What controls the range of hosts a fish parasite infects?

Tad Dallas, Andrew Park, and John M. Drake

Knowledge gap

- What constrains the range of hosts that a parasite can infect? Is there a simple range of host functional traits that can determine the likelihood that a parasite infects a given host species? How well can we predict parasite occurrences given host life history traits? How about using solely information on parasite community structure?
- Does the importance of different host functional traits or parasite community information differ with parasite type? (supplement)
- Since geographic variables are important, what if we try to predict parasite niche breadth in a specific biogeographic region? (supplement)

Thesis paragraph

Here, I apply a series of predictive models in order to predict parasite occurrence across a range of potential host species for a large set of parasites of freshwater fish, using host functional traits, geographic location, and parasite community information. Parasite occurrence on a given host could be constrained by space (geographic location), patch quality (host characteristics), or through interactions with competing parasites (parasite community structure). It's important to note right up front that the importance of parasite community structure cannot be interpreted as evidence for community interactions, as parasites could infect hosts based on their traits, and the parasite community information could just be serving as a proxy for unmeasured host trait variation. However, predicting parasite occurrence based solely on parasite community information does remove some importance of the patch (host), and is easy to sell, as it may be possible to predict spillover of parasites, or the degree of biotic resistance a community offers to a potential invader, simply by having presence-absence data on parasite communities.

I examine the role of parasite group (Acanthocephala, Cestoda, Monogenea, Nematoda, Trematoda) on model performance, and on the relative contribution of different variables to prediction.

thorns in my side:

- Absence data aren't true absences. Should I even train on these data if the model treats them as true absences? I think I'm basically treating them as background data, much like MaxEnt.

Methods

Data and processing

We use an existing global database of fish-parasite associations (Strona et al. 2013) consisting of over 38000 parasite records spanning a large diversity of parasites (Acanthocephala, Cestoda, Monogenea, Nematoda, Trematoda). In order to allow for cross-validation and accurate prediction, we constrained our ananlyses to parasites with a minimum of 50 host records. In other words, we only examined parasites that had been recorded more than 50 times, but these occurrences could be on fewer than 50 host species. The inclusion of duplicate occurrences was only permitted if the parasite was recorded on a host in a different location, based on latitude and longitude values. Our response variable was parasite occurrence (binary), and was predicted using only host life history traits, and geographic location of host capture. Host trait information was obtained through the FishPest database (Strona and Lafferty 2012; Strona et al. 2013), and FishBase (Froese and Pauly 2010). Host traits descriptions are provided in Table 1

Model formulation

We trained a series of models in order to compare predictive performance of different techniques. Each model was trained on 70% of the data, and accuracy was determined from the remaining 30%. This process was repeated z times ($z = 20$). We generated background data by randomly sampling host species where parasite i was not recorded. To maintain proportional training data, the number of random samples was selected to be five times greater than the occurrence records.

Models used

discuss null predictions scenario, and then go into other algorithms used (brt, svm, lr, rf)

```r
# storage for model outputs
baseline.auc=vector();
brtModel=list(); brt.best.iter=list(); brt.preds=list(); brt.perf=list(); brt.perfAUC=vector()
svmModel=list(); svm.preds=list(); svm.perf=list(); svm.perfAUC=vector()
lrModel=list(); lr.preds=list(); lr.perf=list(); lr.perfAUC=vector()
rfModel=list(); rf.preds=list(); rf.perf=list(); rf.perfAUC=vector()


## Creating a pointsObject
#hostPest = unique(pest[,-c(1:3)], MARGIN=1)
hostPest = pest[,-c(1:3)]
parPest = names(which(summary(pest1[,'P_SP'], maxsum=500)>20));
parPest=parPest[-max(length(parPest))]

ret=list()
for(i in 1:length(parPest)){
#create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)


#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:4)]
  hostsOfi=dat[,1]

#Impute the data
  impDat=rfImpute(dat[,-1], presence1)
  dat=impDat[,-1]
```

```
flag=0
 while(flag == 0){
  # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
  inds=sample(1:nrow(dat), 0.7*nrow(dat))
  if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

  #Set up a prelim train set and a test set
  train = dat[inds,]
  test = dat[-inds,]
  if(all(unique(train$GEO) %in% unique(test$GEO))){flag=1}
 }

 #Presences
  prezTR=presence1[inds]
  prezTE=presence1[-inds]


  #baseline expectations and null models
  baseline.auc[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), presence[-i

 ##trained models
  #boosted regression trees
    brtModel[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution='bernou
    brt.best.iter[[i]] = gbm.perf(brtModel[[i]], method="OOB")
    brt.preds[[i]] = prediction(predict(brtModel[[i]], newdata=test, n.trees=brt.best.iter[[i]]), prezTE
    brt.perf[[i]] = performance(brt.preds[[i]],"tpr","fpr")
    brt.perfAUC[i]=unlist(performance(brt.preds[[i]], 'auc')@y.values)

  #support vector machines
    svmModel[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
    svm.preds[[i]] = prediction(predict(svmModel[[i]], test), prezTE)
    svm.perf[[i]] = performance(svm.preds[[i]],"tpr","fpr")
    svm.perfAUC[i] = unlist(performance(svm.preds[[i]], 'auc')@y.values)

  #logistic regression
    lrModel[[i]] = glm(prezTR ~ ., data=train, family=binomial)
    lr.preds[[i]] = prediction(predict(lrModel[[i]], test), prezTE)
    lr.perf[[i]] = performance(lr.preds[[i]],"tpr","fpr")
    lr.perfAUC[i] = unlist(performance(lr.preds[[i]], 'auc')@y.values)

  #random forest
    rfModel[[i]] = randomForest(prezTR ~ ., data=train)
    rf.preds[[i]] = prediction(predict(rfModel[[i]], test), prezTE)
    rf.perf[[i]] = performance(rf.preds[[i]],"tpr","fpr")
    rf.perfAUC[i] = unlist(performance(rf.preds[[i]], 'auc')@y.values)
  print(i)
  }

 ret=cbind(baseline.auc[1:241], brt.perfAUC[1:241], svm.perfAUC[1:241], lr.perfAUC[1:241], rf.perfAUC[1
 colnames(ret)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

Training using only host trait data

```
# storage for model outputs
baseline.auc.b=vector();
brtModel.b=list(); brt.best.iter.b=list(); brt.preds.b=list(); brt.perf.b=list(); brt.perfAUC.b=vector(
svmModel.b=list(); svm.preds.b=list(); svm.perf.b=list(); svm.perfAUC.b=vector()
lrModel.b=list(); lr.preds.b=list(); lr.perf.b=list(); lr.perfAUC.b=vector()
rfModel.b=list(); rf.preds.b=list(); rf.perf.b=list(); rf.perfAUC.b=vector()


## Creating a pointsObject
#hostPest = unique(pest[,-c(1:3)], MARGIN=1)
hostPest = pest[,-c(1:3)]
parPest = names(which(summary(pest[,'P_SP'], maxsum=500)>19));
parPest=parPest[-max(length(parPest))]

for(i in 1:length(parPest)){
 #create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)


#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:4)]
  hostsOfi=dat[,1]

#Impute the data
 # impDat=rfImpute(dat[,-1], presence1)
 # dat=impDat[,-1]


###### If you want to include parasite richness (or composition) ######
 psr = rowSums(intMat)
#PCA of parasite communities...otherwise you're including like 500 new variables
pcaHosts = princomp(intMat[,-which(colnames(intMat)==parPest[i])])$scores

psrOut=vector()
pcaHostsOut=matrix(0, nrow=length(hostsOfi), ncol=10)
for(e in 1:length(hostsOfi)){
  psrOut[e] = psr[which(hostsOfi[e] == rownames(intMat))]
  pcaHostsOut[e,] = pcaHosts[which(hostsOfi[e] == rownames(intMat)), 1:10]
}



dat = data.frame(psrOut, pcaHostsOut)
```

```
 # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
 inds=sample(1:nrow(dat), 0.7*nrow(dat))
 if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

 #Set up a prelim train set and a test set
 train = dat[inds,]
 test = dat[-inds,]


#Presences
 prezTR=presence1[inds]
 prezTE=presence1[-inds]


 #baseline expectations and null models
 baseline.auc.b[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), presence

##trained models
 #boosted regression trees
   brtModel.b[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution='bern
   brt.best.iter.b[[i]] = gbm.perf(brtModel.b[[i]], method="OOB")
   brt.preds.b[[i]] = prediction(predict(brtModel.b[[i]], newdata=test, n.trees=brt.best.iter.b[[i]]),
   brt.perf.b[[i]] = performance(brt.preds.b[[i]],"tpr","fpr")
   brt.perfAUC.b[i]=unlist(performance(brt.preds.b[[i]], 'auc')@y.values)

 #support vector machines
   svmModel.b[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
   svm.preds.b[[i]] = prediction(predict(svmModel.b[[i]], test), prezTE)
   svm.perf.b[[i]] = performance(svm.preds.b[[i]],"tpr","fpr")
   svm.perfAUC.b[i] = unlist(performance(svm.preds.b[[i]], 'auc')@y.values)

 #logistic regression
   lrModel.b[[i]] = glm(prezTR ~ ., data=train, family=binomial)
   lr.preds.b[[i]] = prediction(predict(lrModel.b[[i]], test), prezTE)
   lr.perf.b[[i]] = performance(lr.preds.b[[i]],"tpr","fpr")
   lr.perfAUC.b[i] = unlist(performance(lr.preds.b[[i]], 'auc')@y.values)

 #random forest
   rfModel.b[[i]] = randomForest(prezTR ~ ., data=train)
   rf.preds.b[[i]] = prediction(predict(rfModel.b[[i]], test), prezTE)
   rf.perf.b[[i]] = performance(rf.preds.b[[i]],"tpr","fpr")
   rf.perfAUC.b[i] = unlist(performance(rf.preds.b[[i]], 'auc')@y.values)
 print(i)
 }

ret.b=cbind(baseline.auc.b, brt.perfAUC.b, svm.perfAUC.b, lr.perfAUC.b, rf.perfAUC.b)
colnames(ret.b)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

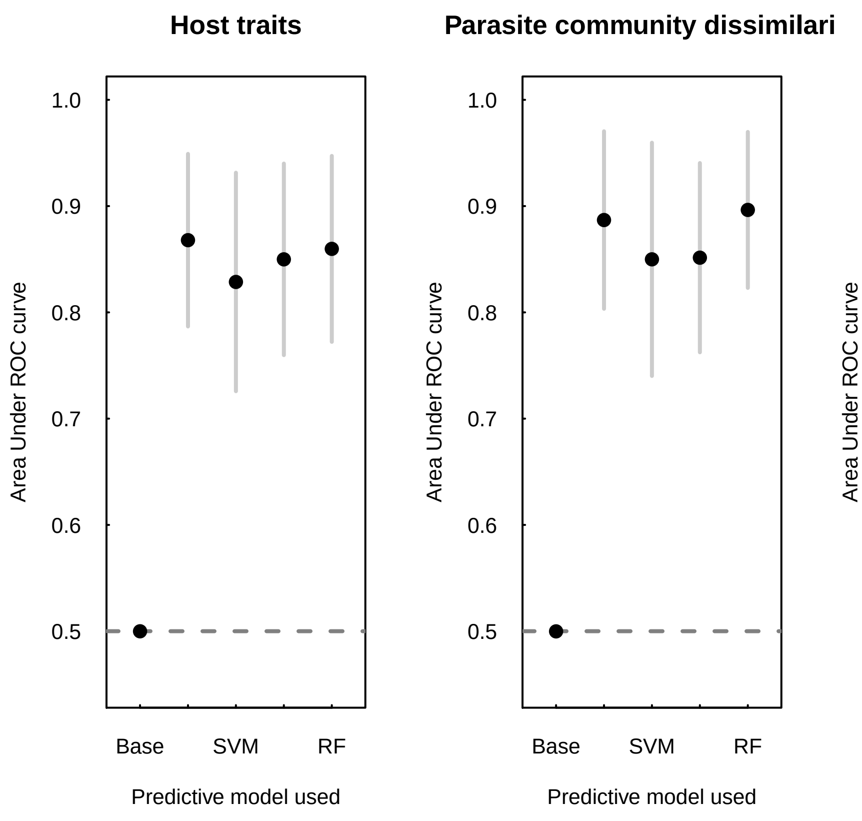
Training using only parasite community data

```
# storage for model outputs
baseline.auc.c=vector();
brtModel.c=list(); brt.best.iter.c=list(); brt.preds.c=list(); brt.perf.c=list(); brt.perfAUC.c=vector(
svmModel.c=list(); svm.preds.c=list(); svm.perf.c=list(); svm.perfAUC.c=vector()
lrModel.c=list(); lr.preds.c=list(); lr.perf.c=list(); lr.perfAUC.c=vector()
rfModel.c=list(); rf.preds.c=list(); rf.perf.c=list(); rf.perfAUC.c=vector()


## Creating a pointsObject
#hostPest = unique(pest[,-c(1:3)], MARGIN=1)
hostPest = pest[,-c(1:3)]
parPest = names(which(summary(pest1[,'P_SP'], maxsum=500)>20));
parPest=parPest[-max(length(parPest))]


for(i in 1:length(parPest)){
#create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)


#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:4)]
  hostsOfi=dat[,1]

#Impute the data
  impDat=rfImpute(dat[,-1], presence1)
  dat=impDat[,-1]


###### If you want to include parasite richness (or composition) ######
 psr = rowSums(intMat)
#PCA of parasite communities...otherwise you're including like 500 new variables
 pcaHosts = princomp(intMat[,-which(colnames(intMat)==parPest[i])])$scores

psrOut=vector()
pcaHostsOut=matrix(0, nrow=length(hostsOfi), ncol=10)
for(e in 1:length(hostsOfi)){
  psrOut[e] = psr[which(hostsOfi[e] == rownames(intMat))]
  pcaHostsOut[e,] = pcaHosts[which(hostsOfi[e] == rownames(intMat)), 1:10]
}

dat = data.frame(dat, psrOut, pcaHostsOut)

flag=0
```

```
while(flag == 0){
 # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
 inds=sample(1:nrow(dat), 0.7*nrow(dat))
 if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

 #Set up a prelim train set and a test set
 train = dat[inds,]
 test = dat[-inds,]
 if(all(unique(train$GEO) %in% unique(test$GEO))){flag=1}
}

#Presences
 prezTR=presence1[inds]
 prezTE=presence1[-inds]


 #baseline expectations and null models
 baseline.auc.c[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), presence

##trained models
 #boosted regression trees
   brtModel.c[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution='berr
   brt.best.iter.c[[i]] = gbm.perf(brtModel.c[[i]], method="OOB")
   brt.preds.c[[i]] = prediction(predict(brtModel.c[[i]], newdata=test, n.trees=brt.best.iter.c[[i]]),
   brt.perf.c[[i]] = performance(brt.preds.c[[i]],"tpr","fpr")
   brt.perfAUC.c[i]=unlist(performance(brt.preds.c[[i]], 'auc')@y.values)

 #support vector machines
   svmModel.c[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
   svm.preds.c[[i]] = prediction(predict(svmModel.c[[i]], test), prezTE)
   svm.perf.c[[i]] = performance(svm.preds.c[[i]],"tpr","fpr")
   svm.perfAUC.c[i] = unlist(performance(svm.preds.c[[i]], 'auc')@y.values)

 #logistic regression
   lrModel.c[[i]] = glm(prezTR ~ ., data=train, family=binomial)
   lr.preds.c[[i]] = prediction(predict(lrModel.c[[i]], test), prezTE)
   lr.perf.c[[i]] = performance(lr.preds.c[[i]],"tpr","fpr")
   lr.perfAUC.c[i] = unlist(performance(lr.preds.c[[i]], 'auc')@y.values)

 #random forest
   rfModel.c[[i]] = randomForest(prezTR ~ ., data=train)
   rf.preds.c[[i]] = prediction(predict(rfModel.c[[i]], test), prezTE)
   rf.perf.c[[i]] = performance(rf.preds.c[[i]],"tpr","fpr")
   rf.perfAUC.c[i] = unlist(performance(rf.preds.c[[i]], 'auc')@y.values)
 print(i)
 }

ret.c=cbind(baseline.auc.c, brt.perfAUC.c, svm.perfAUC.c, lr.perfAUC.c, rf.perfAUC.c)
colnames(ret.c)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

```
testRet=apply(as.matrix(ret), 2, as.numeric)
testRetb=apply(as.matrix(ret.b),2, as.numeric)
```

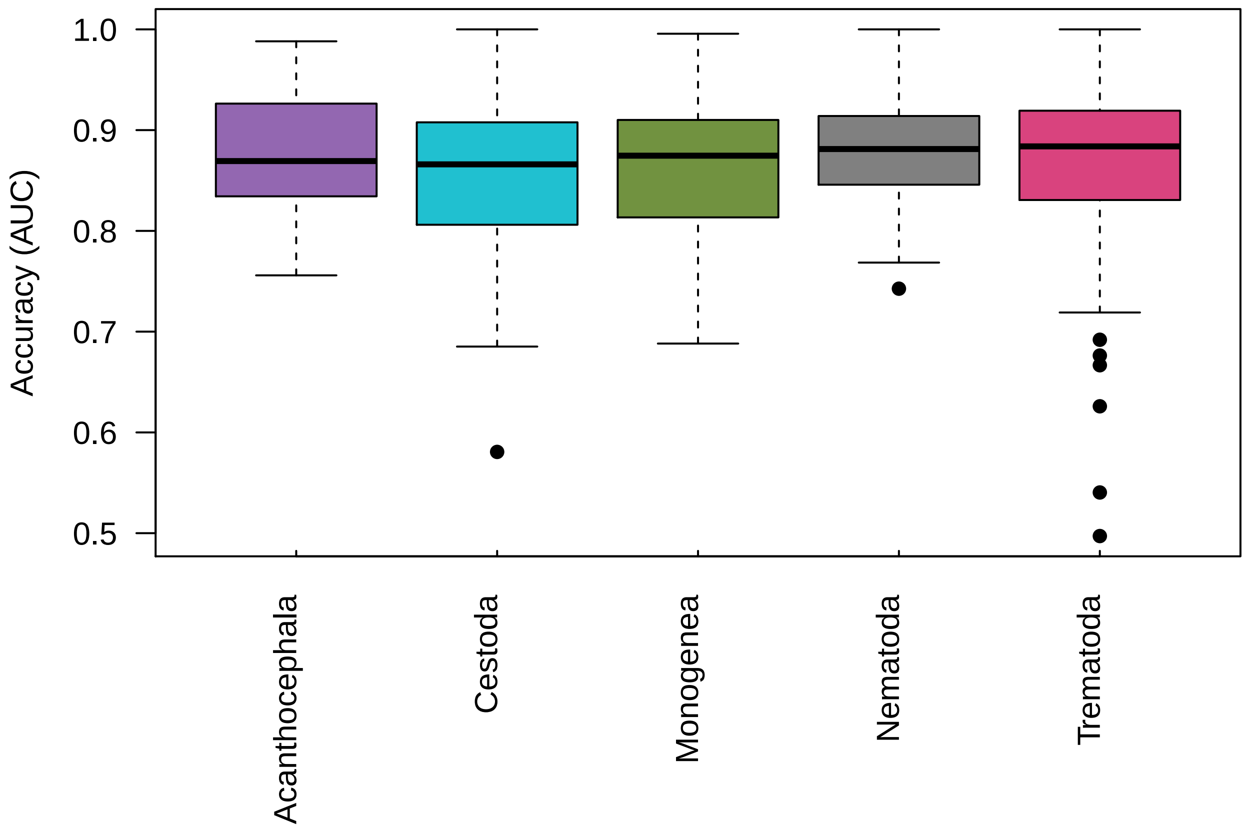
```r
testRetc=apply(as.matrix(ret.c),2, as.numeric)

meanAUC = colMeans(testRet)
meanSE = apply(testRet, 2, sd) / sqrt(nrow(testRet))
meanSD = apply(testRet, 2, sd)

meanAUC.b = colMeans(testRetb)
meanSE.b = apply(testRetb, 2, sd) / sqrt(nrow(testRetb))
meanSD.b = apply(testRetb, 2, sd)

meanAUC.c = colMeans(testRetc)
meanSE.c = apply(testRetc, 2, sd) / sqrt(nrow(testRetc))
meanSD.c = apply(testRetc, 2, sd)


layout(matrix(c(1,2,3),ncol=3))
plot(1:5, meanAUC, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Predictive
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC+meanSD, y1=meanAUC-meanSD, col=grey(0.8),lwd=2)
points(1:5, meanAUC, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)

plot(1:5, meanAUC.b, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Predicti
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC.b+meanSD.b, y1=meanAUC.b-meanSD.b, col=grey(0.8),lwd=2)
points(1:5, meanAUC.b, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)

plot(1:5, meanAUC.c, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Predicti
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC.c+meanSD.c, y1=meanAUC.c-meanSD.c, col=grey(0.8),lwd=2)
points(1:5, meanAUC.c, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)
```



Training using both parasite and host data

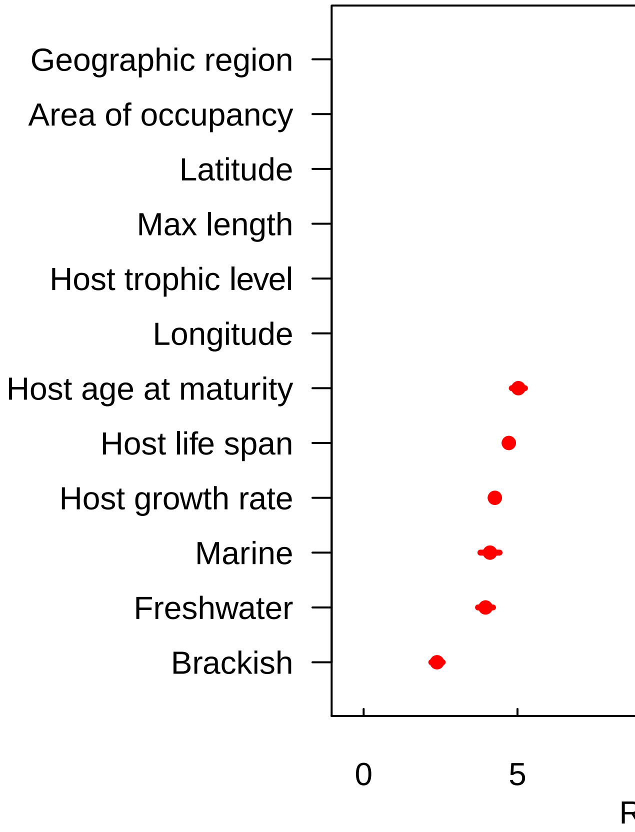
Does predictive power differ systematically among parasites?

```
par(mar=c(8,4,0.25,0.25))
hostCol=c('#9367B1', '#20C0D0','#719240', grey(0.5), '#D9437E')
plot(as.factor(parType[1:239,1]), testRet[1:239,2], col=hostCol, pch=16, las=1, ylab='Accuracy (AUC)', 
parGroups=c('Acanthocephala', 'Cestoda', 'Monogenea', 'Nematoda', 'Trematoda')
axis(1, 1:5, parGroups, las=2, tck=0.01, padj=0)
```



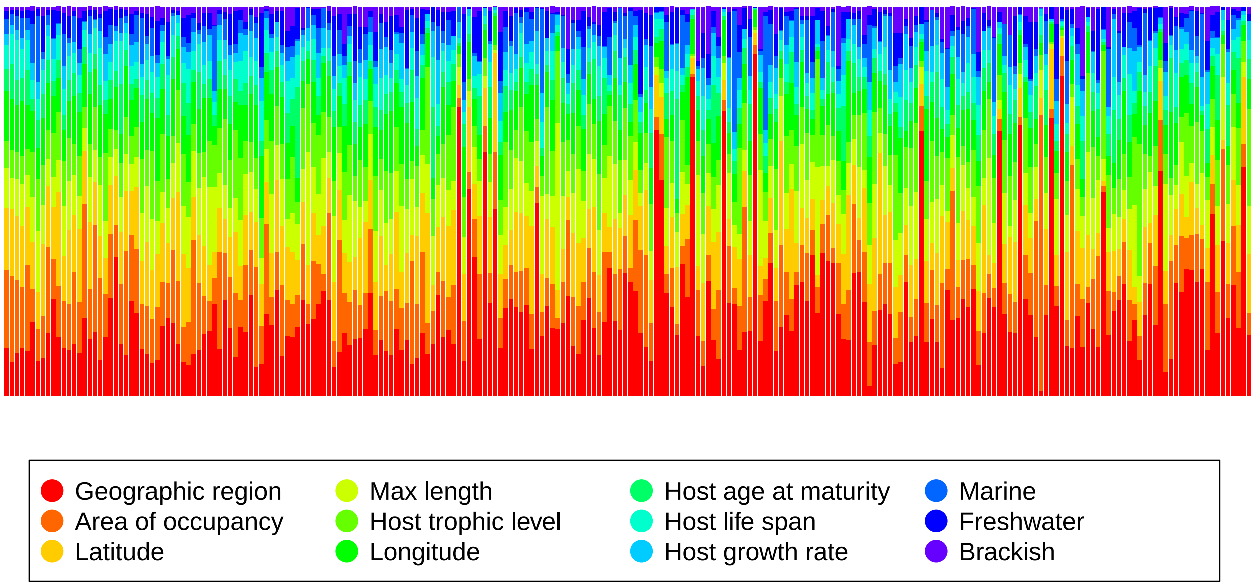
Are parasite distributions shaped by different factors?

```
varNames=c('Geographic region', 'Area of occupancy', 'Latitude', 'Max length', 'Host trophic level', 'Lo
#rownames(brtResults[rev(inds),])
brtResults=brtResults[,1:240]
brtMean = rowMeans(brtResults)
brtSE = apply(brtResults, 1, sd) / sqrt(ncol(brtResults))
brtSD = apply(brtResults, 1, sd)

inds.a=order(brtMean)
cols=rev(c(1,1,1,2,2,1,2,2,2,2,2,2))
par(mar=c(3,9,1,1))
plot(brtMean[inds.a], 1:12, pch=16, xlim=c(0,26), ylim=c(0.5,12.5), las=1, xlab='', ylab='', yaxt='n', c
segments(x0=brtMean[inds.a] - brtSE[inds.a], x1=brtMean[inds.a] + brtSE[inds.a], y0=1:12, col=cols,lwd=3
mtext("Relative contribution", side=1, line=2)
axis(2, at=1:12, labels=rev(varNames), las=1)
```



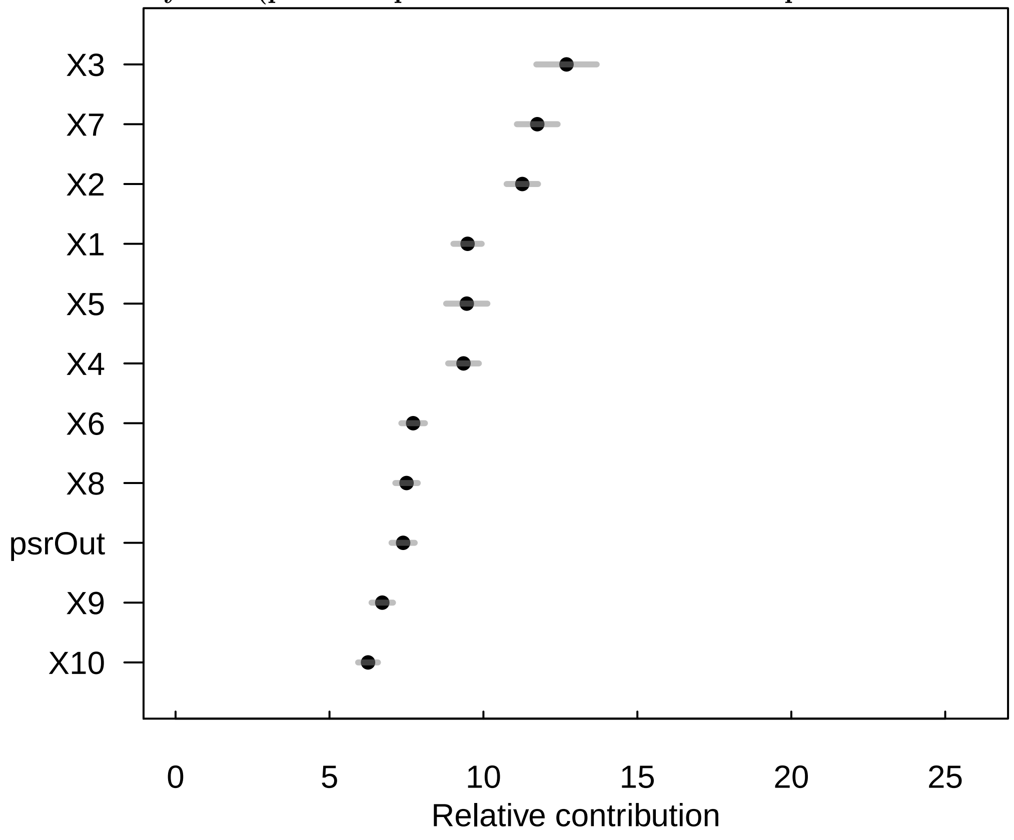
BRT model using only host traits (and geographic variables)

```r
layout(matrix(c(1,2),ncol=1), heights=c(1.5, 1))
par(mar=c(0.25,0.25,0.25,0.25))
barplot(brtResults[rev(inds.a),], col=rainbow(15), xaxt='n', axes=FALSE, border = NA)
plot.new()
legend(0.02,0.85,varNames , pch=16, col=rainbow(15), ncol=4, pt.cex=2, text.width=0.2)
```



```
brtMean.b = rowMeans(brtResults.b)
brtSE.b = apply(brtResults.b, 1, sd) / sqrt(ncol(brtResults.b))
brtSD.b = apply(brtResults.b, 1, sd)

inds=order(brtMean.b)
par(mar=c(3,9,1,1))
plot(brtMean.b[inds], 1:11, pch=16, xlim=c(0,26), ylim=c(0.5,11.5), las=1, xlab='', ylab='', yaxt='n', 
segments(x0=brtMean.b[inds] - brtSE.b[inds], x1=brtMean.b[inds] + brtSE.b[inds], y0=1:12, col=grey(0.5,
mtext("Relative contribution", side=1, line=2)
axis(2, at=1:11, labels=names(brtMean.b[inds]), las=1)
```

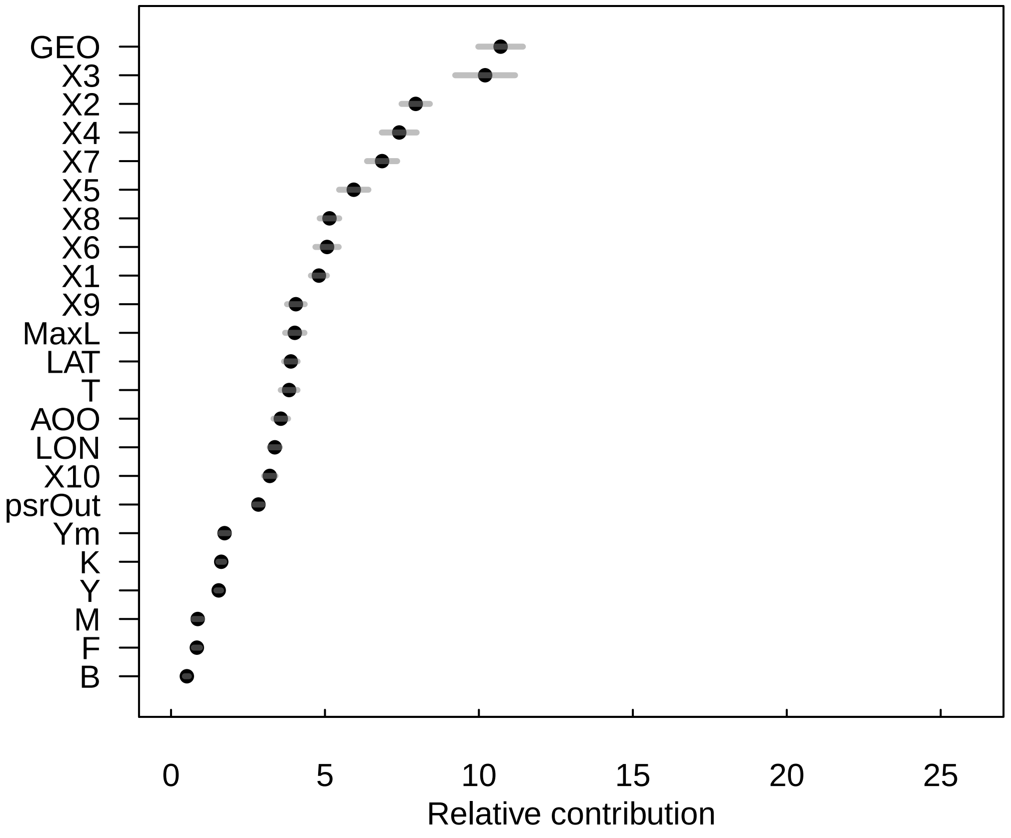
BRT models using only parasite community data (parasite species richness and a PCA of parasite community composition)



BRT models using data on parasite community composition and host traits Host traits still not all that important, though biogeographic region is really important.

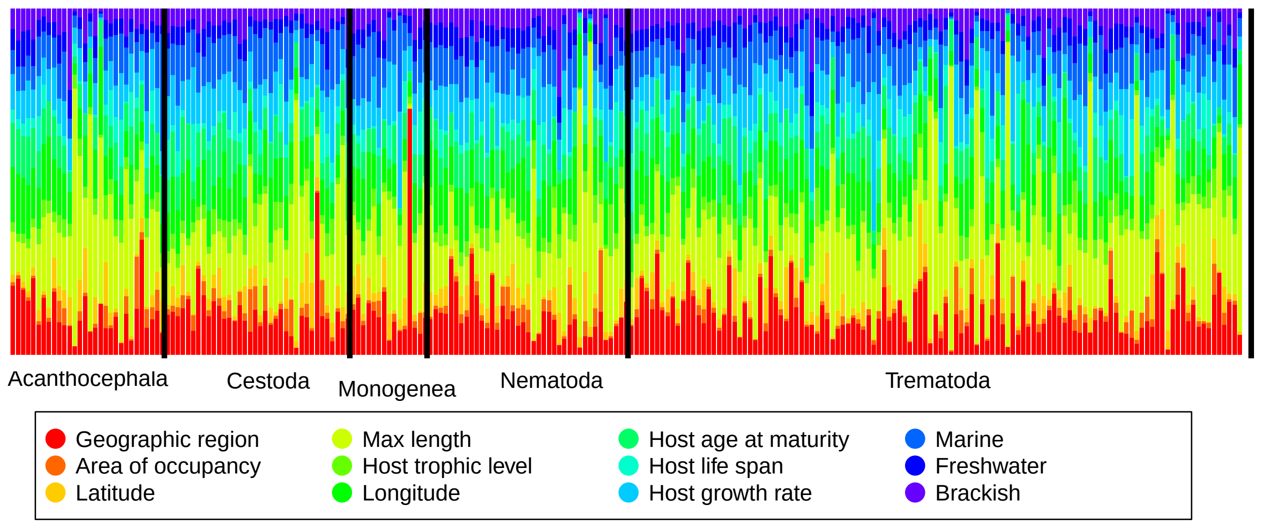
```
brtMean.c = rowMeans(brtResults.c)
brtSE.c = apply(brtResults.c, 1, sd) / sqrt(ncol(brtResults.c))
brtSD.c = apply(brtResults.c, 1, sd)

inds=order(brtMean.c)
par(mar=c(3,9,1,1))
plot(brtMean.c[inds], 1:23, pch=16, xlim=c(0,26), ylim=c(0.5,23.5), las=1, xlab='', ylab='', yaxt='n', 
segments(x0=brtMean.c[inds] - brtSE.c[inds], x1=brtMean.c[inds] + brtSE.c[inds], y0=1:23, col=grey(0.5,
```

```r
mtext("Relative contribution", side=1, line=2)
axis(2, at=1:23, labels=names(brtMean.c[inds]), las=1)
```



Is this variation attributable to parasite type?

```r
layout(matrix(c(1,2),ncol=1), heights=c(1.5, 1))
par(mar=c(0.25,0.25,0.25,0.25))

brtResults.ParType=brtResults[,order(parType[,1])]

barplot(brtResults.ParType, col=rainbow(15), xaxt='n', axes=FALSE, border = NA)
abline(v=c(30,66,81,120,241)*1.199, lwd=4)
plot.new()
text(15/239, 1 , parGroups[1],srt=0)
text(50/239, 1, parGroups[2], srt=0)
text(75/239, 0.95, parGroups[3], srt=0)
text(105/239, 1, parGroups[4], srt=0)
text(180/239, 1, parGroups[5], srt=0)


legend(0.02,0.85,varNames , pch=16, col=rainbow(15), ncol=4, pt.cex=2, text.width=0.2)
```



What if I only train on data from one specific region?

Let's check out the Palearctic.

```
# storage for model outputs
baseline.auc.pal=vector();
brtModel.pal=list(); brt.best.iter.pal=list(); brt.preds.pal=list(); brt.perf.pal=list(); brt.perfAUC.pa
svmModel.pal=list(); svm.preds.pal=list(); svm.perf.pal=list(); svm.perfAUC.pal=vector()
lrModel.pal=list(); lr.preds.pal=list(); lr.perf.pal=list(); lr.perfAUC.pal=vector()
rfModel.pal=list(); rf.preds.pal=list(); rf.perf.pal=list(); rf.perfAUC.pal=vector()


## Creating a pointsObject
pest1 = pest[which(pest$GEO == "PAL"),]
hostPest = pest1[,-c(1:3)]

parPest = names(which(summary(pest1[,'P_SP'], maxsum=500)>20));
parPest = parPest[-max(length(parPest))]


for(i in 1:length(parPest)){
#create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)

#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:6)]
```

```r
#Impute the data
  impDat=rfImpute(dat, presence1)
  dat=impDat[,-1]

flag=0
 while(flag == 0){
  # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
  inds=sample(1:nrow(dat), 0.7*nrow(dat))
  if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

  #Set up a prelim train set and a test set
  train = dat[inds,]
  test = dat[-inds,]
  if(all(unique(train$GEO) %in% unique(test$GEO))){flag=1}
 }

 #Presences
  prezTR=presence1[inds]
  prezTE=presence1[-inds]


  #baseline expectations and null models
  baseline.auc.pal[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), presen

 ##trained models
  #boosted regression trees
    brtModel.pal[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution='b
    brt.best.iter.pal[[i]] = gbm.perf(brtModel.pal[[i]], method="OOB")
    brt.preds.pal[[i]] = prediction(predict(brtModel.pal[[i]], newdata=test, n.trees=brt.best.iter.pal[
    brt.perf.pal[[i]] = performance(brt.preds.pal[[i]],"tpr","fpr")
    brt.perfAUC.pal[i]=unlist(performance(brt.preds.pal[[i]], 'auc')@y.values)

  #support vector machines
    svmModel.pal[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
    svm.preds.pal[[i]] = prediction(predict(svmModel.pal[[i]], test), prezTE)
    svm.perf.pal[[i]] = performance(svm.preds.pal[[i]],"tpr","fpr")
    svm.perfAUC.pal[i] = unlist(performance(svm.preds.pal[[i]], 'auc')@y.values)

  #logistic regression
    lrModel.pal[[i]] = glm(prezTR ~ ., data=train, family=binomial)
    lr.preds.pal[[i]] = prediction(predict(lrModel.pal[[i]], test), prezTE)
    lr.perf.pal[[i]] = performance(lr.preds.pal[[i]],"tpr","fpr")
    lr.perfAUC.pal[i] = unlist(performance(lr.preds.pal[[i]], 'auc')@y.values)

  #random forest
    rfModel.pal[[i]] = randomForest(prezTR ~ ., data=train)
    rf.preds.pal[[i]] = prediction(predict(rfModel.pal[[i]], test), prezTE)
    rf.perf.pal[[i]] = performance(rf.preds.pal[[i]],"tpr","fpr")
    rf.perfAUC.pal[i] = unlist(performance(rf.preds.pal[[i]], 'auc')@y.values)
  print(i)
  }

 ret.pal=cbind(baseline.auc.pal, brt.perfAUC.pal, svm.perfAUC.pal, lr.perfAUC.pal, rf.perfAUC.pal)
```

```r
colnames(ret.pal)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

```r
# storage for model outputs
baseline.auc.pal.b=vector();
brtModel.pal.b=list(); brt.best.iter.pal.b=list(); brt.preds.pal.b=list(); brt.perf.pal.b=list(); brt.pe
svmModel.pal.b=list(); svm.preds.pal.b=list(); svm.perf.pal.b=list(); svm.perfAUC.pal.b=vector()
lrModel.pal.b=list(); lr.preds.pal.b=list(); lr.perf.pal.b=list(); lr.perfAUC.pal.b=vector()
rfModel.pal.b=list(); rf.preds.pal.b=list(); rf.perf.pal.b=list(); rf.perfAUC.pal.b=vector()


## Creating a pointsObject
pest1 = pest[which(pest$GEO == "PAL"),]
hostPest = pest1[,-c(1:3)]

parPest = names(which(summary(pest1[,'P_SP'], maxsum=500)>20));
parPest = parPest[-max(length(parPest))]


for(i in 1:length(parPest)){
 #create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)


#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:4)]
  hostsOfi=dat[,1]


###### If you want to include parasite richness (or composition) ######
 psr = rowSums(intMat)
#PCA of parasite communities...otherwise you're including like 500 new variables
pcaHosts = princomp(intMat[,-which(colnames(intMat)==parPest[i])])$scores

psrOut=vector()
pcaHostsOut=matrix(0, nrow=length(hostsOfi), ncol=10)
for(e in 1:length(hostsOfi)){
  psrOut[e] = psr[which(hostsOfi[e] == rownames(intMat))]
  pcaHostsOut[e,] = pcaHosts[which(hostsOfi[e] == rownames(intMat)), 1:10]
}

dat = data.frame(psrOut, pcaHostsOut)
```

```
  # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
  inds=sample(1:nrow(dat), 0.7*nrow(dat))
  if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

  #Set up a prelim train set and a test set
  train = dat[inds,]
  test = dat[-inds,]


 #Presences
  prezTR=presence1[inds]
  prezTE=presence1[-inds]

  #baseline expectations and null models
  baseline.auc.pal.b[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), prese

 ##trained models
  #boosted regression trees
    brtModel.pal.b[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution=
    brt.best.iter.pal.b[[i]] = gbm.perf(brtModel.pal.b[[i]], method="OOB")
    brt.preds.pal.b[[i]] = prediction(predict(brtModel.pal.b[[i]], newdata=test, n.trees=brt.best.iter.
    brt.perf.pal.b[[i]] = performance(brt.preds.pal.b[[i]],"tpr","fpr")
    brt.perfAUC.pal.b[i]=unlist(performance(brt.preds.pal.b[[i]], 'auc')@y.values)

  #support vector machines
    svmModel.pal.b[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
    svm.preds.pal.b[[i]] = prediction(predict(svmModel.pal.b[[i]], test), prezTE)
    svm.perf.pal.b[[i]] = performance(svm.preds.pal.b[[i]],"tpr","fpr")
    svm.perfAUC.pal.b[i] = unlist(performance(svm.preds.pal.b[[i]], 'auc')@y.values)

  #logistic regression
    lrModel.pal.b[[i]] = glm(prezTR ~ ., data=train, family=binomial)
    lr.preds.pal.b[[i]] = prediction(predict(lrModel.pal.b[[i]], test), prezTE)
    lr.perf.pal.b[[i]] = performance(lr.preds.pal.b[[i]],"tpr","fpr")
    lr.perfAUC.pal.b[i] = unlist(performance(lr.preds.pal.b[[i]], 'auc')@y.values)

  #random forest
    rfModel.pal.b[[i]] = randomForest(prezTR ~ ., data=train)
    rf.preds.pal.b[[i]] = prediction(predict(rfModel.pal.b[[i]], test), prezTE)
    rf.perf.pal.b[[i]] = performance(rf.preds.pal.b[[i]],"tpr","fpr")
    rf.perfAUC.pal.b[i] = unlist(performance(rf.preds.pal.b[[i]], 'auc')@y.values)
  print(i)
  }

 ret.pal.b=cbind(baseline.auc.pal.b, brt.perfAUC.pal.b, svm.perfAUC.pal.b, lr.perfAUC.pal.b, rf.perfAUC
 colnames(ret)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

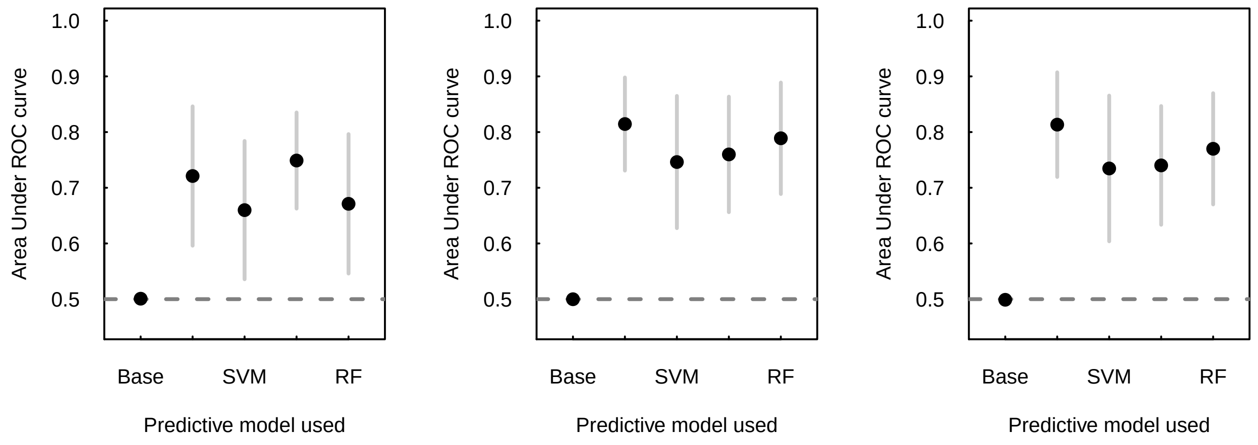
```
# storage for model outputs
baseline.auc.pal.c=vector();
brtModel.pal.c=list(); brt.best.iter.pal.c=list(); brt.preds.pal.c=list(); brt.perf.pal.c=list(); brt.p
svmModel.pal.c=list(); svm.preds.pal.c=list(); svm.perf.pal.c=list(); svm.perfAUC.pal.c=vector()
lrModel.pal.c=list(); lr.preds.pal.c=list(); lr.perf.pal.c=list(); lr.perfAUC.pal.c=vector()
rfModel.pal.c=list(); rf.preds.pal.c=list(); rf.perf.pal.c=list(); rf.perfAUC.pal.c=vector()
```

```r
## Creating a pointsObject
pest1 = pest[which(pest$GEO == "PAL"),]
hostPest = pest1[,-c(1:3)]

parPest = names(which(summary(pest1[,'P_SP'], maxsum=500)>20));
parPest = parPest[-max(length(parPest))]


for(i in 1:length(parPest)){
#create presence vector
  presence=rep(0, nrow(hostPest))
  presence[which(hostPest[,'P_SP'] == parPest[i])]=1

#make some 'na' into actual NAs
  ugh=which(hostPest=='na', arr.ind=TRUE)
  hostPest[ugh]=NA
  hostPest[,7:17]=apply(hostPest[,7:17],2, as.numeric)

#Only train on some of the absences, since they're totally not true absences
  cutDown=c(which(presence==1), sample(which(presence==0), 5*sum(presence)))
  presence1=presence[cutDown]
  dat=hostPest[cutDown, -c(1:4)]
  hostsOfi=dat[,1]

#Impute the data
  impDat=rfImpute(dat[,-1], presence1)
  dat=impDat[,-1]


###### If you want to include parasite richness (or composition) ######
 psr = rowSums(intMat)
#PCA of parasite communities...otherwise you're including like 500 new variables
 pcaHosts = princomp(intMat[,-which(colnames(intMat)==parPest[i])])$scores

psrOut=vector()
pcaHostsOut=matrix(0, nrow=length(hostsOfi), ncol=10)
for(e in 1:length(hostsOfi)){
  psrOut[e] = psr[which(hostsOfi[e] == rownames(intMat))]
  pcaHostsOut[e,] = pcaHosts[which(hostsOfi[e] == rownames(intMat)), 1:10]
}

dat = data.frame(dat[,-1], psrOut, pcaHostsOut)

  # This makes sure that the test set contains at least 4 hosts on which the parasite actually occurs
  inds=sample(1:nrow(dat), 0.7*nrow(dat))
  if(sum(presence1[inds]) < 4){inds[1:4] = which(presence1 == 1)[1:4]}

  #Set up a prelim train set and a test set
  train = dat[inds,]
  test = dat[-inds,]
```

```
#Presences
 prezTR=presence1[inds]
 prezTE=presence1[-inds]


 #baseline expectations and null models
 baseline.auc.pal.c[i] = performance(prediction(sample(presence[-inds], length(presence[-inds])), prese

##trained models
 #boosted regression trees
   brtModel.pal.c[[i]] = gbm(prezTR ~ ., data=train, n.trees=50000, interaction.depth=4, distribution=
   brt.best.iter.pal.c[[i]] = gbm.perf(brtModel.pal.c[[i]], method="OOB")
   brt.preds.pal.c[[i]] = prediction(predict(brtModel.pal.c[[i]], newdata=test, n.trees=brt.best.iter.
   brt.perf.pal.c[[i]] = performance(brt.preds.pal.c[[i]],"tpr","fpr")
   brt.perfAUC.pal.c[i]=unlist(performance(brt.preds.pal.c[[i]], 'auc')@y.values)

 #support vector machines
   svmModel.pal.c[[i]] = svm(prezTR ~ ., data=train, probability=TRUE)
   svm.preds.pal.c[[i]] = prediction(predict(svmModel.pal.c[[i]], test), prezTE)
   svm.perf.pal.c[[i]] = performance(svm.preds.pal.c[[i]],"tpr","fpr")
   svm.perfAUC.pal.c[i] = unlist(performance(svm.preds.pal.c[[i]], 'auc')@y.values)

 #logistic regression
   lrModel.pal.c[[i]] = glm(prezTR ~ ., data=train, family=binomial)
   lr.preds.pal.c[[i]] = prediction(predict(lrModel.pal.c[[i]], test), prezTE)
   lr.perf.pal.c[[i]] = performance(lr.preds.pal.c[[i]],"tpr","fpr")
   lr.perfAUC.pal.c[i] = unlist(performance(lr.preds.pal.c[[i]], 'auc')@y.values)

 #random forest
   rfModel.pal.c[[i]] = randomForest(prezTR ~ ., data=train)
   rf.preds.pal.c[[i]] = prediction(predict(rfModel.pal.c[[i]], test), prezTE)
   rf.perf.pal.c[[i]] = performance(rf.preds.pal.c[[i]],"tpr","fpr")
   rf.perfAUC.pal.c[i] = unlist(performance(rf.preds.pal.c[[i]], 'auc')@y.values)
 print(i)
 }

ret.pal.c=cbind(baseline.auc.pal.c, brt.perfAUC.pal.c, svm.perfAUC.pal.c, lr.perfAUC.pal.c, rf.perfAUC
colnames(ret)=c('BASE', 'BRT', 'SVM', 'LR', 'RF')
```

```
load('/media/drakelab/Lexar/8910Project/Analysis/pest(reduced)_pal_abc.RData')
testRet.pal = as.matrix(ret.pal)
testRet.pal = apply(testRet.pal, 2, as.numeric )
meanAUC.pal = colMeans(testRet.pal)
meanSE.pal = apply(testRet.pal, 2, sd) / sqrt(nrow(testRet.pal))
meanSD.pal = apply(testRet.pal, 2, sd)

testRet.pal.b = as.matrix(ret.pal.b)
testRet.pal.b = apply(testRet.pal.b, 2, as.numeric )
meanAUC.pal.b = colMeans(testRet.pal.b)
meanSE.pal.b = apply(testRet.pal.b, 2, sd) / sqrt(nrow(testRet.pal.b))
meanSD.pal.b = apply(testRet.pal.b, 2, sd)

testRet.pal.c = as.matrix(ret.pal.c)
```

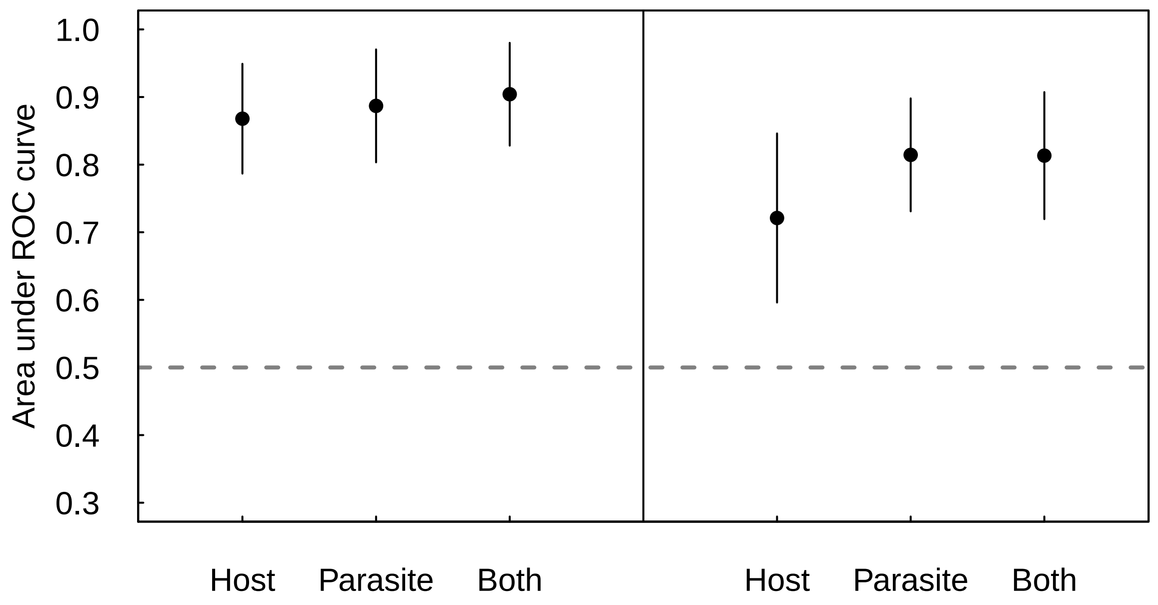
```r
testRet.pal.c = apply(testRet.pal.c, 2, as.numeric )
meanAUC.pal.c = colMeans(testRet.pal.c)
meanSE.pal.c = apply(testRet.pal.c, 2, sd) / sqrt(nrow(testRet.pal.c))
meanSD.pal.c = apply(testRet.pal.c, 2, sd)



layout(matrix(c(1,2,3), ncol=3))
plot(1:5, meanAUC.pal, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Predi
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC.pal+meanSD.pal, y1=meanAUC.pal-meanSD.pal, col=grey(0.8),lwd=2)
points(1:5, meanAUC.pal, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)


plot(1:5, meanAUC.pal.b, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Pre
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC.pal.b+meanSD.pal.b, y1=meanAUC.pal.b-meanSD.pal.b, col=grey(0.8),lwd=2)
points(1:5, meanAUC.pal.b, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)


plot(1:5, meanAUC.pal.c, ylim=c(0.45, 1), xlim=c(0.5,5.5), las=1, pch=16, tck=0.01, xaxt='n', xlab='Pre
abline(h=0.5, col=grey(0.5), lty=2, lwd=2)
segments(x0=1:5, y0=meanAUC.pal.c+meanSD.pal.c, y1=meanAUC.pal.c-meanSD.pal.c, col=grey(0.8),lwd=2)
points(1:5, meanAUC.pal.c, pch=16, cex=1.5)
axis(1, at=1:5, labels=c('Base', 'BRT', 'SVM', 'LR', 'RF'), tck=0.01, srt=30)
```



So my predictive power is reduced a fair bit, and many geographic variables still have high relative contribution values, suggesting that within a biogeographic region, spatial influences still help determine which fish host species are parasitized by a given parasite.

```r
brt.means=c(meanAUC[2], meanAUC.b[2], meanAUC.c[2], meanAUC.pal[2], meanAUC.pal.b[2], meanAUC.pal.c[2])
brt.sd = c(meanSD[2], meanSD.b[2], meanSD.c[2], meanSD.pal[2], meanSD.pal.b[2], meanSD.pal.c[2])

plot(c(1,2,3,5,6,7), brt.means, ylim=c(0.3,1), las=1, xlim=c(0.5,7.5), pch=16, tck=0.01, xlab='', ylab=
segments(x0=c(1,2,3,5,6,7), y0=brt.means+brt.sd, y1=brt.means-brt.sd)
abline(h=0.5, lty=2, lwd=2, col=grey(0.5))
abline(v=4)
```

```r
axis(1, at=c(1,2,3,5,6,7), rep(c('Host', 'Parasite', 'Both'),2), tck=0.01)
box()
#mtext('', side=1, line=2)
mtext('Area under ROC curve', side=2, line=2.5)
```



Results

Discussion

Tables

Table 1: Host traits examined and their corresponding units

Figures

References

Froese, Rainer, and Daniel Pauly. 2010. "FishBase." International Center for Living Aquatic Resources Management.

Strona, Giovanni, and Kevin D Lafferty. 2012. "FishPEST: an Innovative Software Suite for Fish Parasitologists." *Trends in Parasitology* 28 (4): 123.

Strona, Giovanni, Maria Lourdes D Palomares, Nicolas Bailly, Paolo Galli, and Kevin D Lafferty. 2013. "Host Range, Host Ecology, and Distribution of More Than 11 800 Fish Parasite Species: Ecological Archives E094-045." *Ecology* 94 (2): 544–544.