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

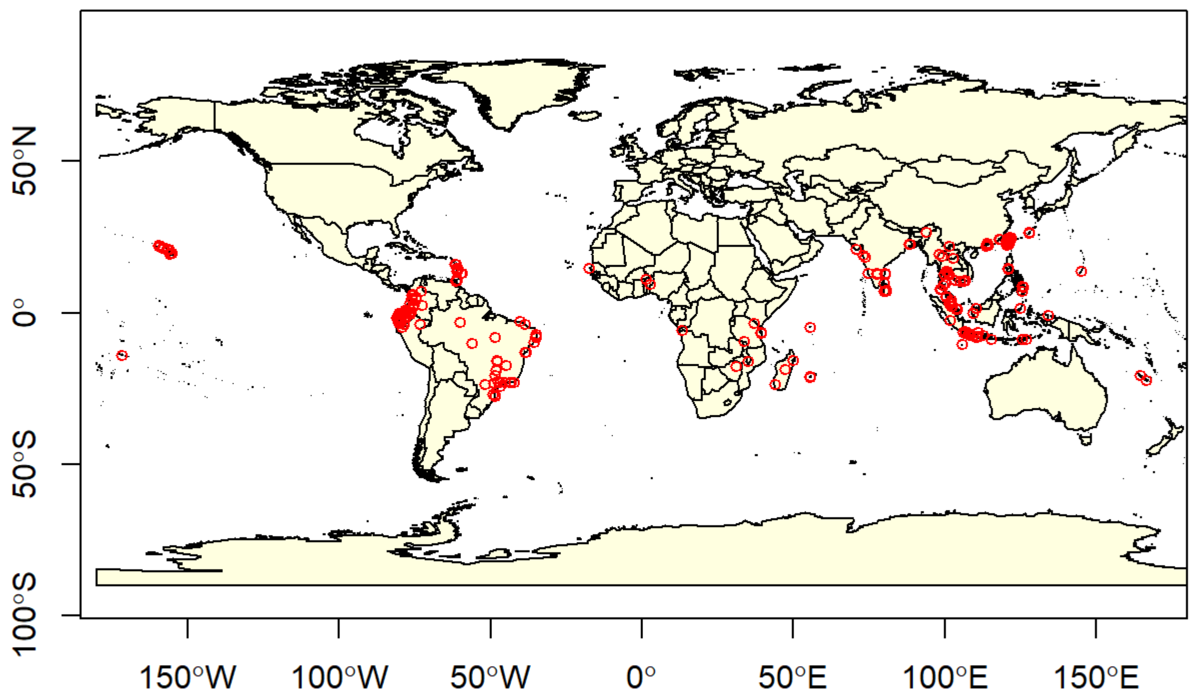
Their are many places in this code where you will need to save files with filenames including the genus and species. We'll save these as strings to automate the creation of file names. Enter your genus name and specific epithet in the quotes below.

```
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 encapsulate the data)  To do this I fi
rst need to convert sdmdata into a spatial points dataframe with the same crs as "wrldsmpl", a giant spatial polygons data f
rame 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 seed 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 integers 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 associated with the number 1 in o
ur 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. These 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.

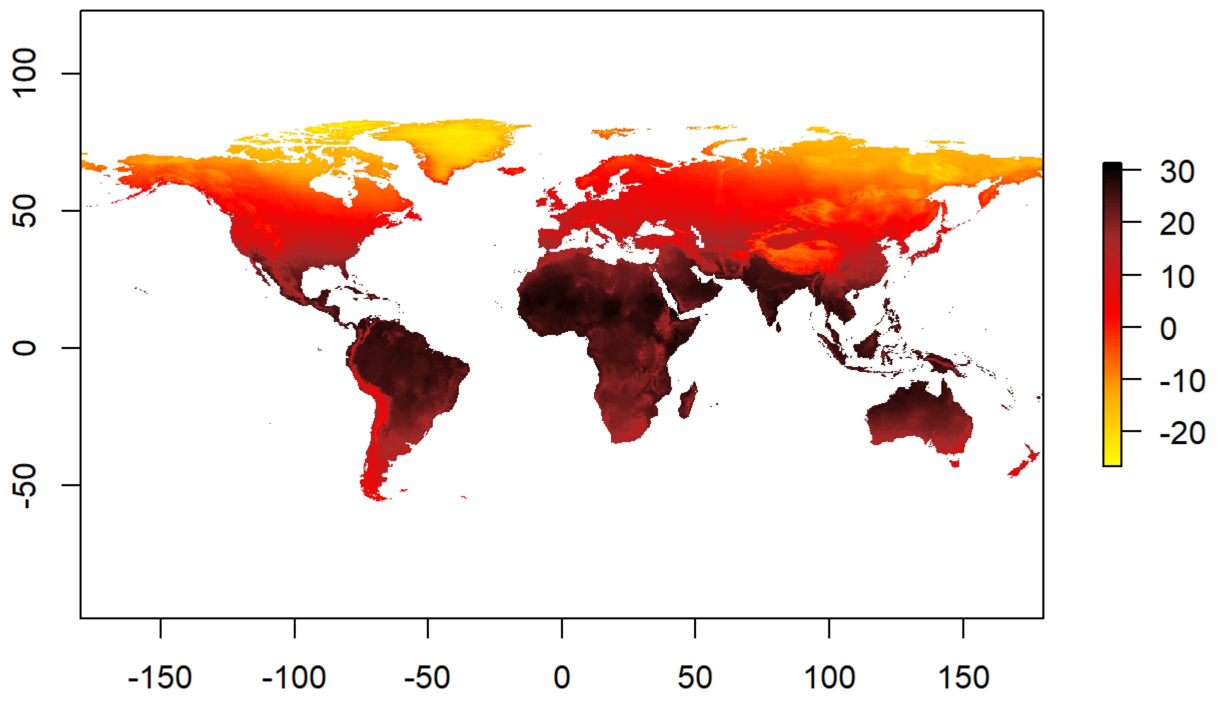
```
#first presdata_train
outdata<-data.frame(pres_train)
colnames(outdata)<-c("lon","lat")
write.csv(outdata, file=paste0(genus,"_",species,"_train.csv"), row.names=FALSE)

#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',  'Min
Temp_ColdestWeek',  'Temp_Ann_Range',   'MeanTemp_WettestQ',    'MeanTemp_DriestQ', 'MeanTemp_WarmestQ',    'MeanTemp_Coldes
tQ',    '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_Seaso
nality',    'Rad_WettestQ', 'Rad_DriestQ',  'Rad_WarmestQ', 'Rad_ColdestQ', 'Ann_Mean_Moisture',    'Highest_Weekly_Moistur
e',  'Lowest_Weekly_Moisture',   'Moisture_Seasonality', 'MeanMoisture_WettestQ',    'MeanMoisture_DriestQ', 'MeanMoisture_W
armestQ',    '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 curvature of the earth). spDis
ts creates a matrix of distances between points. This includes zeros.
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 kilometers, but we need to conv
ert 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)
#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)*3, 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 can 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. NOTE: I had more pres train data than abs train data, and
 so I had to shrink pres train data down to equal abs train instead of the other way around.
#abs_train_data<-abs_train_data[1:nrow(pres_train_data),]
pres_train_data<-pres_train_data[1:nrow(abs_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_DriestWk + Precip_WettestWk + Prec
ip_WettestWk + Precip_Seasonality + Precip_ColdestQ + Lowest_Weekly_Seasonality + Rad_WettestQ + Rad_WarmestQ + Rad_ColdestQ
+ MeanMoisture_WarmestQ + Elev + Human_Impact,
           family = binomial(link = "logit"), data=train_data)
```

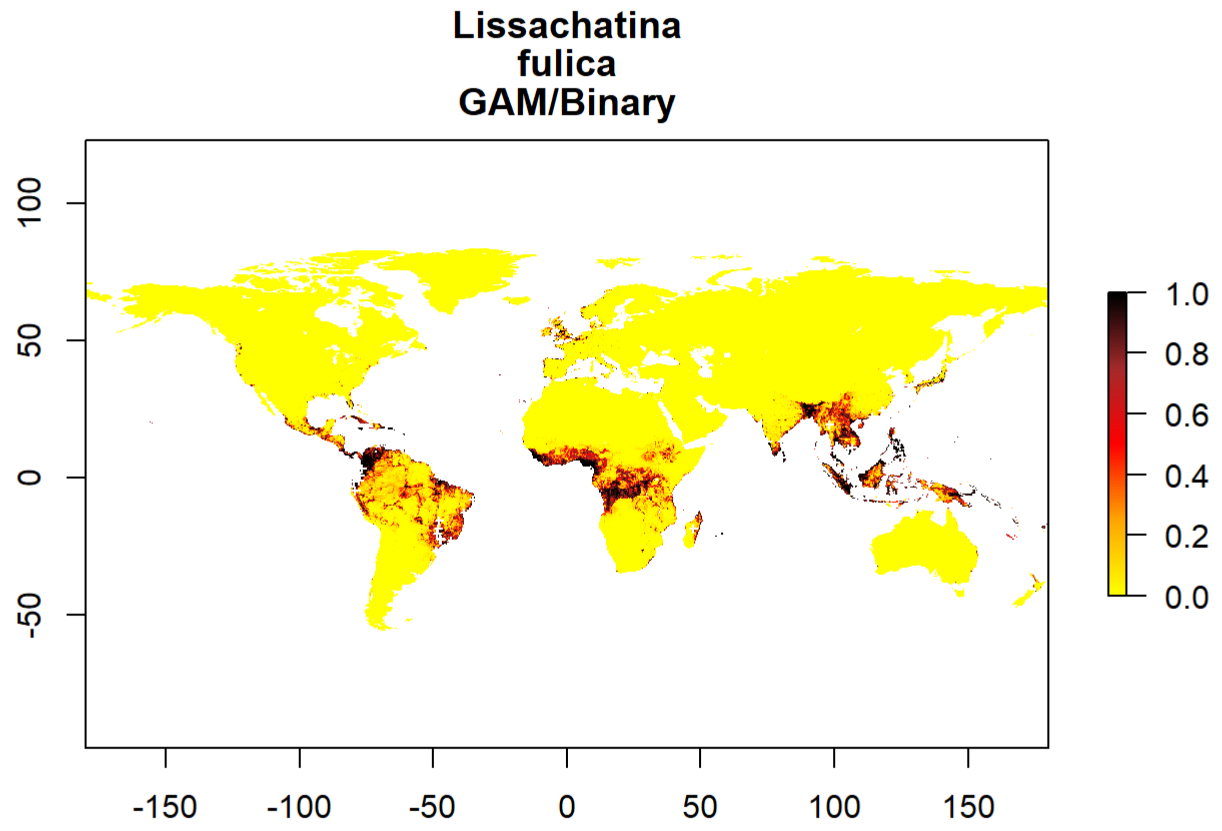
```
## Warning in gam.fit3(x = args$X, y = args$y, sp = lsp, Eb = args$Eb, UrS =
## args$UrS, : fitted probabilities numerically 0 or 1 occurred
```

```
summary(gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## pa ~ Mean_Diurnal_Temp_Range + Temp_Seasonality + MaxTemp_WarmestWeek +
##     Precip_DriestWk + Precip_WettestWk + 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)              -7.289e+00  7.226e+00  -1.009  0.31310
## Mean_Diurnal_Temp_Range   -5.490e-01  2.126e-01  -2.583  0.00981 **
## Temp_Seasonality          -2.745e+02  1.127e+02  -2.435  0.01488 *
## MaxTemp_WarmestWeek        2.902e-01  1.806e-01   1.607  0.10798
## Precip_DriestWk           -1.664e-02  1.646e-02  -1.011  0.31201
## Precip_WettestWk          -7.606e-02  5.290e-02  -1.438  0.15047
## Precip_Seasonality         1.152e+00  1.932e+00   0.596  0.55097
## Precip_ColdestQ            4.930e-03  2.230e-03   2.211  0.02705 *
## Lowest_Weekly_Seasonality  7.766e-01  1.012e+01   0.077  0.93886
## Rad_WettestQ              -1.716e-02  2.660e-02  -0.645  0.51896
## Rad_WarmestQ              -4.411e-02  2.922e-02  -1.510  0.13112
## Rad_ColdestQ               2.227e-02  2.090e-02   1.065  0.28672
## MeanMoisture_WarmestQ      4.420e+00  1.793e+00   2.466  0.01368 *
## Elev                       1.446e-03  1.028e-03   1.407  0.15957
## Human_Impact               3.698e-01  6.053e-02   6.110 9.98e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =  0.883   Deviance explained = 86.6%
## UBRE = -0.74837  Scale est. = 1         n = 454
```

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, 10000, 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 can 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()
```

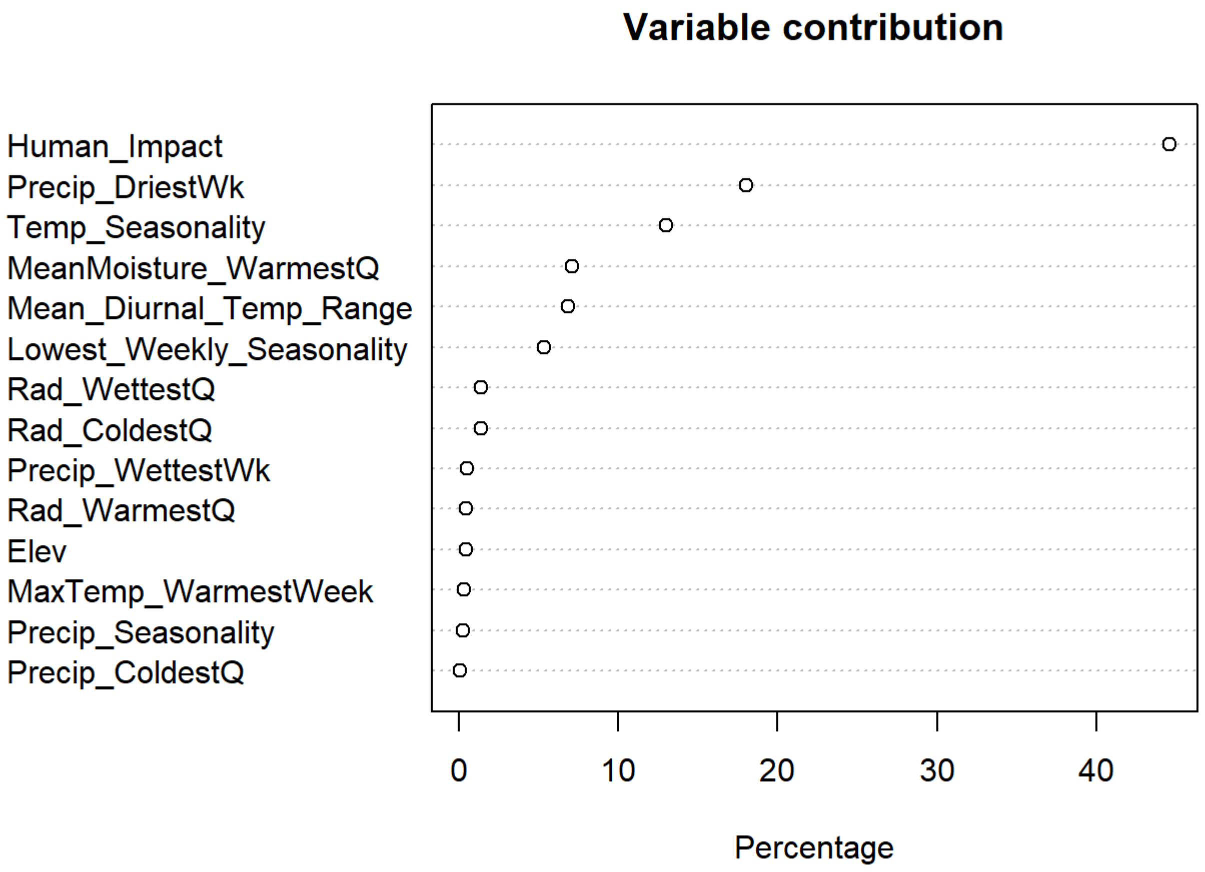
```
## This is MaxEnt version 3.4.1
```

```
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)
```



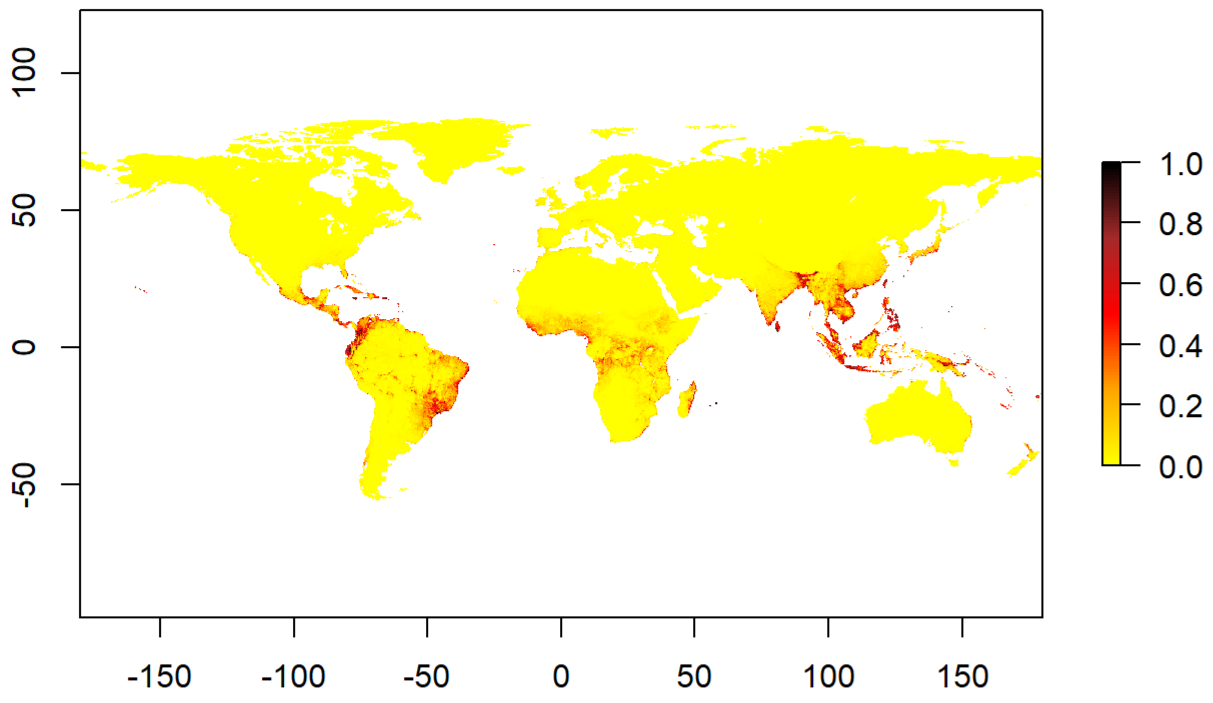
Let's go ahead and make some

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

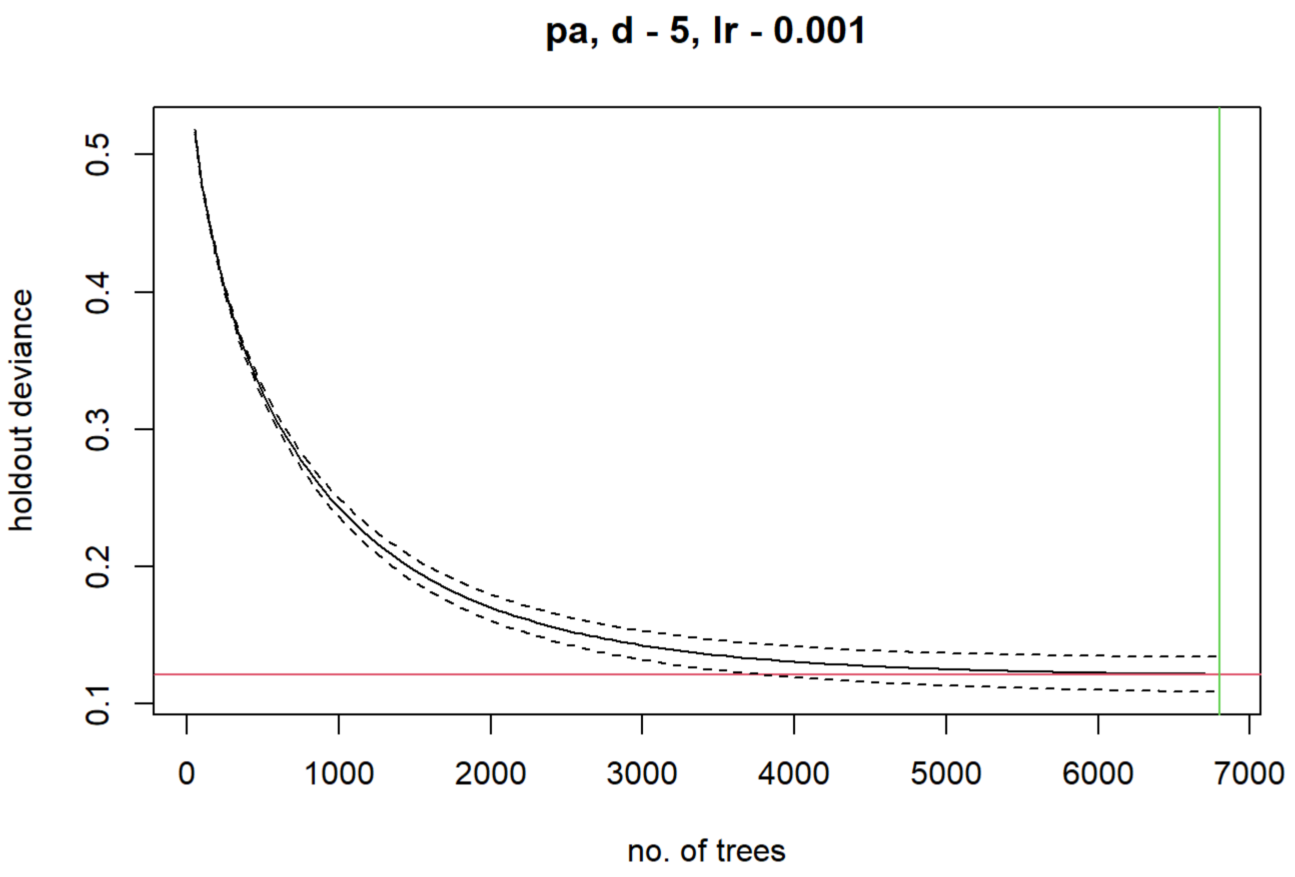
We need to prepare data for BRT in much the same way that we did for GAM, with the exception that we will need a lot more background data. We can use the 10,000 points that we already generated for ME

```
#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.c
omplexity = 5, learning.rate = 0.001, bag.fraction = 0.5)
```

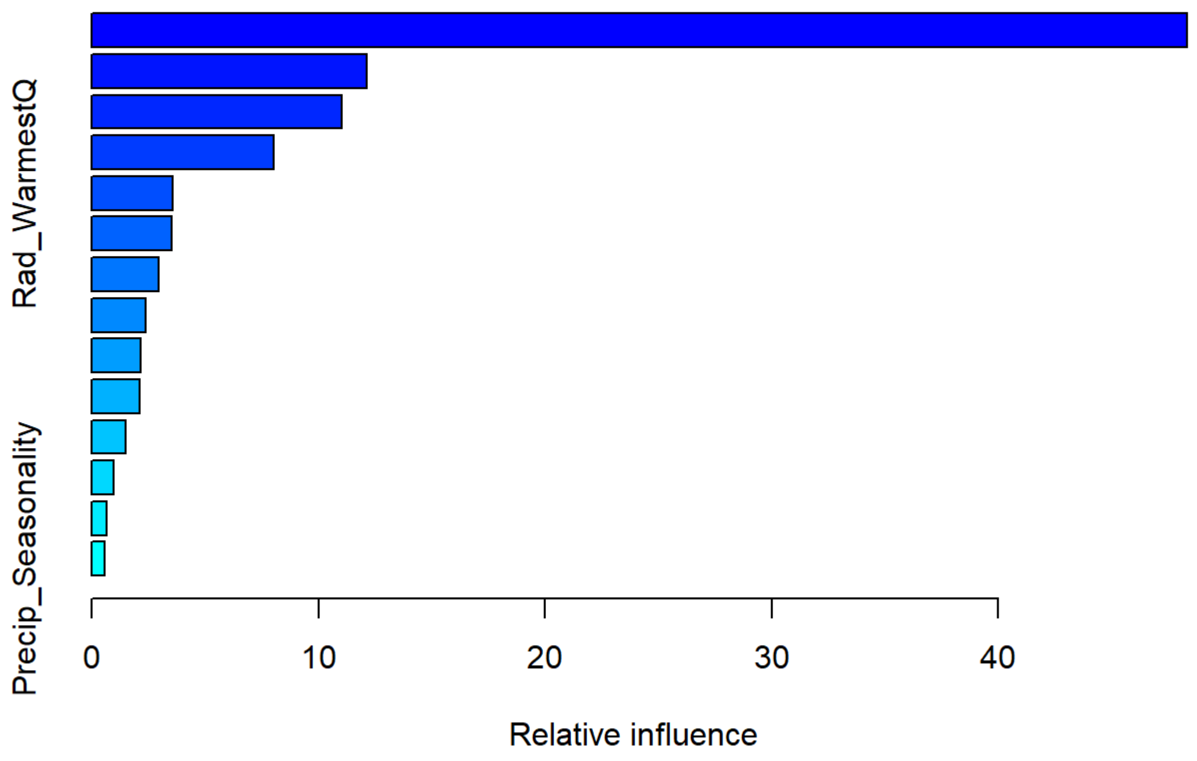
```
##
##
##  GBM STEP - version 2.9
##
## Performing cross-validation optimisation of a boosted regression tree model
## for pa and using a family of bernoulli
## Using 2749 observations and 14 predictors
## creating 10 initial models of 50 trees
##
##  folds are stratified by prevalence
## total mean deviance =  0.57
## tolerance is fixed at  6e-04
## ntrees resid. dev.
## 50    0.5166
## now adding trees...
## 100   0.478
## 150   0.4476
## 200   0.4224
## 250   0.4011
## 300   0.3824
## 350   0.3661
## 400   0.3513
## 450   0.338
## 500   0.3259
## 550   0.3148
## 600   0.3045
## 650   0.2949
## 700   0.2859
## 750   0.2777
## 800   0.2699
## 850   0.2625
## 900   0.2557
## 950   0.2491
## 1000   0.243
## 1050   0.2373
## 1100   0.2318
## 1150   0.2267
## 1200   0.2219
## 1250   0.2172
## 1300   0.2129
## 1350   0.2086
## 1400   0.2047
## 1450   0.2009
## 1500   0.1973
## 1550   0.194
## 1600   0.1908
## 1650   0.1878
## 1700   0.185
## 1750   0.1823
## 1800   0.1797
## 1850   0.1772
## 1900   0.1749
## 1950   0.1726
## 2000   0.1705
## 2050   0.1684
## 2100   0.1665
## 2150   0.1646
## 2200   0.163
## 2250   0.1613
## 2300   0.1597
## 2350   0.1581
## 2400   0.1566
## 2450   0.1551
## 2500   0.1536
## 2550   0.1523
## 2600   0.1511
## 2650   0.1499
## 2700   0.1488
## 2750   0.1477
## 2800   0.1467
## 2850   0.1456
## 2900   0.1447
## 2950   0.1438
## 3000   0.1429
## 3050   0.142
## 3100   0.1412
## 3150   0.1404
## 3200   0.1397
## 3250   0.1389
## 3300   0.1382
## 3350   0.1375
## 3400   0.1368
## 3450   0.1362
## 3500   0.1356
## 3550   0.135
## 3600   0.1345
## 3650   0.1339
## 3700   0.1334
## 3750   0.133
## 3800   0.1325
## 3850   0.1321
## 3900   0.1317
## 3950   0.1311
## 4000   0.1307
## 4050   0.1303
## 4100   0.1299
## 4150   0.1296
## 4200   0.1292
## 4250   0.1289
## 4300   0.1286
## 4350   0.1284
## 4400   0.1281
## 4450   0.1278
## 4500   0.1275
## 4550   0.1272
## 4600   0.1269
## 4650   0.1267
## 4700   0.1265
## 4750   0.1262
## 4800   0.126
## 4850   0.1259
## 4900   0.1257
## 4950   0.1255
## 5000   0.1253
## 5050   0.1252
## 5100   0.125
## 5150   0.1248
## 5200   0.1247
## 5250   0.1246
## 5300   0.1244
## 5350   0.1242
## 5400   0.124
## 5450   0.1239
## 5500   0.1238
## 5550   0.1236
## 5600   0.1235
## 5650   0.1234
## 5700   0.1234
## 5750   0.1232
## 5800   0.1231
## 5850   0.123
## 5900   0.1229
## 5950   0.1228
## 6000   0.1227
## 6050   0.1226
## 6100   0.1224
## 6150   0.1223
## 6200   0.1223
## 6250   0.1222
## 6300   0.1221
## 6350   0.1221
## 6400   0.122
## 6450   0.122
## 6500   0.122
## 6550   0.1219
## 6600   0.1219
## 6650   0.1219
## 6700   0.1219
## 6750   0.1218
## 6800   0.1218
```

```
## fitting final gbm model with a fixed number of 6800 trees for pa
```


pa, d - 5, lr - 0.001
holdout deviance
0.1
0.2
0.3
0.4
0.5
0
1000
2000
3000
4000
5000
6000
7000
no. of trees

```
##
## mean total deviance = 0.57
## mean residual deviance = 0.057
##
## estimated cv deviance = 0.122 ; se = 0.013
##
## training data correlation = 0.955
## cv correlation =  0.891 ; se = 0.01
##
## training data AUC score = 0.999
## cv AUC score = 0.991 ; se = 0.002
##
## elapsed time -  0.05 minutes
```

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


Rad_WarmestQ
Precip_Seasonality
0
10
20
30
40
Relative influence

```
##                                                    var    rel.inf
## Human_Impact                              Human_Impact 48.3184230
## Temp_Seasonality                      Temp_Seasonality 12.1319903
## MeanMoisture_WarmestQ            MeanMoisture_WarmestQ 11.0322261
## Lowest_Weekly_Seasonality Lowest_Weekly_Seasonality  8.0484648
## Rad_WarmestQ                              Rad_WarmestQ  3.5757834
## Mean_Diurnal_Temp_Range        Mean_Diurnal_Temp_Range  3.5558399
## Precip_WettestWk                      Precip_WettestWk  2.9662285
## Precip_DriestWk                        Precip_DriestWk  2.3893975
## Rad_ColdestQ                              Rad_ColdestQ  2.1536378
## MaxTemp_WarmestWeek                MaxTemp_WarmestWeek  2.1335020
## Precip_ColdestQ                        Precip_ColdestQ  1.4916395
## Rad_WettestQ                              Rad_WettestQ  0.9877242
## Elev                                              Elev  0.6573779
## Precip_Seasonality                  Precip_Seasonality  0.5577651
```

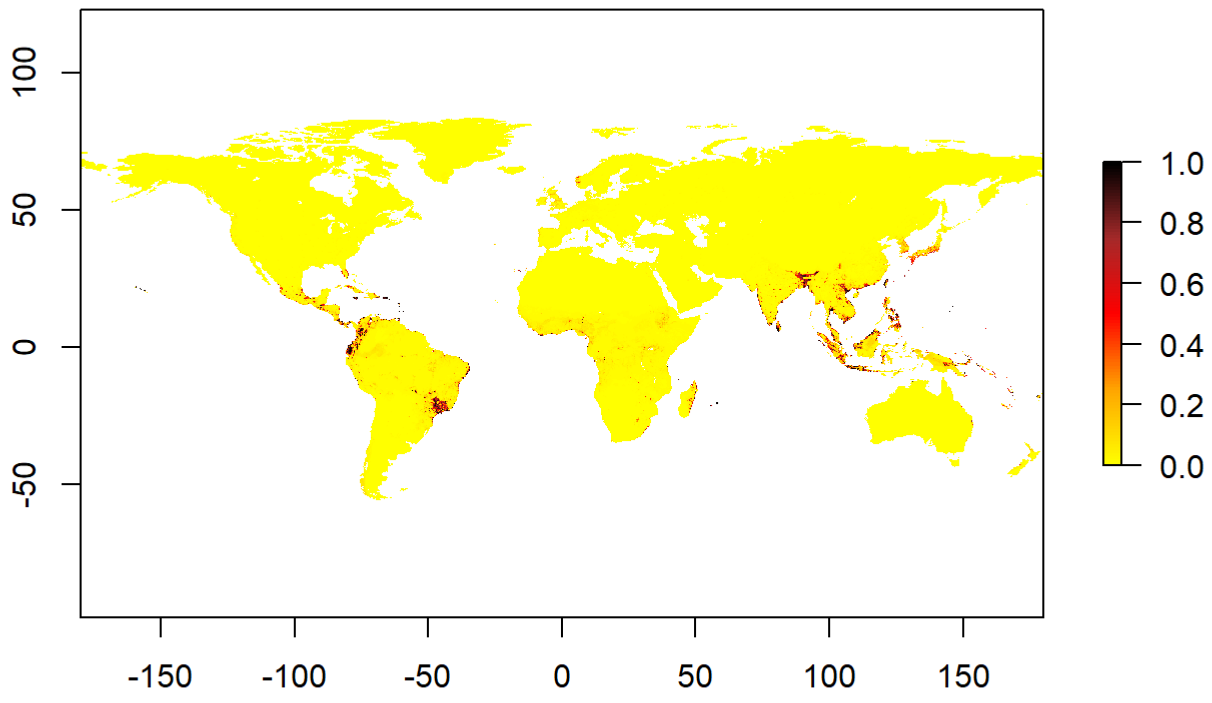
Note: you may want to try different combinations! If your trees are converging too slowly, raise the tree complexity by 1 or two, and back the learning rate down. On the other hand of your holdout deviance drops very quickly and slowly starts to rise, you are overfitting. Drop the tree complexity and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

Let’s make predictions and save them

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

```
## Using 6800 trees...
##
## Using 6800 trees...
##
## Using 6800 trees...
##
## Using 6800 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, 4*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 can 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:  0.909090909090909"
```

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

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

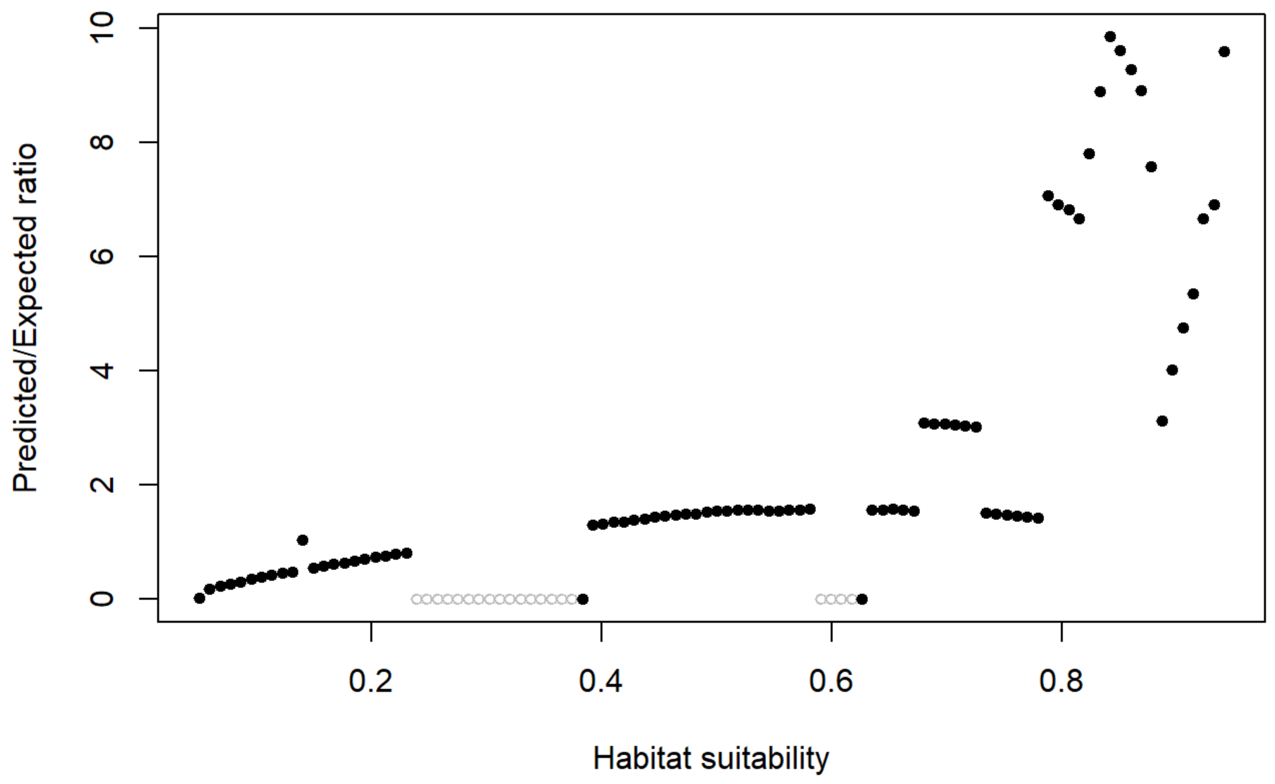
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9917355
## cor            : 0.9266762
## max TPR+TNR at : 0.5362404
```

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
```



```
## $F.ratio
##   [1] 0.01496705 0.18285669 0.23127222 0.27207445 0.30990438 0.34827016
##   [7] 0.38503614 0.42188307 0.45565972 0.48562000 1.03544523 0.55196946
##  [13] 0.58171996 0.61231398 0.64343594 0.67369045 0.70517731 0.73533707
##  [19] 0.76374783 0.78926860 0.81745676 0.00000000 0.00000000 0.00000000
##  [25] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##  [31] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##  [37] 0.00000000 0.00000000 1.29931089 1.31292469 1.34984096 1.36027212
##  [43] 1.38110946 1.41575621 1.43918220 1.45332788 1.47008927 1.48694267
##  [49] 1.49415500 1.52152273 1.54731665 1.54861310 1.56865668 1.57065625
##  [55] 1.56898959 1.55121254 1.55447413 1.55643767 1.57165794 1.57870573
##  [61] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 1.56334932
##  [67] 1.57266091 1.57467070 1.55742131 1.55349421 3.08946049 3.07340634
##  [73] 3.07276764 3.05436023 3.03243599 3.02375417 1.50971594 1.49114138
##  [79] 1.47008927 1.46484623 1.44678649 1.43027258 7.06930479 6.91588302
##  [85] 6.82522059 6.66283250 7.81784873 8.90119584 9.85743860 9.62170678
##  [91] 9.28050080 8.91269313 7.57487597 3.12912484 4.01688614 4.75867594
##  [97] 5.34826063 6.67320677 6.92379922 9.60763996
## 
## $Spearman.cor
## [1] 0.896
## 
## $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
```

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:  0.924242424242424"
```

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

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

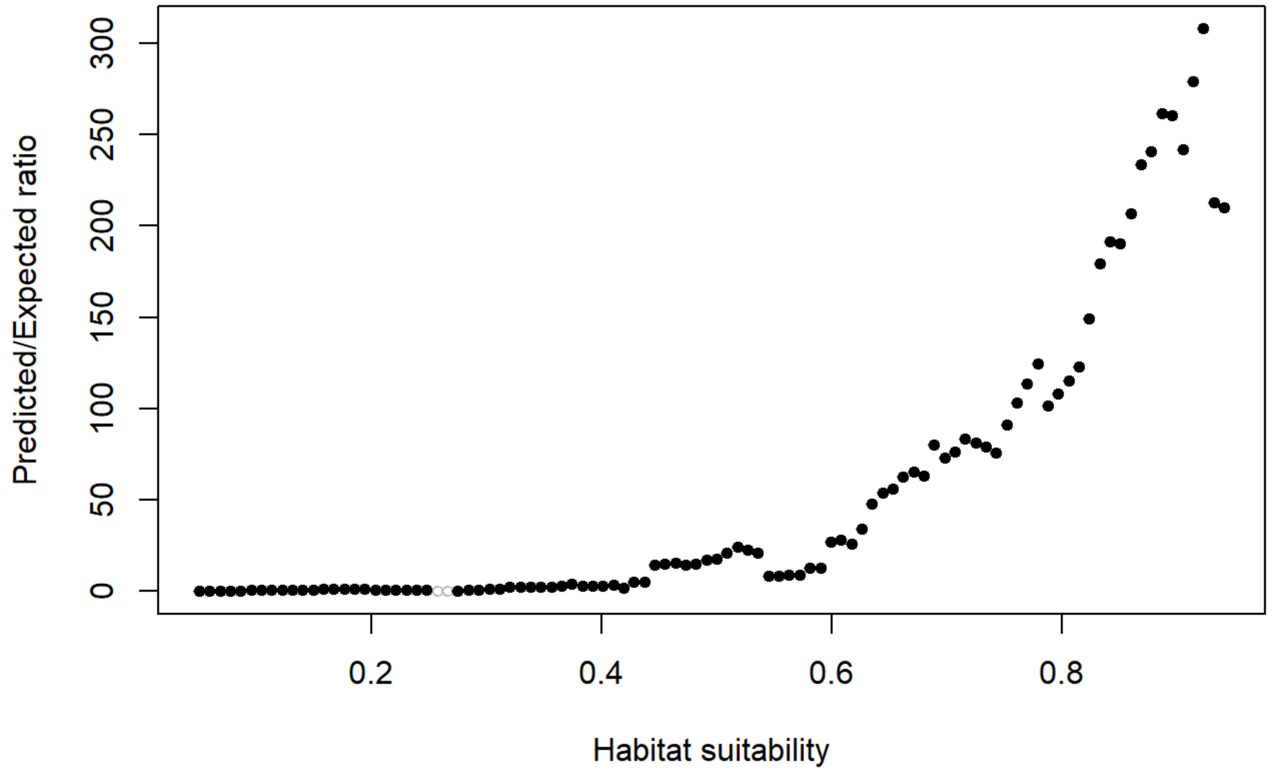
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9951791
## cor            : 0.9053052
## max TPR+TNR at : 0.1349877
```

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
```



```
## $F.ratio
##   [1]   0.01435000   0.09125118   0.12476928   0.15470576   0.36760455
##   [6]   0.42462116   0.48197920   0.53934554   0.58880845   0.64354796
##  [11]   0.69867967   0.74908343   1.20650802   1.28396647   1.36479213
##  [16]   1.43927559   1.01184958   0.53356517   0.56042139   0.58988901
##  [21]   0.62415187   0.65862619   0.69301452   0.00000000   0.00000000
##  [26]   0.00000000   0.84860869   0.89558800   0.94819532   0.99369341
##  [31]   2.09613806   2.18665452   2.27689527   2.36919691   2.47425668
##  [36]   2.58228395   4.03772744   2.81480257   2.94721106   3.05625422
##  [41]   3.14733033   1.63491352   5.03959028   5.22971866  14.61805032
##  [46]  15.03804515  15.60132724  14.22527560  14.79038372  17.37503865
##  [51]  17.89550123  21.18360730  24.25550836  22.67815628  20.79628396
##  [56]   8.10941018   8.45567550   8.77690417   8.99766200  12.44626085
##  [61]  12.90240654  26.81080307  27.89841112  25.67041301  34.13941022
##  [66]  47.74862614  53.72276485  55.78063048  62.42648086  65.38414736
##  [71]  63.23028047  79.92517781  73.09541332  76.04930585  83.38058963
##  [76]  81.30644561  78.97290576  75.63252120  90.95176209 103.35152303
##  [81] 113.41252460 124.68486320 101.70373155 107.98879361 115.37818285
##  [86] 122.90284695 149.45577387 179.54620301 191.29732436 190.48190525
##  [91] 206.79941112 233.77324735 241.07866133 261.54760427 260.71311149
##  [96] 241.73555141 279.38961138 308.04495614 212.75047331 210.37216517
##
## $Spearman.cor
## [1] 0.979
##
## $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:  0.939393939393939"
```

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

```
## [1] "TSS: 0.939393939393939"
```

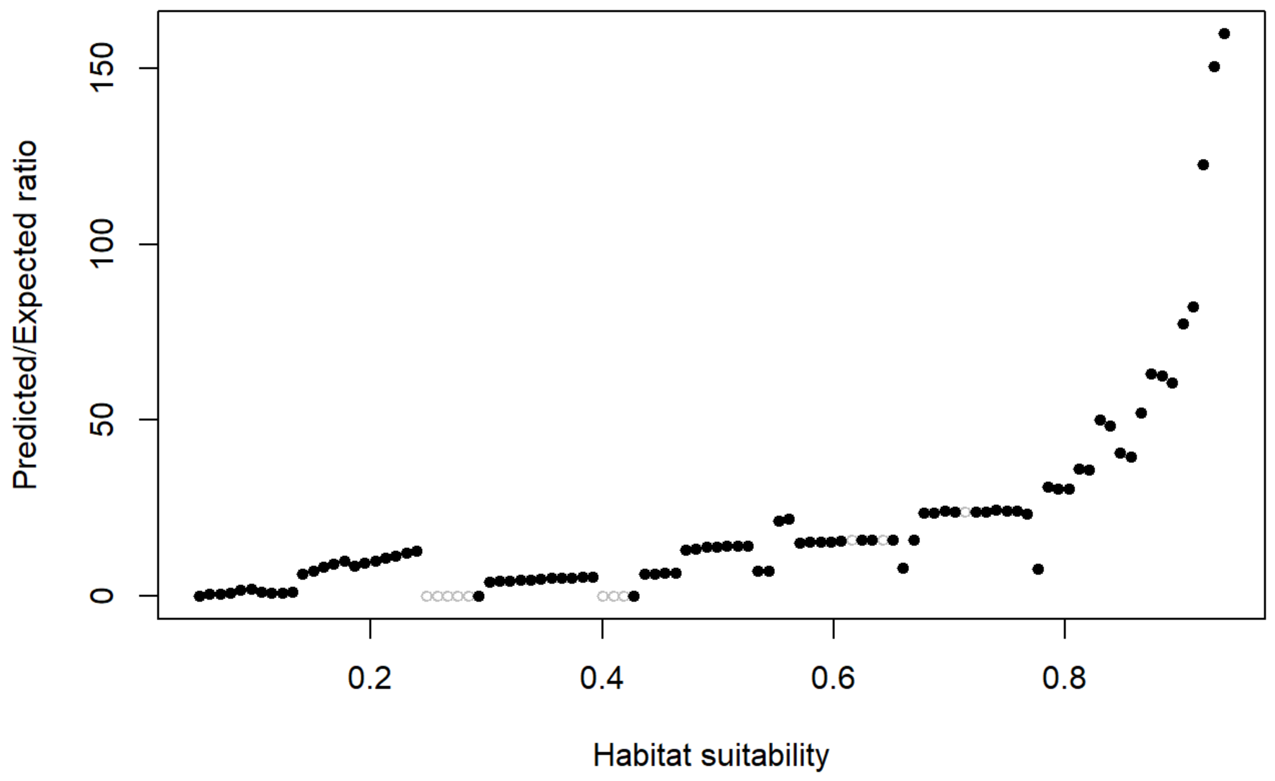
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9933425
## cor            : 0.8652569
## max TPR+TNR at : 0.04760283
```

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
```

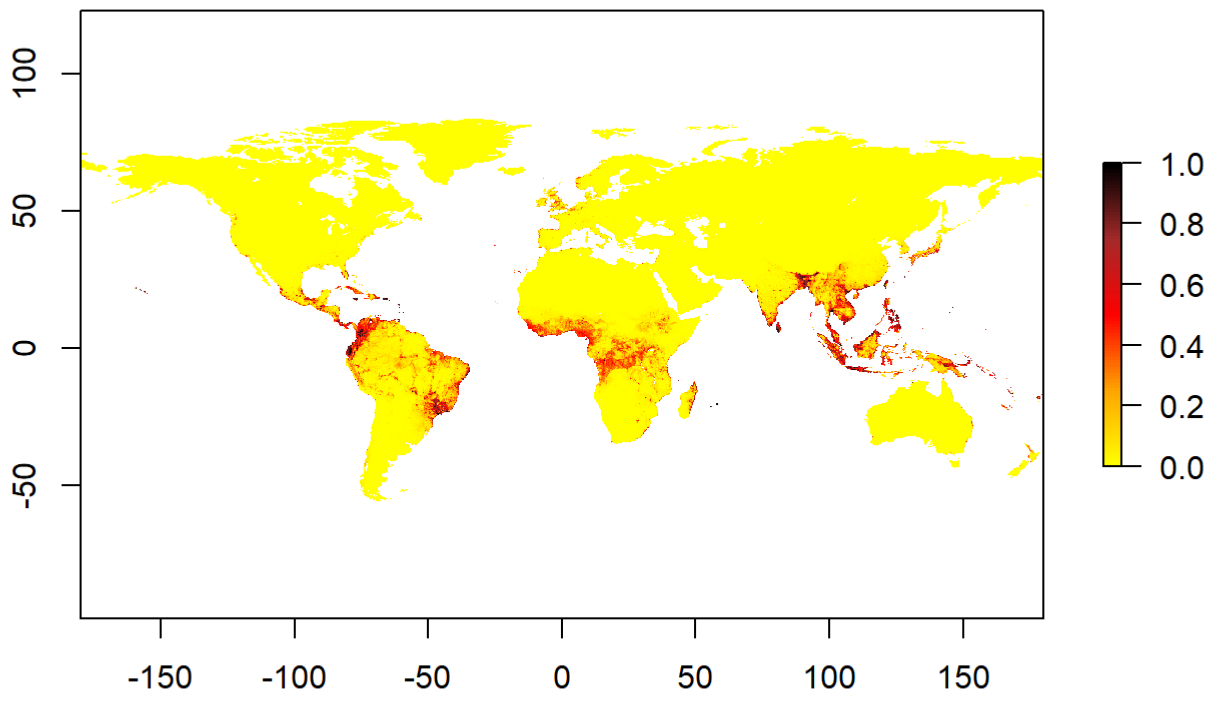


```
## $F.ratio
##   [1]   0.06763294   0.55066413   0.70414746   0.98438759   1.65959458
##   [6]   2.10404239   1.31257505   0.80193936   0.95480808   1.10575515
##  [11]   6.34381238   7.27378881   8.19086965   9.12275290  10.03131472
##  [16]   8.70285927   9.32586433  10.01093967  10.71849068  11.53366450
##  [21]  12.28086204  12.82407450   0.00000000   0.00000000   0.00000000
##  [26]   0.00000000   0.00000000   0.00000000   4.10954916   4.24889595
##  [31]   4.39540960   4.51623638   4.73005691   4.96513025   5.07069887
##  [36]   5.22110095   5.31111993   5.42014586   5.59657755   0.00000000
##  [41]   0.00000000   0.00000000   0.00000000   6.17118443   6.37334392
##  [46]   6.45683751   6.55414800  13.10829601  13.54043763  13.90983809
##  [51]  14.05528317  14.32767238  14.17656558  14.13590621   7.12242673
##  [56]   7.04102757  21.47070362  22.00321115  15.16529015  15.27495650
##  [61]  15.33833807  15.54801040  15.71323900  16.07191076  16.07191076
##  [66]  15.86497628  15.91620871  15.91620871   7.99251778  15.98503556
##  [71]  23.69576586  23.77195803  24.10786613  23.84864177  23.92582184
##  [76]  23.92582184  24.05557141  24.39960049  24.23960311  24.18673592
##  [81]  23.29751769   7.79860648  31.09602081  30.48692349  30.42419320
##  [86]  36.16966217  35.92361005  50.00150013  48.50192374  40.65854600
##  [91]  39.64117398  52.20179310  63.18870895  62.54719922  60.59900777
##  [96]  77.32101876  82.27591833 122.66043502 150.35013287 159.74242155
## 
## $Spearman.cor
## [1] 0.899
## 
## $HS
##   [1] 0.05168500 0.06062743 0.06956985 0.07851228 0.08745471 0.09639713
##   [7] 0.10533956 0.11428199 0.12322442 0.13216684 0.14110927 0.15005170
##  [13] 0.15899412 0.16793655 0.17687898 0.18582140 0.19476383 0.20370626
##  [19] 0.21264868 0.22159111 0.23053354 0.23947596 0.24841839 0.25736082
##  [25] 0.26630325 0.27524567 0.28418810 0.29313053 0.30207295 0.31101538
##  [31] 0.31995781 0.32890023 0.33784266 0.34678509 0.35572751 0.36466994
##  [37] 0.37361237 0.38255479 0.39149722 0.40043965 0.40938208 0.41832450
##  [43] 0.42726693 0.43620936 0.44515178 0.45409421 0.46303664 0.47197906
##  [49] 0.48092149 0.48986392 0.49880634 0.50774877 0.51669120 0.52563362
##  [55] 0.53457605 0.54351848 0.55246091 0.56140333 0.57034576 0.57928819
##  [61] 0.58823061 0.59717304 0.60611547 0.61505789 0.62400032 0.63294275
##  [67] 0.64188517 0.65082760 0.65977003 0.66871245 0.67765488 0.68659731
##  [73] 0.69553974 0.70448216 0.71342459 0.72236702 0.73130944 0.74025187
##  [79] 0.74919430 0.75813672 0.76707915 0.77602158 0.78496400 0.79390643
##  [85] 0.80284886 0.81179128 0.82073371 0.82967614 0.83861856 0.84756099
##  [91] 0.85650342 0.86544585 0.87438827 0.88333070 0.89227313 0.90121555
##  [97] 0.91015798 0.91910041 0.92804283 0.93698526
```

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:  0.939393939393939"
```

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

```
## [1] "TSS: 0.939393939393939"
```

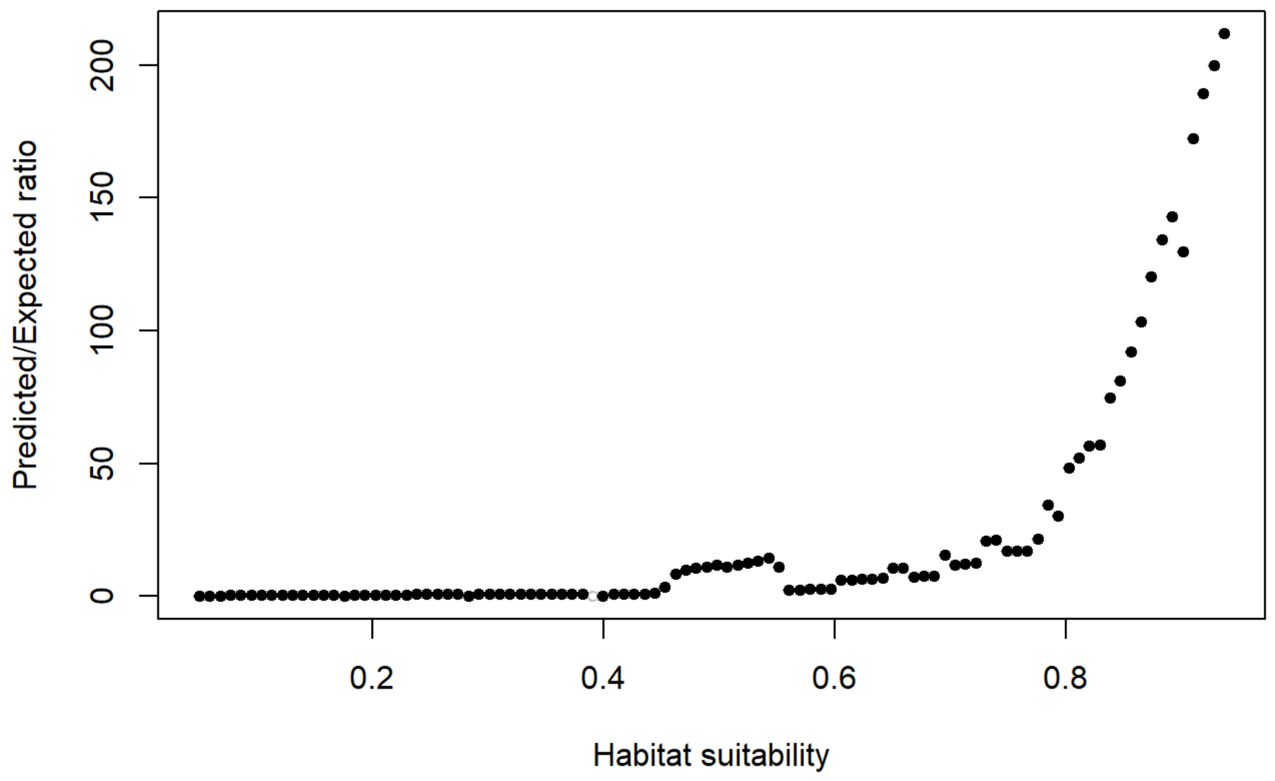
```
e
```

```
## class          : ModelEvaluation
## n presences    : 66
## n absences     : 66
## AUC            : 0.9933425
## cor            : 0.9189526
## max TPR+TNR at : 0.3342835
```

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]   0.01458615   0.11475838   0.15311020   0.37640093   0.44401543
##   [6]   0.50879728   0.28690930   0.31879086   0.34960415   0.37930732
##  [11]   0.40879618   0.43914933   0.46453528   0.49182271   0.00000000
##  [16]   0.54237246   0.56478831   0.58231561   0.59950364   0.61675807
##  [21]   0.63286072   0.64811773   0.65605457   0.66688426   0.67215919
##  [26]   0.68019863   0.00000000   0.69713144   0.69811888   0.70930432
##  [31]   0.71100971   0.71299826   0.71589803   0.72552296   0.72938822
##  [36]   0.74205349   0.76154501   0.77961394   0.00000000   0.00000000
##  [41]   0.85439489   0.89180687   0.93217488   0.98692817   1.03486547
##  [46]   3.31033386   8.20801786   9.86895237  10.46992947  11.10697307
##  [51]  11.77710704  11.01565616  11.79657001  12.52457711  13.36558694
##  [56]  14.31182318  10.96570594   2.33736293   2.49681829   2.62258920
##  [61]   2.74631462   2.87333033   6.02532922   6.22313043   6.42597040
##  [66]   6.63056408   6.82018353  10.46190417  10.62732958   7.24456536
##  [71]   7.46398682   7.64142527  15.58077755  11.81632224  12.20651450
##  [76]  12.41838569  20.95543919  21.25669623  17.01514142  17.03474412
##  [81]  17.08394904  21.49150857  34.20742139  30.37062948  48.46476068
##  [86]  52.03340022  56.80261603  56.90350877  74.72196320  81.24262580
##  [91]  92.18302927 103.46548145 120.33006099 134.23882420 142.96009014
##  [96] 129.70313943 172.31988524 189.41435253 199.98730137 211.83607299
##
## $Spearman.cor
## [1] 0.955
##
## $HS
##   [1] 0.05047038 0.05942409 0.06837780 0.07733151 0.08628521 0.09523892
##   [7] 0.10419263 0.11314634 0.12210005 0.13105376 0.14000747 0.14896118
##  [13] 0.15791489 0.16686859 0.17582230 0.18477601 0.19372972 0.20268343
##  [19] 0.21163714 0.22059085 0.22954456 0.23849827 0.24745197 0.25640568
##  [25] 0.26535939 0.27431310 0.28326681 0.29222052 0.30117423 0.31012794
##  [31] 0.31908165 0.32803535 0.33698906 0.34594277 0.35489648 0.36385019
##  [37] 0.37280390 0.38175761 0.39071132 0.39966503 0.40861873 0.41757244
##  [43] 0.42652615 0.43547986 0.44443357 0.45338728 0.46234099 0.47129470
##  [49] 0.48024841 0.48920211 0.49815582 0.50710953 0.51606324 0.52501695
##  [55] 0.53397066 0.54292437 0.55187808 0.56083179 0.56978549 0.57873920
##  [61] 0.58769291 0.59664662 0.60560033 0.61455404 0.62350775 0.63246146
##  [67] 0.64141517 0.65036887 0.65932258 0.66827629 0.67723000 0.68618371
##  [73] 0.69513742 0.70409113 0.71304484 0.72199855 0.73095225 0.73990596
##  [79] 0.74885967 0.75781338 0.76676709 0.77572080 0.78467451 0.79362822
##  [85] 0.80258193 0.81153563 0.82048934 0.82944305 0.83839676 0.84735047
##  [91] 0.85630418 0.86525789 0.87421160 0.88316531 0.89211901 0.90107272
##  [97] 0.91002643 0.91898014 0.92793385 0.93688756
```

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'))
```