Lissachatina fulica

K. Hankins

05/28/2022

Front Matter

The following script was developed cooperatively by the SHSU SDM working group, including Laura Bianchi, Austin Brenek, Jesus Castillo, Nick Galle, Kayla Hankins, Kenneth Nobleza, Chris Randle, Nico Reger, Alyssa Russell, Ava Stendahl based on tutorials (https://rspatial.org/) provided by Robert Hijmans and Jane Elith. Chris Randle composed the following script from many scripts developed by the SHSU SDM working group.

This works best if your environment is empty at the start.

I have tried to set this up to eliminate required changes to the code. When you see text in **BOLD** below, that will be an indication that you need to make a decision.

Libraries

```
library(dismo)
library(sp)
library(raster)
library(stats)
library(dplyr)
library(knitr)
library(rgeos)
library(maptools)
library(rgdal)
library(ecospat)
library(usdm)
library(mgcv)
setwd("~/School/Thesis/Snail_Data")
```

Genus and Species strings

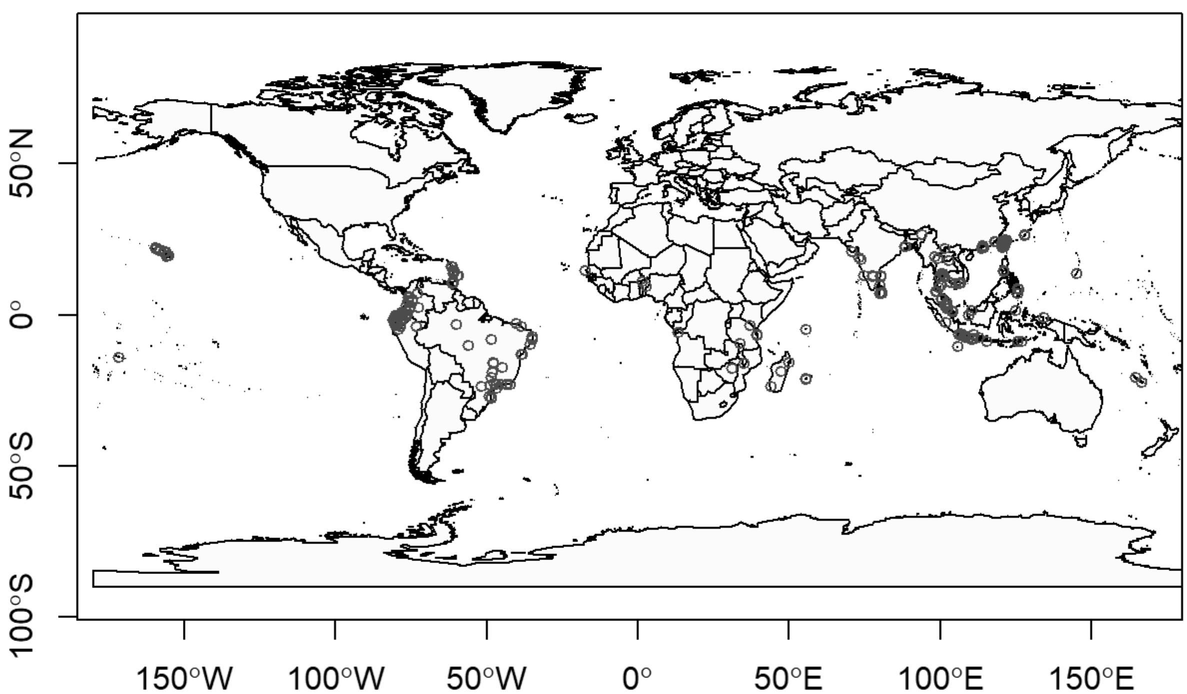
Their are many places in this code where you will need to save files with filenames including the genus and species. We'll save these as strings to automate the creation of file names. Enter your genus name and specific epithet in the quotes below.

```
genus<-"Lissachatina"
species<-"fulica"
```

Occurrence Data

Import occurrence data from csv file already generated (2020-2021), or using the script "Occurrence_Data.rmd" and visualize it.

```
sdmdata<-read.csv(file='C:/Users/kscih/OneDrive/Documents/School/Thesis/Snail_Data/Lissachatina_
fulica_qGIS_clean.csv')
##and visualize the data
#first lets get the extent of the data (the coordinates of the smallest box needed to encapsulat
e the data)  To do this I first need to convert sdmdata into a spatial points dataframe with the
same crs as "wrldsmpl", a giant spatial polygons data frame available from maptools
sdmdataframe<-data.frame(sdmdata)
data(wrld_simpl)
coordinates(sdmdataframe) <- ~lon+lat
crs(sdmdataframe) <- crs(wrld_simpl)
#And then extract the extent
e<-extent(sdmdataframe)
xmin<-xmin(e)
xmax<-xmax(e)
ymin<-ymin(e)
ymax<-ymax(e)
# and then plot a map and add the points from sdmdata
plot(wrld_simpl, xlim=c(xmin,xmax), ylim=c(ymin,ymax), axes=TRUE, col="light yellow")
box()
points(sdmdata$lon, sdmdata$lat, col='red', cex=0.75)
```



Let’s divide the data into training and testing data sets. The following code divides the data set into 80% training and 20% testing.

```
#Let's make sdmdata into a dataframe
data(wrld_simpl)
coordinates(sdmdata) <- ~lon+lat
crs(sdmdata) <- crs(wrld_simpl)

#Let's extract just the coordinates
presence <- coordinates(sdmdata)
#First we'll make a random list of integers from 1-5 as long as our presence data. Setting the s
eed results in a repeatable random process
set.seed(0)
#now make a list as long as the number of rows in presence consisting of a random series of inte
gers from 1-5
group <- kfold(presence, 5)
#Then we want to use this to retrieve the number of the rows in the presence data that are assoc
iated with the number 1 in our group index.
test_indices <- as.integer(row.names(presence[group == 1, ]))
#and create a new list of coordinates including only those rows that are NOT in test indices. Th
ese are all the row numbers NOT corresponding with the test_indices (which is ~80% of the data).
pres_train <- presence[-test_indices,]
#and those that do correspond with test indices (20%) of the data
pres_test <- presence[group ==1,]
```

Save pres_data and test_data as csv files just in case.

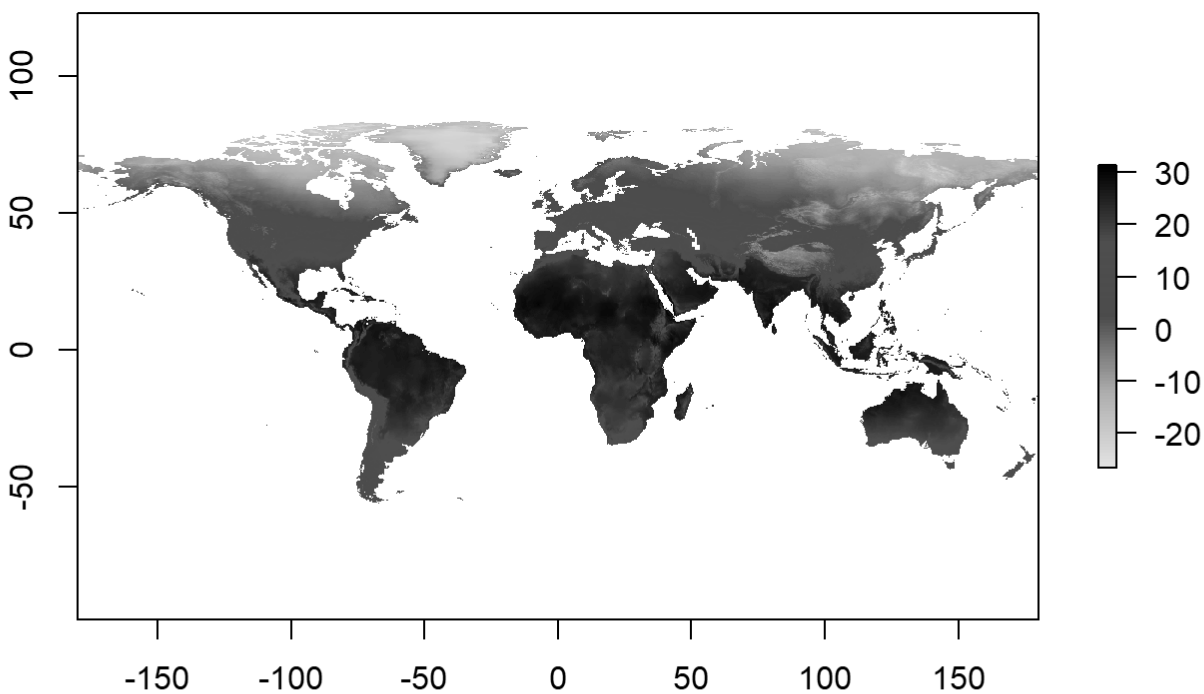
```
#first presdata_train
outdata<-data.frame(pres_train)
colnames(outdata)<-c("lon","lat")
write.csv(outdata, file=paste0(genus,"_",species,"_train.csv"), row.names=FALSE)

#and then presdata_test
outdata<-data.frame(pres_test)
colnames(outdata)<-c("lon","lat")
write.csv(outdata, file=paste0(genus,"_",species,"_test.csv"), row.names=FALSE)
```

Predictor data

Let's get the giant predictor file, name the bands, and generate our raster color schemes. This predictor set consists of all 35 Climond layers and elevation. Get it from Randle and keep it in your directory.

```
predictors<-stack('C:/Users/kscih/OneDrive/Documents/School/Thesis/Snail_Data/Climond_Elev_HI.tif')
bands<-c('Ann_Mean_Temp', 'Mean_Diurnal_Temp_Range', 'Isothermality', 'Temp_Seasonality', 'MaxTemp_WarmestWeek', 'MinTemp_ColdestWeek', 'Temp_Ann_Range', 'MeanTemp_WettestQ', 'MeanTemp_DriestQ', 'MeanTemp_WarmestQ', 'MeanTemp_ColdestQ', 'Ann_Precip', 'Precip_DriestWk', 'Precip_WettestWk', 'Precip_Seasonality', 'Precip_WettestQ', 'Precip_DriestQ', 'Precip_WarmestQ', 'Precip_ColdestQ', 'Ann_Mean_Rad', 'Highest_Weekly_Rad', 'Lowest_Weekly_Rad', 'Lowest_Weekly_Seasonality', 'Rad_WettestQ', 'Rad_DriestQ', 'Rad_WarmestQ', 'Rad_ColdestQ', 'Ann_Mean_Moisture', 'Highest_Weekly_Moisture', 'Lowest_Weekly_Moisture', 'Moisture_Seasonality', 'MeanMoisture_WettestQ', 'MeanMoisture_DriestQ', 'MeanMoisture_WarmestQ', 'MeanMoisture_ColdestQ', 'Elev', 'Human_Impact')
names(predictors)<-bands
cool<-colorRampPalette(c('gray','green','dark green',"blue"))
warm<-colorRampPalette(c('yellow', 'orange', 'red', 'brown', 'black'))
plot(predictors[["Ann_Mean_Temp"]], col=warm(100))
```



And

now we will use the VIFstep function to identify layers contributing most to collinearity (variance inflation factor). Rather than do this from a raster, I think it makes much more sense to do this from a dataframe in which we have sampled all the layers at the presence points only. This is because the larger a species distribution is, the lower the probability of collinearity across the range, even if layers are collinear where the species actually exists in the range.

```
#extract environmental data using the points in sdmdata
env_data<-extract(predictors,sdmdata)
#give names to the columns
colnames(env_data)<-bands
#run the vif
vif<-vifstep(env_data)
#and let's find the layers that were excluded and drop them
excluded<-vif@excluded
predictors<-dropLayer(predictors,excluded)
#and let's just go ahead and see which layers were dropped.
NClayers<-names(predictors)
NClayers
```

```
##  [1] "Mean_Diurnal_Temp_Range"   "Temp_Seasonality"
##  [3] "MaxTemp_WarmestWeek"       "Precip_DriestWk"
##  [5] "Precip_WettestWk"          "Precip_Seasonality"
##  [7] "Precip_ColdestQ"           "Lowest_Weekly_Seasonality"
##  [9] "Rad_WettestQ"              "Rad_WarmestQ"
## [11] "Rad_ColdestQ"              "MeanMoisture_WarmestQ"
## [13] "Elev"                      "Human_Impact"
```

General additive model

Data preparation

Generally speaking, we want to sample absence data from the region in which the presence data occur. There are two ways to do that, and one of them is better than the other. The first is to sample randomly. That may seem like a good idea, but its counter-intuitively not. The reason is that the presence data likely includes sampling bias. This will be inherent in p(hypothesis). If we include the same sampling bias in p(data), they cancel out in Bayes Theorem. The way that we'll do that is to create circles around our data and sample absence points from within those. The circles will have a diameter equal to the average distance between points.

```
#convert presence training data into a spatial points dataframe.
pres_train_SPDF<-SpatialPoints(pres_train)
crs(pres_train_SPDF) <- crs(wrld_simpl)
#Let's get the average distance between points (great circle distance--takes into account the cu
rvature of the earth). spDists creates a matrix of distances between points. This includes zero
s.
dist<-spDists(pres_train_SPDF,longlat = TRUE)
#replace the zeros with NA
dist[dist == 0]<-NA
#and calculate the mean--this is the average distance between points...the result will be in kil
ometers, but we need to convert it to meters so we multiply by 1000
dist<-750*mean(dist, na.rm=TRUE)
#now we are going to make circles using the average distance between points as the diameter.
x <- circles(pres_train_SPDF, d=dist, lonlat=TRUE)
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.26283573488746 80.361953283451641
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.54735671571942 80.784429711792555
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.74959199553766 81.639931613564087
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -162.05559660828493 79.124419633751856
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.40202346746517 81.292441879936433
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.58748675964421 80.753003353513435
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.87136872157416 80.094505347839714
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.25401984775203 80.347805370578087
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.92292131143057 80.162806404057221
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.98181266647279 81.035793430519689
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -162.901395737417 79.06758187250864
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.93797324076581 80.188172893128453
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.9712929702423 80.987703726512436
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -162.07887096759868 79.143938321603372
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.87561403761518 80.943119533377185
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -162.69761302821965 78.859748431524721
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.68150192902428 81.639758383008328
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.8920504405686 80.124332068858394
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.27443164654414 80.420849861656677
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.52529632700424 80.699379691751517
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.97695922086675 81.003720452866318
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.42324292707039 81.464586179401905
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -162.8816873913425 79.052313601336337
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.2647924276219 80.372977335426441
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.54994252061874 80.713453372275779
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.25978938452843 80.35657890970171
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.4212445364125 81.463697902525254
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.65563227875325 81.640456063835586
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.68200213707311 80.78455477445992
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.89137039326479 80.128086217953793
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.57347091282014 80.733085958417774
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.16010304327122 80.302487795675404
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -173.70791533382302 45.639579418644928
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.63896840829869 81.613462944005434
```

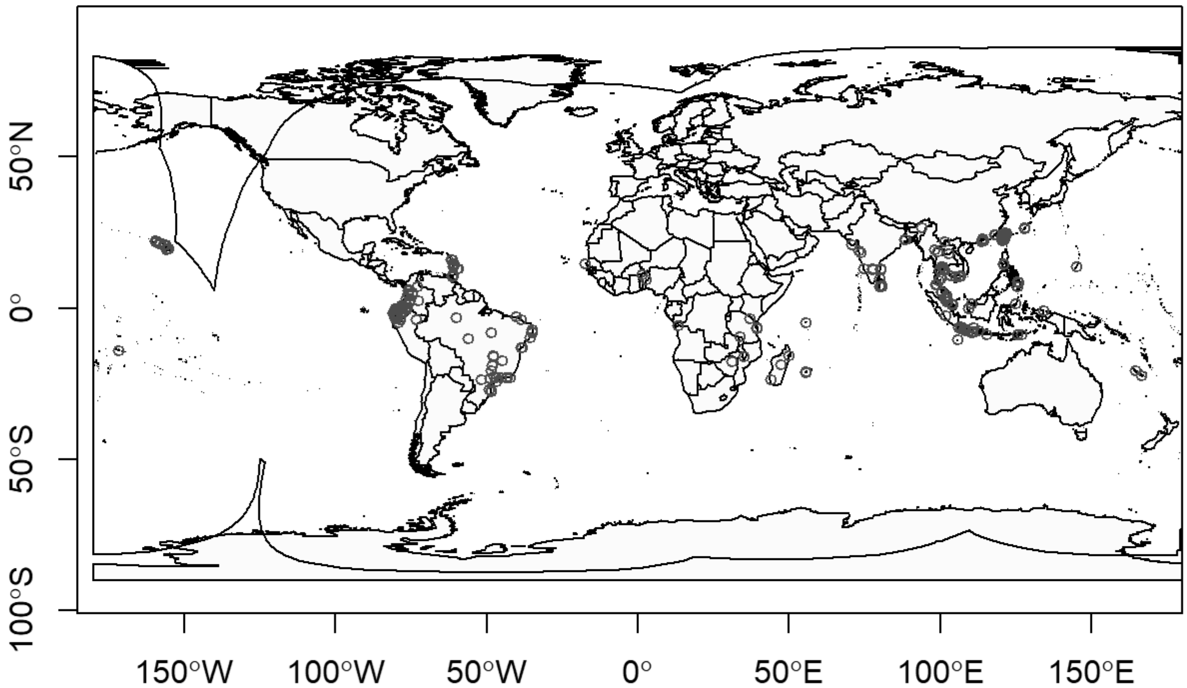
```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.25342791352463 80.342174239590477
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.27050457993334 80.413731049009712
```

```
## ci@polygons is invalid
```

```
## Warning in rgeos::gUnaryUnion(ci@polygons): Invalid objects found; consider
## using set_RGEOS_CheckValidity(2L)
```

```
#and convert those into polygons
pol <- polygons(x)
plot(wrld_simpl, xlim=c(xmin,xmax), ylim=c(ymin,ymax), axes=TRUE, col="light yellow")
box()
points(sdmdata$lon, sdmdata$lat, col='red', cex=0.75)
plot(pol, add=TRUE)
```




```
#and draw a number of samples from that approximately three times the number of presence points.
We'll chop that down at the end.
samp1 <- spsample(pol, nrow(pres_train)*10, type='random', iter=25)
```

```
## Warning in proj4string(obj): CRS object has comment, which is lost in output
```

```
#and get the cell numbers from the raster stack (right to left, up to down)
cells <- cellFromXY(predictors, samp1)
#and transform each of those to the center of its cell.
abs_train <- xyFromCell(predictors, cells)
#You'll get a warning saying that your CRS object has lost a comment. This is unimportant and ca
n be ignored.
```

And let's go ahead and extract the presence data, remove rows with NA values, and add a column of 1s.

```
pres_train_data<-extract(predictors,pres_train)
complete<-complete.cases(pres_train_data)
pres_train_data<-pres_train_data[complete,]
pres_train_data<-cbind(pres_train_data,1)
```

Now we want to extract predictors for the absence data, remove rows with NA values and chop it down to the size of our presence training data, and combine these into one data frame with column names (pa is the last column of 0,1 which indicates presence or absence)

```
abs_train_data<-extract(predictors,abs_train)
#remove rows with NA values
complete<-complete.cases(abs_train_data)
abs_train_data<-abs_train_data[complete,]
#and select a number of rows equal to the presence training data.
abs_train_data<-abs_train_data[1:nrow(pres_train_data),]
#and add a column of zeros to the end.
abs_train_data<-cbind(abs_train_data,0)
#put the two matrices together and name the colmns
train_data<-rbind(pres_train_data,abs_train_data)
colnames(train_data)<-c(names(predictors),"pa")
train_data<-as.data.frame(train_data)
```

Training the GAM and making predictions.

This is a pain in the neck because all of the layers have to be specified. I recommend printing the column names in the console `colnames(train_data)` and then copying them and formatting them

```
colnames(train_data)
```

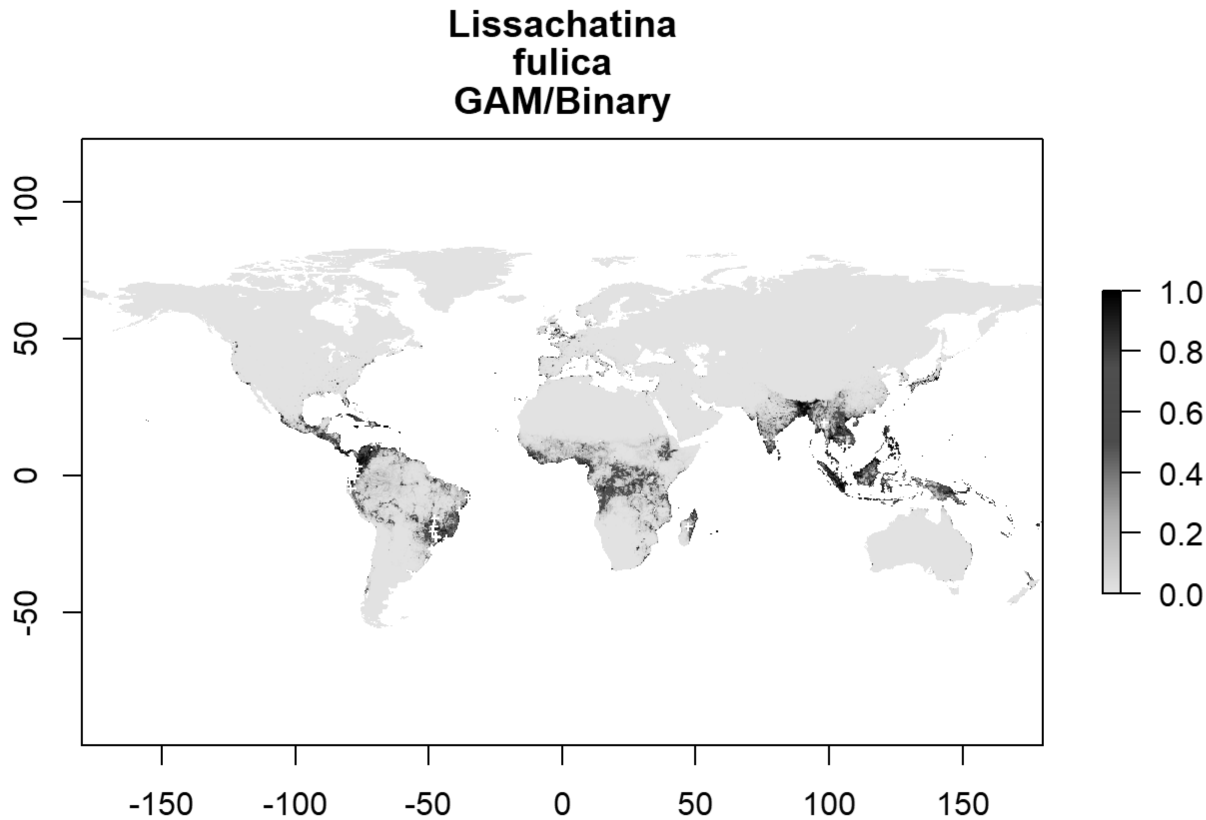
```
##  [1] "Mean_Diurnal_Temp_Range"   "Temp_Seasonality"
##  [3] "MaxTemp_WarmestWeek"       "Precip_DriestWk"
##  [5] "Precip_WettestWk"          "Precip_Seasonality"
##  [7] "Precip_ColdestQ"           "Lowest_Weekly_Seasonality"
##  [9] "Rad_WettestQ"              "Rad_WarmestQ"
## [11] "Rad_ColdestQ"              "MeanMoisture_WarmestQ"
## [13] "Elev"                      "Human_Impact"
## [15] "pa"
```

```
gam <- gam(pa ~ Mean_Diurnal_Temp_Range + Temp_Seasonality + MaxTemp_WarmestWeek + Precip_Driest
Wk + Precip_WettestWk + Precip_Seasonality + Precip_ColdestQ + Lowest_Weekly_Seasonality + Rad_W
ettestQ + Rad_WarmestQ + Rad_ColdestQ + MeanMoisture_WarmestQ + Elev + Human_Impact,
family = binomial(link = "logit"), data=train_data)
summary(gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## pa ~ Mean_Diurnal_Temp_Range + Temp_Seasonality + MaxTemp_WarmestWeek +
##     Precip_DriestWk + Precip_WettestWk + Precip_Seasonality +
##     Precip_ColdestQ + Lowest_Weekly_Seasonality + Rad_WettestQ +
##     Rad_WarmestQ + Rad_ColdestQ + MeanMoisture_WarmestQ + Elev +
##     Human_Impact
##
## Parametric coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -6.704e+00  5.052e+00  -1.327  0.18447
## Mean_Diurnal_Temp_Range   -2.937e-01  1.402e-01  -2.095  0.03619 *
## Temp_Seasonality          -1.365e+02  7.926e+01  -1.723  0.08497 .
## MaxTemp_WarmestWeek        8.299e-02  1.187e-01   0.699  0.48442
## Precip_DriestWk           -6.595e-03  1.301e-02  -0.507  0.61220
## Precip_WettestWk          -1.252e-02  3.094e-02  -0.405  0.68571
## Precip_Seasonality         6.915e-01  1.560e+00   0.443  0.65758
## Precip_ColdestQ            1.394e-03  1.444e-03   0.966  0.33418
## Lowest_Weekly_Seasonality -7.251e+00  6.321e+00  -1.147  0.25136
## Rad_WettestQ              -2.626e-02  1.687e-02  -1.556  0.11970
## Rad_WarmestQ               5.465e-03  1.820e-02   0.300  0.76398
## Rad_ColdestQ               1.918e-02  1.283e-02   1.495  0.13488
## MeanMoisture_WarmestQ      3.680e+00  1.206e+00   3.052  0.00228 **
## Elev                       3.898e-04  7.307e-04   0.533  0.59369
## Human_Impact               2.610e-01  3.489e-02   7.480 7.43e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =   0.83   Deviance explained = 80.4%
## UBRE = -0.67103  Scale est. = 1         n = 528
```

Let’s make some predictions and export them to a file

```
GAMpreds <- predict(predictors, gam, type = 'response')
writeRaster(GAMpreds, filename = paste0(genus,"_",species,"_GAM.tif"), overwrite=TRUE)
plot(GAMpreds, main=c(genus,species,'GAM/Binary'),col=warm(100), zlim=c(0,1))
points(pres_test, col='white', cex =.4, pch=3)
```



MaxEnt

We need many more background points for MaxEnt and BRT than we needed for GAM. Let's go ahead and generate those.

```
samp1 <- spsample(pol, 30000, type='random', iter=25)
```

```
## Warning in proj4string(obj): CRS object has comment, which is lost in output
```

```
#and get the cell numbers from the raster stack (right to left, up to down)
cells <- cellFromXY(predictors, samp1)
#and transform each of those to the center of its cell.
background_train <- xyFromCell(predictors, cells)
#You'll get a warning saying that your CRS object has lost a comment. This is unimportant and ca
n be ignored.

#If the background data has too many NA values, first get the predictor data associated with the
points
background_train_data<-extract(predictors,background_train)
#and remove all of the points that don't have data
complete<-complete.cases(background_train_data)
background_train<-background_train[complete,]
```

Let’s go ahead and set a locations for java **This will obviously be specialized for your computer. Try to find the ‘home’ folder in java and specify the path below**

```
#Sys.setenv(JAVA_HOME='')
```

First we let the program know to start up maxent using the command maxent. After that, all we need to do is to make a model oject (me_model), from the raster data and the presence training data.

```
library(rJava)
```

```
## Warning: package 'rJava' was built under R version 4.1.2
```

```
maxent()
```

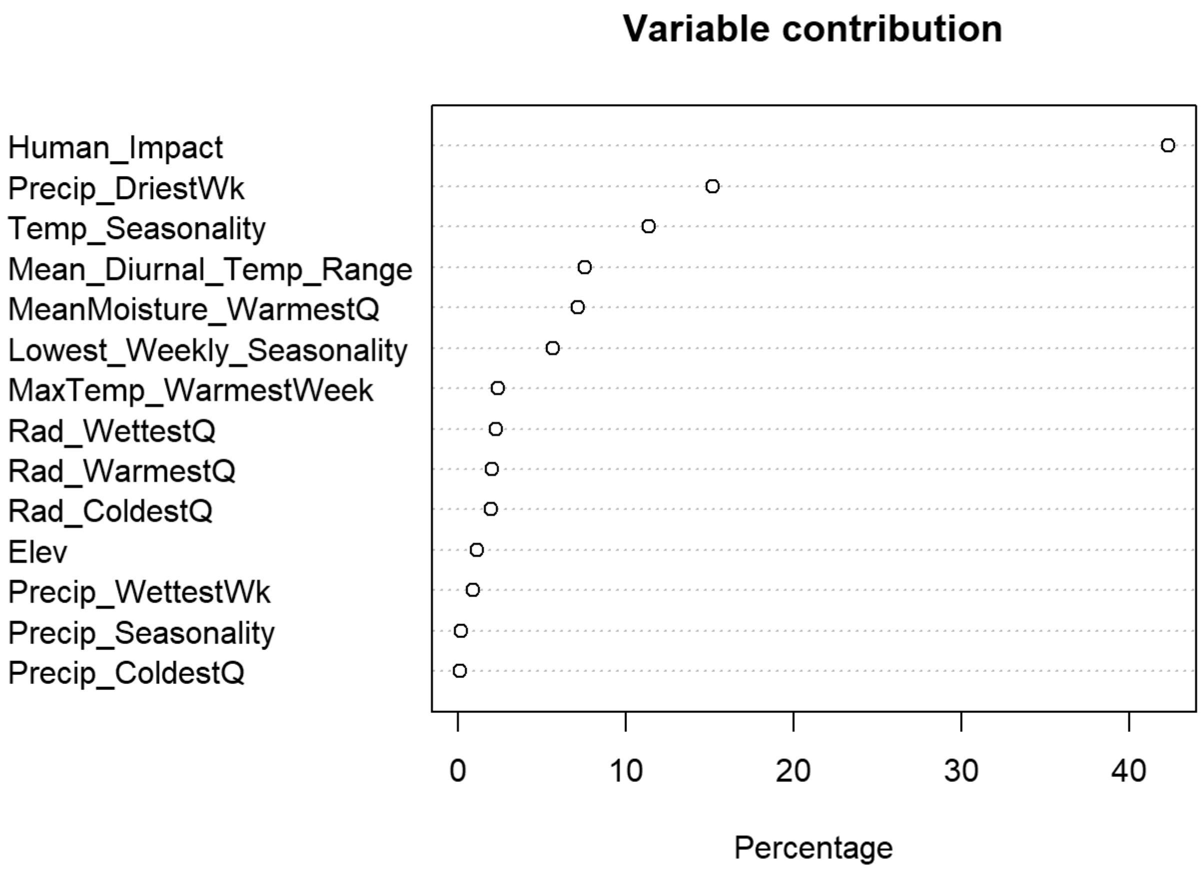
```
## This is MaxEnt version 3.4.1
```

```
me_model <- maxent(predictors, pres_train, a=background_train)
```

```
## Warning in .local(x, p, ...): 25 (12.32%) of the presence points have NA
## predictor values
```

```
## This is MaxEnt version 3.4.1
```

```
#and plot the models most important layers
par(mfrow=c(1,1))
plot(me_model)
```

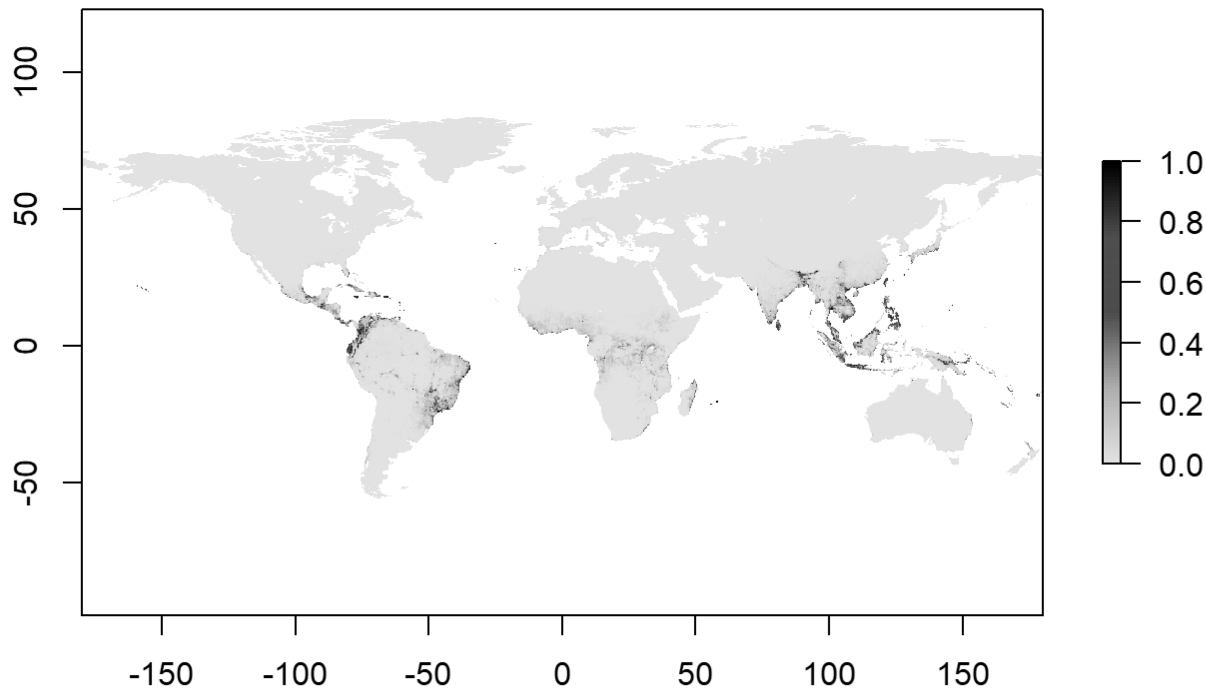


Let's go ahead and make some predictions

```
MEpreds<-predict(predictors, me_model, type='response')
```

```
## This is MaxEnt version 3.4.1
```

```
writeRaster(MEpreds, filename=paste0(genus,"_",species,"_ME.tif"), overwrite=TRUE)
#and plot
plot(MEpreds, col=warm(100), zlim=c(0,1))
```



Boosted regression trees

We need to prepare data for BRT in much the same way that we did for GAM, with the exception that we will need a lot more background data. We can use the 10,000 points that we already generated for ME

```
#let's get the data from our predictors
bg_train_data<-extract(predictors,background_train)
#and bind a column of 0 to the end of it
bg_train_data<-cbind(bg_train_data,0)
#and convert it to a data frame
bg_train_data<-as.data.frame(bg_train_data)
#and then combine it withe the presence training data
pres_train_data<-as.data.frame(pres_train_data)
BRT_data<-rbind(pres_train_data, bg_train_data)
colnames(BRT_data)<-c(names(predictors),"pa")
```

```
sdm.tc5.lr001 <- gbm.step(data=BRT_data, gbm.x = 1:nlayers(predictors), gbm.y = ncol(BRT_data),
 family = "bernoulli", tree.complexity = 5, learning.rate = 0.001, bag.fraction = 0.5)
```

```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for pa and using a family of bernoulli
## Using 7932 observations and 14 predictors
## creating 10 initial models of 50 trees
##
##  folds are stratified by prevalence
## total mean deviance =  0.2919
## tolerance is fixed at  3e-04
## ntrees resid. dev.
## 50    0.2584
## now adding trees...
## 100   0.2389
## 150   0.2251
## 200   0.214
## 250   0.2049
## 300   0.1972
## 350   0.1904
## 400   0.1843
## 450   0.1788
## 500   0.1739
## 550   0.1693
## 600   0.1652
## 650   0.1613
## 700   0.1577
## 750   0.1543
## 800   0.1512
## 850   0.1482
## 900   0.1454
## 950   0.1427
## 1000   0.1403
## 1050   0.138
## 1100   0.1358
## 1150   0.1337
## 1200   0.1317
## 1250   0.1299
## 1300   0.1281
## 1350   0.1264
## 1400   0.1248
## 1450   0.1232
## 1500   0.1218
## 1550   0.1203
## 1600   0.119
## 1650   0.1176
## 1700   0.1164
## 1750   0.1151
## 1800   0.1139
## 1850   0.1127
## 1900   0.1115
```

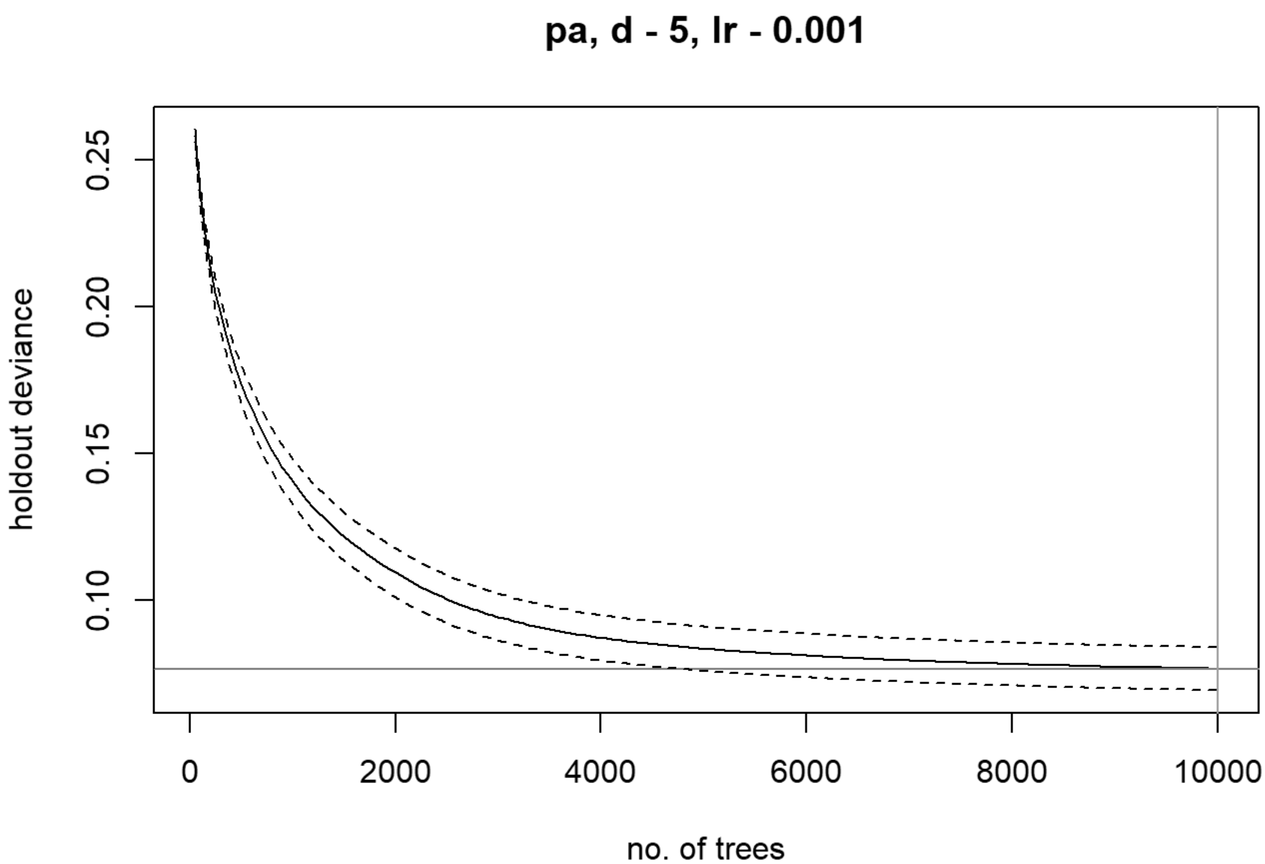
```
## 1950   0.1104
## 2000   0.1094
## 2050   0.1084
## 2100   0.1073
## 2150   0.1063
## 2200   0.1054
## 2250   0.1045
## 2300   0.1036
## 2350   0.1028
## 2400   0.102
## 2450   0.1012
## 2500   0.1004
## 2550   0.0997
## 2600   0.099
## 2650   0.0983
## 2700   0.0977
## 2750   0.097
## 2800   0.0964
## 2850   0.0959
## 2900   0.0953
## 2950   0.0948
## 3000   0.0943
## 3050   0.0938
## 3100   0.0933
## 3150   0.0929
## 3200   0.0924
## 3250   0.092
## 3300   0.0916
## 3350   0.0912
## 3400   0.0908
## 3450   0.0904
## 3500   0.0901
## 3550   0.0897
## 3600   0.0894
## 3650   0.0891
## 3700   0.0888
## 3750   0.0885
## 3800   0.0882
## 3850   0.0879
## 3900   0.0876
## 3950   0.0874
## 4000   0.0871
## 4050   0.0869
## 4100   0.0867
## 4150   0.0864
## 4200   0.0862
## 4250   0.086
## 4300   0.0858
## 4350   0.0856
## 4400   0.0854
## 4450   0.0852
## 4500   0.085
```

```
## 4550   0.0849
## 4600   0.0847
## 4650   0.0845
## 4700   0.0843
## 4750   0.0842
## 4800   0.084
## 4850   0.0839
## 4900   0.0837
## 4950   0.0836
## 5000   0.0834
## 5050   0.0833
## 5100   0.0831
## 5150   0.083
## 5200   0.0829
## 5250   0.0827
## 5300   0.0826
## 5350   0.0825
## 5400   0.0824
## 5450   0.0822
## 5500   0.0822
## 5550   0.082
## 5600   0.0819
## 5650   0.0818
## 5700   0.0817
## 5750   0.0816
## 5800   0.0815
## 5850   0.0814
## 5900   0.0813
## 5950   0.0812
## 6000   0.0811
## 6050   0.081
## 6100   0.0809
## 6150   0.0808
## 6200   0.0807
## 6250   0.0806
## 6300   0.0805
## 6350   0.0804
## 6400   0.0803
## 6450   0.0802
## 6500   0.0801
## 6550   0.0801
## 6600   0.08
## 6650   0.0799
## 6700   0.0798
## 6750   0.0797
## 6800   0.0797
## 6850   0.0796
## 6900   0.0795
## 6950   0.0795
## 7000   0.0794
## 7050   0.0793
## 7100   0.0793
```

```
## 7150   0.0792
## 7200   0.0791
## 7250   0.0791
## 7300   0.079
## 7350   0.0789
## 7400   0.0789
## 7450   0.0788
## 7500   0.0788
## 7550   0.0787
## 7600   0.0786
## 7650   0.0786
## 7700   0.0785
## 7750   0.0785
## 7800   0.0784
## 7850   0.0784
## 7900   0.0783
## 7950   0.0783
## 8000   0.0782
## 8050   0.0782
## 8100   0.0781
## 8150   0.0781
## 8200   0.078
## 8250   0.0779
## 8300   0.0779
## 8350   0.0778
## 8400   0.0778
## 8450   0.0777
## 8500   0.0777
## 8550   0.0777
## 8600   0.0776
## 8650   0.0776
## 8700   0.0775
## 8750   0.0774
## 8800   0.0774
## 8850   0.0774
## 8900   0.0773
## 8950   0.0773
## 9000   0.0772
## 9050   0.0772
## 9100   0.0772
## 9150   0.0772
## 9200   0.0771
## 9250   0.0771
## 9300   0.077
## 9350   0.077
## 9400   0.077
## 9450   0.077
## 9500   0.0769
## 9550   0.0769
## 9600   0.0769
## 9650   0.0768
## 9700   0.0768
```

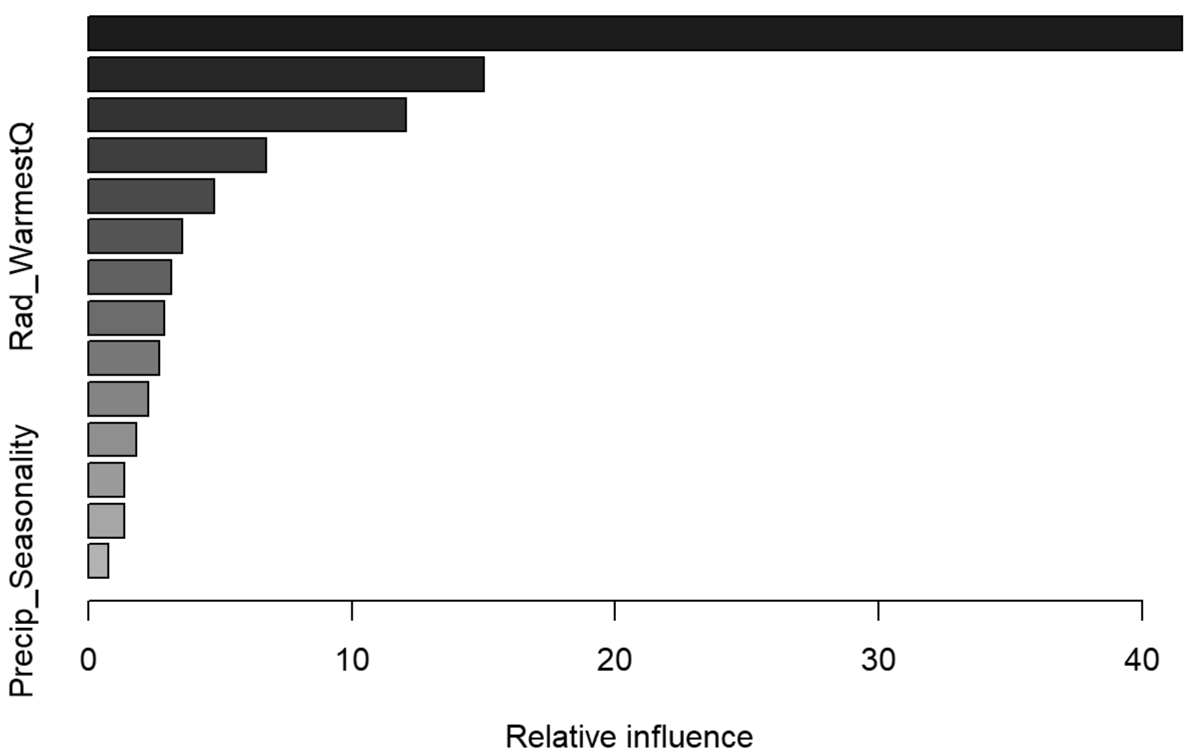
```
## 9750   0.0768
## 9800   0.0767
## 9850   0.0767
## 9900   0.0766
## 9950   0.0766
## 10000   0.0766
```

```
## fitting final gbm model with a fixed number of 10000 trees for pa
```



```
##
## mean total deviance = 0.292
## mean residual deviance = 0.035
##
## estimated cv deviance = 0.077 ; se = 0.007
##
## training data correlation = 0.935
## cv correlation =  0.828 ; se = 0.018
##
## training data AUC score = 0.998
## cv AUC score = 0.989 ; se = 0.002
##
## elapsed time -  0.21 minutes
##
##  ########### warning ##########
##
## maximum tree limit reached - results may not be optimal
##   - refit with faster learning rate or increase maximum number of trees
```

```
summary(sdm.tc5.lr001)
```



```
##                                                   var    rel.inf
## Human_Impact                             Human_Impact 41.5198689
## Temp_Seasonality                     Temp_Seasonality 15.0175530
## Lowest_Weekly_Seasonality Lowest_Weekly_Seasonality 12.0515600
## MeanMoisture_WarmestQ           MeanMoisture_WarmestQ  6.7444416
## Mean_Diurnal_Temp_Range       Mean_Diurnal_Temp_Range  4.7777670
## Rad_WarmestQ                             Rad_WarmestQ  3.5525244
## MaxTemp_WarmestWeek               MaxTemp_WarmestWeek  3.1569868
## Precip_DriestWk                       Precip_DriestWk  2.8755962
## Rad_ColdestQ                             Rad_ColdestQ  2.6851879
## Precip_WettestWk                     Precip_WettestWk  2.2760519
## Precip_ColdestQ                       Precip_ColdestQ  1.8403940
## Rad_WettestQ                             Rad_WettestQ  1.3748755
## Elev                                             Elev  1.3714597
## Precip_Seasonality                 Precip_Seasonality  0.7557332
```

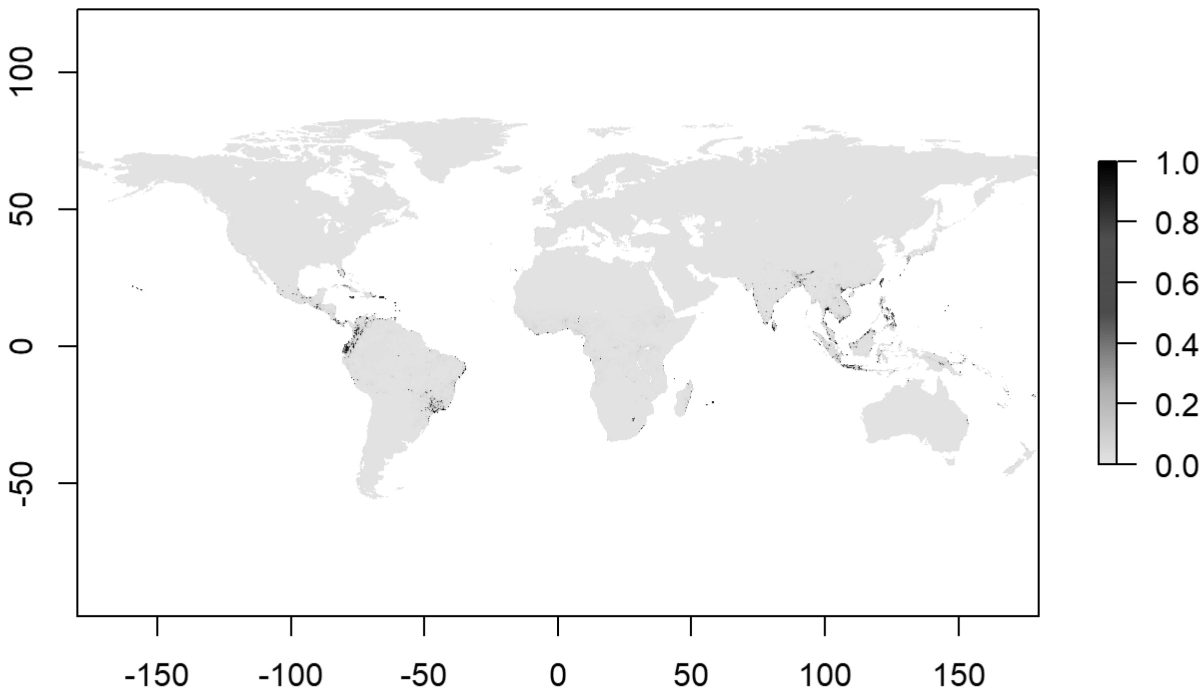
Note: you may want to try different combinations! If your trees are converging too slowly, raise the tree complexity by 1 or two, and back the learning rate down. On the other hand of your holdout deviance drops very quickly and slowly starts to rise, you are overfitting. Drop the tree complexity and raise the learning rate.

Let's make predictions and save them

```
BRTpreds<-predict(predictors, sdm.tc5.lr001, type='response')
```

```
## Using 10000 trees...
##
## Using 10000 trees...
##
## Using 10000 trees...
##
## Using 10000 trees...
```

```
writeRaster(BRTpreds, filename=paste0(genus,"_", species,"_BRT.tif"), overwrite=TRUE)
#and plot
plot(BRTpreds, col=warm(100), zlim=c(0,1))
```



Evaluation

We want to generate the following metrics for each of the three models: AUC, COR, maximum Kappa, TRS, and it wouldn't kill us to have a Boyce graph either.

#Absence Testing Data First we'll use the pres_test data to generate absence test data. This time we want about the same number of points for both. To do that, we'll generate 4x the number of absence points as presence points and chop it to size.

```
pres_test_SPDF<-SpatialPoints(pres_test)
data("wrld_simpl")
crs(pres_test_SPDF) <- crs(wrld_simpl)
#now we are going to make circles of about a degree (110000 meters at the equator). I'm working
 in a relatively small area, but if your data are widespread, you can increase this by changing
 d.
x <- circles(pres_test_SPDF, d=dist, lonlat=TRUE)
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.92465351602306 80.331841678003272
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.57490825658775 80.724666585853939
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -165.96250324059662 81.049982421934672
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.8755613926634 80.102265114318243
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.42239754643987 81.486124485795401
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.87507622326319 80.106061415665266
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -163.93386200461785 80.176185778586202
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -167.39704259290019 81.289009767167229
```

```
## Warning in RGEOSUnaryPredFunc(spgeom, byid, "rgeos_isvalid"): Self-intersection
## at or near point -164.26026450605025 80.35629174157549
```

```
## ci@polygons is invalid
```

```
## Warning in rgeos::gUnaryUnion(ci@polygons): Invalid objects found; consider
## using set_RGEOS_CheckValidity(2L)
```

```
#and convert those into polygons
pol <- polygons(x)
#and draw a number of samples from that...because
samp1 <- spsample(pol, 3*length(pres_test), type='random', iter=25)
```

```
## Warning in proj4string(obj): CRS object has comment, which is lost in output
```

```
#and get the cell numbers from the raster stack (right to left, up to down)
cells <- cellFromXY(predictors, samp1)
#and transform each of those to the center of its cell.
abs_test <- xyFromCell(predictors, cells)
#You'll get a warning saying that your CRS object has lost a comment. This is unimportant and ca
n be ignored.
```

GAM evaluation

```
p<-extract(GAMpreds,pres_test)
a<-extract(GAMpreds,abs_test)
#And let's get rid of nasty NA values and shrink a to the size of p
p<-p[!is.na(p)]
a<-a[!is.na(a)]
a<-a[1:length(p)]
#Let's look at the shape of these data
#lets weld all the data together
all_vals<-c(p,a)
e<-evaluate(p=p,a=a)
AUC_GAM<-e@auc
COR_GAM<-e@cor
pa<-c(replicate(length(p),1),replicate(length(a),0))
kappaGAM<-ecospat.max.kappa(all_vals,pa)
TSS_GAM<-ecospat.max.tss(all_vals,pa)
print(paste('Max kappa: ', kappaGAM[2] ))
```

```
## [1] "Max kappa:  0.924242424242424"
```

```
print(paste('TSS:', TSS_GAM[[2]]))
```

```
## [1] "TSS: 0.924242424242424"
```

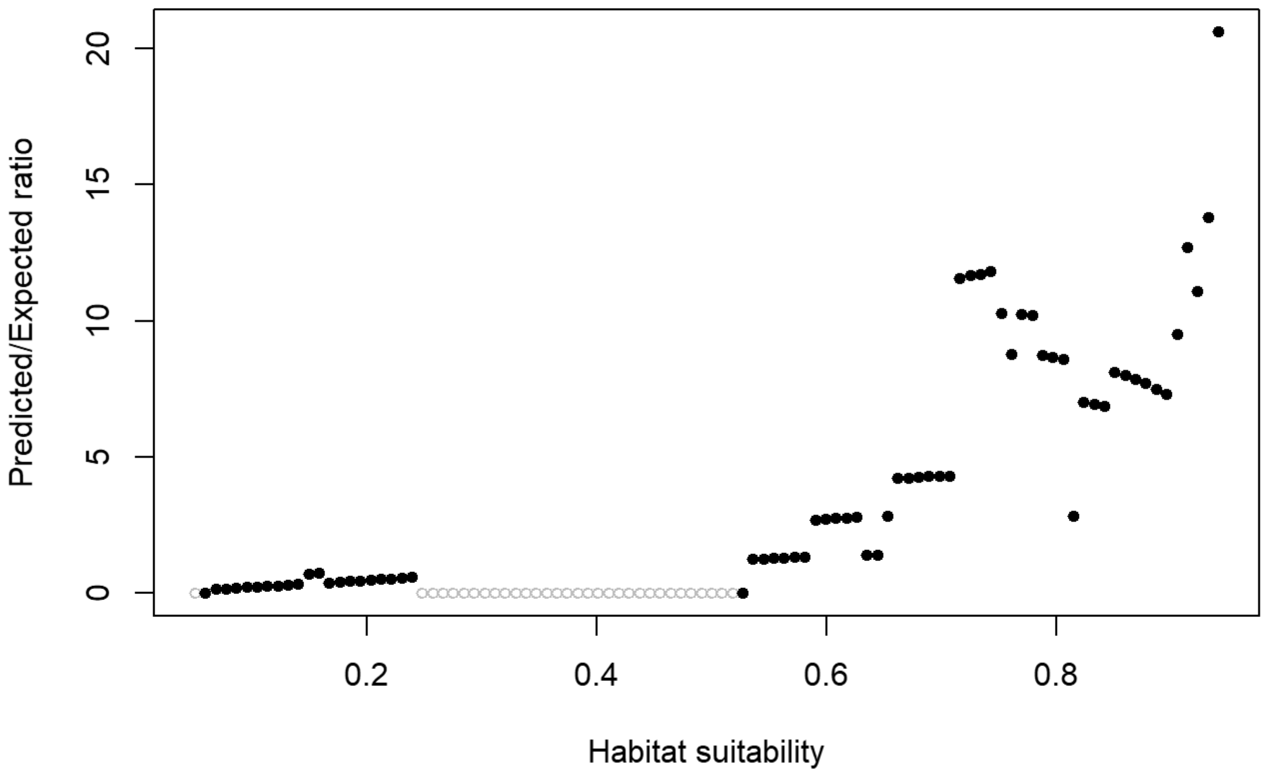
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.979798
## cor            : 0.9061495
## max TPR+TNR at : 0.5786069
```

And let's go ahead and estimate the Boyce Index

```
ecospat.boyce(fit=GAMpreds,pres_test,nclass=0,PEplot = TRUE)
```

```
## Warning in if (class(obs) == "data.frame" | class(obs) == "matrix") {: the
## condition has length > 1 and only the first element will be used
```


Predicted/Expected ratio
0
5
10
15
20
0.2
0.4
0.6
0.8
Habitat suitability

```
## $F.ratio
##   [1]  0.0000000  0.0000000  0.1418119  0.1664096  0.1893913  0.2108994
##   [7]  0.2331025  0.2541363  0.2758817  0.2999464  0.3228983  0.6911034
##  [13]  0.7380532  0.3920810  0.4145264  0.4387584  0.4612315  0.4855881
##  [19]  0.5114548  0.5358080  0.5560797  0.5791226  0.0000000  0.0000000
##  [25]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
##  [31]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
##  [37]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
##  [43]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
##  [49]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
##  [55]  1.2475665  1.2609720  1.2810742  1.3004536  1.3218450  1.3342499
##  [61]  2.7006681  2.7341268  2.7684250  2.7720581  2.7824911  1.3983505
##  [67]  1.3999392  2.8169476  4.2375309  4.2189912  4.2578685  4.2874999
##  [73]  4.2974689  4.3175466 11.5788237 11.6587092 11.6955965 11.8147486
##  [79] 10.2988164  8.7838562 10.2579886 10.2255587  8.7233970  8.6519356
##  [85]  8.5982698  2.8288039  7.0276416  6.9523029  6.8517877  8.0990458
##  [91]  8.0098363  7.8566195  7.7131757  7.5107473  7.3174651  9.5041992
##  [97] 12.6890105 11.0741147 13.8208263 20.6136315
##
## $Spearman.cor
## [1] 0.922
##
## $HS
##   [1] 0.04999985 0.05899982 0.06799980 0.07699977 0.08599974 0.09499972
##   [7] 0.10399969 0.11299966 0.12199964 0.13099961 0.13999958 0.14899956
##  [13] 0.15799953 0.16699950 0.17599948 0.18499945 0.19399942 0.20299940
##  [19] 0.21199937 0.22099934 0.22999931 0.23899929 0.24799926 0.25699923
##  [25] 0.26599921 0.27499918 0.28399915 0.29299913 0.30199910 0.31099907
##  [31] 0.31999905 0.32899902 0.33799899 0.34699897 0.35599894 0.36499891
##  [37] 0.37399889 0.38299886 0.39199883 0.40099881 0.40999878 0.41899875
##  [43] 0.42799872 0.43699870 0.44599867 0.45499864 0.46399862 0.47299859
##  [49] 0.48199856 0.49099854 0.49999851 0.50899848 0.51799846 0.52699843
##  [55] 0.53599840 0.54499838 0.55399835 0.56299832 0.57199830 0.58099827
##  [61] 0.58999824 0.59899822 0.60799819 0.61699816 0.62599813 0.63499811
##  [67] 0.64399808 0.65299805 0.66199803 0.67099800 0.67999797 0.68899795
##  [73] 0.69799792 0.70699789 0.71599787 0.72499784 0.73399781 0.74299779
##  [79] 0.75199776 0.76099773 0.76999771 0.77899768 0.78799765 0.79699763
##  [85] 0.80599760 0.81499757 0.82399755 0.83299752 0.84199749 0.85099746
##  [91] 0.85999744 0.86899741 0.87799738 0.88699736 0.89599733 0.90499730
##  [97] 0.91399728 0.92299725 0.93199722 0.94099720
```

ME Evaluation

```
p<-extract(MEpreds,pres_test)
a<-extract(MEpreds,abs_test)
#And let's get rid of nasty NA values and shrink a to the size of p
p<-p[!is.na(p)]
a<-a[!is.na(a)]
a<-a[1:length(p)]
#Let's look at the shape of these data
#Lets weld all the data together
all_vals<-c(p,a)
e<-evaluate(p=p,a=a)
AUC_ME<-e@auc
COR_ME<-e@cor
pa<-c(replicate(length(p),1),replicate(length(a),0))
kappaME<-ecospat.max.kappa(all_vals,pa)
TSS_ME<-ecospat.max.tss(all_vals,pa)
print(paste('Max kappa: ', kappaME[2] ))
```

```
## [1] "Max kappa:  0.893939393939394"
```

```
print(paste('TSS:', TSS_ME[[2]]))
```

```
## [1] "TSS: 0.893939393939394"
```

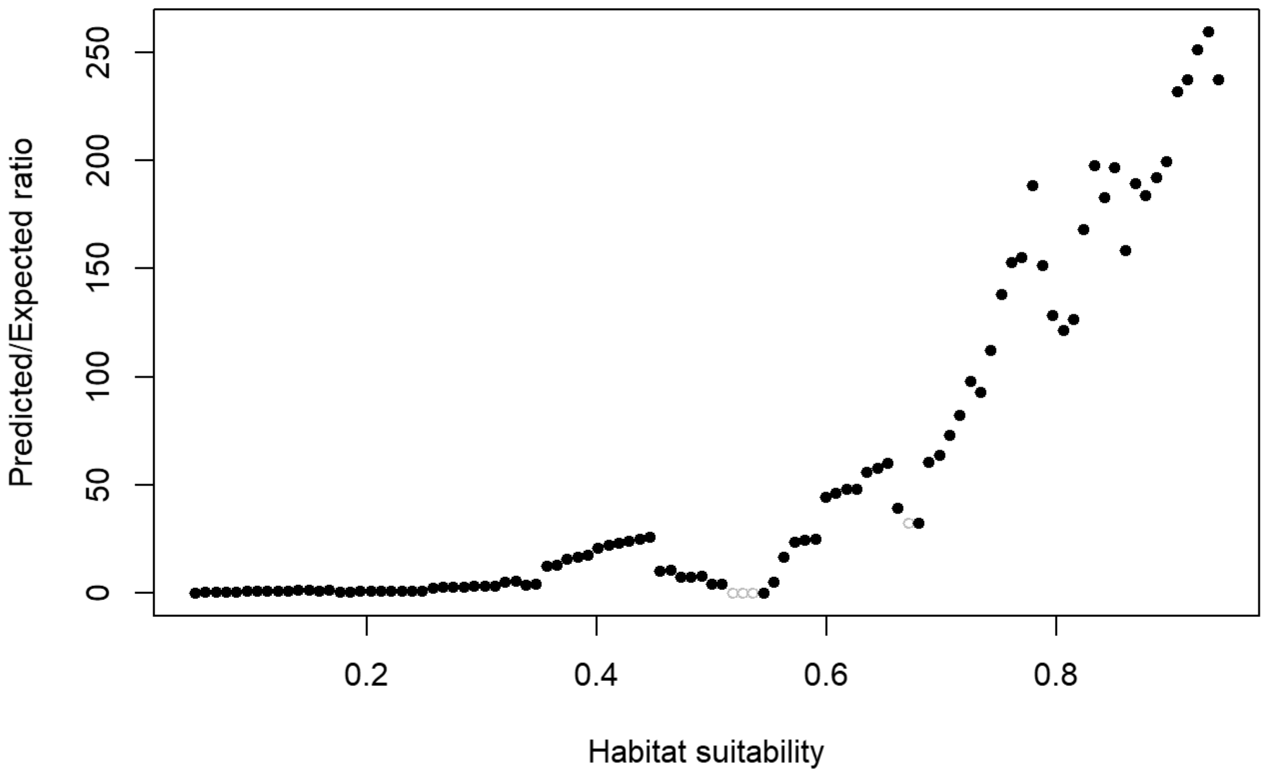
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9791093
## cor            : 0.8485187
## max TPR+TNR at : 0.07696153
```

And let's go ahead and estimate the Boyce Index

```
ecospat.boyce(fit=MEpreds,pres_test,nclass=0,PEplot = TRUE)
```

```
## Warning in if (class(obs) == "data.frame" | class(obs) == "matrix") {: the
## condition has length > 1 and only the first element will be used
```


Predicted/Expected ratio
0
50
100
150
200
250
0.2
0.4
0.6
0.8
Habitat suitability

```
## $F.ratio
##   [1]   0.0277580   0.3494774   0.3256146   0.6167409   0.7356541   0.8529492
##   [7]   0.9795617   1.1020739   1.2330018   0.9145323   1.5152857   1.6544261
##  [13]   1.2067378   1.3066594   0.7076748   0.7701926   0.8376477   0.8951542
##  [19]   0.9620142   1.0321205   1.0941363   1.1557103   1.2284943   2.5940628
##  [25]   2.7407151   2.8969745   3.0280889   3.1319970   3.2539960   3.3882122
##  [31]   5.3123921   5.5158510   3.8666731   4.0267315  12.5271036  13.0812367
##  [37]  15.9480902  16.6510787  17.4600380  21.0779157  22.0606608  22.9687890
##  [43]  24.0425332  25.2215913  26.0663868  10.1646365  10.5766509   7.3126399
##  [49]   7.5982312   7.9155021   4.1651149   4.3875839   0.0000000   0.0000000
##  [55]   0.0000000   0.0000000   5.3846169  16.8407265  23.5448374  24.6435965
##  [61]  25.2323514  44.5366202  46.0423066  47.9181043  48.0515809  55.8495105
##  [67]  57.7584293  60.0453112  39.4507948  32.4970503  32.4970503  60.5281317
##  [73]  63.6550463  72.8382162  82.2468610  97.7920496  92.8001542 112.1708530
##  [79] 138.0891183 153.1111025 155.1770694 188.3586993 151.5931014 128.4737258
##  [85] 121.3970271 126.8109596 168.0245215 197.6759077 182.7186812 196.7739644
##  [91] 158.5260593 189.5661269 183.7667807 192.2200526 199.3983948 232.0003854
##  [97] 237.4650625 251.1449324 259.4062789 237.4650625
##
## $Spearman.cor
## [1] 0.955
##
## $HS
##   [1] 0.050 0.059 0.068 0.077 0.086 0.095 0.104 0.113 0.122 0.131 0.140 0.149
##  [13] 0.158 0.167 0.176 0.185 0.194 0.203 0.212 0.221 0.230 0.239 0.248 0.257
##  [25] 0.266 0.275 0.284 0.293 0.302 0.311 0.320 0.329 0.338 0.347 0.356 0.365
##  [37] 0.374 0.383 0.392 0.401 0.410 0.419 0.428 0.437 0.446 0.455 0.464 0.473
##  [49] 0.482 0.491 0.500 0.509 0.518 0.527 0.536 0.545 0.554 0.563 0.572 0.581
##  [61] 0.590 0.599 0.608 0.617 0.626 0.635 0.644 0.653 0.662 0.671 0.680 0.689
##  [73] 0.698 0.707 0.716 0.725 0.734 0.743 0.752 0.761 0.770 0.779 0.788 0.797
##  [85] 0.806 0.815 0.824 0.833 0.842 0.851 0.860 0.869 0.878 0.887 0.896 0.905
##  [97] 0.914 0.923 0.932 0.941
```

BRT Evaluation

```
p<-extract(BRTpreds,pres_test)
a<-extract(BRTpreds,abs_test)
#And let's get rid of nasty NA values and shrink a to the size of p
p<-p[!is.na(p)]
a<-a[!is.na(a)]
a<-a[1:length(p)]
#Let's look at the shape of these data
#lets weld all the data together
all_vals<-c(p,a)
e<-evaluate(p=p,a=a)
AUC_BRT<-e@auc
COR_BRT<-e@cor
pa<-c(replicate(length(p),1),replicate(length(a),0))
kappaBRT<-ecospat.max.kappa(all_vals,pa)
TSS_BRT<-ecospat.max.tss(all_vals,pa)
print(paste('Max kappa: ', kappaBRT[2] ))
```

```
## [1] "Max kappa:  0.893939393939394"
```

```
print(paste('TSS:', TSS_BRT[[2]]))
```

```
## [1] "TSS: 0.893939393939394"
```

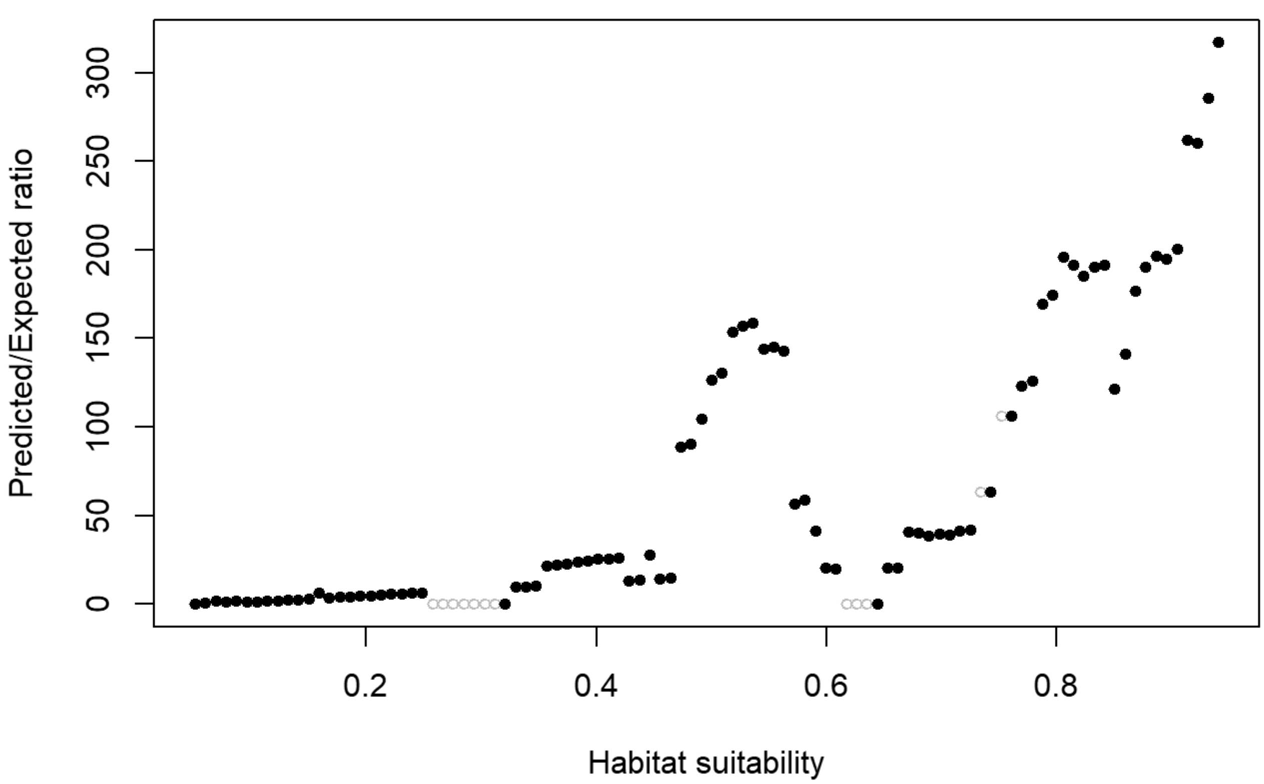
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.979798
## cor            : 0.8519668
## max TPR+TNR at : 0.02506784
```

And let's go ahead and estimate the Boyce Index

```
ecospat.boyce(fit=BRTpreds,pres_test,nclass=0,PEplot = TRUE)
```

```
## Warning in if (class(obs) == "data.frame" | class(obs) == "matrix") {: the
## condition has length > 1 and only the first element will be used
```

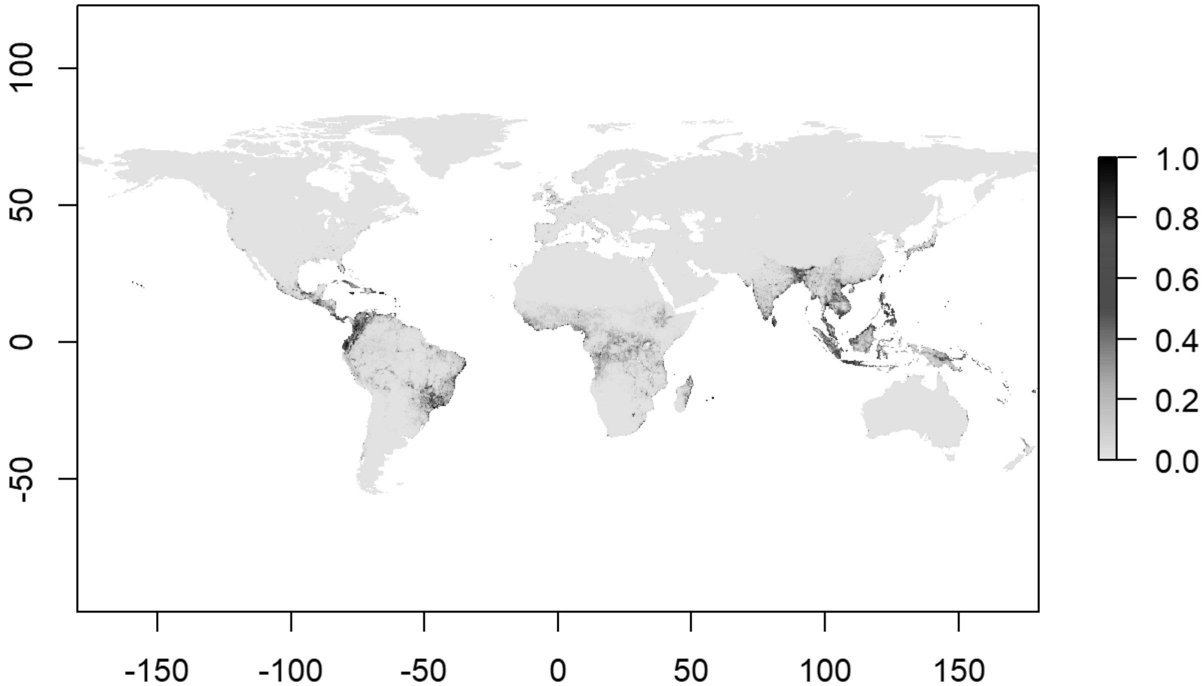


```
## $F.ratio
##   [1]   0.06663589   0.72448020   1.91518139   1.43904213   1.91134409
##   [6]   1.19416555   1.44933914   1.71533154   2.00029192   2.30027347
##  [11]   2.59862177   2.87333033   6.37609224   3.54754268   3.86869647
##  [16]   4.18397224   4.51072541   4.85110167   5.11631761   5.49262923
##  [21]   5.78035883   6.09487135   6.41760325   0.00000000   0.00000000
##  [26]   0.00000000   0.00000000   0.00000000   0.00000000   0.00000000
##  [31]   0.00000000   9.61388680   9.88379538  10.42747383  21.42921434
##  [36]  22.26831008  22.60880412  23.96459951  24.52099153  25.27548358
##  [41]  25.67041301  26.21659201  13.32086297  13.89676494  27.63767831
##  [46]  14.43960732  14.75664461  88.89473684  90.15949936 104.54859117
##  [51] 126.37741790 130.56210062 153.31269707 156.92856256 158.80121844
##  [56] 144.25519897 144.96233230 142.86142893  56.72439090  58.67522974
##  [61]  41.18706934  20.42287002  19.87386814   0.00000000   0.00000000
##  [66]   0.00000000   0.00000000  20.36660867  20.31065645  40.51002163
##  [71]  39.96258890  38.70721962  39.64117398  39.22057797  41.07266082
##  [76]  41.76880761  63.18870895  63.18870895 106.22239867 106.22239867
##  [81] 122.87665841 125.66139854 169.46885839 174.46793976 195.69914861
##  [86] 191.20031760 185.34181205 190.10774436 191.20031760 121.19801553
##  [91] 141.39768479 176.96199608 190.05344338 196.43446479 194.55470914
##  [96] 200.64723657 261.95949026 260.31968125 285.66765639 316.99232001
##
## $Spearman.cor
## [1] 0.882
##
## $HS
##   [1] 0.05089184 0.05987968 0.06886752 0.07785536 0.08684320 0.09583104
##   [7] 0.10481887 0.11380671 0.12279455 0.13178239 0.14077023 0.14975807
##  [13] 0.15874591 0.16773375 0.17672159 0.18570943 0.19469727 0.20368511
##  [19] 0.21267295 0.22166079 0.23064863 0.23963647 0.24862431 0.25761215
##  [25] 0.26659999 0.27558783 0.28457566 0.29356350 0.30255134 0.31153918
##  [31] 0.32052702 0.32951486 0.33850270 0.34749054 0.35647838 0.36546622
##  [37] 0.37445406 0.38344190 0.39242974 0.40141758 0.41040542 0.41939326
##  [43] 0.42838110 0.43736894 0.44635678 0.45534462 0.46433245 0.47332029
##  [49] 0.48230813 0.49129597 0.50028381 0.50927165 0.51825949 0.52724733
##  [55] 0.53623517 0.54522301 0.55421085 0.56319869 0.57218653 0.58117437
##  [61] 0.59016221 0.59915005 0.60813789 0.61712573 0.62611357 0.63510141
##  [67] 0.64408924 0.65307708 0.66206492 0.67105276 0.68004060 0.68902844
##  [73] 0.69801628 0.70700412 0.71599196 0.72497980 0.73396764 0.74295548
##  [79] 0.75194332 0.76093116 0.76991900 0.77890684 0.78789468 0.79688252
##  [85] 0.80587036 0.81485819 0.82384603 0.83283387 0.84182171 0.85080955
##  [91] 0.85979739 0.86878523 0.87777307 0.88676091 0.89574875 0.90473659
##  [97] 0.91372443 0.92271227 0.93170011 0.94068795
```

Making the ensemble and evaluation

The ensemble is simply the average of GAM, ME, and BRT predictions weighted by AUC.

```
ENSpreds<-(GAMpreds*AUC_GAM+MEpreds*AUC_ME+BRTpreds*AUC_BRT)/(AUC_GAM+AUC_ME+AUC_BRT)
writeRaster(ENSpreds, filename=paste0(genus,"_",species,"_ENS.tif"), overwrite=TRUE)
plot(ENSpreds, col=warm(100), zlim=c(0,1))
```



And

let's evaluate

```
p<-extract(ENSpreds,pres_test)
a<-extract(ENSpreds,abs_test)
#And let's get rid of nasty NA values and shrink a to the size of p
p<-p[!is.na(p)]
a<-a[!is.na(a)]
a<-a[1:length(p)]
#Let's look at the shape of these data
#Lets weld all the data together
all_vals<-c(p,a)
e<-evaluate(p=p,a=a)
AUC_ENS<-e@auc
COR_ENS<-e@cor
pa<-c(replicate(length(p),1),replicate(length(a),0))
kappaENS<-ecospat.max.kappa(all_vals,pa)
TSS_ENS<-ecospat.max.tss(all_vals,pa)
print(paste('Max kappa: ', kappaENS[2] ))
```

```
## [1] "Max kappa:  0.909090909090909"
```

```
print(paste('TSS:', TSS_ENS[[2]]))
```

```
## [1] "TSS: 0.909090909090909"
```

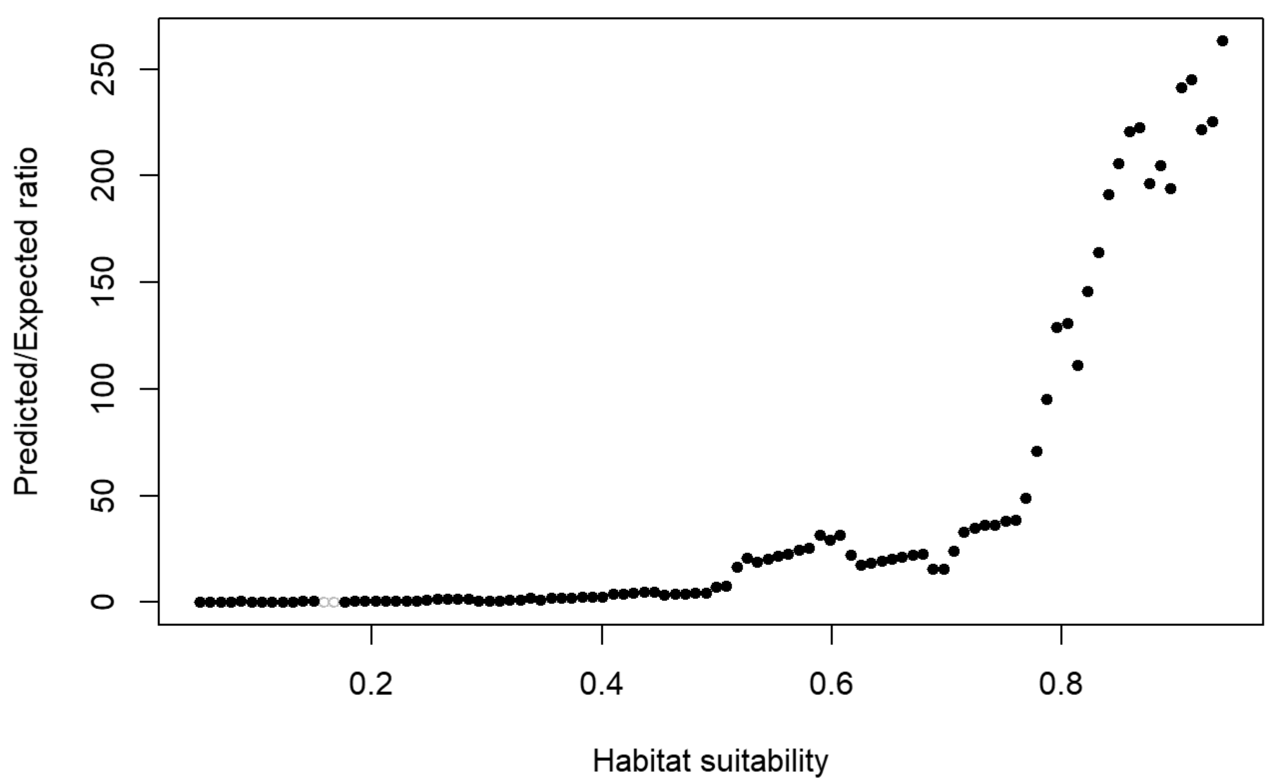
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9802571
## cor            : 0.88864
## max TPR+TNR at : 0.2343614
```

and the Boyce ploy for the ensemble

```
ecospat.boyce(fit=ENSpreds,pres_test,nclass=0,PEplot = TRUE)
```

```
## Warning in if (class(obs) == "data.frame" | class(obs) == "matrix") {: the
## condition has length > 1 and only the first element will be used
```



```
## $F.ratio
##   [1]   0.01443312   0.19919920   0.25777821   0.31081640   0.36254801
##   [6]   0.20841426   0.23470092   0.26097211   0.28669791   0.31251126
##  [11]   0.33989605   0.36657472   0.00000000   0.00000000   0.00000000
##  [16]   0.46970006   0.49247795   0.51444429   0.53674161   0.55965776
##  [21]   0.57898653   0.59756539   1.23940971   1.28497070   1.33076752
##  [26]   1.37712191   1.41901707   0.73424163   0.75826451   0.78117909
##  [31]   0.81385722   0.84415151   1.75378459   0.91645952   1.89833841
##  [36]   1.99220667   2.08578895   2.20491469   2.31395272   2.44318538
##  [41]   3.86667309   4.03552344   4.26605825   4.54213329   4.82787045
##  [46]   3.43544561   3.67723400   3.92726637   4.21258060   4.50385559
##  [51]   7.17774655   7.67447642  16.47788770  20.58534313  18.61455043
##  [56]  20.01736177  21.42921434  22.52842747  24.23960311  25.53740569
##  [61]  31.49820611  29.06846244  31.34874465  22.06889238  17.36823559
##  [66]  18.39074365  19.16960833  20.03544430  21.20385931  21.93791973
##  [71]  22.51699172  15.32244341  15.69655827  23.97755334  32.93130934
##  [76]  34.83193850  36.01987307  36.15197529  38.01068868  38.55582241
##  [81]  48.83143294  70.79555764  95.05387218 129.13675017 130.85095482
##  [86] 111.00719140 145.84222506 164.29064327 191.40657469 206.03662640
##  [91] 220.68892380 222.53414892 196.49379262 205.02829055 194.23524328
##  [96] 241.60388717 245.26802195 221.68152766 225.66083996 263.28628730
##
## $Spearman.cor
## [1] 0.982
##
## $HS
##   [1] 0.05023631 0.05922122 0.06820613 0.07719104 0.08617595 0.09516086
##   [7] 0.10414577 0.11313068 0.12211559 0.13110050 0.14008541 0.14907032
##  [13] 0.15805523 0.16704014 0.17602505 0.18500996 0.19399487 0.20297978
##  [19] 0.21196469 0.22094960 0.22993451 0.23891942 0.24790433 0.25688924
##  [25] 0.26587415 0.27485905 0.28384396 0.29282887 0.30181378 0.31079869
##  [31] 0.31978360 0.32876851 0.33775342 0.34673833 0.35572324 0.36470815
##  [37] 0.37369306 0.38267797 0.39166288 0.40064779 0.40963270 0.41861761
##  [43] 0.42760252 0.43658743 0.44557234 0.45455725 0.46354216 0.47252707
##  [49] 0.48151198 0.49049689 0.49948180 0.50846671 0.51745162 0.52643653
##  [55] 0.53542144 0.54440635 0.55339126 0.56237617 0.57136108 0.58034599
##  [61] 0.58933090 0.59831580 0.60730071 0.61628562 0.62527053 0.63425544
##  [67] 0.64324035 0.65222526 0.66121017 0.67019508 0.67917999 0.68816490
##  [73] 0.69714981 0.70613472 0.71511963 0.72410454 0.73308945 0.74207436
##  [79] 0.75105927 0.76004418 0.76902909 0.77801400 0.78699891 0.79598382
##  [85] 0.80496873 0.81395364 0.82293855 0.83192346 0.84090837 0.84989328
##  [91] 0.85887819 0.86786310 0.87684801 0.88583292 0.89481783 0.90380274
##  [97] 0.91278764 0.92177255 0.93075746 0.93974237
```

and finally let’s make a table of evaluation metrics

```
#Let's go in this order of columns, left to right: AUC, COR, Kappa, TSS
eGAM<-c(AUC_GAM,COR_GAM,kappaGAM[2], TSS_GAM[[2]])
eME<-c(AUC_ME, COR_ME, kappaME[2],TSS_ME[[2]])
eBRT<-c(AUC_BRT, COR_BRT, kappaBRT[2],TSS_BRT[[2]])
eENS<-c(AUC_ENS, COR_ENS, kappaENS[2], TSS_ENS[[2]])
all_evals<-rbind(eGAM,eME,eBRT,eENS)
colnames(all_evals)<-c("AUC", "COR","MaxKappa","TSS")
rownames(all_evals)<-c("GAM","MaxEnt", "BRT", "Ensemble")
write.csv(all_evals, file=paste0(genus,"_",species, '_eval.csv'))
```