Practical 4 Species Distribution Modelling I

MD

20/02/2020

In this week's pratical we will take our first step into species distribution modelling. We will see that there are multiple options for modelling the distribution of species and that all depend on point data and the extraction of environmental factors to those points.

First, let's set the working directory and install the packages that we will use for today's work (notice the use of the c() function to call multiple packages at once):

```
setwd("P:/GEOG70922/Week5")
```

```
install.packages(c("dismo","maptools","glmnet","maxnet","raster","sp","randomForest"))
```

We will be using raster layers from the Worldclim dataset of environmental variables that come ready-to-use once the dismo package is installed. Information on these layers can be found at: http://worldclim.org/bioclim (http://worldclim.org/bioclim) Follow this link now to see how the different layer names are coded. The 'biome' layer included here comes from the 'Terrestrial Biome' dataset by the WWF. You can get more information on this layer here: https://www.worldwildlife.org/publications/terrestrial-ecoregions-of-the-world (https://www.worldwildlife.org/publications/terrestrial-ecoregions-of-the-world)

For now let's load these layers, name them and create a raster stack object (a raster made of multiple bands) before plotting them along with some world map data to see where in the world we are this week. Thes files are inside the 'ex' folder of the dismo package and all have the extention '.grd'. We can access these using the list.files() function and concatonating the file paths using the paste() function. Type '?paste' into the command line for more information on this function. Thisis how they appear when called on my own machine:

```
library(dismo)
```

```
## Loading required package: raster
```

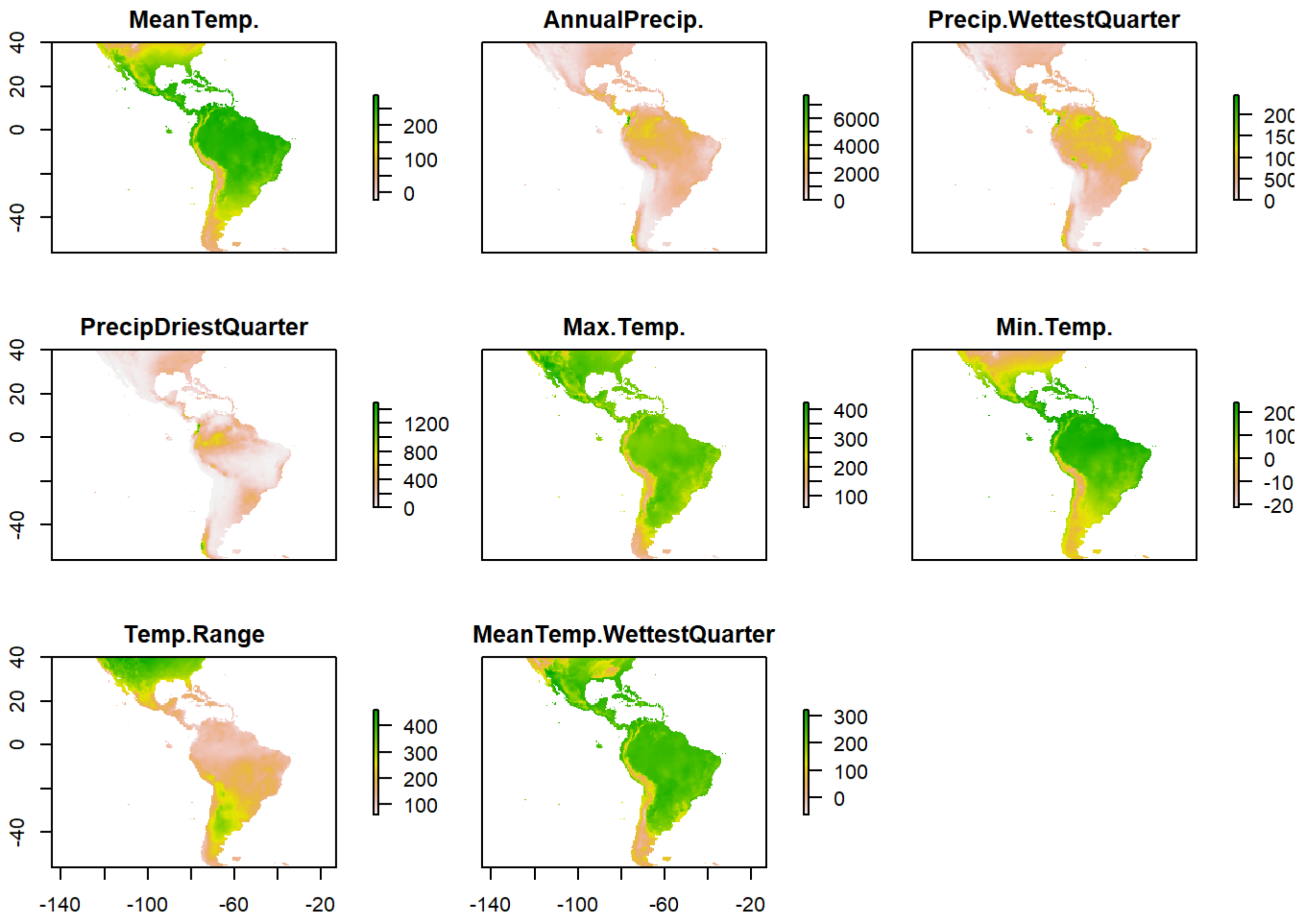
```
## Loading required package: sp
```

```
files <- list.files(path=paste(system.file(package="dismo"),
                              '/ex', sep=''), pattern='grd', full.names=TRUE )

files
```

```
## [1] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio1.grd"
## [2] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio12.grd"
## [3] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio16.grd"
## [4] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio17.grd"
## [5] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio5.grd"
## [6] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio6.grd"
## [7] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio7.grd"
## [8] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/bio8.grd"
## [9] "C:/Users/mlibxmda/Documents/R/win-library/4.0/dismo/ex/biome.grd"
```

Now let's use the stack() function to create a multi-band raster and add names to our bands.

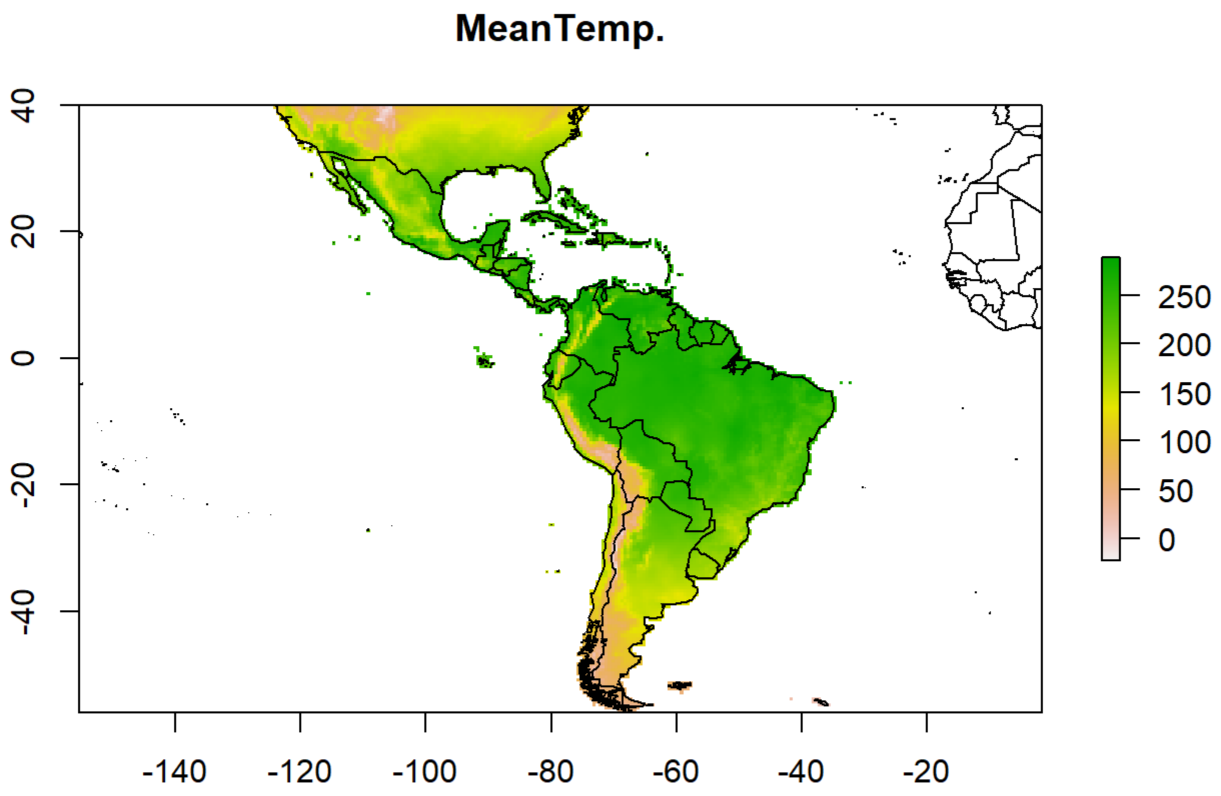
```
env<-stack(files[1:8])


#use the names() function to label each covariate layer

names(env) <- c("MeanTemp.","AnnualPrecip.","Precip.WettestQuarter","PrecipDriestQuart
er","Max.Temp.","Min.Temp.","Temp.Range","MeanTemp.WettestQuarter")



plot(env)
```



```
library(maptools)
```

```
## Checking rgeos availability: TRUE
```

```
 data(wrld_simpl)

#And now plot:
 # first layer of the RasterStack
 plot(env, 1)
 # note the "add=TRUE" argument with plot
 plot(wrld_simpl,add=TRUE)
```



Now we need some species data. Rather than providing our own data we will use the gbif() function to download species record data directly from the on-line portal: Global Biodiversity Information Facility [www.gbif.org]. Of course it is useful and enjoyable to explore species records on the GBIF website, but it we are looking for data on a specific species the gbif() function provide us with instant access and even gives us he option of downloaded the data as SpatialPoints object directly into R.

Today we will practise our species modelling techniques on a well known dataset for this purpose - point occurrence data for a mammal species native to areas of South America, Bradypus variegatus.

```
bradypus <- gbif("Bradypus", "variegatus*",sp=TRUE)
```

```
## Loading required namespace: jsonlite
```

```
## 2680 records found
```

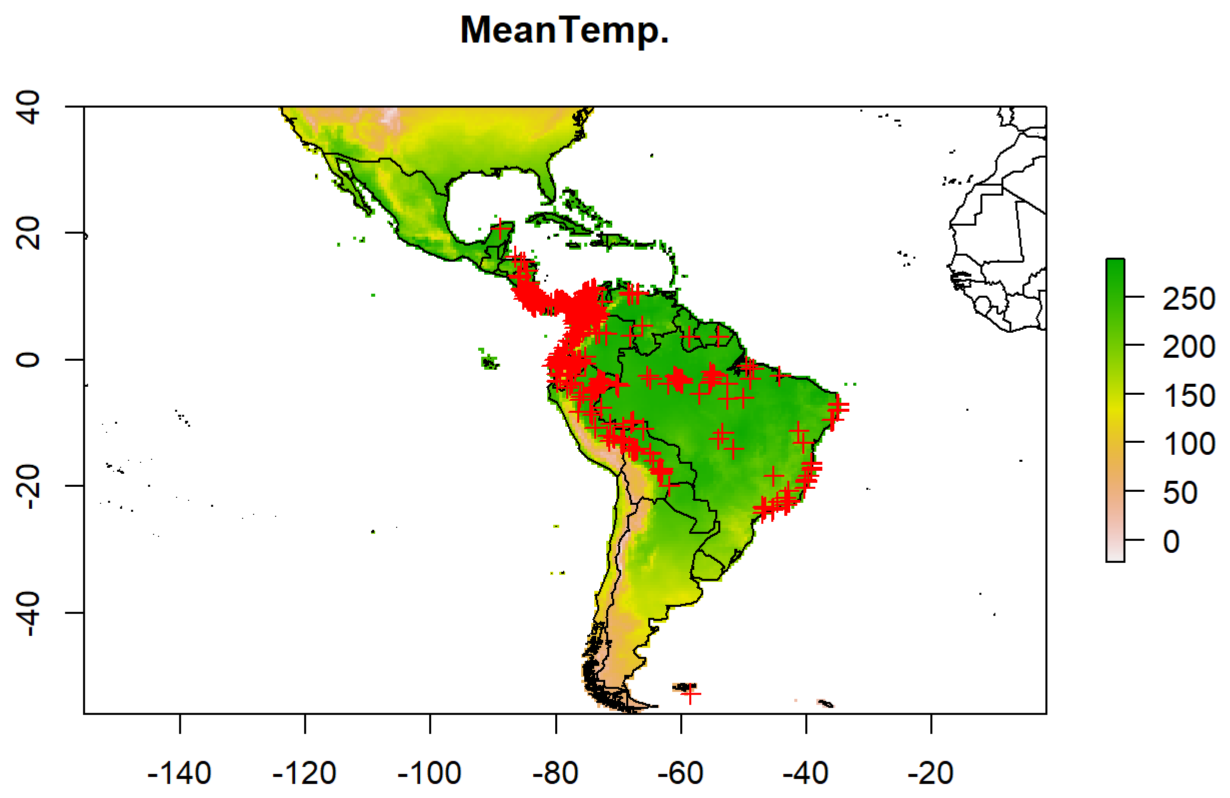
```
## 0-300-600-900-1200-1500-1800-2100-2400-2680 records downloaded
```

Now we’ll assign a crs to the lat/lon coordinates of the Bradypus object so we can plot it on our worldclim data and wrld_simpl data (all in the WGS84 crs).

```
wgs84<-crs.latlong<-crs("+init=epsg:4326")


crs(bradypus)
```

```
## CRS arguments: NA
```

```
plot(env, 1)

plot(wrld_simpl,add=TRUE)

plot(bradypus,add=TRUE,col='red')
```



Now, we know that Bradypus variegatus is a new world species (we aren’t expecting it to show up in Europe or Asia), so lets subset the Wrld_simpl data to give only South and Central America.

Fist let’s look at the data attribute table of the wrld_simple object:

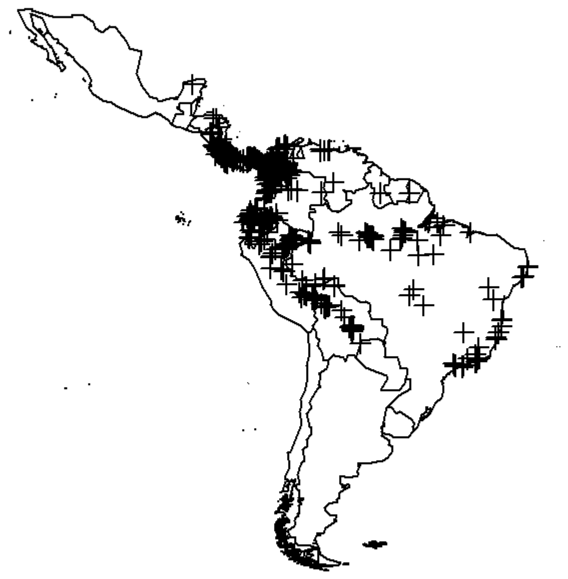
```
View(wrld_simpl@data)
```

Looks like we want SUBREGIONS 5 and 13 from the data. We use the subset() function to select entries in the wrld_simpl data SUBREGION column that are 5 or 13. Note the use of the OR operator ‘|’ and the ‘==’ signifying ‘is equal to’ (as in the Python language).

```
SA<-subset(wrld_simpl,wrld_simpl$SUBREGION==5|wrld_simpl$SUBREGION==13)

plot(SA)
```

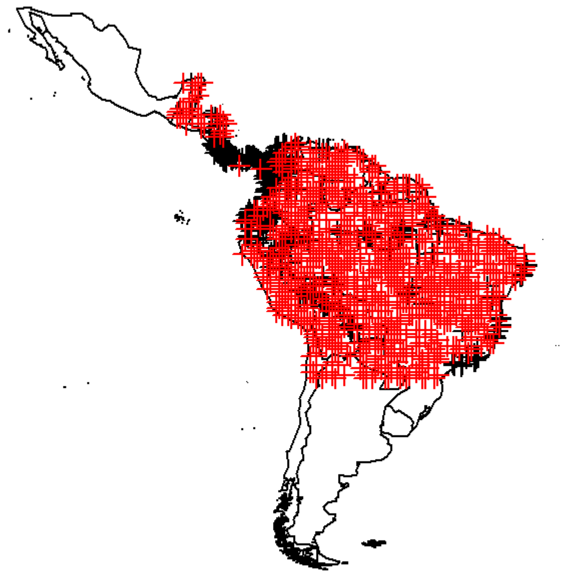
Next we’ll set the Bradypus location data to the same CRS as the world map and subset our points to make sure they are all within the boundary of the SA object (i.e. to remove any erroneous records with incorrect coordinates). Here we take advantage of the spatial extension of the square bracket ‘[]’ notation. We can very conveniently subset spatial data occuring within the extent of another spatial data layer using ‘[]’ without the need for an overlay function as would be required in a desktop GIS (and in previous versions of R). This simple trick is a great strength of R for spatial data manipulation. Note that we renew the object ‘bradypus’ by re-creating the object as a subset of the existing ‘bradypus’object (i.e. ’bradypus<- bradypus[SA,]’). If we wanted to keep two versions of this object we could have given the subsetted data a different name (such as ‘bradypus_sub’), leaving us with both ‘bradypus’ and ‘bradypus_sub’ objects. However, as we will only be working with this subsetted version, it makes sense just to keep the one object.

```
#Set the bradypus crs to that of the SA object created above.
crs(bradypus)<-crs(SA)

#subset our points to only those inside SA
bradypus<-bradypus[SA, ]

plot(SA)

plot(bradypus, add=TRUE)
```



Now we have the presence data and environmental covariates necessary to fit a presence-only model of species distribution. However, we will also carry out the analysis using algorithms that take presence and background points so lets create some in a similar way as we did in Week 2, using the randomPoints() function from the dismo package.

```
set.seed(11)

#create random points on cells of the 'env' object within the extent of bradypus and a
voiding cells containing points from bradypus

back.xy <- randomPoints(env, n=1000,p= bradypus,ext = extent(bradypus))


#Create Spatial Points layer from back.xy

back<-SpatialPoints(back.xy,crs(bradypus))

#make sure crs matches other data

crs(back)<-crs(SA)


#subset to SA object

back<-back[SA,]

plot(SA)

plot(bradypus,add=TRUE)

plot(back,add=TRUE, col='red')
```



Now we’ll extract information from our ‘env’ raster stack to both the presence and background points, add a column denoting presence (‘1’) and absence (‘0’) and combine them into a data frame for use in further analysis.

```
#absence values

eA<-extract(env,back)

#presence values
eP<-extract(env,bradypus)

#create data frames from values
Pres.cov<-data.frame(eP,Pres=1)


Back.cov<-data.frame(eA,Pres=0)


#combine
all.cov<-rbind(Pres.cov,Back.cov)



head(all.cov)
```

```
##   MeanTemp. AnnualPrecip. Precip.WettestQuarter PrecipDriestQuarter Max.Temp.
## 1       230          1238                   589                  83       295
## 2       260          4353                  1918                 247       336
## 3       244          1072                   340                 224       306
## 4       276          1690                   740                  59       342
## 5       259          2574                   812                 495       316
## 6       230          1238                   589                  83       295
##   Min.Temp. Temp.Range MeanTemp.WettestQuarter Pres
## 1       160        135                     232    1
## 2       193        142                     254    1
## 3       184        122                     254    1
## 4       220        122                     274    1
## 5       206        110                     252    1
## 6       160        135                     232    1
```

```
tail(all.cov)
```

```
##      MeanTemp. AnnualPrecip. Precip.WettestQuarter PrecipDriestQuarter
## 2834       268          2354                   716                 393
## 2835       250          2059                   785                 208
## 2836       234          1419                   545                 131
## 2837       260          1526                   744                  10
## 2838       234          1684                   649                 179
## 2839       254          2403                  1106                  62
##      Max.Temp. Min.Temp. Temp.Range MeanTemp.WettestQuarter Pres
## 2834       329       202        127                     269    0
## 2835       323       172        151                     250    0
## 2836       310       139        171                     251    0
## 2837       347       170        177                     258    0
## 2838       309       151        158                     235    0
## 2839       348       166        182                     249    0
```

```
summary(all.cov)
```

```
##    MeanTemp.     AnnualPrecip.  Precip.WettestQuarter PrecipDriestQuarter
##  Min.   : 29.0   Min.   :   2   Min.   :   2.0        Min.   :   0.0     
##  1st Qu.:237.5   1st Qu.:1732   1st Qu.: 749.0        1st Qu.:  93.0     
##  Median :257.0   Median :2489   Median : 928.0        Median : 149.0     
##  Mean   :244.3   Mean   :2405   Mean   : 955.3        Mean   : 218.8     
##  3rd Qu.:263.0   3rd Qu.:2956   3rd Qu.:1183.0        3rd Qu.: 300.0     
##  Max.   :284.0   Max.   :7682   Max.   :2458.0        Max.   :1496.0     
##    Max.Temp.       Min.Temp.        Temp.Range    MeanTemp.WettestQuarter
##  Min.   :125.0   Min.   :-116.0   Min.   : 78.0   Min.   : 44.0          
##  1st Qu.:303.0   1st Qu.: 161.0   1st Qu.:110.0   1st Qu.:238.0          
##  Median :316.0   Median : 196.0   Median :122.0   Median :254.0          
##  Mean   :310.7   Mean   : 179.4   Mean   :131.2   Mean   :244.5          
##  3rd Qu.:329.0   3rd Qu.: 209.0   3rd Qu.:141.0   3rd Qu.:260.0          
##  Max.   :361.0   Max.   : 231.0   Max.   :293.0   Max.   :284.0          
##       Pres       
##  Min.   :0.0000  
##  1st Qu.:0.0000  
##  Median :1.0000  
##  Mean   :0.6788  
##  3rd Qu.:1.0000  
##  Max.   :1.0000
```

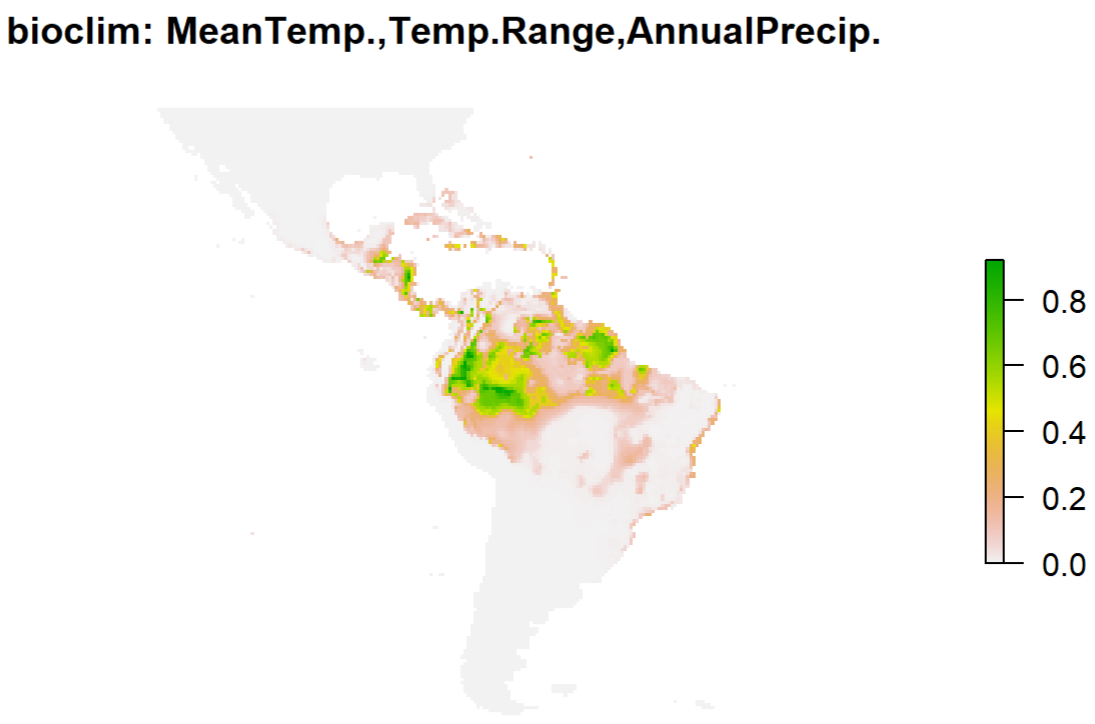
Now let’s create our first model. The Biolclim() function performs a distribution model based on the envelope modelling approach and takes presence data only. We specify the model therefore using only the Pres.cov data frame and, for now, we will use co-variates MeanTemp., Temp.Range. and AnnualPrecip.:

```
bc_bradypus <- bioclim(Pres.cov[,c('MeanTemp.', 'Temp.Range', 'AnnualPrecip.')])



bc_bradypus
```

```
## class    : Bioclim
##
## variables: MeanTemp. Temp.Range AnnualPrecip.
##
##
## presence points: 1927
##    MeanTemp. Temp.Range AnnualPrecip.
## 1        230        135          1238
## 2        260        142          4353
## 3        244        122          1072
## 4        276        122          1690
## 5        259        110          2574
## 6        230        135          1238
## 7        263        112          1878
## 8        234        113          3490
## 9        254        119          2084
## 10       268        137          1369
##   (... ...  ...)
```

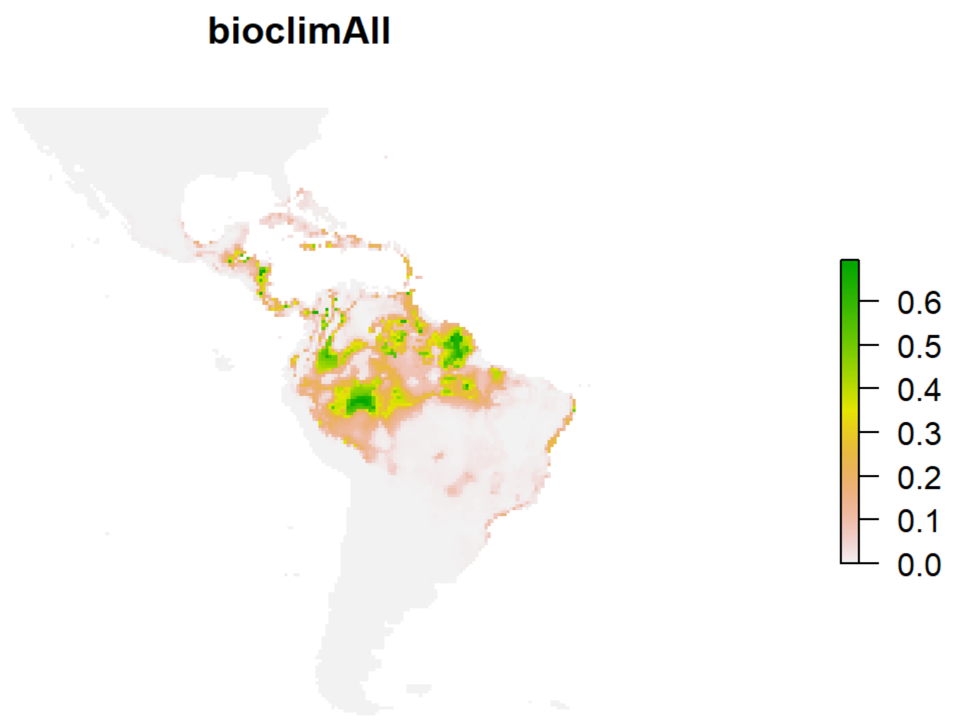
We can map our prediction using the predict() function, using our covariate raster stack as the object and the bc_bradypus as the model. Note that for a bioclim() model the predicted values range from 0 to 1. It is scaled such that a value of 1 would be locations that have the median value of all covariates considered, while a value of zero would reflect locations where at least one covariate is outside the range of environmental covariates at presence locations.

```
bioclim.map <- predict(env, bc_bradypus)
plot(bioclim.map, axes = F, box = F, main = "bioclim: MeanTemp.,Temp.Range,AnnualPreci
p.")
```



Now we'll repeat our bioclim model but using all predictors (i.e. all columns of co-variates in Pres.cov)

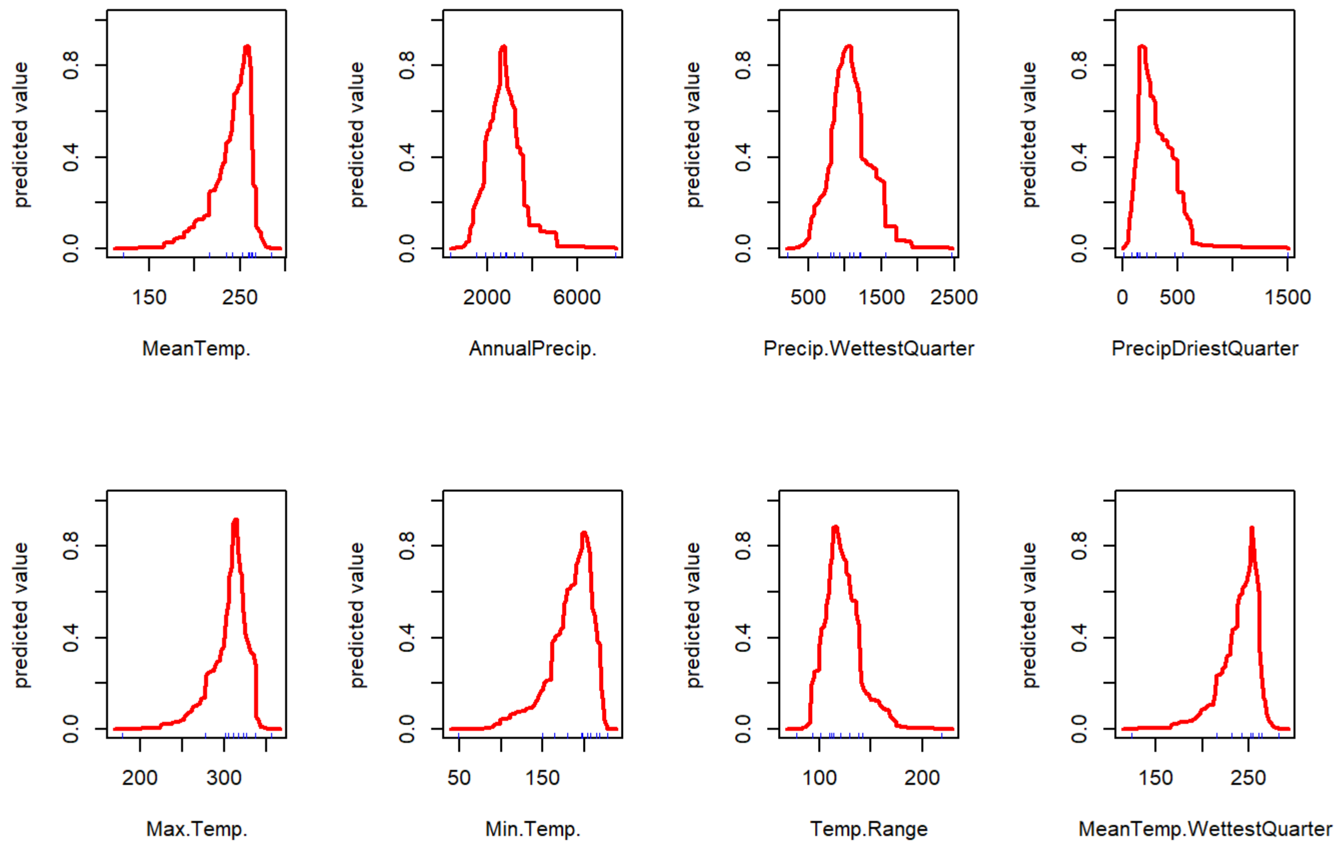
```
bc_bradypus2<-bioclim(Pres.cov[,1:8])


bioclim.map2 <- predict(env, bc_bradypus2)
plot(bioclim.map2, axes = F, box = F, main = "bioclimAll")
```



We can see that our models, as expected, are very similar though we might assume that, given the relatively higher values and using fewer predictors, the first model (bc_bradypus) may be over-predicting our species occurence.

We can also use the response() function to give the response plots for each covariate.

```
response(bc_bradypus2)
```



The response plots show values for each covariate at which prediction is highest.It might help to think of these plots as being similar in principle to a histogram, with values for co-variates on the X axis and proportion of presence points on the Y axis. For most of our covariates, the relationship appears to be non-linear in the presence-only bioclim model.

Next we'll look at fitting some models that accept presence and (psuedo)-absence data. We will use the glm() function (glm stands for 'generalized linear model') and set the family of functions to 'binomial' (i.e. a logistic regression) as our outcome is binary. We set the first argument (the response variable) as 'Pres' and separate this from our predictor variable with the tilda symbol '~'. We can opt to include all variables (i.e. the remaining columns in the data frame) simply by placing a point '.' after the tilda symbol.

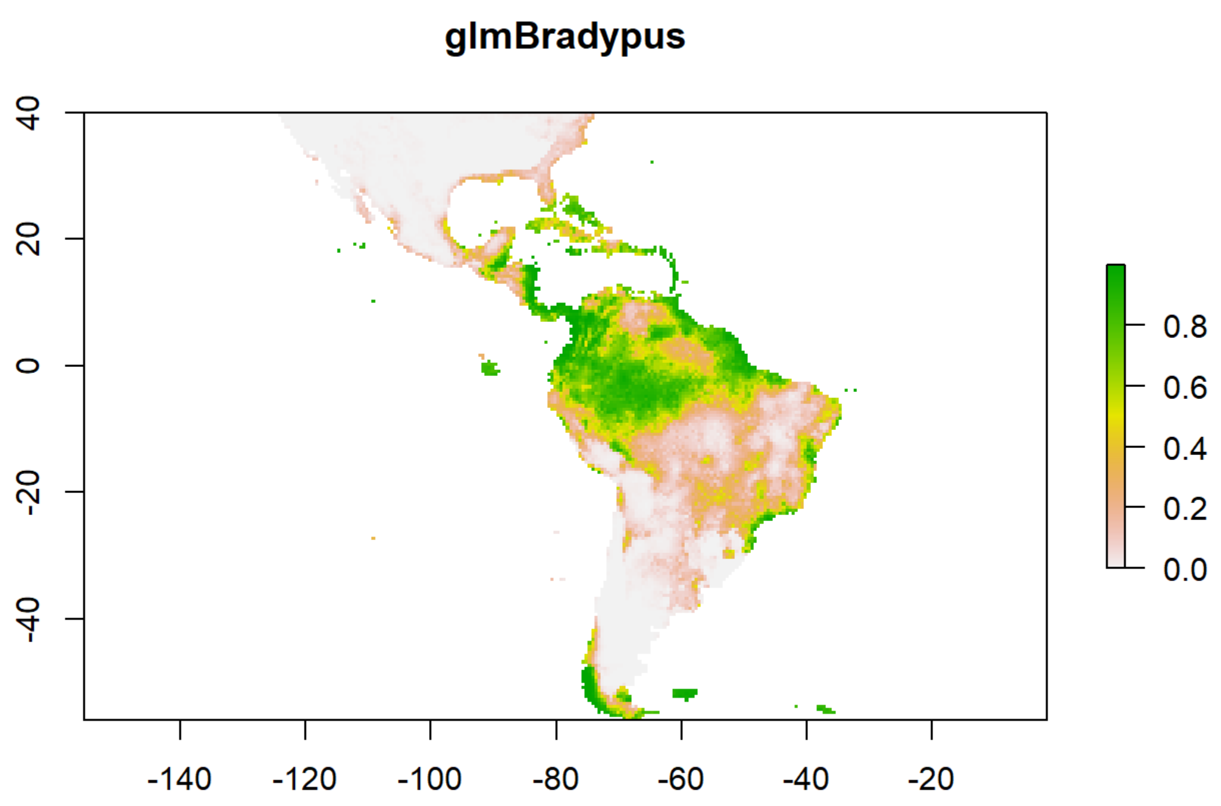
```
#Specify model
glm.bradypus<-glm(Pres~.,binomial(link='logit'), data=all.cov)


#get summary of outputs

summary(glm.bradypus)
```

```
##
## Call:
## glm(formula = Pres ~ ., family = binomial(link = "logit"), data = all.cov)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3506  -0.4765   0.3083   0.6237   2.7509
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               8.3769139  0.7519123  11.141  < 2e-16 ***
## MeanTemp.                -0.0905168  0.0218849  -4.136 3.53e-05 ***
## AnnualPrecip.             0.0040558  0.0003858  10.512  < 2e-16 ***
## Precip.WettestQuarter    -0.0056898  0.0008164  -6.969 3.19e-12 ***
## PrecipDriestQuarter      -0.0087847  0.0008728 -10.065  < 2e-16 ***
## Max.Temp.                 0.4780591  0.1054964   4.532 5.86e-06 ***
## Min.Temp.                -0.4975262  0.1065683  -4.669 3.03e-06 ***
## Temp.Range               -0.5473636  0.1061224  -5.158 2.50e-07 ***
## MeanTemp.WettestQuarter   0.1022126  0.0099477  10.275  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3564.7  on 2838  degrees of freedom
## Residual deviance: 2272.1  on 2830  degrees of freedom
## AIC: 2290.1
##
## Number of Fisher Scoring iterations: 5
```

The statistics provided by the summary() function from the glm model are the co-efficient values estimated for each predictor variable and the AIC (Akaike Information Criterion) value. For the purpose of model comparison, a lower AIC denotes better model fit.

We can map the probability predictions of our glm.bradypus output in the same way as we did for the bioclim model.

```
#Use the predict function to estimate probabilities for our study area based on the ra
ster stack object 'env'.
glm.map <- predict(env, glm.bradypus, type = "response")

#plot the map
plot(glm.map,main="glmBradypus")
```



```
glm.map
```

```
## class       : RasterLayer
## dimensions  : 192, 186, 35712  (nrow, ncol, ncell)
## resolution  : 0.5, 0.5  (x, y)
## extent      : -125, -32, -56, 40  (xmin, xmax, ymin, ymax)
## crs         : +proj=longlat +datum=WGS84 +no_defs
## source      : memory
## names       : layer
## values      : 3.129009e-12, 0.9997035  (min, max)
```

Now we will look at the application of a Maxent algorithm to our data. Maxent is a popular species distribution modelling algorithm, partly due to the open source package that can be freely downloaded and used “out of the box”. It can also be used within the R environment where, if the programme is saved inside the dismo package folder in the R directory, we can call maxent() from the dismo library in the same way as we have already called bioclim() and glm() from their respective libraries. However, this can be problematic if working on a computer without admin rights and in any case there is a way to implement a maxent model without outsourcing the algorithm from within R. The glmnet package contains a function maxnet() that allows us to do this.

For maxnet, we provide our presence data followed by the columns of our data frame containing predictors and, in this case, we choose “lq” (meaning linear and quadratic) from the list of available feature classes for model estimation.

In truth, the maxnet() function approximates a maxent formula using an “infinitely weighted logistic regression” - an inhomogenous point process (IPP) approach carried out with the glmnet package (see Fithian and Hastie, 2013). This is why we also installed glmnet at the start of the practical. As you can see from the topics covered in the lecture and suggested reading for this week, there is a strong theoretical basis for using point process models in ecology.

```
 library(maxnet)
 library(glmnet)
```

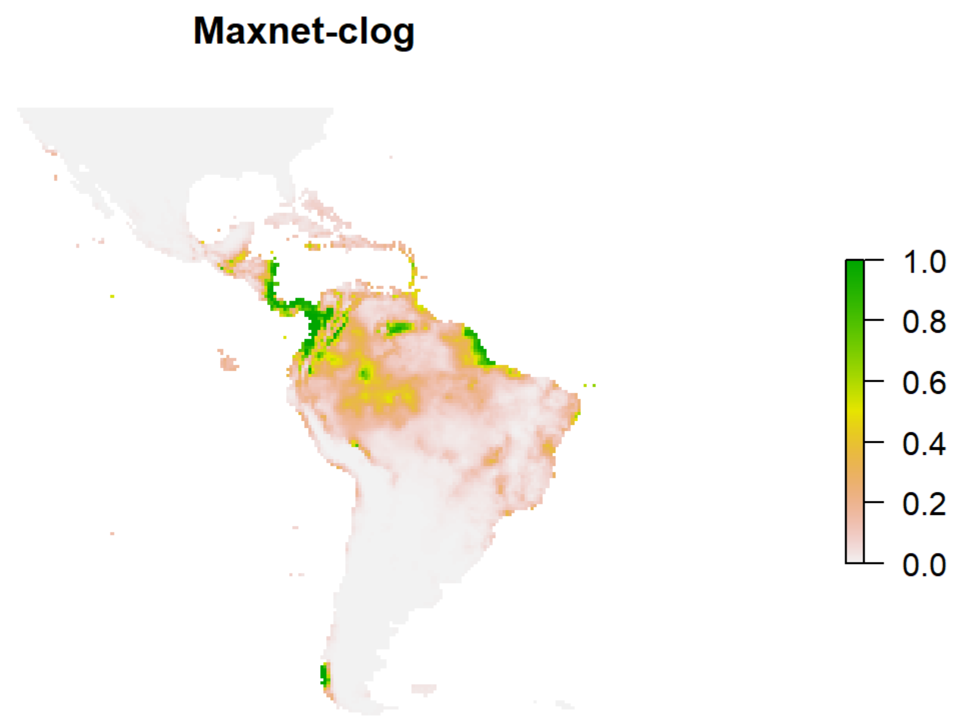
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1
```

```
 maxnetMod <- maxnet(all.cov$Pres, all.cov[,1:8],maxnet.formula(all.cov$Pres,all.cov[,
1:8],classes="lq"))
```

To transform the raw model output values (an exponential model) for mapping to a 0 to 1 probability scale, we have several options. The default option in the orginal Maxent software was type “logistic” for ploting response curves though recent work comparing Maxent to an IPP formulation suggests that a cloglog (complementary log-log transform) approach is more suitable for mapping presence/absence predictions. Consequently, cloglog is now the default for the open source Maxent tool and we will plot it here.

```
 maxnet.cloglog.map <- predict(env, maxnetMod, clamp=F, type="cloglog")

plot(maxnet.cloglog.map, axes=F, box=F, main="Maxnet-clog")
```



Model Validation

An important part of species distribution modelling is model validation and this is a subject we will come back to repeatedly throughout this and next week's practical. Although we briefly looked at the usefulness of the AIC statistic for comparing models, the AIC does not give us any idea about the predictive power of our model. To evaluate this, we need to use information provided in the confusion matrix, as introduced in the lecture.

The most readily available function (before creating our own later) for generating a confusion matrix and returning associated statistics is the evaluate() function that comes with the dismo package. To use this function, we supply three arguments to the evaluate() function: a data frame of presence points, a data frame of absence points and the model we wish to evaluate.

Let's evaluate our presence-only model (bioclim) first.

```
#For evaluating the first bioclim model we only povide the MeanTemp.,Temp.Range and An
nualPrecip. predictors which are columns 1,2 and 7 in the Pres.cov data frame


eBC <- evaluate(Pres.cov[,c(1,2,7)],Back.cov,bc_bradypus)

#For model two validation, provide all predictors
eBC2 <- evaluate(Pres.cov,Back.cov,bc_bradypus2)

#inspect both evaluation reults

eBC
```

```
## class          : ModelEvaluation
## n presences    : 1927
## n absences     : 912
## AUC            : 0.7161624
## cor            : 0.2327633
## max TPR+TNR at : 0.1095212
```

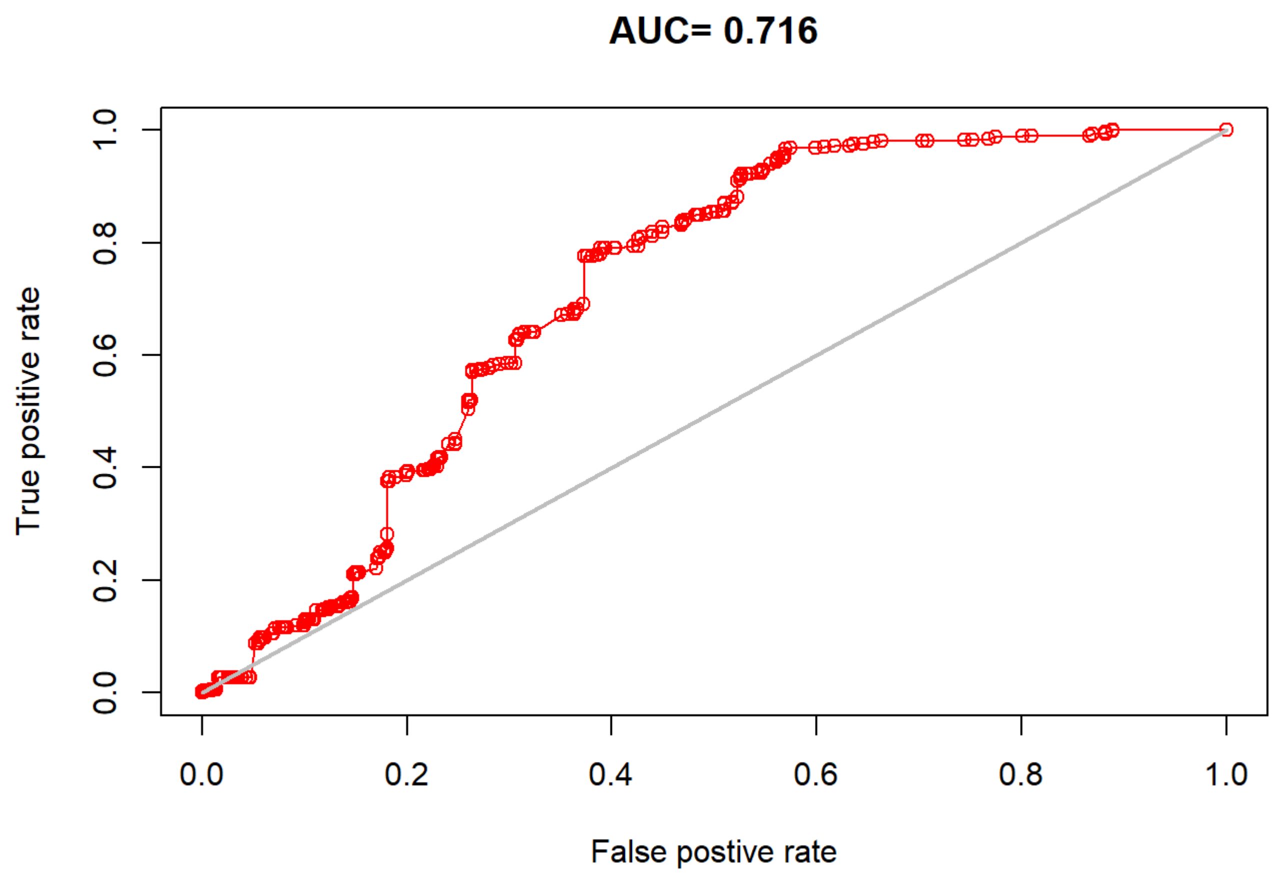
```
eBC2
```

```
## class          : ModelEvaluation
## n presences    : 1927
## n absences     : 912
## AUC            : 0.736629
## cor            : 0.2500237
## max TPR+TNR at : 0.01458604
```

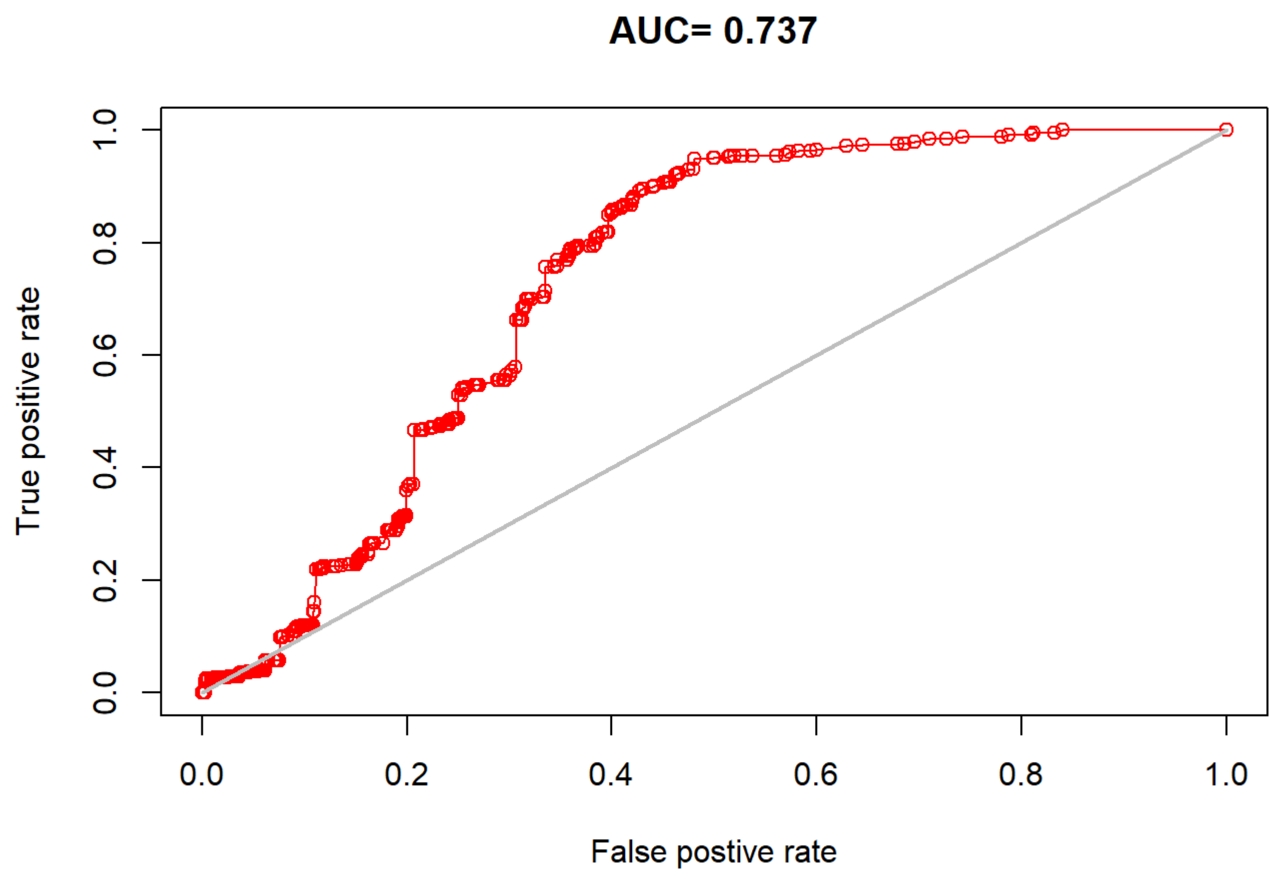
The main outputs of interest here are the AUC, and max TPR+TNR statistics. Recall from the lecture that AUC stands for Area Under the Curve and is a test of model performance where higher values indicate greater accuracy in our predictions.This statistic is often used as a mark of model performance and we will use it to assess the model we run today. Remember also that the max TPR+TNR denotes the probability threshold at which our model maximizes the True Positive Rate and the True Negative Rate. It is generally accepted that this is an optimum value at which to set the threshold for binary classification (i.e. 1/0 or presence/absence) in mapping outputs. We will use these values later when symbolising our final mapped predictions. For now, we can safely assume that our second bioclim model, with a higher AUC statistic, is the better performing model.

We can plot the ROC (Receiver Operating Curve) to visualize these results:

```
 plot(eBC,'ROC')
```



```
plot(eBC2,'ROC')
```

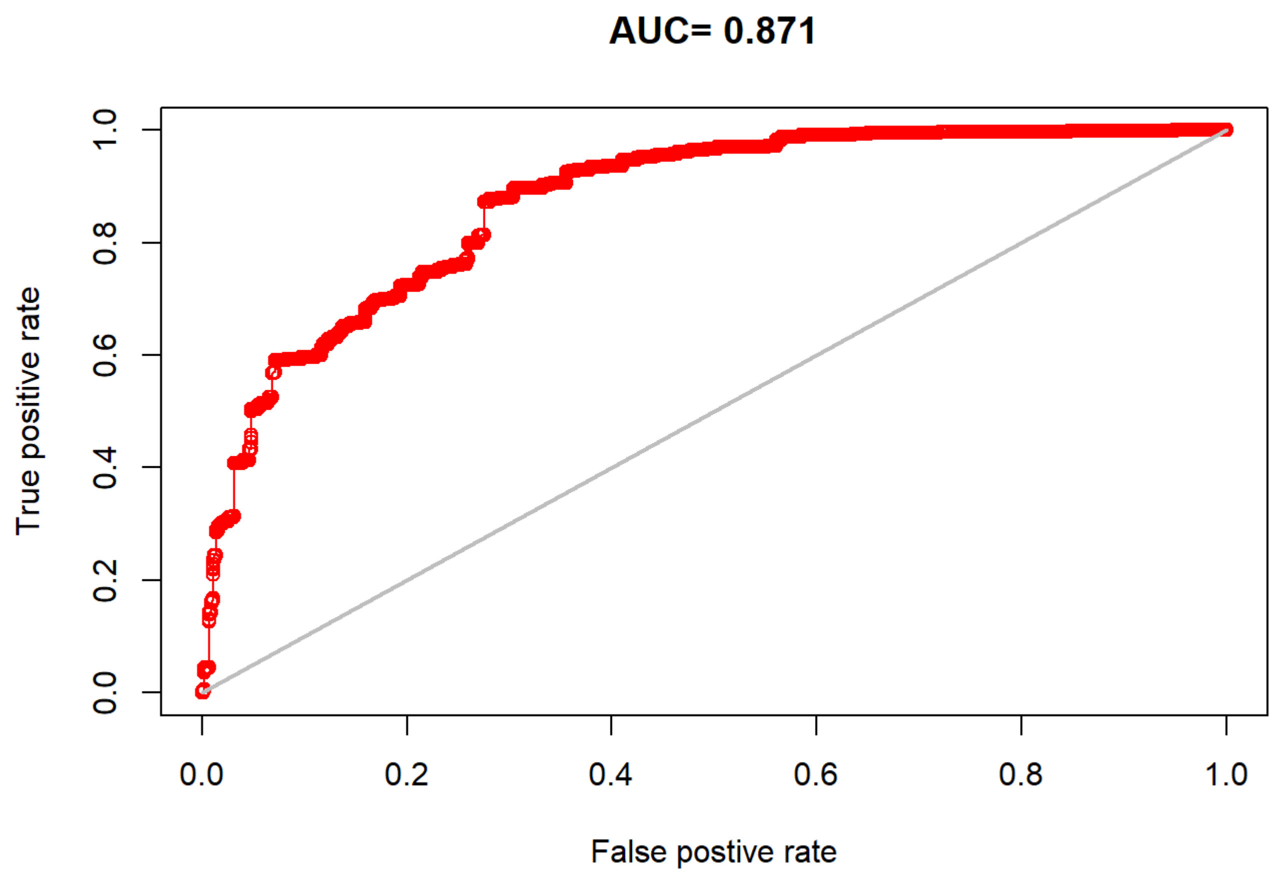


Now let's evaluate our glm models.

```
#GLM evaluation

#Model with all predictors
eglmAll <- evaluate(all.cov[all.cov$Pres==1,],all.cov[all.cov$Pres==0,], glm.bradypus)


#inspect
eglmAll
```

```
## class          : ModelEvaluation
## n presences    : 1927
## n absences     : 912
## AUC            : 0.8712047
## cor            : 0.6218117
## max TPR+TNR at : 0.4916525
```

```
plot(eglmAll,'ROC')
```

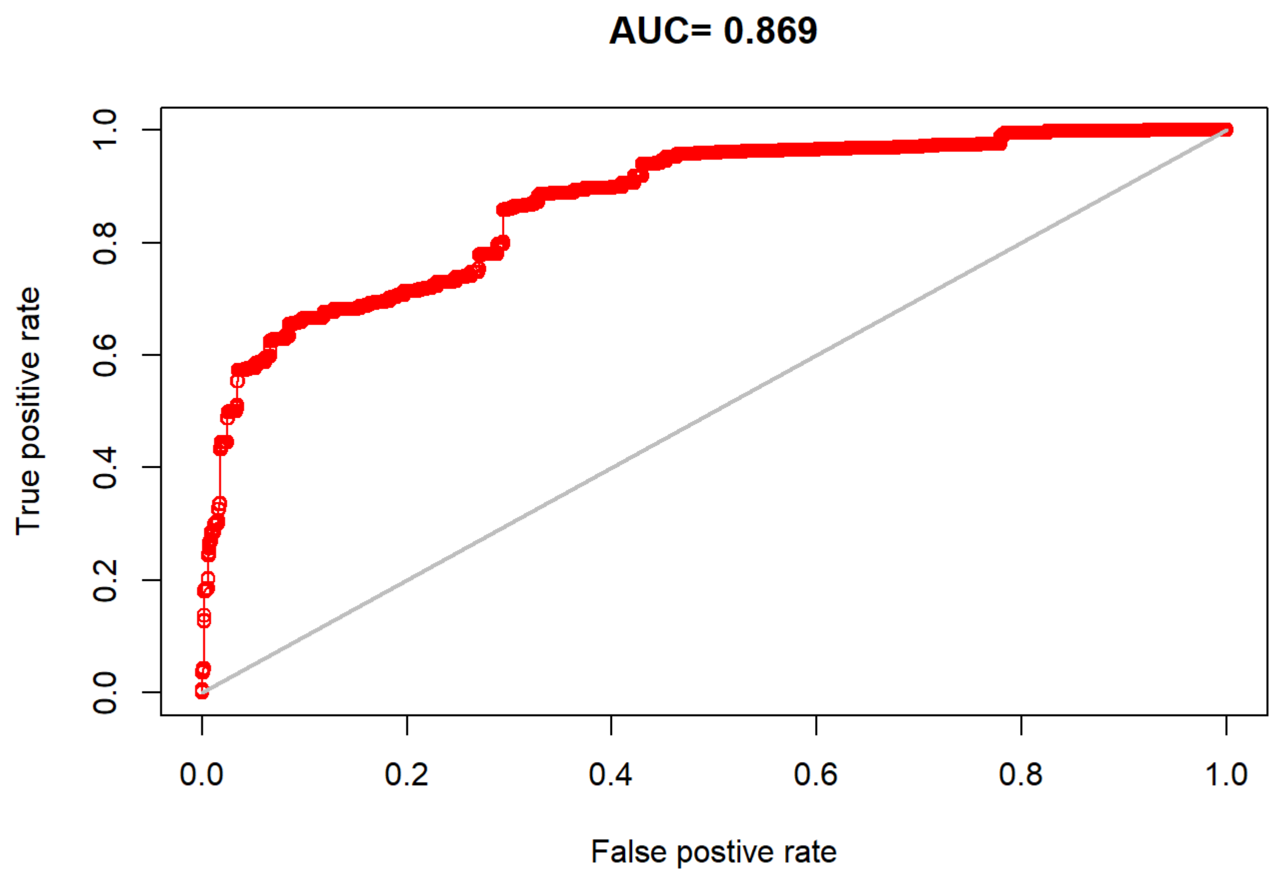


Finally we apply the evaluate() function to our maxnet model.

```
maxeval<-evaluate(p=bradypus, a=back, maxnetMod, env)

maxeval
```

```
## class          : ModelEvaluation
## n presences    : 1927
## n absences     : 912
## AUC            : 0.8690583
## cor            : 0.5800945
## max TPR+TNR at : -6.116487
```

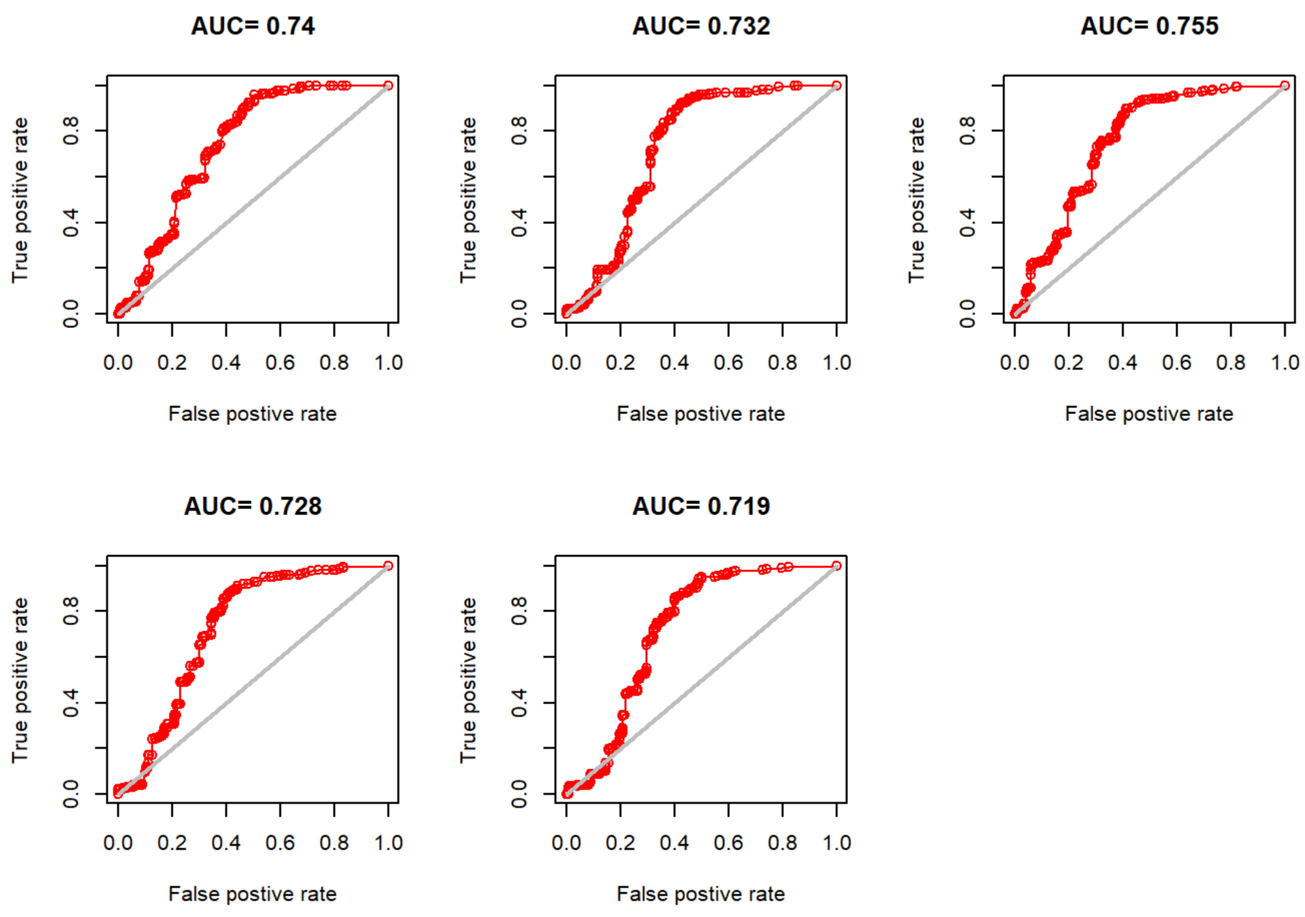
```
plot(maxeval,'ROC')
```



So far the evaluation results suggest good performance by all of our models (high AUC), particularly the glm approach. However, we have almost certainly over-estimated the quality of our models given that we have tested them with the same data that was used for training them. A more robust approach is to train the model on a subset of the data and test (evaluate) the model on the remaining cases. For example, we might use 70% of the data to train our model and use the remaining 30% of cases to test our model's ability to correctly classify those cases. Although more robust, this approach only gives us a single attempt at model validation on which to base our assessment. To overcome this, the k-fold approach is commonly used where the entire dataset is divided into folds (groups), say five, where one fold is chosen for model testing and the remaining four used for model training. This is then repeated in an iterative process until all folds have been used for model testing. We can then base our assessment on the mean values derived from all iterations. The kfold() function in dismo provided us with a neat way of partitioning our data.

```
set.seed(5)

#set number of folds to use
folds=5

#partiction presence and absence data according to folds using the kfold() function.

kfold_pres <- kfold(Pres.cov, folds)
kfold_back <- kfold(Back.cov, folds)


#create an empty list to hold our results (remember there will be five sets)
bc_K<-list()
par(mfrow=c(2,3))
# for loop to iterate over folds
for (i in 1:folds) {
  train <- Pres.cov[kfold_pres!= i,]#for presence values, select all folds which are n
ot 'i' to train the model
  test <- Pres.cov[kfold_pres == i,]#the remaining fold is used to test the model
  backTrain<-Back.cov[kfold_back!=i,]#now for background values, select training folds
  backTest<-Back.cov[kfold_back==i,]#use the remainder for model testing
  dataTrain<-rbind(train,backTrain)#bind presence and background training data togethe
r
  dataTest<-rbind(test,backTest)#bind test data together
  bc_bradypus_K<-bioclim(train[,1:8])#for bioclim we only need presence data to train
 the model!
  bc_K[[i]] <- evaluate(p=test[,1:8],a=backTest[,1:9], bc_bradypus_K)#use testing data
(kfold==i) for model evaluation

  #check the AUC by plotting ROC values

  plot(bc_K[[i]],'ROC')

}


#inspect

bc_K
```

```
## [[1]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.7400599
## cor            : 0.2821893
## max TPR+TNR at : 0.01481569
##
## [[2]]
## class          : ModelEvaluation
## n presences    : 386
## n absences     : 183
## AUC            : 0.7315538
## cor            : 0.2322336
## max TPR+TNR at : 0.02585717
##
## [[3]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.755102
## cor            : 0.2946908
## max TPR+TNR at : 0.01935525
##
## [[4]]
## class          : ModelEvaluation
## n presences    : 386
## n absences     : 183
## AUC            : 0.7280359
## cor            : 0.2498861
## max TPR+TNR at : 0.02780396
##
## [[5]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.719031
## cor            : 0.1935418
## max TPR+TNR at : 0.03751349
```



Next we use the slot() function to acces the elements in the list bc_K that are of use to us. In this case we want to return all auc values from our 5 evaluation runs. We 'loop' through all elements in the list using the sapply() function (remember this is similar to a for loop but applies to elements in a list or a vector). Effectively the below code is iterating over each element in the list 'bc_K' and applying a function that selects the 'auc' statistics from the correpsonding slot in the model results. Slots are basically attributes of objects in R. Here bc_k is a list of objects of type Formal Class ModelEvaluation. So we iterate of each and extract the 'auc' slot. You can type 'help(slot)' into the console for more information on slots.

Get model evaluation statistics:

```
aucBIO <- sapply( bc_K, function(x){slot(x, 'auc')} )

#calculate the mean values for coparison with other models


mean(aucBIO)
```

```
## [1] 0.7347565
```

In order to map the results of our model we need to set a threshold over which locations (cells on the map) are allocated 1 and, beneath which, 0. One way to determine this threshold is to calculate the threshold (the model output) at which the TPR (true positive rate) and TNR (true negative rate) are maximized. Remember we need both of these to be high in order to ensure a successful model.

We use the sapply() function to iterate over each item in our eGLM list and select the ‘t’ value (this is the slot on bc_K which denotes a given threshold) for which corressponding values for TPR+TNR are highest (hence we use ‘which.max’).

```
Opt_BIO<-sapply( bc_K, function(x){ x@t[which.max(x@TPR + x@TNR)] } )

Opt_BIO
```

```
## [1] 0.01481569 0.02585717 0.01935525 0.02780396 0.03751349
```

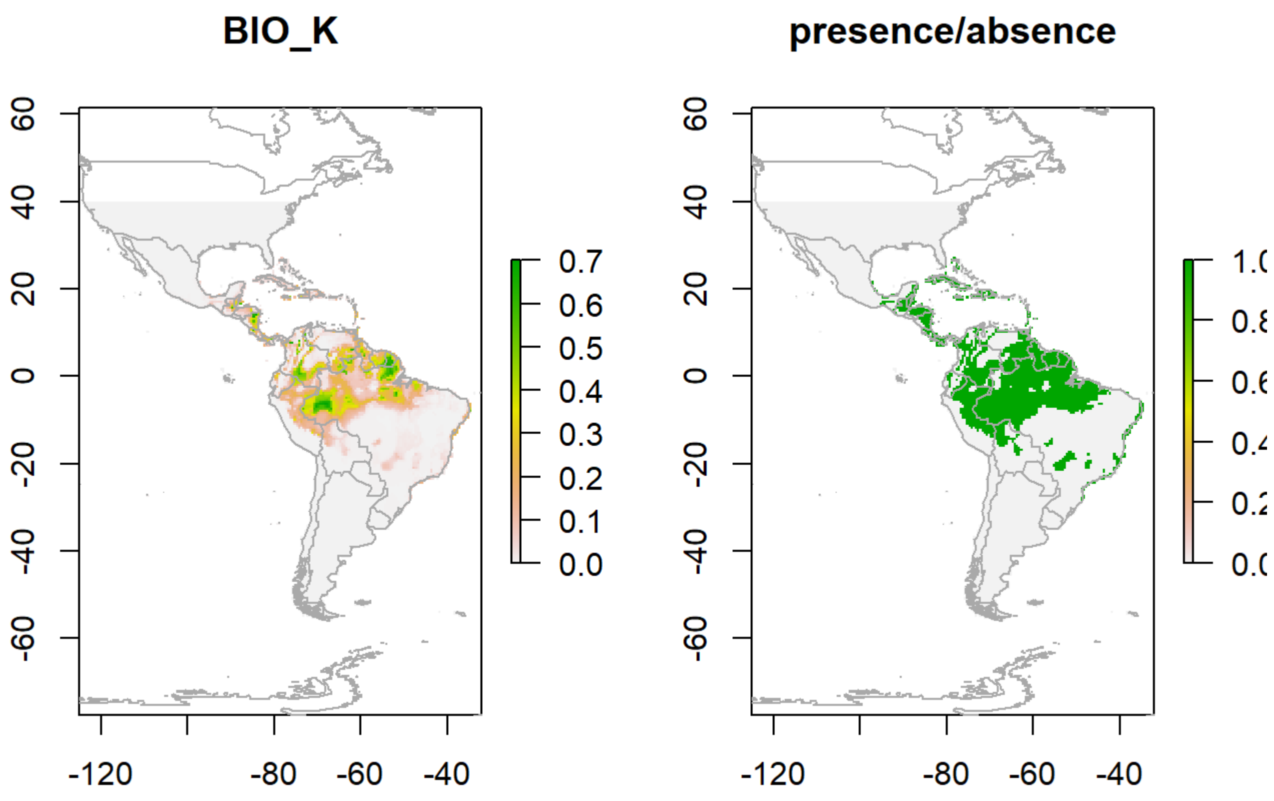
We will use these threshold values to make final predictions for areas of 1 and 0 (presence/absence) and map them.

```
#take the mean of the OptBIO list for application in our predictions

Mean_OptBIO<- mean(Opt_BIO)



Mean_OptBIO
```

```
## [1] 0.02506911
```

Now we just need to make a prediction based on our environmental raster stack and our bioclim model, setting 1 and 0 (presence/absence) according to the above threshold.

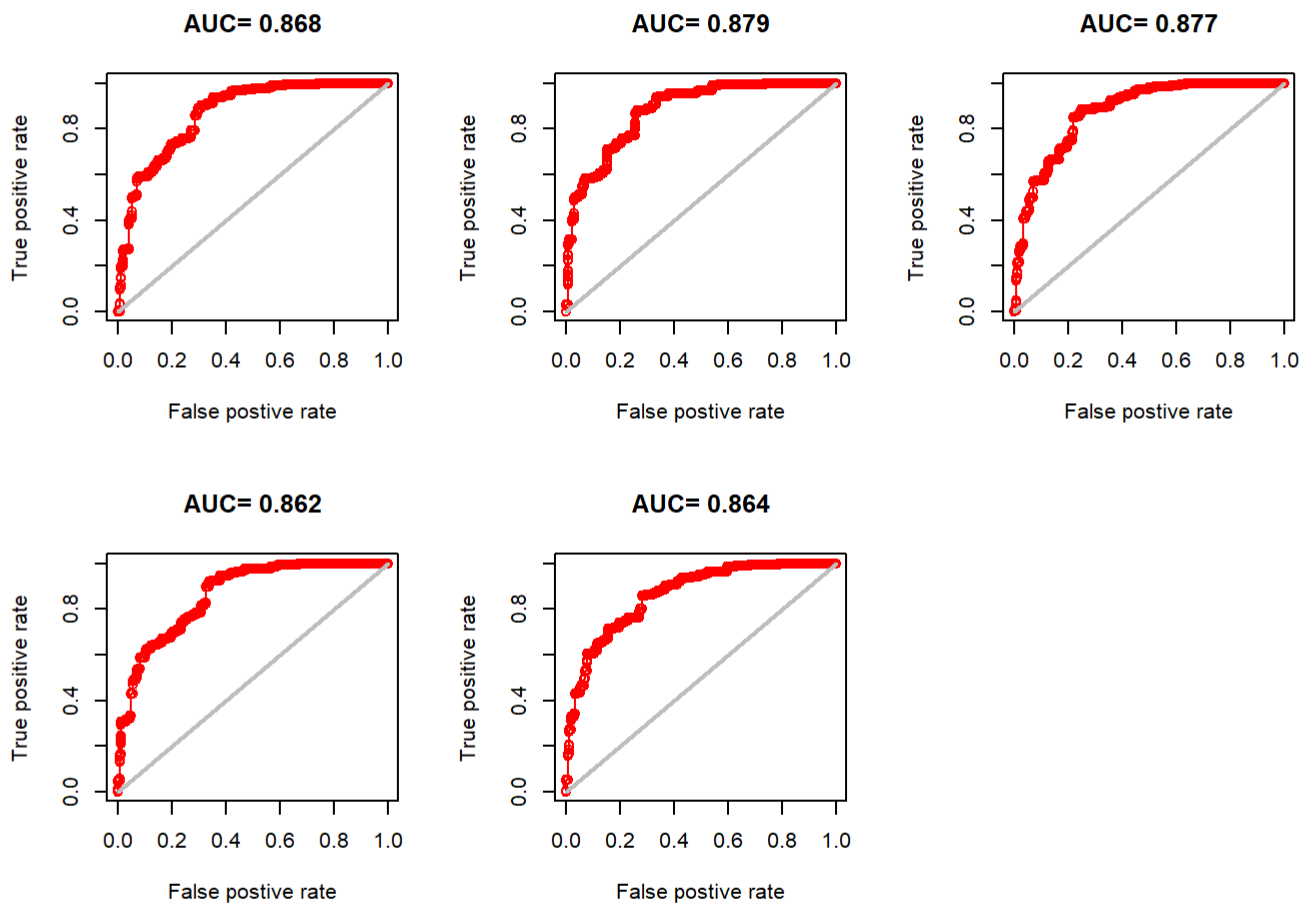
```
prBIO <- predict(env, bc_bradypus_K,type = "response")
par(mfrow=c(1,2))
plot(prBIO, main='BIO_K')
plot(wrld_simpl, add=TRUE, border='dark grey')
plot(prBIO > Mean_OptBIO, main='presence/absence')
plot(wrld_simpl,add=TRUE, border ='dark grey')
```



Now we'll do exactly the same for our GLM model, only remember that this alorithm takes presence and absence data so we'll modify our for loop slightly.

```r
#create an empty list to hold our results (remember there will be five sets)
eGLM<-list()
par(mfrow=c(2,3))

for (i in 1:folds) {
  train <- Pres.cov[kfold_pres!= i,]
  test <- Pres.cov[kfold_pres == i,]
  backTrain<-Back.cov[kfold_back!=i,]
  backTest<-Back.cov[kfold_back==i,]
  dataTrain<-rbind(train,backTrain)
  dataTest<-rbind(test,backTest)
  glm_eval <- glm(Pres~.,binomial(link = "logit"), data=dataTrain)#this is our glm model trained on presence and absence points
  eGLM[[i]] <- evaluate(p=dataTest[ which(dataTest$Pres==1),],a=dataTest[which(dataTest$Pres==0),], glm_eval)#use testing data (kfold==i) for model evaluation

  #check the AUC by plotting ROC values

  plot(eGLM[[i]],'ROC')

}


#inspect

eGLM
```

```
## [[1]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.8675467
## cor            : 0.6133703
## max TPR+TNR at : 0.3734796
##
## [[2]]
## class          : ModelEvaluation
## n presences    : 386
## n absences     : 183
## AUC            : 0.8793709
## cor            : 0.6393888
## max TPR+TNR at : 0.5073879
##
## [[3]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.8765663
## cor            : 0.6230226
## max TPR+TNR at : 0.311382
##
## [[4]]
## class          : ModelEvaluation
## n presences    : 386
## n absences     : 183
## AUC            : 0.8623687
## cor            : 0.6146204
## max TPR+TNR at : 0.4364676
##
## [[5]]
## class          : ModelEvaluation
## n presences    : 385
## n absences     : 182
## AUC            : 0.8636792
## cor            : 0.6085366
## max TPR+TNR at : 0.5904025
```



Get model evaluation statistics:

```
aucGLM <- sapply( eGLM, function(x){slot(x, 'auc')} )

#calculate the mean values for coparison with other models


mean(aucGLM)
```

```
## [1] 0.8699064
```

Again we use the sapply() function to iterate over each item in our eGLM list and select the ‘t’ value (this is the slot on eGLM which denotes a given threshold) for which corressponding values for TPR+TNR are highest (hence we use ‘which.max’).

```
Opt_GLM<-sapply( eGLM, function(x){ x@t[which.max(x@TPR + x@TNR)] } )

Opt_GLM
```

```
## [1] 0.3734796 0.5073879 0.3113820 0.4364676 0.5904025
```

We can now use these threshold values to make final predictions for areas of 1 and 0 (presence/absence) and map them. You may be wondering why our some of the Opt_GLM values are <0 rather than somewhere between 0 and 1. This is because the output of the binomial (logistic) regression is on a log-scale. We can back-transform these values to approximate probabilities (i.e. values ranging from 0 to 1) using the plogis() function like so:

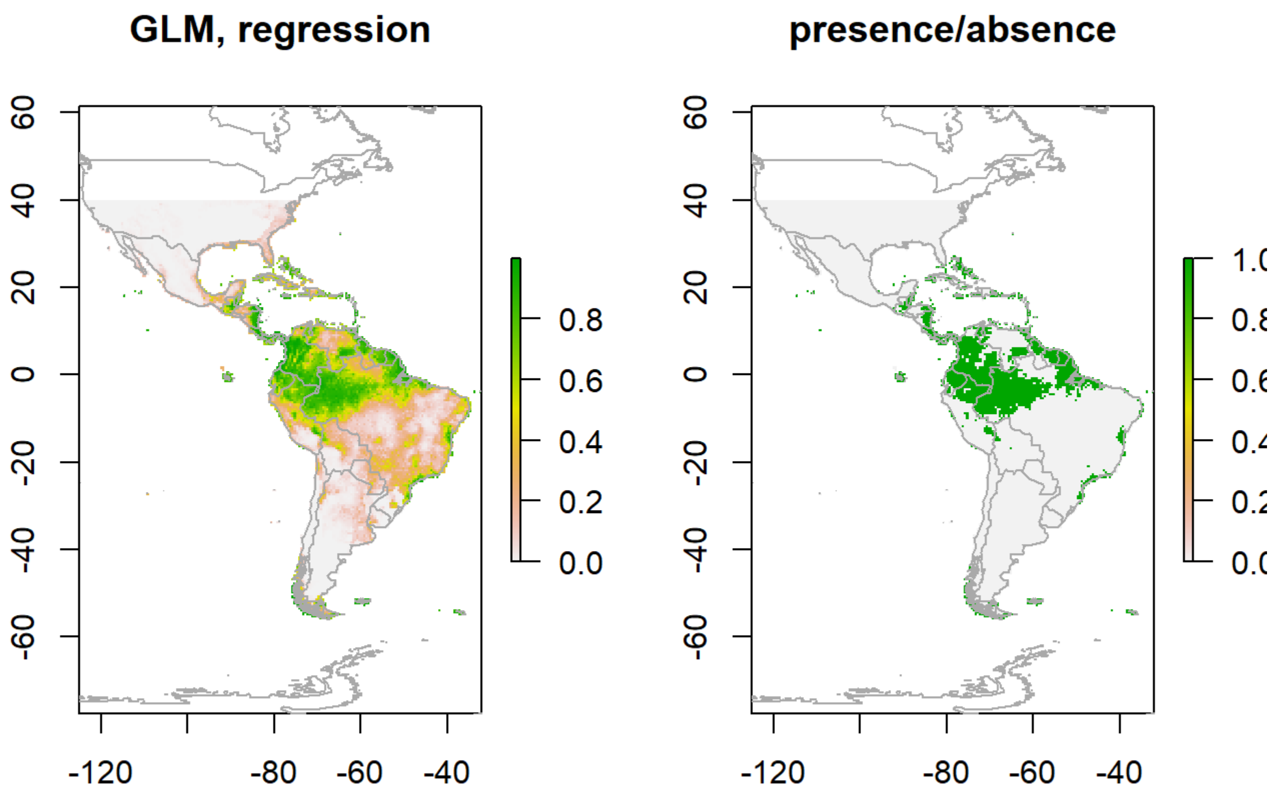
```
#take the mean to be applied to our predictions

Mean_OptGLM<- mean(Opt_GLM)

trGLM<-plogis(Mean_OptGLM)

trGLM
```

```
## [1] 0.6091698
```

We now have a value with which to set the threshold for our presence/absence maps.

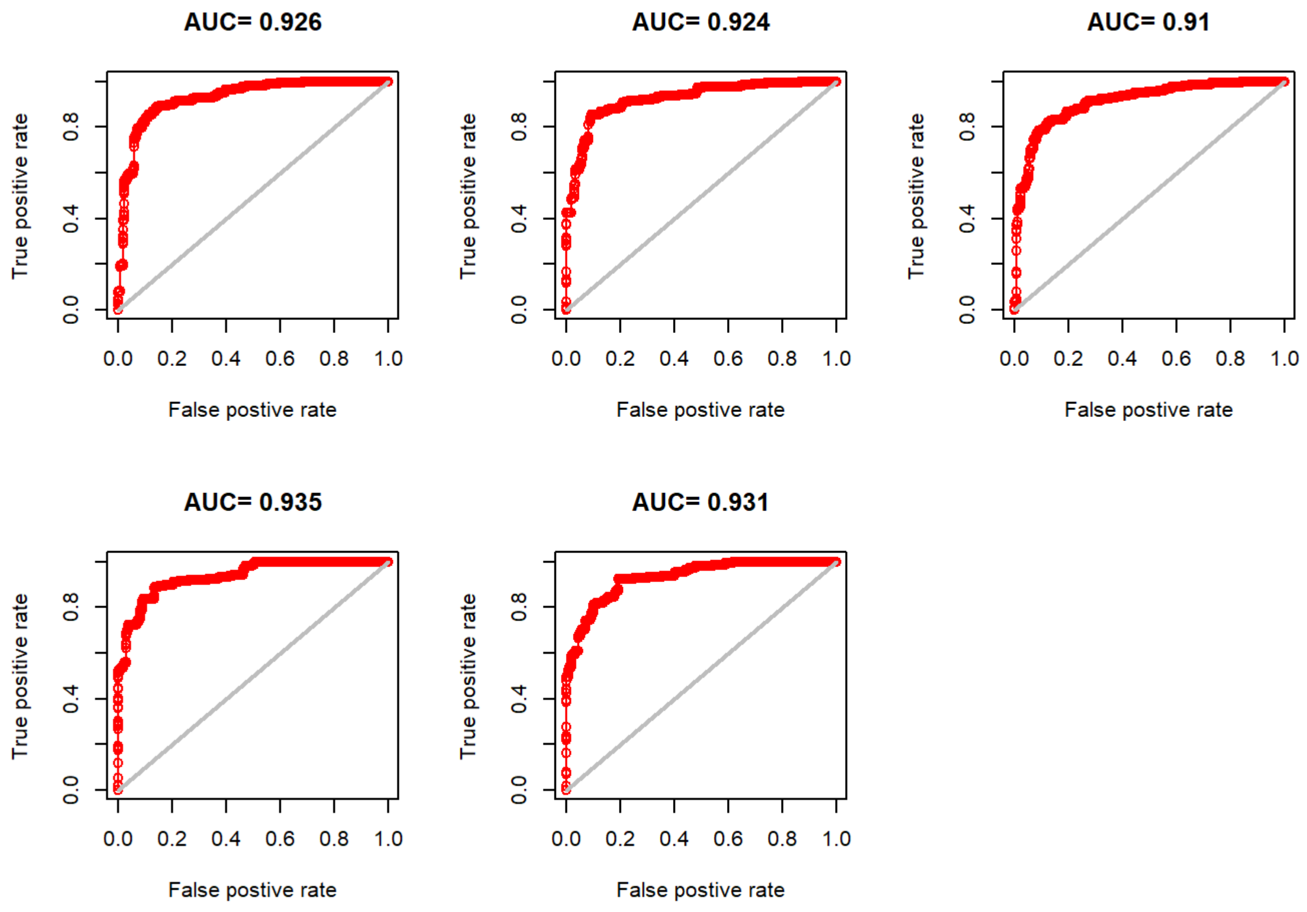
We use the predict() function to estimate our output based on the 'env' raster stack and the information from our glm_eval model. For plotting purposes we then restict the output to show only areas above our threshold (presence), leaving all other areas blank.

```
prGLM <- predict(env, glm_eval,type = "response")
par(mfrow=c(1,2))
plot(prGLM, main='GLM, regression')
plot(wrld_simpl, add=TRUE, border='dark grey')
plot(prGLM > trGLM, main='presence/absence')
plot(wrld_simpl,add=TRUE, border ='dark grey')
```



We can perform the same evaluation process on our maxent model also (we will leave k-fold validation of the lesser-performing bioclim model).

```
par(mfrow=c(2,3))

eMAX<-list()

folds=5

kfold_pres <- kfold(Pres.cov, folds)
kfold_back <- kfold(Back.cov, folds)


for (i in 1:folds) {
  train <- Pres.cov[kfold_pres!= i,]
  test <- Pres.cov[kfold_pres == i,]
  backTrain<-Back.cov[kfold_back!=i,]
  backTest<-Back.cov[kfold_back==i,]
  dataTrain<-rbind(train,backTrain)
  dataTest<-rbind(test,backTest)
  maxnet_eval <- maxnet(dataTrain$Pres, dataTrain[,1:8])
  eMAX[[i]] <- evaluate(p=dataTest[which(dataTest$Pres==1),],a=dataTest[which(dataTest
$Pres==0),], maxnet_eval)
  plot(eMAX[[i]],'ROC')

}
```



Again, we can access our AUC and MaxTPR+TNR statistics for this latest model (for reference, the equivalent AUC evaluation for the same analysis using the stand-alone Maxent package was 0.901).

```
aucMAX <- sapply( eMAX, function(x){slot(x, 'auc')} )

#calculate the mean values for comparison with other models

aucMAX
```

```
## [1] 0.9259883 0.9237946 0.9101613 0.9353323 0.9305552
```

```
mean(aucMAX)
```

```
## [1] 0.9251663
```

```
#Get maxTPR+TNR for the maxnet model

Opt_MAX<-sapply( eMAX, function(x){ x@t[which.max(x@TPR + x@TNR)] } )

Opt_MAX
```

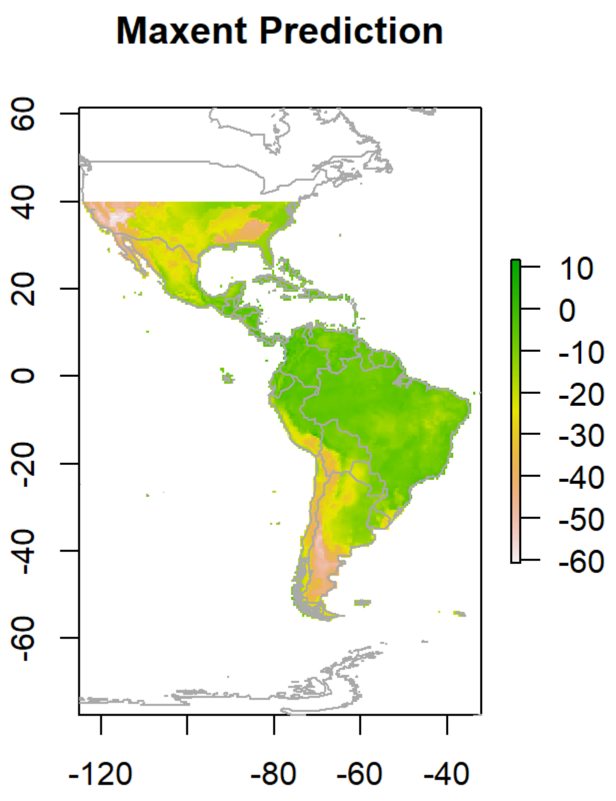
```
## [1] -6.159259 -5.688821 -5.617520 -6.009064 -6.264698
```

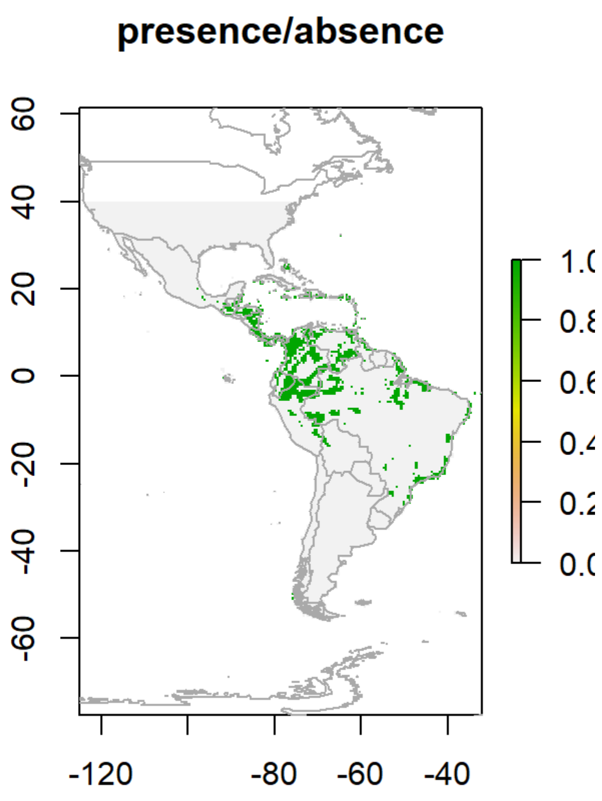
```
Mean_OptMAX<-mean(Opt_MAX)

Mean_OptMAX
```

```
## [1] -5.947872
```

The threshold values for the maxnet model relate to the raw outputs (prior to transformation). We can achieve our presence/absence map simply by applying our threshold to these raw values. To do so, we map the raw results of the model (i.e, this time we do not opt for a ‘cloglog’ or other transform) and then remove all values below our threshold.

```
prMAX <- predict(env, maxnet_eval)
par(mfrow=c(1,2))
plot(prMAX, main='Maxent Prediction')
plot(wrld_simpl, add=TRUE, border='dark grey')
plot(prMAX > Mean_OptMAX, main='presence/absence')
plot(wrld_simpl,add=TRUE, border ='dark grey')
```





Finally we will look at how to fit a random forest model to the data using the randomForest() package. The randomForest model takes two arguments - parameters that can be tuned - mtry and ntree. mtry refers to the number of predictor variables to try at each iteration of the model and and ntree specifies the number of trees grown. 500 is the default value often used and will suffices for the size of our data. Model performance is though to be more sensitive to mtry than ntree and this can be tuned with the tuneRF function:

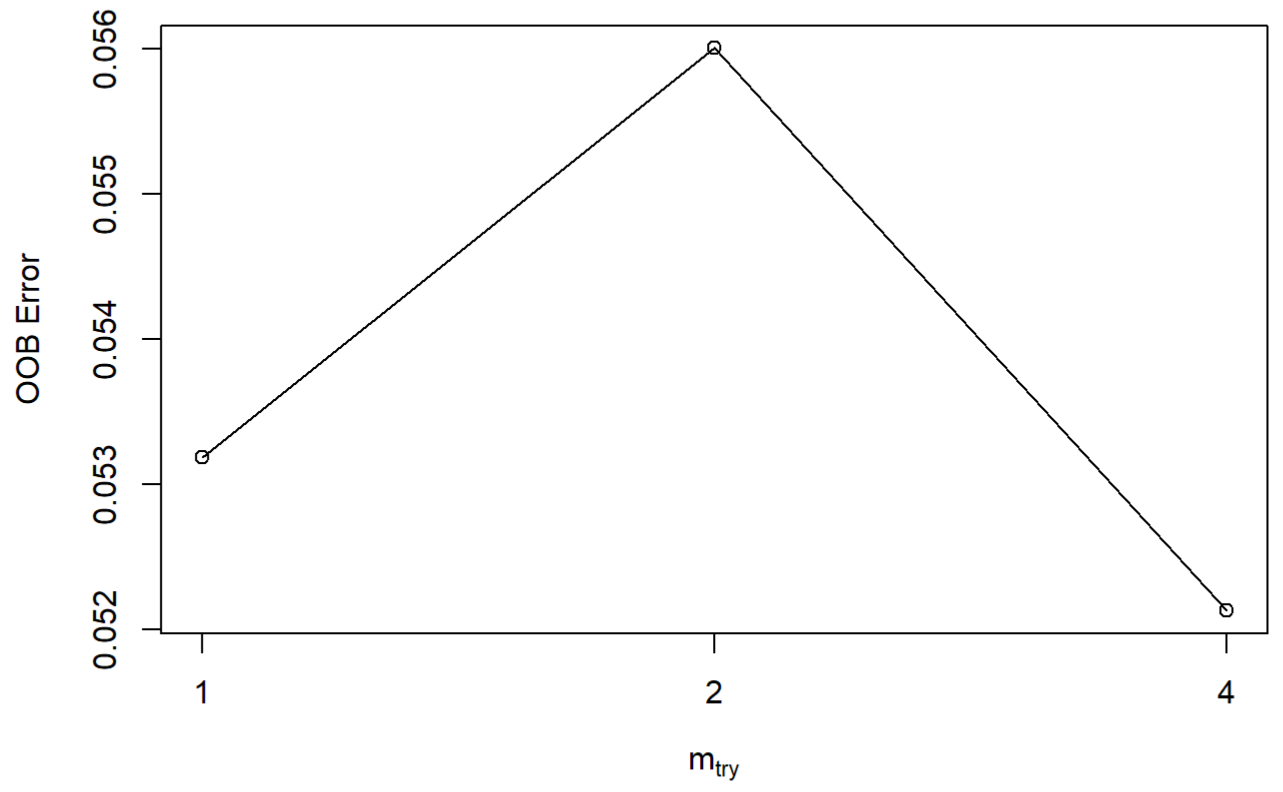
```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
tuneRF(x=all.cov[,1:8],y=as.factor(all.cov$Pres))
```

```
## mtry = 2  OOB error = 5.6%
## Searching left ...
## mtry = 1     OOB error = 5.32%
## 0.05031447 0.05
## Searching right ...
## mtry = 4     OOB error = 5.21%
## 0.01986755 0.05
```



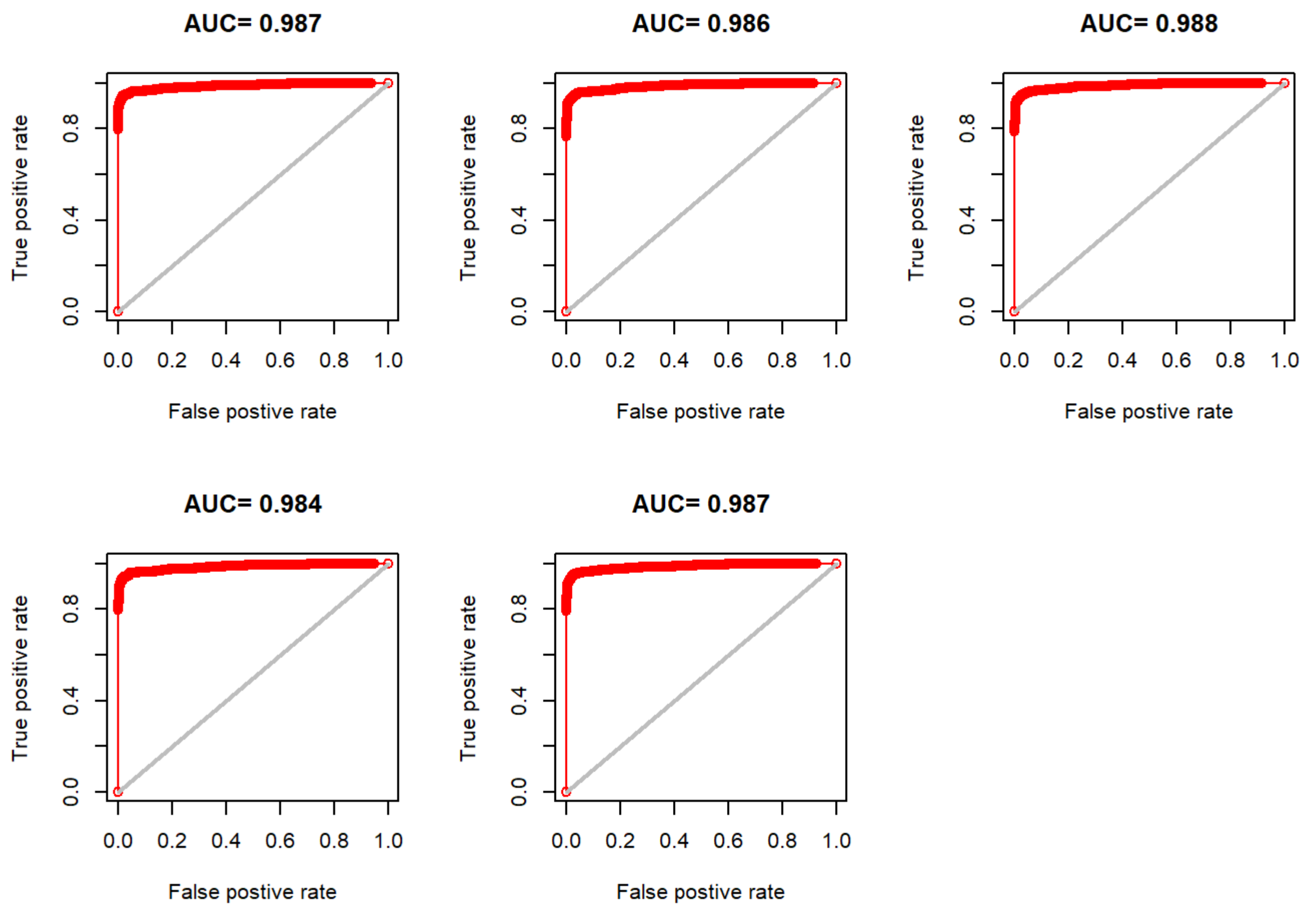
```
##       mtry   OOBError
## 1.OOB    1 0.05318774
## 2.OOB    2 0.05600564
## 4.OOB    4 0.05213103
```

```
rf.Bradypus<-randomForest(as.factor(Pres)~.,mtry=4,ntree=500,data=all.cov)
```

To evaluate our random forest model we can still use the evaluate() function from the Dismo package but in this case we first make a prediction using the model output. This is because random forest, as used here, is a classifier and so we have to give it the response variable as a factor. However, factors don’t work well with ROC calculations so the predict() function here allows us to first make the predictions (i.e. where all predicted and actual values are returned as numerics rather than factors) before passing these to the evaluate() function.

```r
#create an empty list to hold our results (remember there will be five sets)
eRF<-list()
par(mfrow=c(2,3))

for (i in 1:folds) {
  train <- Pres.cov[kfold_pres!= i,]
  test <- Pres.cov[kfold_pres == i,]
  backTrain<-Back.cov[kfold_back!=i,]
  backTest<-Back.cov[kfold_back==i,]
  dataTrain<-rbind(train,backTrain)
  dataTest<-rbind(test,backTest)
  RF_eval <- randomForest(as.factor(Pres)~., data=dataTrain)#this is our RF model
  rf.pred <- predict(RF_eval, type="prob")[,2]#make prediction
  eRF[[i]]<-evaluate(p = rf.pred[which(dataTrain$Pres == "1")], #set p to be 1s from t
he dataTrain object
         a = rf.pred[which(dataTrain$Pres == "0")])# set a to be 0s from the dataTrain
object

  #check the AUC by plotting ROC values

  plot(eRF[[i]],'ROC')

}


#inspect

eRF
```

```
## [[1]]
## class          : ModelEvaluation
## n presences    : 1542
## n absences     : 730
## AUC            : 0.9865061
## cor            : 0.9056285
## max TPR+TNR at : 0.7581046
##
## [[2]]
## class          : ModelEvaluation
## n presences    : 1541
## n absences     : 729
## AUC            : 0.9857187
## cor            : 0.90009
## max TPR+TNR at : 0.7148557
##
## [[3]]
## class          : ModelEvaluation
## n presences    : 1542
## n absences     : 730
## AUC            : 0.9875522
## cor            : 0.9053616
## max TPR+TNR at : 0.7496618
##
## [[4]]
## class          : ModelEvaluation
## n presences    : 1541
## n absences     : 729
## AUC            : 0.9844689
## cor            : 0.899205
## max TPR+TNR at : 0.7675153
##
## [[5]]
## class          : ModelEvaluation
## n presences    : 1542
## n absences     : 730
## AUC            : 0.9865901
## cor            : 0.9039349
## max TPR+TNR at : 0.704081
```



Get model evaluation statistics:

```
aucRF <- sapply( eRF, function(x){slot(x, 'auc')} )

#calculate the mean values for comparison with other models


mean(aucRF)
```

```
## [1] 0.9861672
```

```
#Get maxTPR+TNR for the Random Forest model

Opt_RF<-sapply( eRF, function(x){ x@t[which.max(x@TPR + x@TNR)] } )

Opt_RF
```

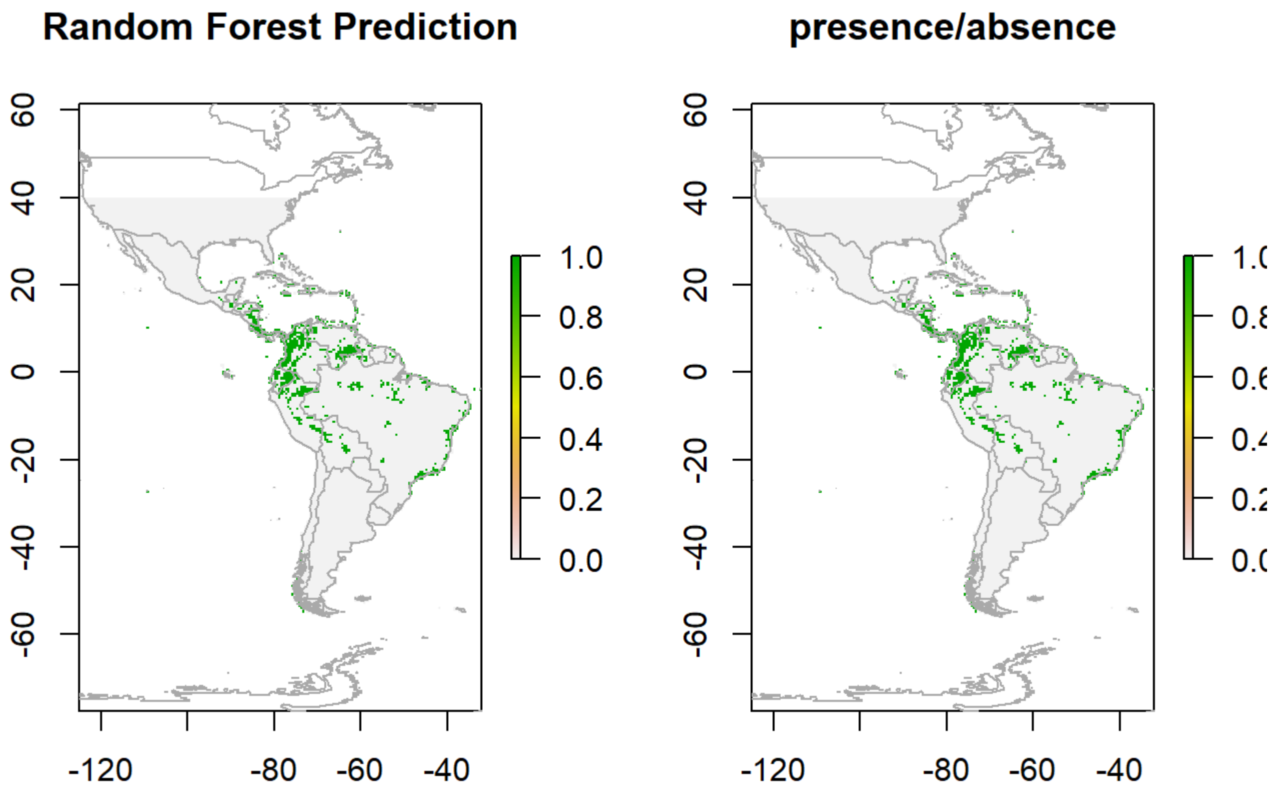
```
## [1] 0.7581046 0.7148557 0.7496618 0.7675153 0.7040810
```

```
Mean_OptRF<-mean(Opt_RF)

Mean_OptRF
```

```
## [1] 0.7388437
```

We can predict and map our distribution estimate as for the other models:

```
prRF <- predict(env, RF_eval)
par(mfrow=c(1,2))
plot(prRF, main='Random Forest Prediction')
plot(wrld_simpl, add=TRUE, border='dark grey')
plot(prRF > Mean_OptRF, main='presence/absence')
plot(wrld_simpl,add=TRUE, border ='dark grey')
```



Now we have a clearer view of the quality of our models, with the random forest model performing better overall. Both the glm and maxnet approaches still demonstrated good predictive power even after the k-fold validation. Next week we will look at how this method of cross-validation may still be overly optimistic given that we have not accounted for spatial autocorrelation in our evaluation, and at methods for handling such problems.

For now, proceed to this week's independent exercises.

Week 4 Exercise

1. Use the gbif() function to download records for *Solanum acaule* and generate species distribution models using the maxnet() and glm() approaches. Use the same environment data and approach to backgroud sampling as used in this practical. Evaluate your models uing a k-fold approach based on 5 folds. Select the best performing model and create 2 mapped outputs - 1. a probablility plot of your predictions and 2. a presence/absence map based on a sensible threshold.

2. Explain the importance of species distribution modelling in the context of nature conservation. Summarize the opportunities and challenges of applying species distribution models to species presence data (300 words).