Making a Prediction Map using leaflet()

14 March 2024

This document outlines how to plot Maxent predictions onto a `leaflet` map — which differs from the `ggplot` version in that it displays the predictions over a base map. Most of this was covered by Jeremy in class on 3/12, so all the code until the leaflet section should be what was covered in class.

Install and Load Packages

These packages will be necessary for performing the code that follows, so here we will install (if not done already) and load them:

Installation (only needs to be done once)

```
install.packages("dismo")
install.packages("dplyr")
install.packages("geodata")
install.packages("rJava")
install.packages("ggplot2")
install.packages("leaflet")
install.packages("maps")
install.packages("mapview")
install.packages("webshot2")
```

Load libraries (needs to be done every session)

```
library(dismo)
library(dplyr)
library(geodata)
library(rJava)
library(ggplot2)
library(leaflet)
library(maps)
library(mapview)
library(webshot2)
```

Predictions from Maxent

Current SDM

I believe the only difference between the current and future SDMs is which climate (WorldClim or CMIP6) is being used, so I will only put the current SDM instructions in this document.

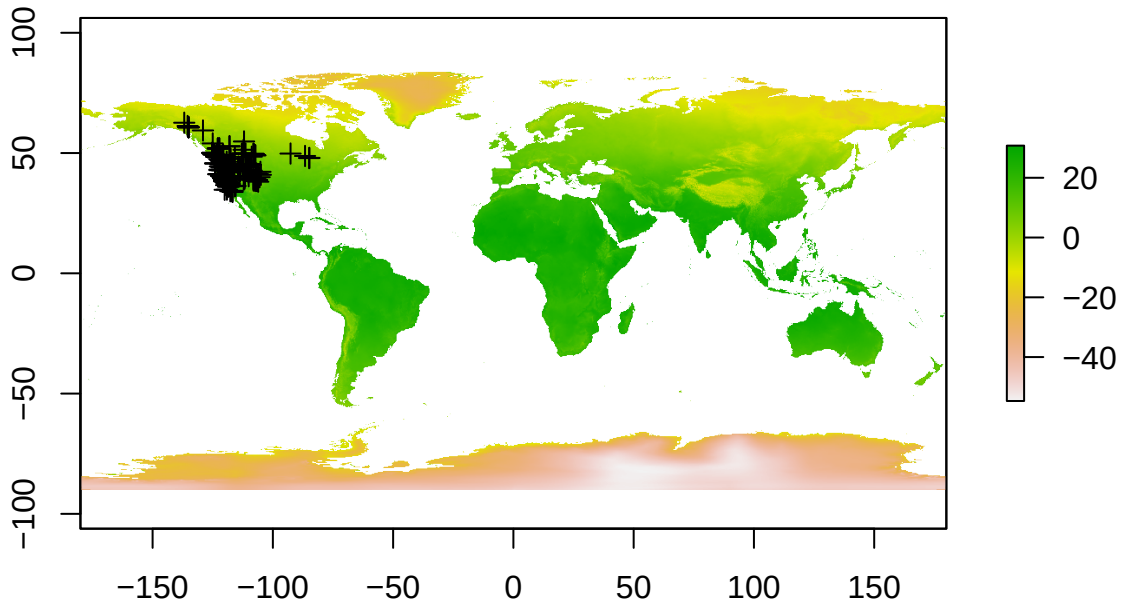
1. Get occurrence data

```
# Get latitude and longitude from our species data
occurrenceCoords<-read.csv("data/cleanedData.csv") %>%
  dplyr::select(decimalLongitude, decimalLatitude)

# Convert to spatial points, necessary for modelling and mapping
occurrenceSpatialPts <- SpatialPoints(occurrenceCoords,
                                      proj4string = CRS("+proj=longlat"))
# Now get the climate data

# Make sure RAM is bumped up (I'm using 16GB)

# This command downloads 19 different raster files, one for each climate variable
# raster: image file comprised of pixels (locations), with each pixel associated with some value (e.g.

# worldclim_global(var = "bio", res = 2.5, path = "data/", version = "2.1")

# Update .gitignore to prevent huge files getting pushed to github!
# Add: data/wc2.1_2.5m/
```

Here are the meanings of the bioclimatic variables (bio1 to bio19) provided by WorldClim:

- bio1: Mean annual temperature
- bio2: Mean diurnal range (mean of monthly (max temp - min temp))
- bio3: Isothermality (bio2/bio7) (*100)
- bio4: Temperature seasonality (standard deviation *100)
- bio5: Max temperature of warmest month
- bio6: Min temperature of coldest month
- bio7: Temperature annual range (bio5-bio6)
- bio8: Mean temperature of wettest quarter
- bio9: Mean temperature of driest quarter
- bio10: Mean temperature of warmest quarter
- bio11: Mean temperature of coldest quarter
- bio12: Annual precipitation
- bio13: Precipitation of wettest month
- bio14: Precipitation of driest month
- bio15: Precipitation seasonality (coefficient of variation)
- bio16: Precipitation of wettest quarter
- bio17: Precipitation of driest quarter
- bio18: Precipitation of warmest quarter
- bio19: Precipitation of coldest quarter

```r
# We need to feed the model a "raster stack", or a bunch of rasters laid on top of each other.

# Let's make a list of the raster files:
climList <- list.files(path = "data/wc2.1_2.5m/",
                       pattern = ".tif$",
                       full.names = T)

# Stacking the bioclim variables to process them at one go
# Each "layer" of the stack is a file for one environmental variable
currentClimRasterStack <- raster::stack(climList)

# Show one environmental layer ( = annual precepitation)
plot(currentClimRasterStack[[1]])
plot(occurrenceSpatialPts, add = TRUE)
```



Looks good, we can see where our data is.

2. Create pseudo-absence points

```r
# First we need a raster layer to make the points up on, just picking 1
# mask: raster object that determines the area where we are generating pts
mask <- raster(climList[[1]])

# Determine geographic extent of our data
# (so we generate random points reasonably nearby)
```

```r
geographicExtent <- extent(x = occurrenceSpatialPts)

# Random points for background (same number as our observed points we will use)
# Seed set so we get the same background points each time we run this code
set.seed(45)

# When you set the seed to a specific value,
# R will generate the same sequence of random numbers every time you run your code,
# provided that you use the same seed value. This is useful for debugging,
# sharing code, and ensuring that others can reproduce your results exactly.

backgroundPoints <- randomPoints(mask = mask,
                                 n = nrow(occurrenceCoords), # same n
                                 ext = geographicExtent,
                                 extf = 1.25, # draw a slightly larger area
                                 warn = 0)

# Add col names (can click and see right now they are x and y)
colnames(backgroundPoints) <- c("longitude", "latitude")
```

3. Convert occurrence and environmental data into format for model

```r
# Data for observation sites (presence and background), with climate data
# Creates grids of climate measurements, per point
occEnv <- na.omit(raster::extract(x = currentClimRasterStack, y = occurrenceCoords))

# Same for absence
absenceEnv<- na.omit(raster::extract(x = currentClimRasterStack, y = backgroundPoints))

# (0 = abs, 1 = pres)
# Bunch of true/false... we'll use later
presenceAbsenceV <- c(rep(1, nrow(occEnv)), rep(0, nrow(absenceEnv)))

# Create a single data frame with both presence and absence points for model training
presenceAbsenceEnvDf <- as.data.frame(rbind(occEnv, absenceEnv))
```

4. Current SDM with dismo::maxent

```r
# Create a new folder called maxent_outputs
# If you get a Java error, restart R, and reload the packages
habronattusCurrentSDM <-
  dismo::maxent(x = presenceAbsenceEnvDf,    # env conditions
                p = presenceAbsenceV,         # 1:presence or 0:absence
                path = paste("maxent_outputs"))     # maxent output dir
```

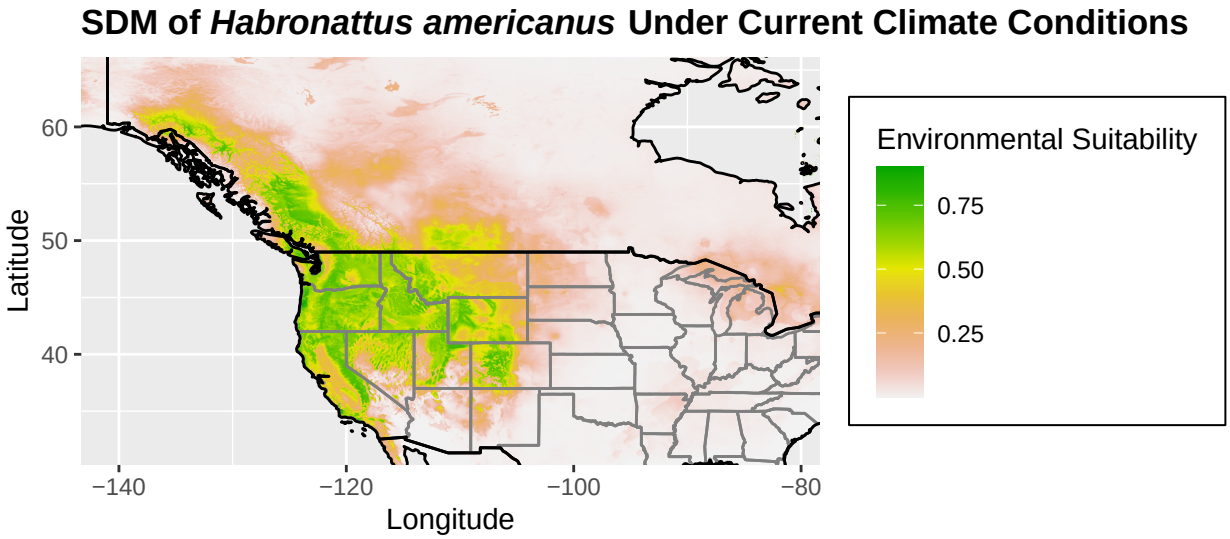
Plotting with ggplot

The ggplot version requires the results to be turned into a dataframe object first.

```r
# currentClimRasterStack is huge and it isn't reasonable to predict over
# whole world
# First we will make it smaller

# Choose here what is reasonable for your pts (where you got background pts from)
predictExtent <- 1.25 * geographicExtent
geographicArea <- crop(currentClimRasterStack, predictExtent, snap = "in")

# Crop currentClimRasterStack to the extent of the map you want
# Makes a RasterLayer with a prediction based on a a fitted model object.
habronattusPredictPlot <- raster::predict(habronattusCurrentSDM, geographicArea)

# For ggplot, we need the prediction to be a data frame
raster.spdf <- as(habronattusPredictPlot, "SpatialPixelsDataFrame")
habronattusPredictDf <- as.data.frame(raster.spdf)
```

Now we can plot it!

```r
# Plot in ggplot
wrld <- ggplot2::map_data("world")

# Determine the bounding box of the map
xmax <- max(habronattusPredictDf$x)
xmin <- min(habronattusPredictDf$x)
ymax <- max(habronattusPredictDf$y)
ymin <- min(habronattusPredictDf$y)

# Using ggplot, b/c I couldn't get it to work in leaflet :(
ggplot() +
  geom_polygon(data = wrld, mapping = aes(x = long, y = lat, group = group),
               fill = "grey75") +
  geom_raster(data = habronattusPredictDf, aes(x = x, y = y, fill = layer)) +
  scale_fill_gradientn(colors = terrain.colors(10, rev = T)) +
  coord_fixed(xlim = c(xmin, xmax), ylim = c(ymin, ymax), expand = F) +
  scale_size_area() +
  borders("state") +
  borders("world", colour = "black", fill = NA) +
  labs(title = bquote(paste(bold("SDM of"),
                            bolditalic(" Habronattus americanus"),
                            bold(" Under Current Climate Conditions"))),
       x = "Longitude",
       y = "Latitude",
       fill = "Environmental Suitability")+
  theme(legend.box.background = element_rect(),
        legend.box.margin = margin(5,5,5,5))
```




```
# Save the file
# Dimensions may require some tinkering
ggsave("output/habronattusCurrentSdm.jpg", width = 35, height = 25, units = "cm")
```

Plotting with `leaflet`

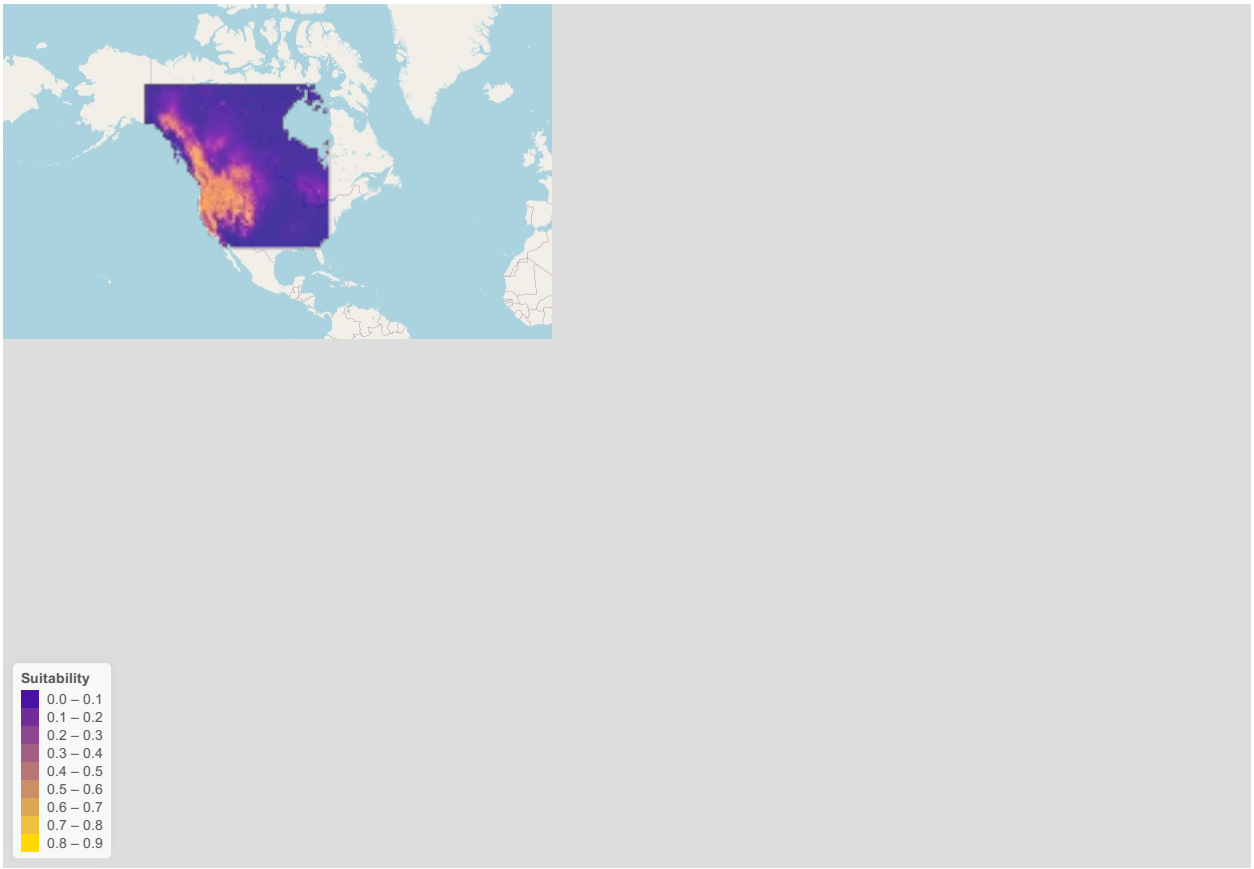
Using leaflet requires the raster result prior to conversion in addition to the data frame version. Here I will present two options for mapping.

1. No modifications to suitability results:

```
# Color palette for legend
pal <- colorBin(palette = c("#4A12A4", "#FFD900"),
                    domain = habronattusPredictDf$layer,
                    bins = 10)

# Plot map
map1 <-
leaflet(options = leafletOptions(zoomControl = FALSE, # Removes zoom button
                                 attributionControl = FALSE, # Removes banner
                                 maxZoom = 19)) %>%
  # Add basemap
  addProviderTiles("OpenStreetMap.Mapnik") %>%
  # Add Maxent results
```

```
  addRasterImage(habronattusPredictPlot,
                 colors = viridis::plasma(99),
                 opacity = 0.80) %>%
  # Add legend
  addLegend(position = "bottomleft",
            pal = pal,
            values = habronattusPredictDf$layer,
            title = "Suitability",
            opacity = 1)

map1
```

```
## Google Chrome was not found. Try setting the 'CHROMOTE_CHROME' environment variable to the executable
```

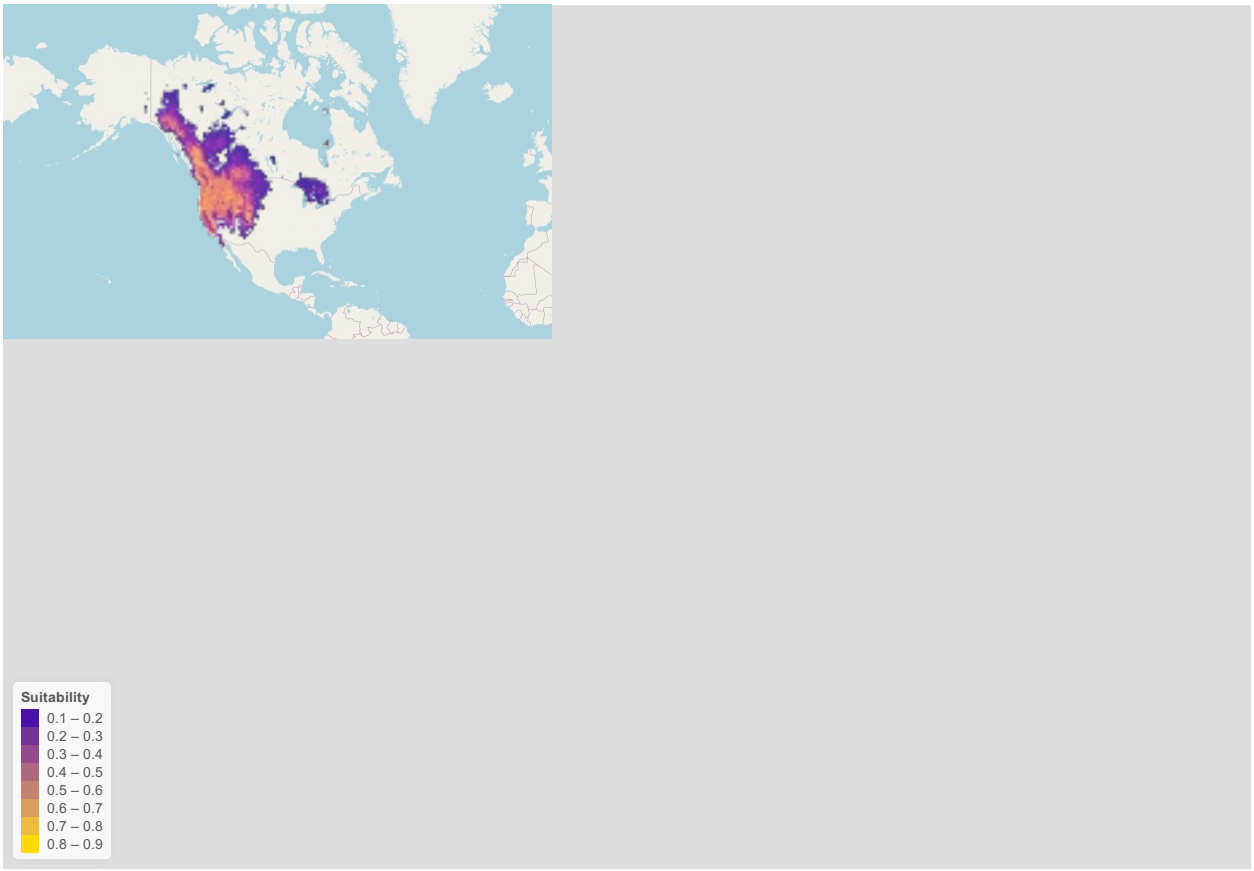



```
# Save this map
# mapshot2(map1, file = "output/habro_current_sdm_version1.png")
```

2. Limiting result presentation to those only above 0.1 suitability:

   For better visiblity of the map under the results, we can choose to omit suitability values that are below 0.10. ***Pay attention to whether the raster or data frame version of our results is being used***.

```r
# We'll use a copy of the original in case we ever decide we don't want to do that
habronattusPredictPlot_mod <- habronattusPredictPlot

# Then we remove the values below 0.1
habronattusPredictPlot_mod[habronattusPredictPlot_mod < 0.10] <- NA

# Convert this to a data frame (for the legend)
raster.spdf2 <- as(habronattusPredictPlot_mod, "SpatialPixelsDataFrame")
habronattusPredictDf_mod <- as.data.frame(raster.spdf2)

# Color palette for legend
pal2 <- colorBin(palette = c("#4A12A4", "#FFD900"),
                 domain = habronattusPredictDf_mod$layer,
                 bins = 10)

# Plot map
map2 <-
leaflet(options = leafletOptions(zoomControl = FALSE, # Removes zoom button
                                 attributionControl = FALSE, # Removes banner
                                 maxZoom = 19)) %>%
  # Add basemap
  addProviderTiles("OpenStreetMap.Mapnik") %>%
  # Add Maxent results
  addRasterImage(habronattusPredictPlot_mod,
                 colors = viridis::plasma(99),
                 opacity = 0.80) %>%
  # Add legend
  addLegend(position = "bottomleft",
            pal = pal2,
            values = habronattusPredictDf_mod$layer,
            title = "Suitability",
            opacity = 1)

map2
```




```
# Save this map
# mapshot2(map1, file = "output/habro_current_sdm_version2.png")
```