SMT1
Script para el cálculo de cobertura arbórea
SMT1
Script to calculate tree cover

```
// Get the forest loss image.
var gfc2014 = ee.Image('UMD/hansen/global_forest_change_2021_v1_9');
var lossImage = gfc2014.select(['loss']);
var areaImage = lossImage.multiply(ee.Image.pixelArea());

// Sum the values of forest loss pixels.
var stats = areaImage.reduceRegion({
 reducer: ee.Reducer.sum(),
 geometry: table,
 scale: 30,
 maxPixels: 1e9
});
print('pixels representing loss: ', stats.get('loss'), 'square meters');


// Get the loss image.
// This dataset is updated yearly, so we get the latest version.
var gfc2021g = ee.Image('UMD/hansen/global_forest_change_2021_v1_9');
var lossImage = gfc2021g.select(['loss']);
var lossAreaImage = lossImage.multiply(ee.Image.pixelArea());

var lossYear = gfc2021g.select(['lossyear']);
var lossByYear = lossAreaImage.addBands(lossYear).reduceRegion({
 reducer: ee.Reducer.sum().group({
groupField: 1
 }),
 geometry: table,
 scale: 30,
maxPixels: 1e9
});
print(lossByYear);


var statsFormatted = ee.List(lossByYear.get('groups'))
.map(function(el) {
 var d = ee.Dictionary(el);
 return [ee.Number(d.get('group')).format("20 %02d"), d.get('sum')];
 });
var statsDictionary = ee.Dictionary(statsFormatted.flatten());
print(statsDictionary);


var chart = ui.Chart.array.values({
 array: statsDictionary.values(),
 axis: 0,
xLabels: statsDictionary.keys()
}).setChartType('ColumnChart')
.setOptions({
 title: 'Yearly Forest Loss',
hAxis: {title: 'Year', format: '####'},
vAxis: {title: 'Area (square meters)'},
 legend: { position: "none" },
lineWidth: 1,
pointSize: 3
});
print(chart);
```

SMT2
Script del modelo empleado en Wallace 2.0
SMT2
Script of the model used in Wallace 2.0

```
library(spocc)
library(spThin)
library(dismo)
library(rgeos)
library(ENMeval)
library(wallace)


# just type the function name and press Return to see its source code
# paste this code into a new script to edit it
occs_queryDb

Youranalyses are below.

```

```
# NOTE: provide the folder path of the .csv file
occs_path<- ""
occs_path<- file.path(occs_path, "Presas_japu_processed_occs.csv")
# get a list of species occurrence data
userOccs_Pj<- occs_userOccs(
txtPath = occs_path,
txtName = "Presas_japu_processed_occs.csv",
txtSep = ",",
txtDec = ".")
occs_Pj<- userOccs_Pj$Presas_japu$cleaned

# Download environmental data
envs_Pj<- envs_worldclim(
bcRes = 2.5,
bcSel = c('bio01', 'bio02', 'bio03', 'bio04', 'bio05', 'bio07', 'bio12', 'bio13', 'bio14', 'bio15'),
mapCntr = c(-79.74, -1.047), # Mandatory for 30 arcsec resolution
doBrick = TRUE)
occs_xy_Pj<- occs_Pj[c('longitude', 'latitude')]
occs_vals_Pj<- as.data.frame(raster::extract(envs_Pj, occs_xy_Pj, cellnumbers = TRUE))
# Remove duplicated same cell values
occs_Pj<- occs_Pj[!duplicated(occs_vals_Pj[, 1]), ]
occs_vals_Pj<- occs_vals_Pj[!duplicated(occs_vals_Pj[, 1]), -1]
# remove occurrence records with NA environmental values
occs_Pj<- occs_Pj[!(rowSums(is.na(occs_vals_Pj)) >= 1), ]
# also remove variable value rows with NA environmental values
occs_vals_Pj<- na.omit(occs_vals_Pj)
# add columns for env variable values for each occurrence record
occs_Pj<- cbind(occs_Pj, occs_vals_Pj)

# Load the user provided shapefile or csv file with the desired extent.
 ##User must input the path to shapefile or csv file and the file name
# Define path
bgPath_Pj<- ""
bgExt_Pj<- penvs_userBgExtent(
bgShp_path = paste0(bgPath_Pj, "Occidente", ".shp"),
bgShp_name = paste0("Occidente", c(".shp", ".shx", ".dbf")),
userBgBuf = 0,
occs = occs_Pj)
# Mask environmental data to provided extent
bgMask_Pj<- penvs_bgMask(
occs = occs_Pj,
envs = envs_Pj,
bgExt = bgExt_Pj)
# Sample background points from the provided area
bgSample_Pj<- penvs_bgSample(
occs = occs_Pj,
bgMask = bgMask_Pj,
bgPtsNum = 1000)
# Extract values of environmental layers for each background point
bgEnvsVals_Pj<- as.data.frame(raster::extract(bgMask_Pj, bgSample_Pj))
##Add extracted values to background points table
bgEnvsVals_Pj<- cbind(scientific_name = paste0("bg_", "Presas japu"), bgSample_Pj,
occID = NA, year = NA, institution_code = NA, country = NA,
state_province = NA, locality = NA, elevation = NA,
record_type = NA, bgEnvsVals_Pj)

# R code to get partitioned data
groups_Pj<- part_partitionOccs(
occs = occs_Pj ,
bg = bgSample_Pj,
 method = "block",
bgMask = bgMask_Pj,
aggFact = 2)

# Run maxent model for the selected species
model_Pj<- model_maxent(
occs = occs_Pj,
bg = bgEnvsVals_Pj,
user.grp = groups_Pj,
bgMsk = bgMask_Pj,
 rms = c(0.5, 5),
rmsStep = 0.5,
 fcs = c('L', 'LQ', 'H', 'LQH'),
clampSel = TRUE,
algMaxent = "maxnet",
 parallel = FALSE,
numCores = 7)

# Select current model and obtain raster prediction
m_Pj<- model_Pj@models[["fc.H_rm.0.5"]]
predSel_Pj<- predictMaxnet(m_Pj, bgMask_Pj,
 type = "cloglog",
 clamp = TRUE)
#Get values of prediction
mapPredVals_Pj<- getRasterVals(predSel_Pj, "cloglog")
#Define colors and legend
rasCols<- c("#2c7bb6", "#abd9e9", "#ffffbf", "#fdae61", "#d7191c")
legendPal<- colorNumeric(rev(rasCols), mapPredVals_Pj, na.color = 'transparent')
rasPal<- colorNumeric(rasCols, mapPredVals_Pj, na.color = 'transparent')
#Generate map
m <- leaflet() %> %addProviderTiles(providers$Esri.WorldTopoMap)
m %> %
 leaflet::addLegend("bottomright", pal = legendPal,
 title = "Predicted Suitability<br>(Training)",
 values = mapPredVals_Pj, layerId = "train",
labFormat = reverseLabel(2, reverse_order = TRUE)) %> %
 #add occurrence data
addCircleMarkers(data = occs_Pj, lat = ~latitude, lng = ~longitude,
 radius = 5, color = 'red', fill = TRUE, fillColor = "red",
fillOpacity = 0.2, weight = 2, popup = ~pop) %> %
 ##Add model prediction
addRasterImage(predSel_Pj, colors = rasPal, opacity = 0.7,
 group = 'vis', layerId = 'mapPred', method = "ngb") %> %
 ##add background polygons
addPolygons(data = bgExt_Pj,fill = FALSE,
 weight = 4, color = "blue", group = 'proj')

#Download variables for transferring
xferTimeEnvs_Pj<- raster::getData(
 'CMIP5',
 var = "bio",
 res = round((raster::res(bgMask_Pj) * 60)[1],1),
rcp = 85,
 model = "CC",
 year = 50)
```

```
names(xferTimeEnvs_Pj) <- paste0('bio', c(paste0('0',1:9), 10:19))
# Select variables for transferring to match variables used for modelling
xferTimeEnvs_Pj<- xferTimeEnvs_Pj[[names(bgMask_Pj)]]

# Generate a transfer of the model to the desired area and time
xfer_time_Pj<-xfer_time(
evalOut = model_Pj,
curModel = "fc.H_rm.0.5",
envs = xferTimeEnvs_Pj,
xfExt = bgExt_Pj,
alg = "maxnet",
outputType = "cloglog",
 clamp = TRUE
 )
# store the cropped variables of transfer
xferExt_Pj<- xfer_time_Pj$xferExt


###Make map of transfer
bb_Pj<- bgExt_Pj@bbox
bbZoom<- polyZoom(bb_Pj[1, 1], bb_Pj[2, 1], bb_Pj[1, 2],
bb_Pj[2, 2], fraction = 0.05)
mapXferVals_Pj<- getRasterVals(xfer_time_Pj$xferTime,"cloglog")
rasCols_Pj<- c("#2c7bb6", "#abd9e9", "#ffffbf", "#fdae61", "#d7191c")
# if no threshold specified
legendPal<- colorNumeric(rev(rasCols_Pj), mapXferVals_Pj, na.color = 'transparent')
rasPal_Pj<- colorNumeric(rasCols_Pj, mapXferVals_Pj, na.color = 'transparent')
m <- leaflet() %> %addProviderTiles(providers$Esri.WorldTopoMap)
m %> %
fitBounds(bbZoom[1], bbZoom[2], bbZoom[3], bbZoom[4]) %> %
 leaflet::addLegend("bottomright", pal = legendPal,
 title = "Predicted Suitability<br>(Transferred)",
 values = mapXferVals_Pj, layerId = 'xfer',
labFormat = reverseLabel(2, reverse_order = TRUE)) %> %
# map model prediction raster and polygon of transfer
clearMarkers() %> %clearShapes() %> %removeImage('xferRas') %> %
addRasterImage(xfer_time_Pj$xferTime, colors = rasPal_Pj, opacity = 0.7,
layerId = 'xferRas', group = 'xfer', method = "ngb") %> %
 ##add polygon of transfer (same modeling area)
addPolygons(data = bgExt_Pj, fill = FALSE,
 weight = 4, color = "blue", group = 'xfer')

# R code to generate MESS raster
xferMess_Pj<- xfer_mess(
occs = occs_Pj,
bg = bgEnvsVals_Pj ,
bgMsk = bgMask_Pj,
xferExtRas = xferExt_Pj)

# Generate MESS map
rasVals_Pj<- getRasterVals(xferMess_Pj)

 # define colorRamp for mess
 if (max(rasVals_Pj) > 0 & min(rasVals_Pj) < 0) {
 rc1 <- colorRampPalette(colors = rev(RColorBrewer::brewer.pal(n = 3, name = 'Reds')),
 space = "Lab")(abs(min(rasVals_Pj)))
 rc2 <- colorRampPalette(colors = RColorBrewer::brewer.pal(n = 3, name = 'Blues'),
 space = "Lab")(max(rasVals_Pj))
rasCols_Pj<- c(rc1, rc2)
 } else if (max(rasVals_Pj) < 0 & min(rasVals_Pj) < 0) {
rasCols_Pj<- colorRampPalette(colors = rev(RColorBrewer::brewer.pal(n = 3, name = 'Reds')),
 space = "Lab")(abs(min(rasVals)))
 } else if (max(rasVals_Pj) > 0 & min(rasVals_Pj) > 0) {
rasColsPj<- colorRampPalette(colors = RColorBrewer::brewer.pal(n = 3, name = 'Blues'),
 space = "Lab")(max(rasVals_Pj))
 }
legendPal_Pj<- colorNumeric(rev(rasCols_Pj), rasVals_Pj, na.color='transparent')
rasPal_Pj<- colorNumeric(rasCols_Pj, rasVals_Pj, na.color='transparent')
 #Create map
 m <- leaflet() %> %addProviderTiles(providers$Esri.WorldTopoMap)
m %> %
```

```
 leaflet::addLegend("bottomright", pal = legendPal_Pj, title = "MESS Values",
 values = rasVals_Pj, layerId = 'xfer',
labFormat = reverseLabel(2, reverse_order=TRUE)) %> %
 # map model prediction raster and transferring polygon
clearMarkers() %> %clearShapes() %> %removeImage('xferRas') %> %
addRasterImage(xferMess_Pj, colors = rasPal_Pj , opacity = 0.9,
layerId = 'xferRas', group = 'xfer', method = "ngb") %> %
 ##add transferring polygon: this we need to fix for now please replace bgExt_Pj for the name of your transferring polygon.
addPolygons(data = bgExt_Pj, fill = FALSE,
 weight = 4, color = "blue", group = 'xfer')
```