Helicoverpa\_armigera

Nico

2/18/2022

## 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)

## Loading required package: raster

## Loading required package: sp

library(sp)  
library(raster)  
library(stats)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:raster':  
##   
## intersect, select, union

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(knitr)  
library(rgeos)

## rgeos version: 0.5-8, (SVN revision 679)  
## GEOS runtime version: 3.9.1-CAPI-1.14.2   
## Please note that rgeos will be retired by the end of 2023,  
## plan transition to sf functions using GEOS at your earliest convenience.  
## GEOS using OverlayNG  
## Linking to sp version: 1.4-5   
## Polygon checking: TRUE

library(maptools)

## Checking rgeos availability: TRUE  
## Please note that 'maptools' will be retired by the end of 2023,  
## plan transition at your earliest convenience;  
## some functionality will be moved to 'sp'.

library(rgdal)

## Please note that rgdal will be retired by the end of 2023,  
## plan transition to sf/stars/terra functions using GDAL and PROJ  
## at your earliest convenience.  
##   
## rgdal: version: 1.5-27, (SVN revision 1148)  
## Geospatial Data Abstraction Library extensions to R successfully loaded  
## Loaded GDAL runtime: GDAL 3.2.1, released 2020/12/29  
## Path to GDAL shared files: C:/Users/User/Documents/R/win-library/4.1/rgdal/gdal  
## GDAL binary built with GEOS: TRUE   
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ\_VERSION: 721]  
## Path to PROJ shared files: C:/Users/User/Documents/R/win-library/4.1/rgdal/proj  
## PROJ CDN enabled: FALSE  
## Linking to sp version:1.4-5  
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,  
## use options("rgdal\_show\_exportToProj4\_warnings"="none") before loading sp or rgdal.  
## Overwritten PROJ\_LIB was C:/Users/User/Documents/R/win-library/4.1/rgdal/proj

library(ecospat)

## Loading required package: ade4

## Loading required package: ape

##   
## Attaching package: 'ape'

## The following objects are masked from 'package:raster':  
##   
## rotate, zoom

## Loading required package: gbm

## Loaded gbm 2.1.8

## Registered S3 methods overwritten by 'adehabitatMA':  
## method from  
## print.SpatialPixelsDataFrame sp   
## print.SpatialPixels sp

library(usdm)  
library(mgcv)

## Loading required package: nlme

##   
## Attaching package: 'nlme'

## The following object is masked from 'package:usdm':  
##   
## Variogram

## The following object is masked from 'package:dplyr':  
##   
## collapse

## The following object is masked from 'package:raster':  
##   
## getData

## This is mgcv 1.8-38. For overview type 'help("mgcv-package")'.

# Genus and Species strings

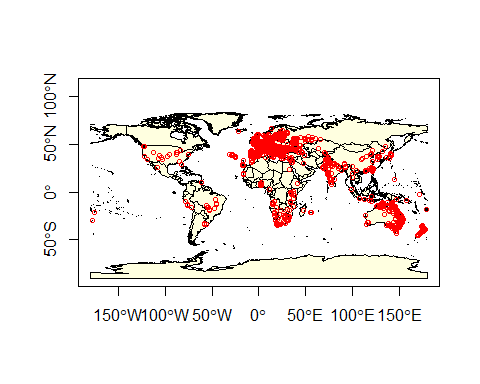
Their are many places in this code where you will need to save files with filenames including the genus and species. We’ll save these as strings to automate the creation of file names. Enter your genus name and specific epithet in the quotes below.

genus<-"Helicoverpa"  
species<-"armigera"

## 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 = "Helicoverpa armigera.csv")  
sdmdata <- subset(sdmdata, !is.na(lon) & !is.na(lat))  
dups <-duplicated(sdmdata[ c('lon', 'lat')])  
sdmdata<-sdmdata[!dups,]  
#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 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 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 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. 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,]

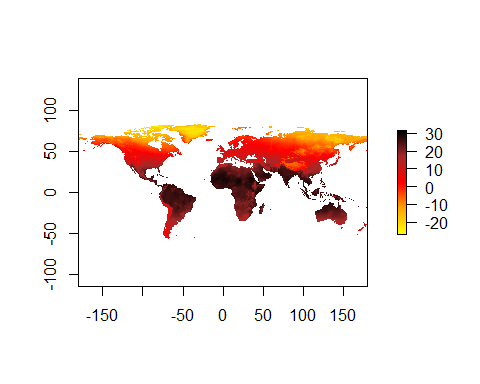
Save pres\_data and test\_data as csv files just in case.

#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('Climond\_elev\_HI.tif')  
bands<-c("Ann\_mean\_temp", "Mean\_durnal\_temp\_range", "Isothermality", "Temp\_Seasonality", "MaxTemp\_WarmestWeek", "MinTemp\_ColdestWeek", "Temp\_Annual\_Range", "Mean\_temp\_wettest\_q", "Mean\_temp\_driest\_q", "Mean\_temp\_warmiest\_q", "Mean\_temp\_coldest\_q", "Ann\_ precip", "Precip\_driest\_week", "Precip\_wettest\_week", "Precip\_Seasonality", "Precip\_wettest\_q", "Precip\_driest\_q", "Precip\_warmest\_q", "Precip\_coldest\_q", "Ann\_mean\_rad", "Hghest\_weekly\_rad", "Lowest\_weekly\_rad", "Rad\_seasonality", "Rad\_wettest\_q", "Rad\_driest\_q", "Rad\_warmest\_q", "Rad\_coldest\_q", "Ann\_mean\_moisture", "Highsets\_weekly\_moisture", "Lowest\_weekly\_moisture", "Moisture\_seasonality", "Mean\_moisture\_wettest\_q", "Mean\_moisture\_driest\_q", "Mean\_moisture\_warmest\_q", "Mean\_moisture\_coldest\_q", "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 collinier 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\_durnal\_temp\_range" "Temp\_Seasonality" "Ann\_.precip"   
## [4] "Precip\_driest\_week" "Precip\_wettest\_week" "Precip\_Seasonality"   
## [7] "Precip\_warmest\_q" "Precip\_coldest\_q" "Hghest\_weekly\_rad"   
## [10] "Rad\_seasonality" "Rad\_wettest\_q" "Rad\_driest\_q"   
## [13] "Moisture\_seasonality" "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). spDists creates a matrix of distances between points. This includes zeros.   
#I have to use a smaller distance in this case becacuse of the raster geometry  
dist<-110000  
x <- circles(pres\_train\_SPDF, d=110000, lonlat=TRUE)  
#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  
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)  
colnames(train\_data)

## [1] "Mean\_durnal\_temp\_range" "Temp\_Seasonality" "Ann\_.precip"   
## [4] "Precip\_driest\_week" "Precip\_wettest\_week" "Precip\_Seasonality"   
## [7] "Precip\_warmest\_q" "Precip\_coldest\_q" "Hghest\_weekly\_rad"   
## [10] "Rad\_seasonality" "Rad\_wettest\_q" "Rad\_driest\_q"   
## [13] "Moisture\_seasonality" "Elev" "human\_impact"   
## [16] "pa"

# Training the GAM and making predictions.

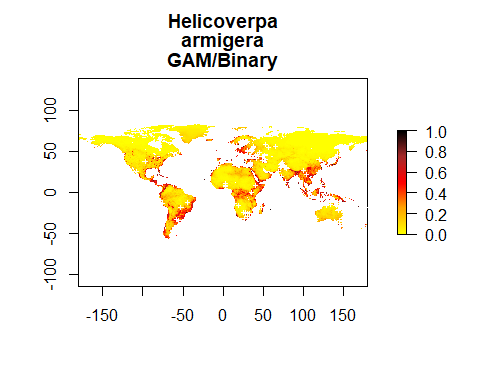
**This is a pain in the neck because all of the layers have to be specified. I recommend printing the column names in the console colnames(train\_data) and then copying them and formatting them**

gam <- gam(pa ~ Mean\_durnal\_temp\_range + Temp\_Seasonality + Ann\_.precip + Precip\_driest\_week + Precip\_wettest\_week + Precip\_Seasonality + Precip\_warmest\_q + Precip\_coldest\_q +   
Hghest\_weekly\_rad + Rad\_seasonality + Rad\_wettest\_q + Rad\_driest\_q + Moisture\_seasonality + Elev + human\_impact,  
 family = binomial(link = "logit"), data=train\_data)  
summary(gam)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## pa ~ Mean\_durnal\_temp\_range + Temp\_Seasonality + Ann\_.precip +   
## Precip\_driest\_week + Precip\_wettest\_week + Precip\_Seasonality +   
## Precip\_warmest\_q + Precip\_coldest\_q + Hghest\_weekly\_rad +   
## Rad\_seasonality + Rad\_wettest\_q + Rad\_driest\_q + Moisture\_seasonality +   
## Elev + human\_impact  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -4.168e-01 5.762e-01 -0.723 0.46945   
## Mean\_durnal\_temp\_range -2.482e-01 2.272e-02 -10.927 < 2e-16 \*\*\*  
## Temp\_Seasonality -9.113e+01 8.064e+00 -11.300 < 2e-16 \*\*\*  
## Ann\_.precip -8.673e-04 4.150e-04 -2.090 0.03661 \*   
## Precip\_driest\_week 2.181e-03 5.544e-03 0.393 0.69396   
## Precip\_wettest\_week 3.196e-03 1.437e-02 0.222 0.82398   
## Precip\_Seasonality 2.151e-01 2.376e-01 0.905 0.36527   
## Precip\_warmest\_q 2.339e-03 4.744e-04 4.931 8.18e-07 \*\*\*  
## Precip\_coldest\_q -1.106e-03 6.873e-04 -1.609 0.10753   
## Hghest\_weekly\_rad 2.128e-02 1.960e-03 10.856 < 2e-16 \*\*\*  
## Rad\_seasonality -9.362e-01 4.836e-01 -1.936 0.05287 .   
## Rad\_wettest\_q -4.203e-03 9.624e-04 -4.367 1.26e-05 \*\*\*  
## Rad\_driest\_q -1.618e-03 1.213e-03 -1.334 0.18223   
## Moisture\_seasonality -3.706e+00 3.935e-01 -9.419 < 2e-16 \*\*\*  
## Elev 1.683e-04 5.819e-05 2.893 0.00382 \*\*   
## human\_impact 8.714e-02 4.295e-03 20.290 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
##   
## R-sq.(adj) = 0.317 Deviance explained = 25.5%  
## UBRE = 0.03875 Scale est. = 1 n = 5206

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.

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='/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/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.

maxent()

## Loading required namespace: rJava

## java.home option:

## JAVA\_HOME environment variable: /Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home

## Warning in fun(libname, pkgname): Java home setting is INVALID, it will be ignored.  
## Please do NOT set it unless you want to override system settings.

## This is MaxEnt version 3.4.3

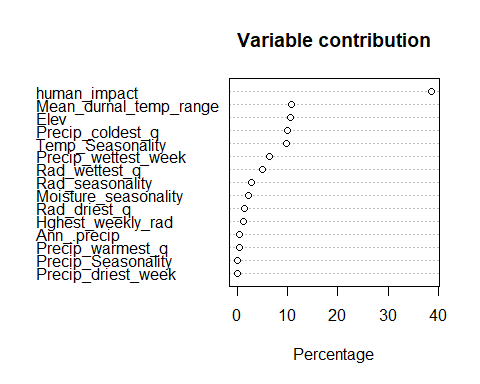
me\_model <- maxent(predictors, pres\_train, a=background\_train)

## Warning in .local(x, p, ...): 170 (9.91%) of the presence points have NA  
## predictor values

## Warning in .local(x, p, ...): 2554 (25.54%) of the presence points have NA  
## predictor values

## This is MaxEnt version 3.4.3

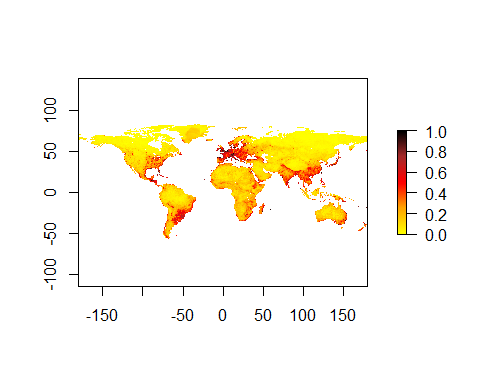
#and plot the models most important layers  
par(mfrow=c(1,1))  
plot(me\_model)

 Let’s go ahead and make some predictions

MEpreds<-predict(predictors, me\_model, type='response')

## This is MaxEnt version 3.4.3

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

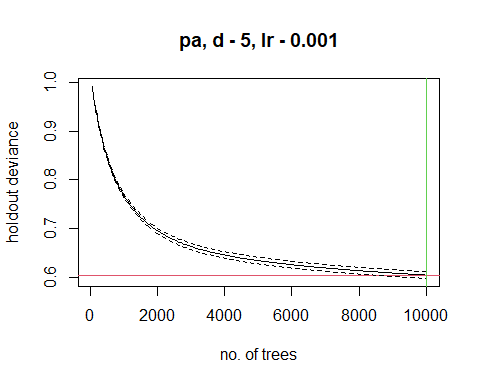
#let's get the data from our predictors  
bg\_train\_data<-extract(predictors,background\_train)  
#and bind a column of 0 to the end of it  
bg\_train\_data<-cbind(bg\_train\_data,0)  
#and convert it to a data frame  
bg\_train\_data<-as.data.frame(bg\_train\_data)  
#and then combine it withe the 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")

And now we can train a model using the first X columns of train\_data and the 6th as the response. Let’s start with tree complexity of 5 and learning rate of 0.001

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)

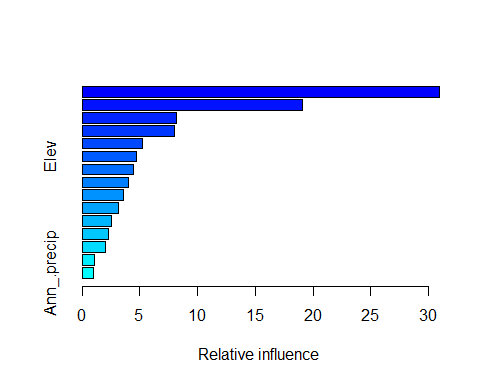
##   
##   
## GBM STEP - version 2.9   
##   
## Performing cross-validation optimisation of a boosted regression tree model   
## for pa and using a family of bernoulli   
## Using 12603 observations and 15 predictors   
## creating 10 initial models of 50 trees   
##   
## folds are stratified by prevalence   
## total mean deviance = 1.0187   
## tolerance is fixed at 0.001   
## ntrees resid. dev.   
## 50 0.992   
## now adding trees...   
## 100 0.9686   
## 150 0.9478   
## 200 0.9291   
## 250 0.9122   
## 300 0.8968   
## 350 0.8826   
## 400 0.8696   
## 450 0.8576   
## 500 0.8465   
## 550 0.836   
## 600 0.8264   
## 650 0.8174   
## 700 0.8089   
## 750 0.801   
## 800 0.7935   
## 850 0.7866   
## 900 0.7801   
## 950 0.7739   
## 1000 0.7681   
## 1050 0.7626   
## 1100 0.7573   
## 1150 0.7524   
## 1200 0.7476   
## 1250 0.7431   
## 1300 0.7389   
## 1350 0.7348   
## 1400 0.7309   
## 1450 0.7272   
## 1500 0.7236   
## 1550 0.7201   
## 1600 0.7169   
## 1650 0.7138   
## 1700 0.7108   
## 1750 0.708   
## 1800 0.7052   
## 1850 0.7026   
## 1900 0.7   
## 1950 0.6976   
## 2000 0.6953   
## 2050 0.6931   
## 2100 0.6909   
## 2150 0.6888   
## 2200 0.6869   
## 2250 0.685   
## 2300 0.6831   
## 2350 0.6814   
## 2400 0.6797   
## 2450 0.678   
## 2500 0.6765   
## 2550 0.6749   
## 2600 0.6734   
## 2650 0.6719   
## 2700 0.6706   
## 2750 0.6693   
## 2800 0.6679   
## 2850 0.6667   
## 2900 0.6654   
## 2950 0.6642   
## 3000 0.6631   
## 3050 0.6619   
## 3100 0.6609   
## 3150 0.6598   
## 3200 0.6587   
## 3250 0.6577   
## 3300 0.6568   
## 3350 0.6558   
## 3400 0.6549   
## 3450 0.654   
## 3500 0.6531   
## 3550 0.6523   
## 3600 0.6515   
## 3650 0.6507   
## 3700 0.6498   
## 3750 0.6491   
## 3800 0.6483   
## 3850 0.6476   
## 3900 0.6469   
## 3950 0.6462   
## 4000 0.6455   
## 4050 0.6448   
## 4100 0.6442   
## 4150 0.6435   
## 4200 0.6428   
## 4250 0.6422   
## 4300 0.6416   
## 4350 0.641   
## 4400 0.6404   
## 4450 0.6398   
## 4500 0.6392   
## 4550 0.6386   
## 4600 0.6381   
## 4650 0.6375   
## 4700 0.6369   
## 4750 0.6364   
## 4800 0.6359   
## 4850 0.6354   
## 4900 0.6349   
## 4950 0.6344   
## 5000 0.6339   
## 5050 0.6335   
## 5100 0.633   
## 5150 0.6325   
## 5200 0.6321   
## 5250 0.6316   
## 5300 0.6311   
## 5350 0.6307   
## 5400 0.6303   
## 5450 0.6299   
## 5500 0.6294   
## 5550 0.629   
## 5600 0.6286   
## 5650 0.6282   
## 5700 0.6278   
## 5750 0.6274   
## 5800 0.627   
## 5850 0.6266   
## 5900 0.6262   
## 5950 0.6259   
## 6000 0.6255   
## 6050 0.6251   
## 6100 0.6248   
## 6150 0.6244   
## 6200 0.624   
## 6250 0.6237   
## 6300 0.6233   
## 6350 0.623   
## 6400 0.6227   
## 6450 0.6223   
## 6500 0.622   
## 6550 0.6217   
## 6600 0.6213   
## 6650 0.621   
## 6700 0.6207   
## 6750 0.6204   
## 6800 0.6201   
## 6850 0.6198   
## 6900 0.6194   
## 6950 0.6191   
## 7000 0.6188   
## 7050 0.6185   
## 7100 0.6182   
## 7150 0.6179   
## 7200 0.6176   
## 7250 0.6173   
## 7300 0.617   
## 7350 0.6167   
## 7400 0.6164   
## 7450 0.6162   
## 7500 0.6159   
## 7550 0.6156   
## 7600 0.6153   
## 7650 0.615   
## 7700 0.6148   
## 7750 0.6145   
## 7800 0.6143   
## 7850 0.614   
## 7900 0.6137   
## 7950 0.6135   
## 8000 0.6132   
## 8050 0.613   
## 8100 0.6127   
## 8150 0.6125   
## 8200 0.6122   
## 8250 0.612   
## 8300 0.6117   
## 8350 0.6115   
## 8400 0.6113   
## 8450 0.611   
## 8500 0.6108   
## 8550 0.6105   
## 8600 0.6103   
## 8650 0.61   
## 8700 0.6098   
## 8750 0.6096   
## 8800 0.6093   
## 8850 0.6091   
## 8900 0.6089   
## 8950 0.6086   
## 9000 0.6084   
## 9050 0.6082   
## 9100 0.6079   
## 9150 0.6077   
## 9200 0.6075   
## 9250 0.6073   
## 9300 0.6071   
## 9350 0.6069   
## 9400 0.6067   
## 9450 0.6064   
## 9500 0.6062   
## 9550 0.606   
## 9600 0.6058   
## 9650 0.6056   
## 9700 0.6054   
## 9750 0.6052   
## 9800 0.605   
## 9850 0.6048   
## 9900 0.6046   
## 9950 0.6044   
## 10000 0.6042

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



##   
## mean total deviance = 1.019   
## mean residual deviance = 0.564   
##   
## estimated cv deviance = 0.604 ; se = 0.007   
##   
## training data correlation = 0.697   
## cv correlation = 0.66 ; se = 0.007   
##   
## training data AUC score = 0.914   
## cv AUC score = 0.896 ; se = 0.003   
##   
## elapsed time - 0.54 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  
## Mean\_durnal\_temp\_range Mean\_durnal\_temp\_range 30.9674246  
## human\_impact human\_impact 19.0662590  
## Temp\_Seasonality Temp\_Seasonality 8.1196781  
## Rad\_seasonality Rad\_seasonality 7.9945035  
## Precip\_wettest\_week Precip\_wettest\_week 5.1955405  
## Elev Elev 4.7021121  
## Moisture\_seasonality Moisture\_seasonality 4.4008046  
## Precip\_coldest\_q Precip\_coldest\_q 4.0177919  
## Rad\_wettest\_q Rad\_wettest\_q 3.5706682  
## Hghest\_weekly\_rad Hghest\_weekly\_rad 3.1029105  
## Precip\_warmest\_q Precip\_warmest\_q 2.5061938  
## Rad\_driest\_q Rad\_driest\_q 2.2791286  
## Precip\_Seasonality Precip\_Seasonality 2.0191062  
## Precip\_driest\_week Precip\_driest\_week 1.0978267  
## Ann\_.precip Ann\_.precip 0.9600518

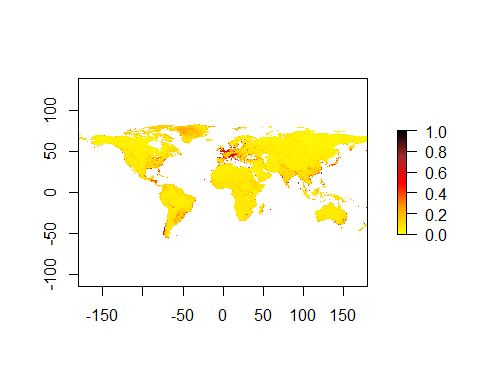
**Note: you may want to try different combinations! If your trees are converging too slowly, raise the tree complexity by 1 or two, and back the learning rate down. On the other hand of your holdout deviance drops very quickly and slowly starts to rise, you are overfitting. Drop the tree complexity and 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)  
#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: 0.446455505279035"

print(paste('TSS:', TSS\_GAM[[2]]))

## [1] "TSS: 0.446455505279035"

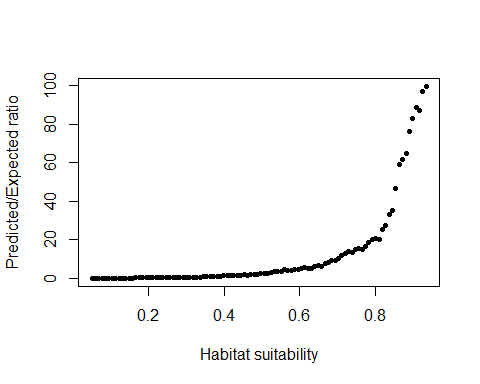
e

## class : ModelEvaluation   
## n presences : 663   
## n absences : 663   
## AUC : 0.7866649   
## cor : 0.490907   
## max TPR+TNR at : 0.5948367

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] 0.03189177 0.05358138 0.05633832 0.08403966 0.10449602 0.12075713  
## [7] 0.13294024 0.14513901 0.14402999 0.16421413 0.19476875 0.23910427  
## [13] 0.23765603 0.27564132 0.26158007 0.29103077 0.28440111 0.38693233  
## [19] 0.42823880 0.40768170 0.46074075 0.43065537 0.46768546 0.43012786  
## [25] 0.44831160 0.48154651 0.49839280 0.57892254 0.63891368 0.58722826  
## [31] 0.62416738 0.61466668 0.70254272 0.79146770 0.80773544 0.97761297  
## [37] 1.16265640 1.21803636 1.19760894 1.28603773 1.34895554 1.48024294  
## [43] 1.59557275 1.61382845 1.59947052 1.89984648 1.78617978 1.81310218  
## [49] 1.91577614 2.14757612 2.38572874 2.48335514 2.58519442 3.10834141  
## [55] 3.37688668 3.53577598 3.57058114 4.41018234 4.36281568 4.36065004  
## [61] 4.64994987 4.83532014 5.05457890 5.45152631 5.06704323 5.36613270  
## [67] 5.95212146 6.75798283 6.22268480 7.99939066 8.53886464 9.30751933  
## [73] 9.44622628 10.45901937 12.08475775 12.84562230 13.77164859 13.62753337  
## [79] 15.15883150 15.54500713 15.26382586 16.82709497 18.74006504 20.29040269  
## [85] 20.91886763 20.26560025 25.25895973 27.39572044 32.92954964 35.23967491  
## [91] 46.81607735 59.14971692 61.62705980 65.06183418 76.23875009 82.90436123  
## [97] 88.75615630 87.01208299 97.14629776 99.63800798  
##   
## $Spearman.cor  
## [1] 0.999  
##   
## $HS  
## [1] 0.04984290 0.05878936 0.06773582 0.07668228 0.08562874 0.09457520  
## [7] 0.10352166 0.11246812 0.12141459 0.13036105 0.13930751 0.14825397  
## [13] 0.15720043 0.16614689 0.17509335 0.18403981 0.19298627 0.20193273  
## [19] 0.21087919 0.21982565 0.22877211 0.23771857 0.24666504 0.25561150  
## [25] 0.26455796 0.27350442 0.28245088 0.29139734 0.30034380 0.30929026  
## [31] 0.31823672 0.32718318 0.33612964 0.34507610 0.35402256 0.36296902  
## [37] 0.37191548 0.38086195 0.38980841 0.39875487 0.40770133 0.41664779  
## [43] 0.42559425 0.43454071 0.44348717 0.45243363 0.46138009 0.47032655  
## [49] 0.47927301 0.48821947 0.49716593 0.50611240 0.51505886 0.52400532  
## [55] 0.53295178 0.54189824 0.55084470 0.55979116 0.56873762 0.57768408  
## [61] 0.58663054 0.59557700 0.60452346 0.61346992 0.62241638 0.63136285  
## [67] 0.64030931 0.64925577 0.65820223 0.66714869 0.67609515 0.68504161  
## [73] 0.69398807 0.70293453 0.71188099 0.72082745 0.72977391 0.73872037  
## [79] 0.74766683 0.75661330 0.76555976 0.77450622 0.78345268 0.79239914  
## [85] 0.80134560 0.81029206 0.81923852 0.82818498 0.83713144 0.84607790  
## [91] 0.85502436 0.86397082 0.87291728 0.88186375 0.89081021 0.89975667  
## [97] 0.90870313 0.91764959 0.92659605 0.93554251

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: 0.475113122171946"

print(paste('TSS:', TSS\_ME[[2]]))

## [1] "TSS: 0.475113122171946"

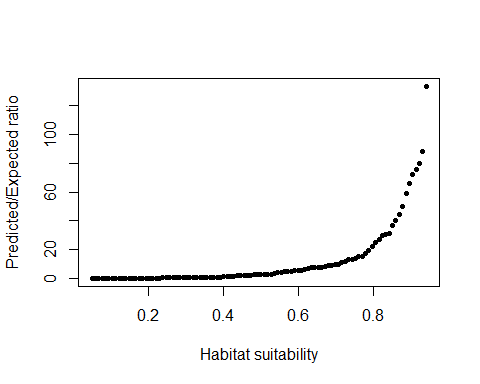
e

## class : ModelEvaluation   
## n presences : 663   
## n absences : 663   
## AUC : 0.8033312   
## cor : 0.5240787   
## max TPR+TNR at : 0.5813309

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] 2.762878e-03 3.190377e-03 1.066368e-02 1.530857e-02 2.030250e-02  
## [6] 2.997807e-02 3.180379e-02 3.846200e-02 5.630845e-02 7.094067e-02  
## [11] 8.127752e-02 9.862417e-02 1.246765e-01 1.189637e-01 1.416781e-01  
## [16] 1.518770e-01 1.732051e-01 1.953247e-01 2.081066e-01 2.102311e-01  
## [21] 2.558367e-01 3.170003e-01 3.713692e-01 3.518313e-01 3.975798e-01  
## [26] 4.318007e-01 5.438569e-01 5.694088e-01 5.946879e-01 6.358068e-01  
## [31] 6.440949e-01 6.358715e-01 5.841884e-01 5.689174e-01 6.382077e-01  
## [36] 8.424717e-01 9.069990e-01 8.506801e-01 9.142605e-01 9.862229e-01  
## [41] 1.206602e+00 1.421577e+00 1.592842e+00 1.680315e+00 1.907993e+00  
## [46] 1.928019e+00 1.996697e+00 2.217578e+00 2.389639e+00 2.508732e+00  
## [51] 2.633199e+00 2.715535e+00 2.892290e+00 3.027797e+00 3.372326e+00  
## [56] 4.011904e+00 4.388167e+00 4.700586e+00 5.117426e+00 4.914866e+00  
## [61] 5.306844e+00 5.648435e+00 5.779863e+00 5.974520e+00 6.558479e+00  
## [66] 7.385113e+00 7.492037e+00 7.620892e+00 7.411558e+00 7.956955e+00  
## [71] 8.652983e+00 9.295154e+00 9.485284e+00 9.956497e+00 1.070567e+01  
## [76] 1.205866e+01 1.318481e+01 1.345950e+01 1.372048e+01 1.542025e+01  
## [81] 1.547263e+01 1.756881e+01 1.924250e+01 2.232812e+01 2.499017e+01  
## [86] 2.676432e+01 2.997272e+01 3.038074e+01 3.110251e+01 3.642524e+01  
## [91] 4.014793e+01 4.421578e+01 4.993801e+01 5.899695e+01 6.596992e+01  
## [96] 7.183601e+01 7.570778e+01 8.008650e+01 8.835594e+01 1.333420e+02  
##   
## $Spearman.cor  
## [1] 0.999  
##   
## $HS  
## [1] 0.05089383 0.05988349 0.06887315 0.07786281 0.08685246 0.09584212  
## [7] 0.10483178 0.11382144 0.12281109 0.13180075 0.14079041 0.14978006  
## [13] 0.15876972 0.16775938 0.17674904 0.18573869 0.19472835 0.20371801  
## [19] 0.21270767 0.22169732 0.23068698 0.23967664 0.24866630 0.25765595  
## [25] 0.26664561 0.27563527 0.28462493 0.29361458 0.30260424 0.31159390  
## [31] 0.32058356 0.32957321 0.33856287 0.34755253 0.35654218 0.36553184  
## [37] 0.37452150 0.38351116 0.39250081 0.40149047 0.41048013 0.41946979  
## [43] 0.42845944 0.43744910 0.44643876 0.45542842 0.46441807 0.47340773  
## [49] 0.48239739 0.49138705 0.50037670 0.50936636 0.51835602 0.52734567  
## [55] 0.53633533 0.54532499 0.55431465 0.56330430 0.57229396 0.58128362  
## [61] 0.59027328 0.59926293 0.60825259 0.61724225 0.62623191 0.63522156  
## [67] 0.64421122 0.65320088 0.66219054 0.67118019 0.68016985 0.68915951  
## [73] 0.69814917 0.70713882 0.71612848 0.72511814 0.73410779 0.74309745  
## [79] 0.75208711 0.76107677 0.77006642 0.77905608 0.78804574 0.79703540  
## [85] 0.80602505 0.81501471 0.82400437 0.83299403 0.84198368 0.85097334  
## [91] 0.85996300 0.86895266 0.87794231 0.88693197 0.89592163 0.90491128  
## [97] 0.91390094 0.92289060 0.93188026 0.94086991

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: 0.536953242835596"

print(paste('TSS:', TSS\_BRT[[2]]))

## [1] "TSS: 0.536953242835596"

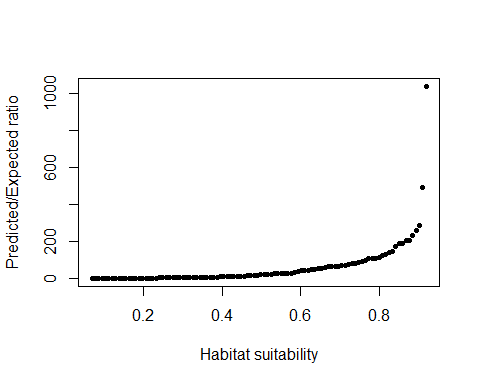
e

## class : ModelEvaluation   
## n presences : 663   
## n absences : 663   
## AUC : 0.8322596   
## cor : 0.5771344   
## max TPR+TNR at : 0.2371633

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

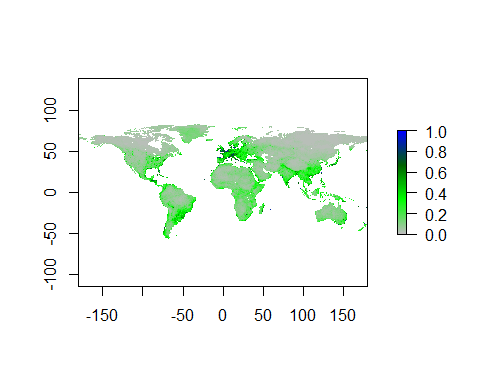


## $F.ratio  
## [1] 0.1029209 0.1179585 0.1450679 0.1854147 0.2661591  
## [6] 0.3508095 0.4226243 0.5274196 0.6215070 0.7430985  
## [11] 0.8010353 0.8652182 0.9691901 1.1218345 1.3016551  
## [16] 1.3552604 1.3640324 1.4967495 1.7545380 1.9323289  
## [21] 2.4082867 2.8034383 3.1438732 3.3872107 3.6457153  
## [26] 4.1684282 4.4624922 4.8430789 5.2207998 5.4031938  
## [31] 5.6151251 5.5303137 5.1470723 5.2849022 5.2413076  
## [36] 6.0893093 6.0545752 6.7040841 7.8704377 8.4337486  
## [41] 8.3929436 9.0811033 9.7291652 10.7945712 10.2254824  
## [46] 12.9549668 13.9877304 16.6397018 17.1544961 18.0967347  
## [51] 20.0860251 20.3411518 22.7298925 23.1036293 25.6503444  
## [56] 26.9438703 24.5231320 27.1849962 27.3527241 28.6685283  
## [61] 30.5807001 35.1676156 40.4066782 41.5646305 45.2952510  
## [66] 46.5031244 48.9716121 52.6879297 53.0141435 59.0443213  
## [71] 63.2737715 63.5448181 63.2985909 66.5891274 69.5785383  
## [76] 67.8396922 73.2945414 80.4078828 82.8875133 86.6864200  
## [81] 92.7531213 95.0816103 105.6381070 109.9825313 107.3012142  
## [86] 111.3600317 122.3216176 129.7916400 138.4835688 147.8106585  
## [91] 173.4070157 186.3758031 187.7368956 206.7326395 202.2030024  
## [96] 230.8059481 259.5679201 284.0012237 489.4078813 1036.9781524  
##   
## $Spearman.cor  
## [1] 0.999  
##   
## $HS  
## [1] 0.06859255 0.07719241 0.08579227 0.09439214 0.10299200 0.11159187  
## [7] 0.12019173 0.12879159 0.13739146 0.14599132 0.15459118 0.16319105  
## [13] 0.17179091 0.18039077 0.18899064 0.19759050 0.20619036 0.21479023  
## [19] 0.22339009 0.23198995 0.24058982 0.24918968 0.25778954 0.26638941  
## [25] 0.27498927 0.28358913 0.29218900 0.30078886 0.30938873 0.31798859  
## [31] 0.32658845 0.33518832 0.34378818 0.35238804 0.36098791 0.36958777  
## [37] 0.37818763 0.38678750 0.39538736 0.40398722 0.41258709 0.42118695  
## [43] 0.42978681 0.43838668 0.44698654 0.45558640 0.46418627 0.47278613  
## [49] 0.48138599 0.48998586 0.49858572 0.50718559 0.51578545 0.52438531  
## [55] 0.53298518 0.54158504 0.55018490 0.55878477 0.56738463 0.57598449  
## [61] 0.58458436 0.59318422 0.60178408 0.61038395 0.61898381 0.62758367  
## [67] 0.63618354 0.64478340 0.65338326 0.66198313 0.67058299 0.67918285  
## [73] 0.68778272 0.69638258 0.70498245 0.71358231 0.72218217 0.73078204  
## [79] 0.73938190 0.74798176 0.75658163 0.76518149 0.77378135 0.78238122  
## [85] 0.79098108 0.79958094 0.80818081 0.81678067 0.82538053 0.83398040  
## [91] 0.84258026 0.85118012 0.85977999 0.86837985 0.87697971 0.88557958  
## [97] 0.89417944 0.90277931 0.91137917 0.91997903

# 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=cool(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: 0.503770739064857"

print(paste('TSS:', TSS\_ENS[[2]]))

## [1] "TSS: 0.503770739064857"

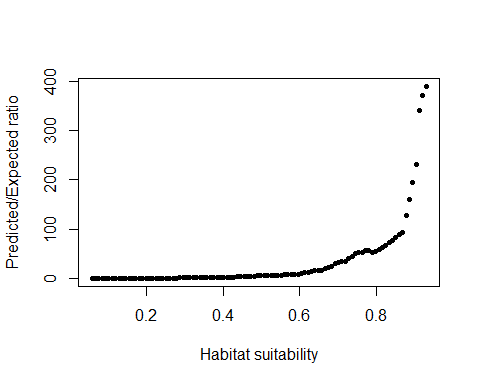
e

## class : ModelEvaluation   
## n presences : 663   
## n absences : 663   
## AUC : 0.8149744   
## cor : 0.5515558   
## max TPR+TNR at : 0.523557

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] 0.01830330 0.02219860 0.03620331 0.05771891 0.06598004  
## [6] 0.07645879 0.09035108 0.10934669 0.13603581 0.16605564  
## [11] 0.17023356 0.20148581 0.23819932 0.26473762 0.26288847  
## [16] 0.29440270 0.33863781 0.42018159 0.46169051 0.53230894  
## [21] 0.51090757 0.55852351 0.62370070 0.66040287 0.71509677  
## [26] 0.78796538 0.85016422 0.99666341 1.01001699 1.04245265  
## [31] 1.12705242 1.37121318 1.50312808 1.53313937 1.56544641  
## [36] 1.72547628 1.83501373 2.10308349 2.21020816 2.36043233  
## [41] 2.62693181 2.70895063 2.82443939 2.97783305 3.42402553  
## [46] 3.84220910 4.04813139 4.54633605 4.77160835 5.00091416  
## [51] 5.29652666 5.55326608 5.79072850 6.22151340 6.25247313  
## [56] 6.28537112 6.80660973 8.14836608 8.30852539 8.58514244  
## [61] 8.60831587 8.74813626 9.83468733 11.50329233 12.49526049  
## [66] 13.87517445 15.73621556 16.96970819 16.97102559 20.54541061  
## [71] 21.74343188 24.95364738 30.34000864 32.54422131 34.36600369  
## [76] 35.03304569 40.40851865 43.89406434 49.94489368 51.54352792  
## [81] 52.60346553 55.65859733 56.79365997 53.13068541 55.51588906  
## [86] 57.68284357 63.37249227 66.93791380 72.05199657 77.11809971  
## [91] 82.07036415 88.09341863 93.53008477 127.39289342 160.38486319  
## [96] 195.35727082 231.25194991 339.98282161 371.85402746 389.41996052  
##   
## $Spearman.cor  
## [1] 1  
##   
## $HS  
## [1] 0.05686533 0.06569656 0.07452778 0.08335900 0.09219022 0.10102145  
## [7] 0.10985267 0.11868389 0.12751511 0.13634634 0.14517756 0.15400878  
## [13] 0.16284000 0.17167123 0.18050245 0.18933367 0.19816490 0.20699612  
## [19] 0.21582734 0.22465856 0.23348979 0.24232101 0.25115223 0.25998345  
## [25] 0.26881468 0.27764590 0.28647712 0.29530834 0.30413957 0.31297079  
## [31] 0.32180201 0.33063323 0.33946446 0.34829568 0.35712690 0.36595812  
## [37] 0.37478935 0.38362057 0.39245179 0.40128301 0.41011424 0.41894546  
## [43] 0.42777668 0.43660791 0.44543913 0.45427035 0.46310157 0.47193280  
## [49] 0.48076402 0.48959524 0.49842646 0.50725769 0.51608891 0.52492013  
## [55] 0.53375135 0.54258258 0.55141380 0.56024502 0.56907624 0.57790747  
## [61] 0.58673869 0.59556991 0.60440113 0.61323236 0.62206358 0.63089480  
## [67] 0.63972603 0.64855725 0.65738847 0.66621969 0.67505092 0.68388214  
## [73] 0.69271336 0.70154458 0.71037581 0.71920703 0.72803825 0.73686947  
## [79] 0.74570070 0.75453192 0.76336314 0.77219436 0.78102559 0.78985681  
## [85] 0.79868803 0.80751925 0.81635048 0.82518170 0.83401292 0.84284414  
## [91] 0.85167537 0.86050659 0.86933781 0.87816904 0.88700026 0.89583148  
## [97] 0.90466270 0.91349393 0.92232515 0.93115637

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'))