Exercise 14 Other Algorithms

Adam B. Smith, Danielle Svehla, & Camilo Sanín

July 6, 2018

Thus far we have used Maxent exclusively. Maxent is the most popular SDM/ENM algorithm, but it is only one of about 30 that are found in the ecological literature. As we have mentioned, the fundamental difference between algorithms is the "shape" they use to identify areas of suitable habitat. In this exercise you will:

1. Learn how to apply several **SDM/ENM algorithms** other than Maxent.
2. Learn the difference between "1-class" and "2-class" model algorithms.

# 1-class algorithms

Maxent is a 2-class algorithm in the sense that it requires presences and background sites. As you have seen, the selection of background sites can be tricky. Another genre of algorithms uses *only* presence sites. These so-called "1-class" models were the first kind of SDMs. Although they are conceptually easier to use, they often perform poorly compared to flexible 2-class models. And because they don't use background sites they offer fewer ways to address bias in presence sites.

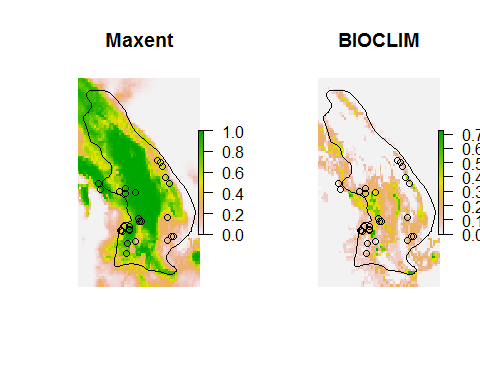
## BIOCLIM

BIOCLIM was the first SDM/ENM algorithm, and it shares its name with the 19 [BIOCLIM](http://www.worldclim.org/bioclim) variables. The first version of BIOCLIM simply demarcated "suitable" from "unsuitable" (i.e., it did not assign relative suitability scores). For a set of presences it found the minimum and maximum along each value along environmental axis *i*. Let's call this range *Ei*. If a site was within *all* *Ei*, then it was suitable. If it was outside *Ei* for *any* axis *i*, then it was not (sometimes the 95th percentile was used instead of the minimum/maximum).

Robert Hijman's version (in the [dismo](https://cran.r-project.org/web/packages/dismo/index.html) R package we've been using) assigns relative suitability scores between 0 and 1 by assuming the median of the distribution along an environmental axis is the most suitable (a score of 1) and sites at the very limits of the distribution are barely suitable (a score of 0). Everything else between these extremes is scaled approximately linearly. The suitability for a site is thus the minimum value of these scores across all environmental axes.

Let's train a BIOCLIM model and compare it to a Maxent model.

# training data is just environment at presences  
envData <- records[ , predictors]  
  
# train BIOCLIM model  
bioclimModel <- bioclim(envData)  
  
# create raster map  
bioclimMap <- predict(bioclimModel, climate[[predictors]])  
  
# save results  
dirCreate('./Models/Model 12 Other Algorithms')  
save(bioclimModel, file='./Models/Model 12 Other Algorithms/Model - BIOCLIM.Rdata', compress=TRUE)  
writeRaster(bioclimMap, './Models/Model 12 Other Algorithms/bioclimMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='BIOCLIM')  
plot(bioclimMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



You can see the BIOCLIM prediction is much less generous than Maxent. This occurs because BIOCLIM assumes if a site falls outside the occupied environmental range of *any* variable it is unsuitable.

BIOCLIM assumes:

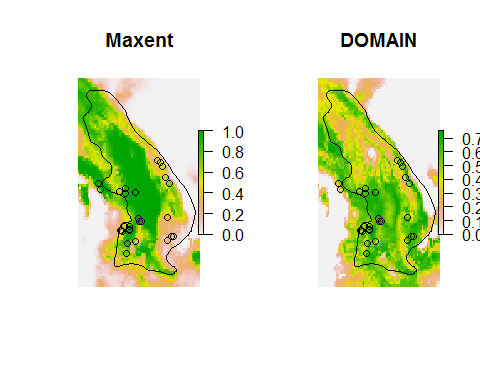
* The environmental variables affect the species completely independently of one another.
* Presences reflect the outermost extremes of the environment the species can occupy. Other algorithms make this assumption, too, but are more flexible and so can project suitable habitat outside the currently occupied suitable habitat.

The effect is to assume a species occupies a "hyperrectangle" in environmental space (see the first figure in this exercise).

## DOMAIN

Unlike BIOCLIM DOMAIN accounts for interactions between variables and attempts to fit a hyper-ellipsoid to the presence sites in environmental space.

domainModel <- domain(envData)  
  
# create raster map  
domainMap <- predict(domainModel, climate[[predictors]])  
  
# save results  
save(bioclimModel, file='./Models/Model 12 Other Algorithms/Model - DOMAIN.Rdata', compress=TRUE)  
writeRaster(bioclimMap, './Models/Model 12 Other Algorithms/domainMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='DOMAIN')  
plot(domainMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



DOMAIN assumes

* The species' response to the environment is stationary around an optimum. For example, if the species' optimum temperature is 20C, then suitability will decline at the same rate as one goes from 20C to 10C or from 20C to 30C.
* Presences reflect the outermost extremes of the environment the species can occupy.

The dismo package contains other 1-class algorithms, too.

# 2-class algorithms

2-class algorithms use either presences and background sites (all algorithms) *or* presences and absences (all 2-class algorithms except Maxent). It is commonly believed that if you have presences and reliable and meaningful absences then a presence/absence algorithm will give you a better result than a 2-class model trained with the same presences and background sites.

Note that in many 2-class algorithms each presence and absence/background can have a different weight. This is one way of addressing sampling bias (downweight overly-sampled presences or upweight their background sites). Regardless, it is helpful to assign weights such that the total weight of presences equals the total weight of absences/background sites. This makes the output between different algorithms (with the same background/absence sites and weighting) comparable. Also, we've found some algorithms perform weirdly when weights >1 are assigned, so it helps to rescale them so the maximum is 1. In the examples below the functions often automatically weight sites so presences and background sites/absences have equal weight. So we won't have to do it explicitly.

Note that weights is a function in R, so avoid using weights as a variable when you work with weighting.

## Generalized Linear Models (GLMs, also known as logistic regresion)

GLMs are commonly employed as SDMs/ENMs, though they tend to perform poorly compared to more complex algorithms explored below (and Maxent). Nonetheless, they are easy to understand because they are based on linear regression.

Here we'll use the trainGlm() function to explore a suite of models and select the best based on AICc. This function is also capable of using Firth logistic regression which can usually accommodate cases where an environmental variable completely separates presences and non-presence sites (background sites or absences). This is problematic for normal GLMs because numerically they require some "mixing" between presences and absences. If complete separation does not occur then Firth's correction does not make a differnce (it's the same as a normal GLM). You can tell if you have complete separation if you get the error glm.fit: fitted probabilities numerically 0 or 1 occurred. We'll use Firth's correction (implemented in the [brglm](https://cran.r-project.org/web/packages/brglm/index.html) package). You may need to install it first:

install.packages('brglm')

library(brglm)

## Loading required package: profileModel

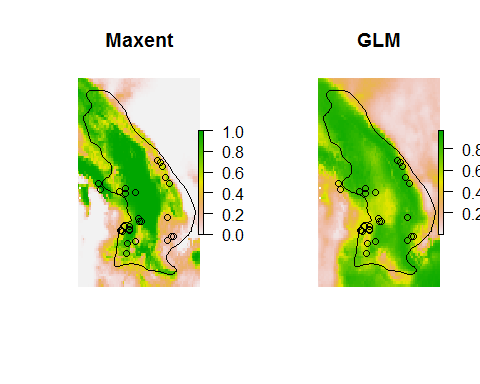
## 'brglm' will gradually be superseded by 'brglm2' (https://cran.r-project.org/package=brglm2), which provides utilities for mean and median bias reduction for all GLMs and methods for the detection of infinite estimates in binomial-response models.

envData <- rbind(records[ , predictors], targetBg[ , predictors])  
  
presBg <- c(rep(1, nrow(records)), rep(0, nrow(targetBg)))  
trainData <- cbind(presBg, envData)

The trainGlm() function first constructs a "full" model by calculating the AICc for small models made from just 1, 2, or 3 terms (e.g., x + x2 + x3). It then enters the terms of the small models into the full model in order of their AICc scores such that there are (by default) >=10 presences per term. It then evaluates all possible submodels of the full model and returns the one with the lowest AICc and with (by default) >=20 presences per term. See ?trainGlm for more information.

# produces warnings that are probably related to non-integer weighting  
glmModel <- trainGlm(  
 data=trainData,  
 resp='presBg',  
 family='binomial',  
 use='brglm',  
 quadratic=TRUE,  
 cubic=TRUE,  
 interaction=TRUE,  
 interQuad=TRUE,  
 w=TRUE,  
 out='model',  
 verbose=TRUE  
)

# create raster map  
glmMap <- predict(climate[[predictors]], glmModel, type='response')  
  
# save results  
save(glmModel, file='./Models/Model 12 Other Algorithms/Model - GLM.Rdata', compress=TRUE)  
writeRaster(glmMap, './Models/Model 12 Other Algorithms/glmMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='GLM')  
plot(glmMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



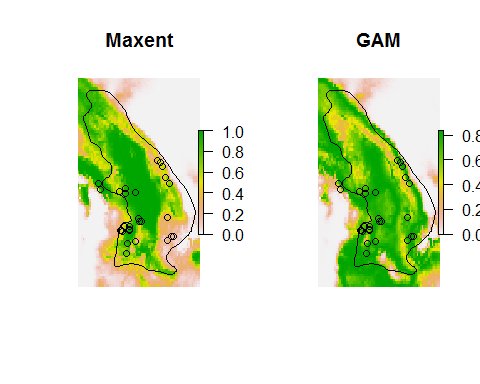
## Generalized Additive Models (GAMs)

GAMs are an extension of GLMs but much more flexible. They tend to perform as well as Maxent, boosted regression trees, and other "complex" models.

The trainGam() function works in a manner very similar to the trainGlm() function--first by evaluating small models, adding them to a "full" model, then selecting the best subset of that.

gamModel <- trainGam(  
 data=trainData,  
 resp='presBg',  
 family='binomial',  
 w=TRUE,  
 out='model',  
 verbose=TRUE  
)

# create raster map  
gamMap <- predict(climate[[predictors]], gamModel, type='response')  
  
# save results  
save(gamModel, file='./Models/Model 12 Other Algorithms/Model - GAM.Rdata', compress=TRUE)  
writeRaster(gamMap, './Models/Model 12 Other Algorithms/gamMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='GAM')  
plot(gamMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



## Boosted Regression Trees (BRTs)

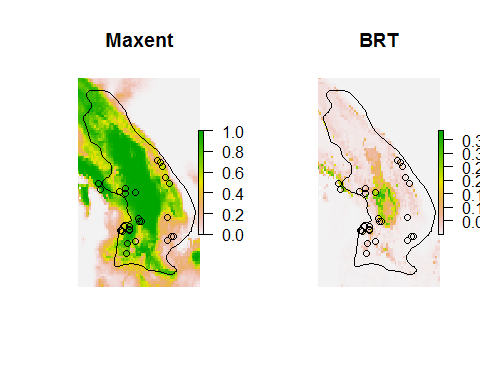
BRTs are constructed from an ensemble of classification/decision trees (CARTs) chained together serially. The first tree is trained on a randomly selected portion of the data (set by the "bag" fraction). Subsequent trees are trained on the residuals of the preceding trees, again using a random portion of the data. Bagging helps address over-fitting.

BRTs are parameterized by the bag fraction (usually between 0.5 and 0.7), maximum number of trees (larger if using a smaller learning rate), the learning rate (the degree to which each successive tree contributes to the final model), and tree complexity (levels in each tree). The trainBrt() function cycles through all combinations of these parameters. It can take a long time to do this... so we'll just examine two levels of maximum number of trees.

Note that because of the randomness involved in choosing bags in each step models trained on the same data will be slightly different. Also note that you may need to install and load the gbm package before trainBrt() will run (use install.packages(‘gbm’) then library(gbm)).

brtModel <- trainBrt(  
 data=trainData,  
 resp='presBg',  
 family='bernoulli',  
 lr=0.01,  
 tc=3,  
 bf=0.7,  
 maxTrees=c(1000, 2000),  
 w=TRUE,  
 verbose=TRUE  
)

# create raster map  
brtMap <- predict(climate[[predictors]], brtModel, type='response', n.trees=brtModel$n.trees)  
  
# save results  
save(brtModel, file='./Models/Model 12 Other Algorithms/Model - BRT.Rdata', compress=TRUE)  
writeRaster(brtMap, './Models/Model 12 Other Algorithms/brtMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='BRT')  
plot(brtMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)

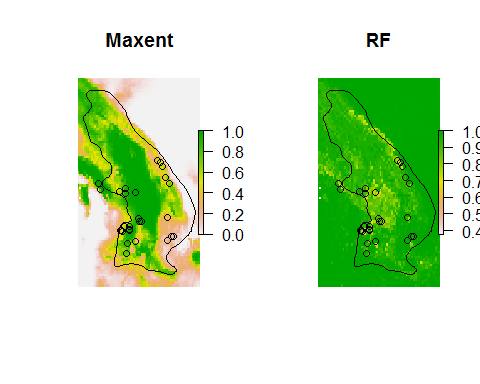


Relevant citation: [Elith, J., J.R. Leathwick, and T. Hastie](http://www.dx.doi.org/10.1111/j.1365-2656.2008.01390.x). 2008. A working guide to boosted regression trees. Journal of Animal Ecology 77:802-813.

## Random Forests (RFs)

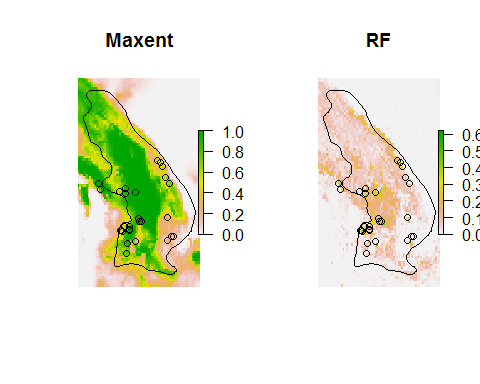
Random forests are like BRTs in that they are also constructed from many classification and regression trees, but whereas in BRTs they are chained togrther serially, in RFs they are combined in parallel. Each tree gets a "vote" in the final outcome. The trainRf() function is a simple wrapper for the randomForest() function in the package of the same name, but it's included here to match the same format as the other trainXXX() functions.

# convert the response (1/0) to a factor  
trainData$presBg <- factor(trainData$presBg)  
  
rfModel <- trainRf(  
 data=trainData,  
 resp='presBg',  
 package='randomForest'  
)  
  
# create raster map  
rfMap <- predict(climate[[predictors]], rfModel, type='prob')  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='RF')  
plot(rfMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



What's going on here? The RF map seems inverted... unsuitable areas have high values and vice versa. The problem lies in the fact that the randomForest function (called by trainRf) requires factors when the response is binary. In this case the function reversed the "meaning" of the 0s and 1s indicating background/presence, so it's giving us the probability that a site belongs to the background group. We need to invert the prediction.

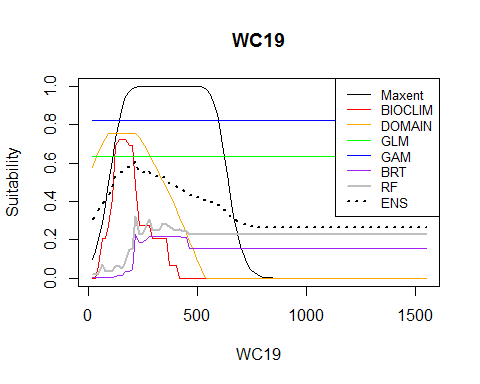
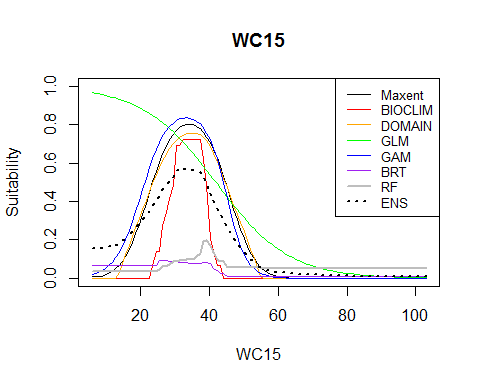
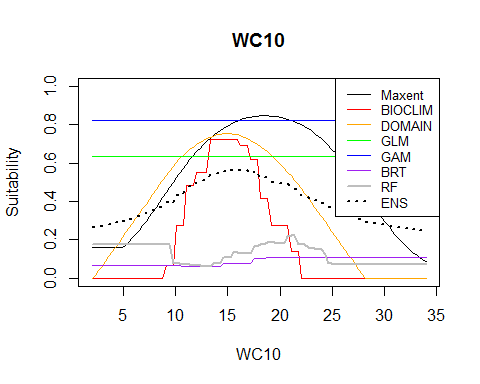
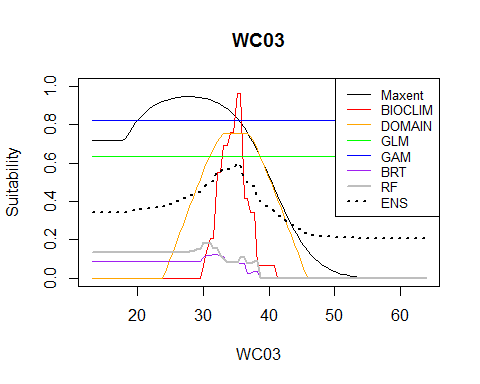
rfMap <- 1 - rfMap  
  
# save results  
save(rfModel, file='./Models/Model 12 Other Algorithms/Model - RF.Rdata', compress=TRUE)  
writeRaster(rfMap, './Models/Model 12 Other Algorithms/rfMap', format='GTiff', overwrite=TRUE)  
  
# plot  
par(mfrow=c(1, 2))  
plot(rangeMap, main='Maxent')  
plot(tunedMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)  
  
plot(rangeMap, main='RF')  
plot(rfMap, add=TRUE)  
plot(rangeMap, add=TRUE)  
points(records$longitude, records$latitude)



# Response curves

So what do the response curves for these models look like? Let's take a look!

envPres <- records[ , predictors]  
medianEnv <- apply(envPres, 2, median)  
medianEnv <- as.data.frame(medianEnv)  
medianEnv <- t(medianEnv)  
medianEnv <- as.data.frame(medianEnv) # yes, again!  
medianEnv <- medianEnv[rep(1, 100), ]  
row.names(medianEnv) <- 1:nrow(medianEnv)  
  
env <- rbind(records[ , predictors], randomBg[ , predictors])  
  
# plot response to each predictor  
for (thisPred in predictors) {  
   
 minEnv <- min(env[ , thisPred])  
 maxEnv <- max(env[ , thisPred])  
   
 thisEnvData <- medianEnv  
 thisEnvData[ , thisPred] <- seq(minEnv, maxEnv, length.out = 100)  
   
 # Maxent  
 maxentPred <- predict(tunedModel, thisEnvData, type='cloglog')  
   
 # BIOCLIM  
 bioclimPred <- predict(bioclimModel, thisEnvData)  
   
 # DOMAIN  
 domainPred <- predict(domainModel, thisEnvData)  
   
 # GLM  
 glmPred <- predict(glmModel, thisEnvData, type='response')  
   
 # GAM  
 gamPred <- predict(gamModel, thisEnvData, type='response')  
   
 # BRT  
 brtPred <- predict(brtModel, thisEnvData, type='response', n.trees=brtModel$n.trees)  
   
 #RFs  
 rfPred <- predict(rfModel, thisEnvData, type='prob')  
 rfPred <- rfPred[ , 2]  
   
 # ensemble prediction  
 allPred <- matrix(  
 c(maxentPred, bioclimPred, domainPred, glmPred, gamPred, brtPred, rfPred),  
 byrow=TRUE, ncol=100  
 )  
   
 ensemblePred <- colMeans(allPred)  
   
 # plot  
 x <- seq(minEnv, maxEnv, length.out = 100)  
   
 plot(x, maxentPred, ylim=c(0, 1), main=thisPred, type='l', xlab=thisPred, ylab='Suitability')  
 lines(x, bioclimPred, col='red')  
 lines(x, domainPred, col='orange')  
 lines(x, glmPred, col='green')  
 lines(x, gamPred, col='blue')  
 lines(x, brtPred, col='purple')  
 lines(x, rfPred, col='gray', lwd=2)  
 lines(x, ensemblePred, lty='dotted', lwd=2)  
   
 legend('topright',  
 legend=c('Maxent', 'BIOCLIM', 'DOMAIN', 'GLM', 'GAM', 'BRT', 'RF', 'ENS'),  
 col=c('black', 'red', 'orange', 'green', 'blue', 'purple', 'gray', 'black'),  
 lty=c(rep('solid', 7), 'dotted'),  
 lwd=c(rep(1, 6), 2, 2),  
 cex=0.8  
 )  
   
}



## Reflection

1. Which model algorithm seems to produce the most biologically plausible responses? How do you know what is plausible?
2. Which model algorithm seems to be most "conservative" (predicts the lowest values)? Which the most generous?
3. Some of the responses are flat. This occurs when the predictor is not selected to be in the model (the non-zero value reflects the prediction across the median value of the other predictors). Which algorithms tend to do this? Do these variables really not matter?
4. Do the different algorithms tend to agree at least in the direction they predict (increase/decrease/stable)?
5. How do the different models extrapolate? What effect would this have when extimating species' vulnerability to climate change?

*~ FINIS ~*