,1))

# find str expansions (relative)
# takes str allele distribution as input
find_expansions = function(matrix,str_id) {
  distr = as.numeric(matrix[str_id,])
  if (length(distr)==0) {return(NA)}
  med = median(na.omit(distr))
  meded = (distr-med) / med
  return(meded)
}

medea = all_genos
medea[!(is.na(medea))] = NA
for (str in rownames(all_genos)) {
  medea[str,] = find_expansions(all_genos,str )
}
max_med = apply(medea,1,max,na.rm=TRUE)
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```











```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

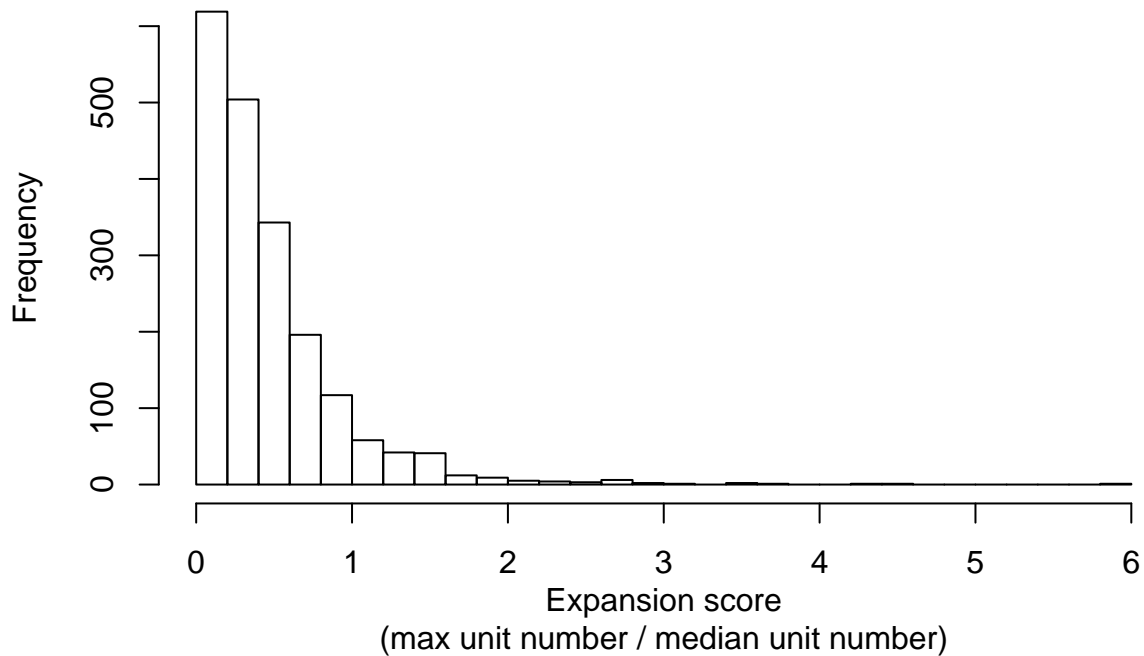
```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

```
max_in_strain = apply(medea,2,max,na.rm=TRUE)
expanded = max_in_strain[max_in_strain>=2]
cat('num strains with >=1 expanded STR (>=2X median copy number):',length(expanded),'of',length(max_in_strain))
```

```
## num strains with >=1 expanded STR (>=2X median copy number): 64 of 96 strains, involving 28 STRs.
```

```
hist(max_med,40,xlab='Expansion score\n(max unit number / median unit number)',main='')

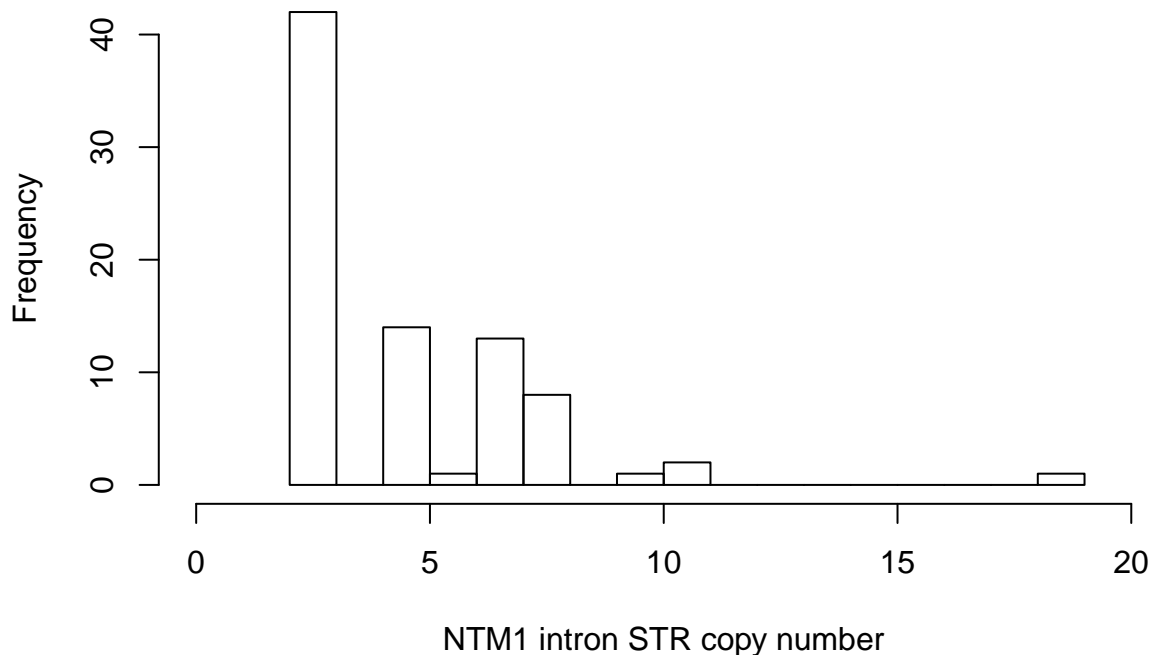
```



```
expanded = na.omit(medea[medea>=2]) #totally arbitrary
```

```
# ntm1 intronic str was interesting (expansion score ~ 6):
```

```
hist(as.numeric(genos['65400',]),20,main='',xlab='NTM1 intron STR copy number',xlim=c(0,20))
```



```
# repopulate this, to get back filtered-out strs
annotation = read.delim('araport_annot/Ath_STRs_full_annotations_031317.tsv',header=T)
rownames(annotation) = annotation$ID
# chars of the expanded strs
expansion_strs = cbind(annotation[names(max_med[max_med>=2]),], max_med[names(which(max_med>=2))])

# write out a table of expanded things
write.table(expansion_strs, 'output/expanded_strs_table_s2_010917.txt',sep='\t',quote=FALSE)
table(expansion_strs$annotation)

##
##      coding Intergenic      intron      UTR
##          6          5          6        11

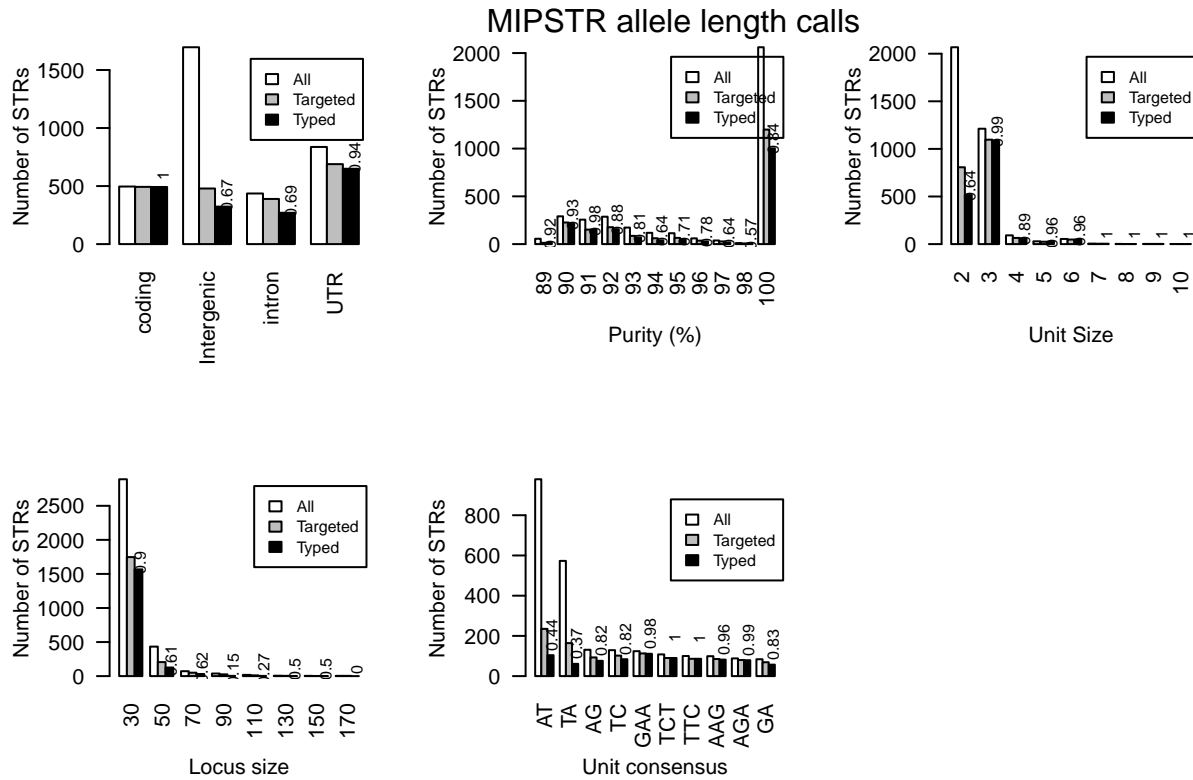
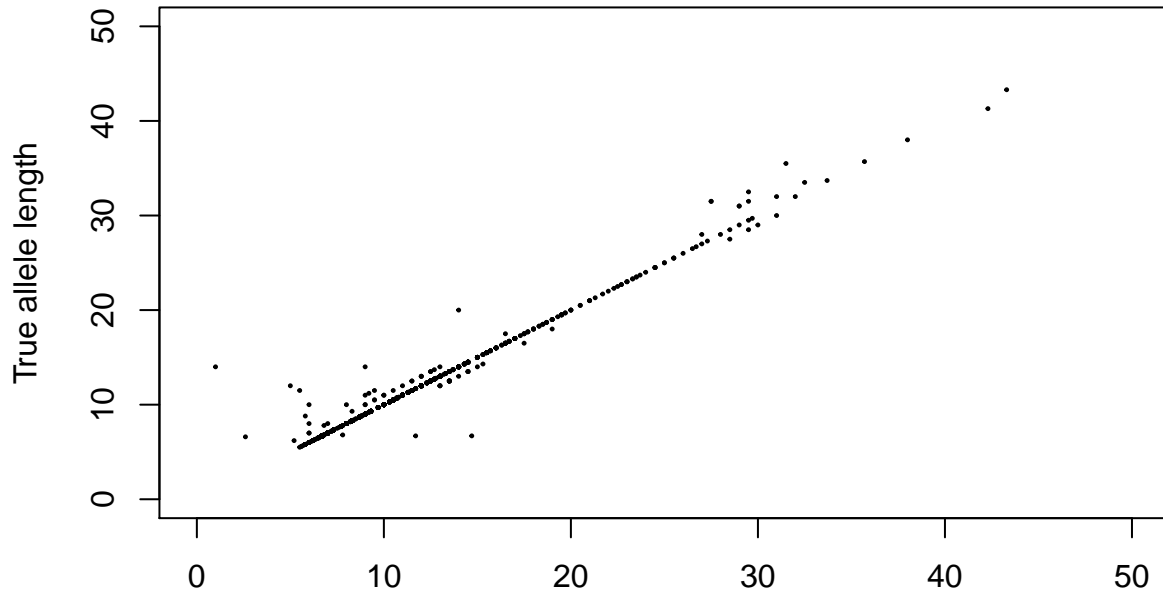
# avg missing genotypes for expanded STRs
exp_missing = sum(nacount[rownames(expansion_strs)])/(nrow(expansion_strs) * 96)

# avg missing genotypes for non-expanded STRs with at least two calls
# (possible to have a non-zero expansion score)
non_exp = names(nacount)[
  !(names(nacount) %in% rownames(expansion_strs))
  & nacount < 95 ]
non_exp_missing = sum(nacount[non_exp])/ (length(non_exp) * 96)

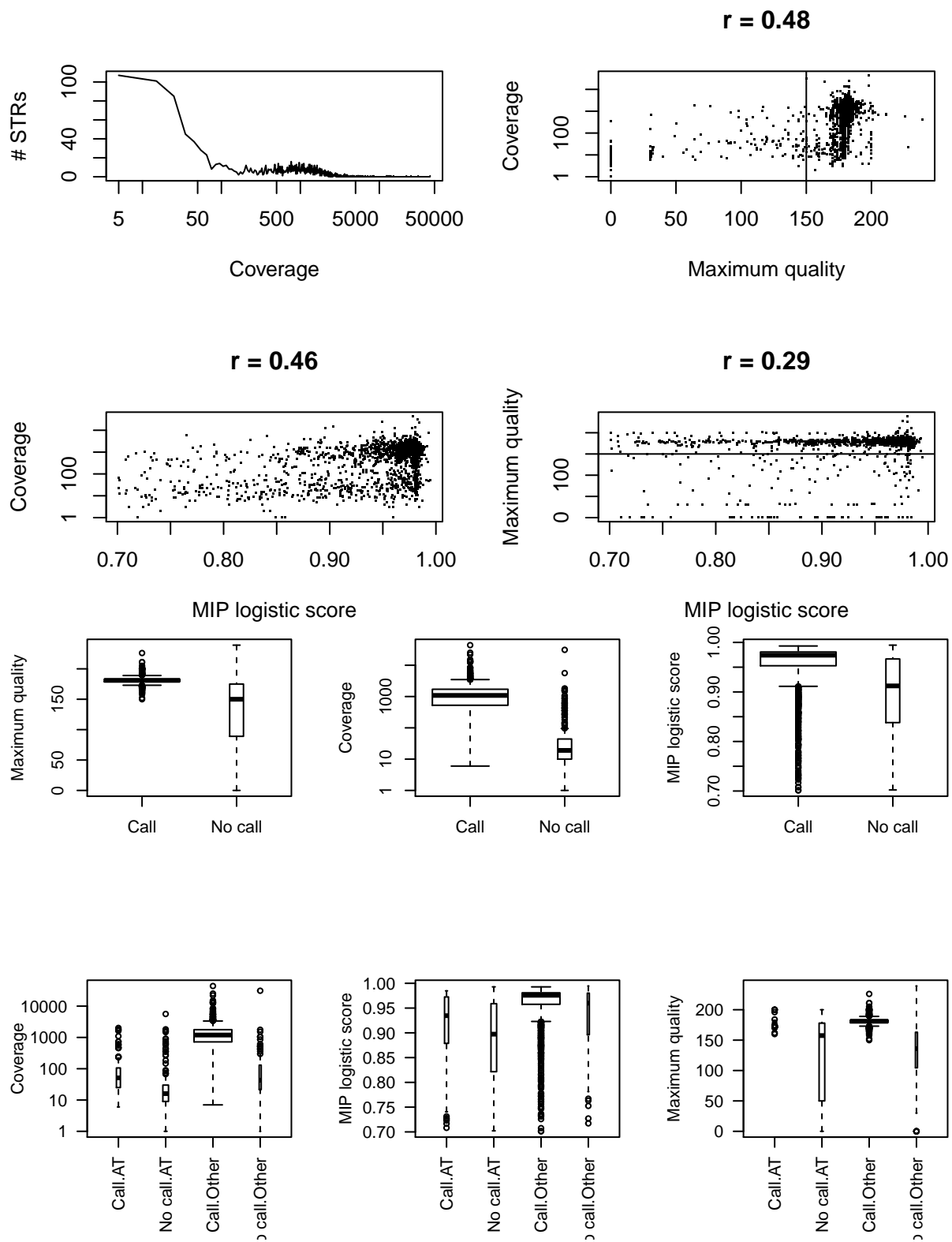
cat('expanded STRs have',exp_missing,'proportion missing data compared to',
    non_exp_missing, 'for comparable STRs\n')

## expanded STRs have 0.2105655 proportion missing data compared to 0.1568653 for comparable STRs
###
# CALLING EXTERNAL SCRIPTS THAT MAKE FIGURES ETC.
###
source('code/AthMips_figs1_design_010217.R')
```

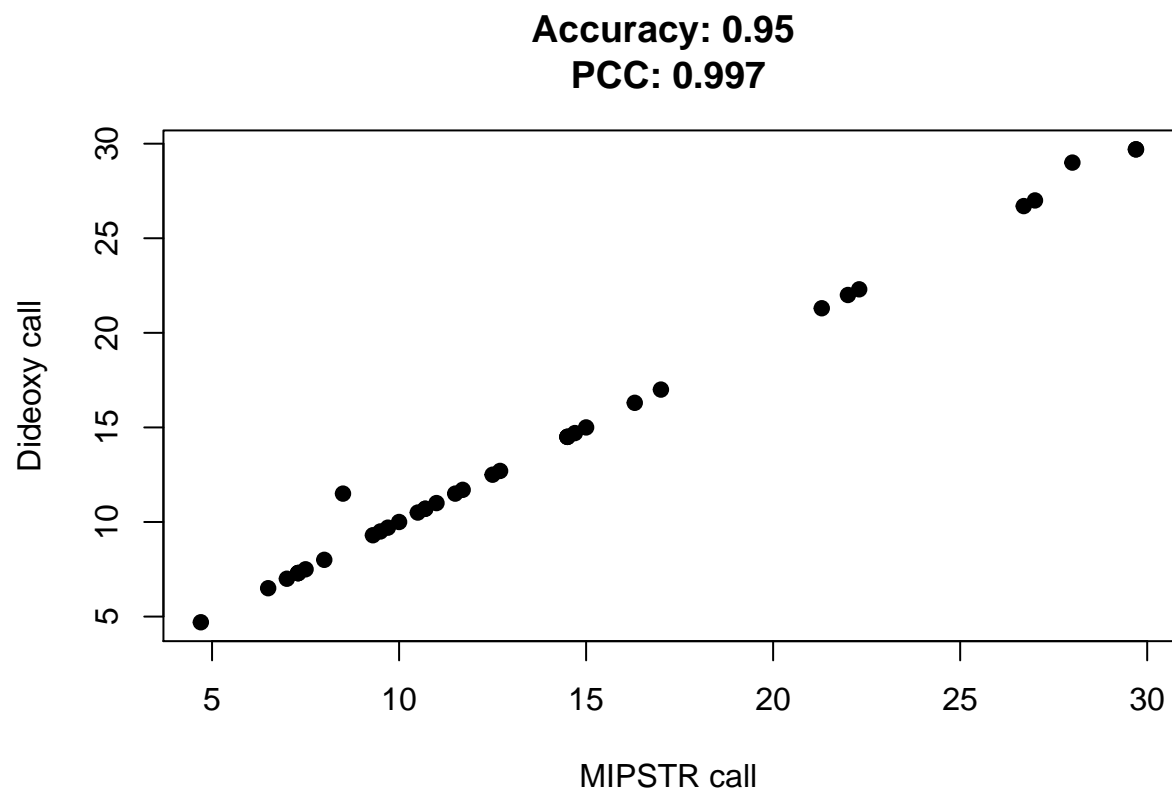
Accuracy = 96%, r = 0.99



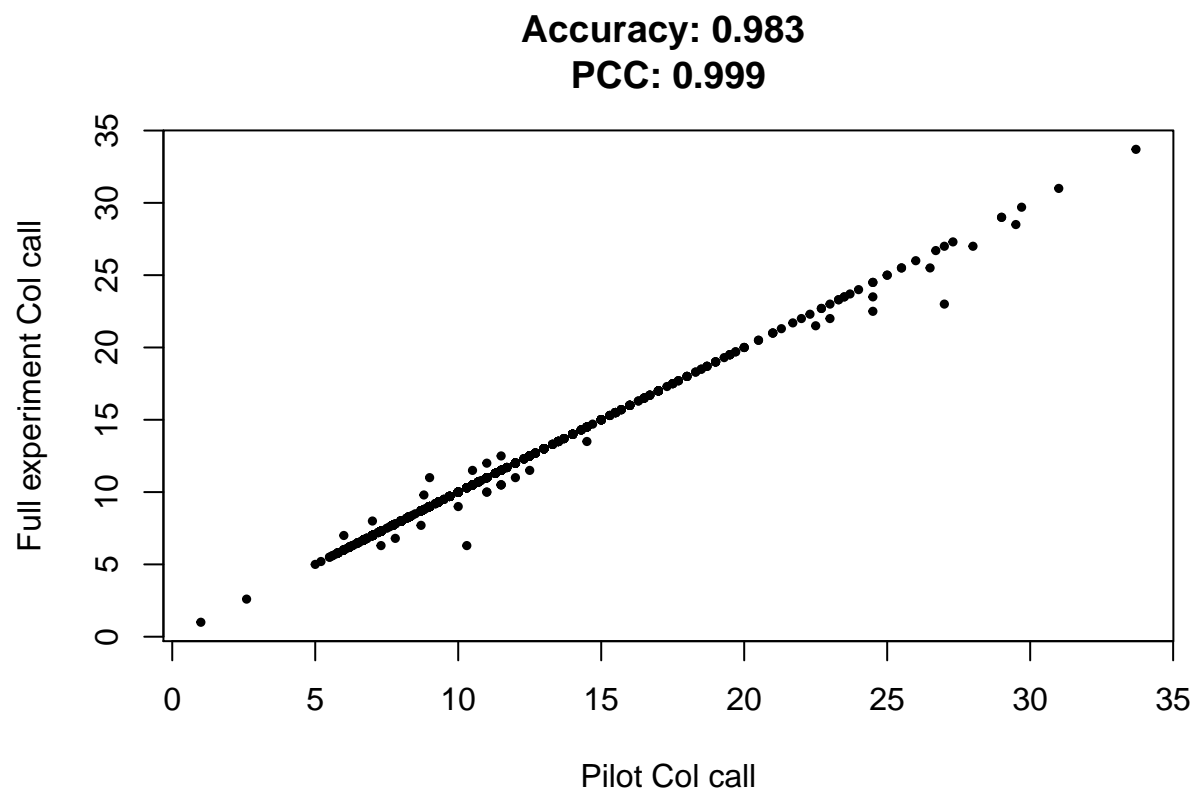
```
## overall genotyping success rate: 0.8458537
## success rate split into AT/TA vs. others:
## AT/TA: 0.4135338
## all others: 0.9503331
```



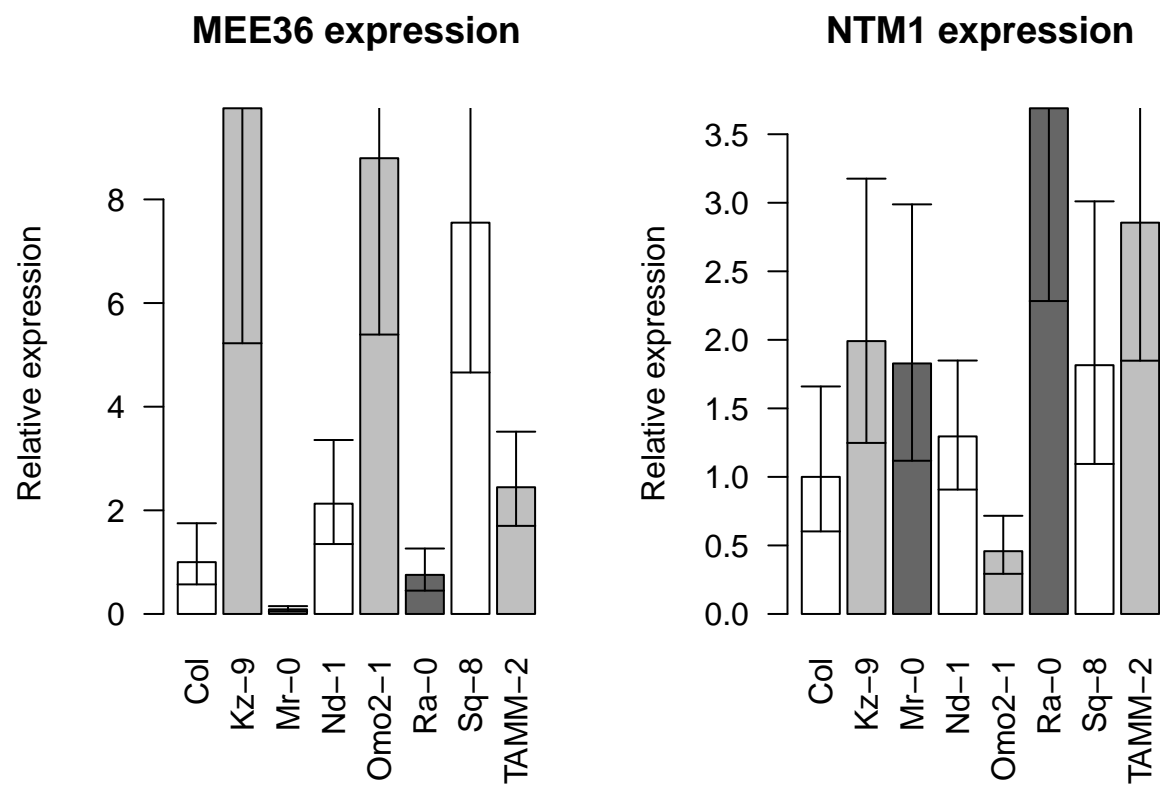




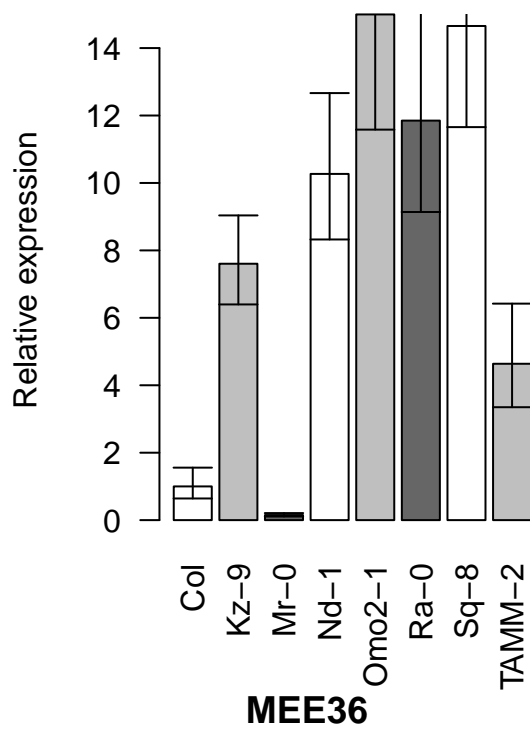
```
## [1] "evaluating technical reproducibility between pilot and big expts for Col"
##
## Pearson's product-moment correlation
##
## data: twocalls[, 1] and twocalls[, 2]
## t = 786.6, df = 1682, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.9985076 0.9987672
## sample estimates:
## cor
## 0.9986435
##
## number calls the same for loci called in both: 1656 out of 1684 total, for accuracy: 0.9833729
```



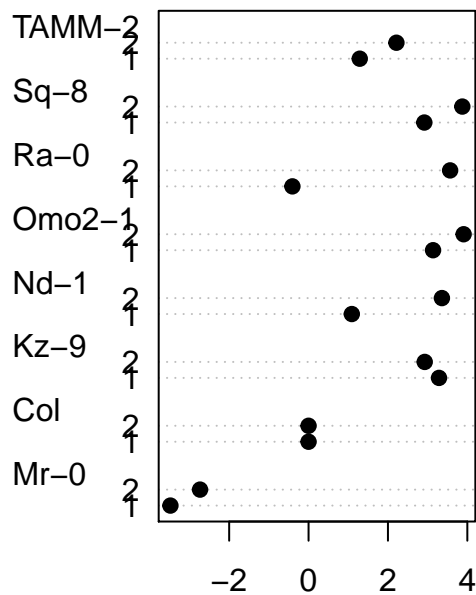
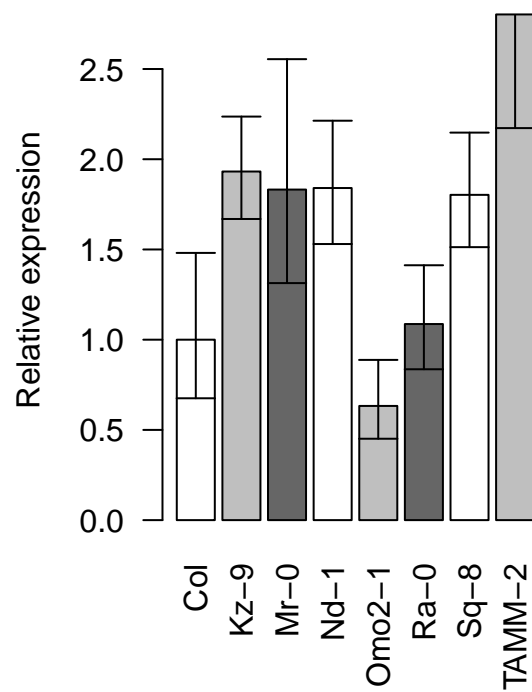
```
source('code/qpcr_analysis_65400_47883_2bioreps_100216.R')
```



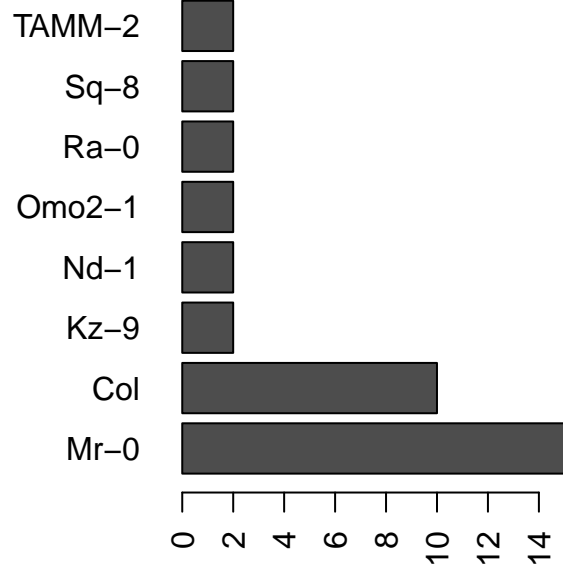
**MEE36 expression**



**NTM1 expression**



Log2(Relative expression)



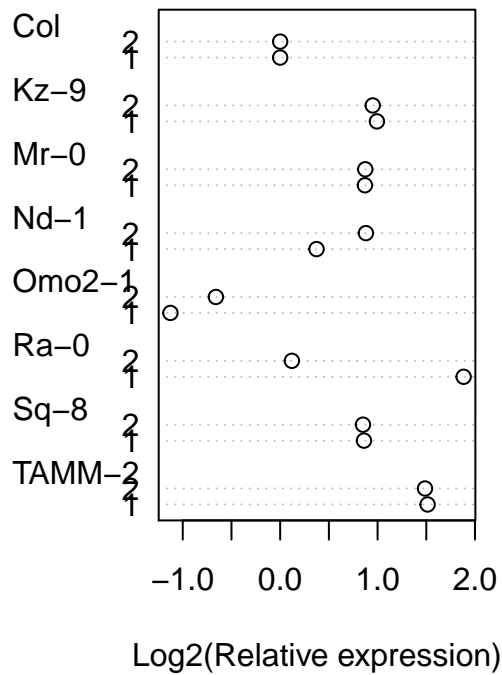
MEE36 STR unit copy number

```
source('code/str_pheno_assoc_post_analysis_032017.R')
```

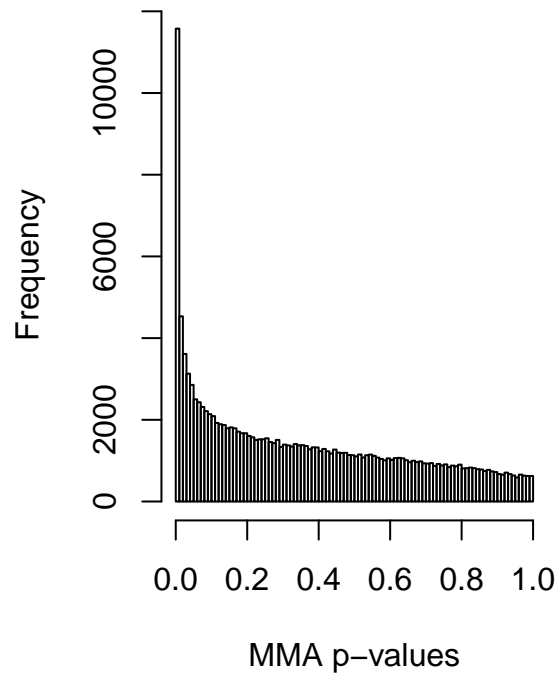
```
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
```

```
##
##      lowess
## Loading required package: survival
##
## Attaching package: 'survival'
## The following object is masked from 'package:DAAG':
##
##      lung
## Loading required package: bdsmatrix
##
## Attaching package: 'bdsmatrix'
## The following object is masked from 'package:base':
##
##      backsolve
```

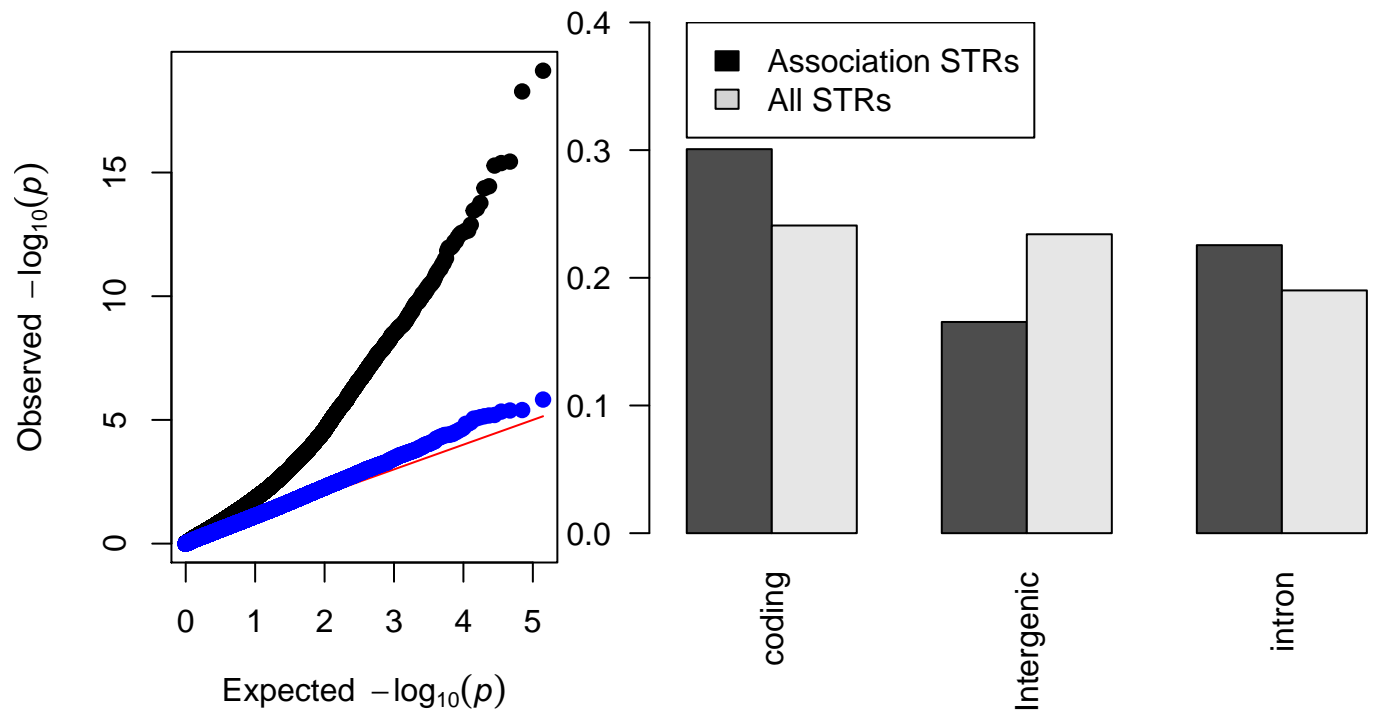
**NTM1**



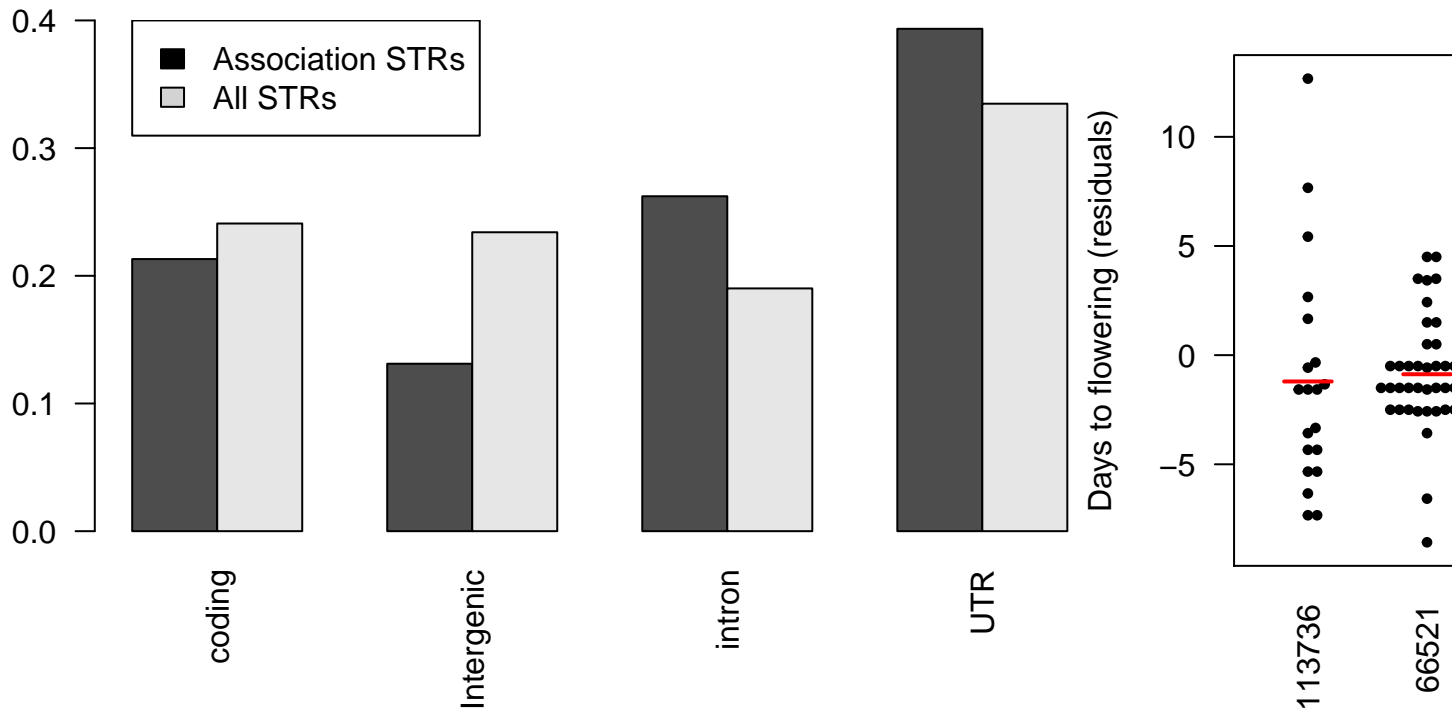
**Histogram of mmaps**



```
## [1] "HEATMAP IS WRITTEN AS PDF"
```



```
## Loading required package: Matrix
##
## Attaching package: 'lmerTest'
## The following object is masked from 'package:lme4':
##
##     lmer
## The following object is masked from 'package:stats':
##
##     step
```



```
## Linear mixed model fit by REML t-tests use Satterthwaite approximations
## to degrees of freedom [lmerMod]
## Formula:
## DTF ~ factor(genotype_label, levels = c("Col", "66521", "113736")) +
## (1 | tray_row) + (1 | expt)
## Data: all_expts
##
## REML criterion at convergence: 651.5
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.1080 -0.4026 -0.1391  0.3876  3.6656
##
## Random effects:
## Groups   Name                Variance Std.Dev.
## tray_row (Intercept)    0.00      0.000
## expt      (Intercept)   20.75      4.555
## Residual                    14.43      3.798
## Number of obs: 118, groups: tray_row, 6; expt, 3
##
## Fixed effects:
##                                     Estimate
## (Intercept)                        35.6405
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521    -1.9292
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736   -2.4498
##                                     Std. Error
## (Intercept)                        2.6857
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521     0.8177
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736     1.1151
##                                     df
## (Intercept)                        2.0800
```

```

## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 113.6300
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 114.0500
##
## (Intercept)
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 -2.359
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 -2.197
##
## Pr(>|t|)
## (Intercept)
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 0.02001
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 0.03004
##
## (Intercept)
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 *
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##
## (Intr) f(,l=c("C","66521","113736"))6
## f(,l=c("C","66521","113736"))6 -0.107
## f(,l=c("C","66521","113736"))1 -0.114 0.153
## Linear mixed model fit by REML t-tests use Satterthwaite approximations
## to degrees of freedom [lmerMod]
## Formula:
## RLN ~ factor(genotype_label, levels = c("Col", "66521", "113736")) +
## (1 | tray_row) + (1 | expt)
## Data: all_expts
##
## REML criterion at convergence: 482.6
##
## Scaled residuals:
## Min 1Q Median 3Q Max
## -3.1761 -0.5138 0.0698 0.4749 3.1241
##
## Random effects:
## Groups Name Variance Std.Dev.
## tray_row (Intercept) 2.053 1.433
## expt (Intercept) 3.562 1.887
## Residual 3.015 1.736
## Number of obs: 118, groups: tray_row, 6; expt, 3
##
## Fixed effects:
##
## Estimate
## (Intercept) 13.0966
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 -0.6932
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 -1.5436
##
## Std. Error
## (Intercept) 1.2705
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 0.3755
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 0.5113
##
## df
## (Intercept) 3.1200
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 106.6400
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 107.3200

```

```
##
## (Intercept) t value
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 -1.846
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 -3.019
## Pr(>|t|)
## (Intercept) 0.00163
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 0.06765
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 0.00317
##
## (Intercept) **
## factor(genotype_label, levels = c("Col", "66521", "113736"))66521 .
## factor(genotype_label, levels = c("Col", "66521", "113736"))113736 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
## (Intr) f(,l=c("C","66521","113736"))6
## f(,l=c("C","66521","113736"))6 -0.102
## f(,l=c("C","66521","113736"))1 -0.112 0.152
```

```
source('code/effectsize_plot_032417.R')
```

