El código utilizado

A continuación, se detalla el código en R que puede utilizarse como base y ejemplo, para comenzar a probar su potencial en este tipo de procedimientos.

```
# Remover toda la info utilizada hasta el momento.
rm(list=ls())

# Instalar los paquetes necesarios.
# Esto sólo se hace una vez, no es necesario instalarlos cada vez que se utilizan.
install.packages("rgdal")
install.packages("raster")
install.packages("caret")
install.packages("kerndwd")
install.packages("mapview")
install.packages("sp")
install.packages("RColorBrewer")
install.packages("dplyr")
install.packages("leaflet")
install.packages("sf")

# Cargar las librerías a utilizar.
library(rgdal)
library(raster)
library(caret)
library(kerndwd)
library(mapview)
library(sp)
library(RColorBrewer)
library(dplyr)
library(leaflet)
library(sf)

# Definir directorio de trabajo, acá se debe poner la ruta en donde está
# la carpeta en la cual vamos a guardar los archivos.
setwd("C:/Usuarios/Carpeta de ejemplo")

# Cargar la imagen satelital sobre la cual se realizará la clasificación.
img <- brick("Sentinel-2L1C_FALSE_COLOR_URBAN_2019-01-01_2019-06-01_-31.364413_-64.483938_-
31.294292_-64.152241_50_mostRecent.tiff")

# Definir las bandas que se utilizarán. En este caso se usan sólo 3 bandas, pero pueden ser
más.
names(img) <- paste0("B", c(1:3))

# Ver el mapa de cada una de las bandas.
plot(img)

# Cargar las muestras que serán utilizadas para la clasificación.
library(sf)
muestras <- st_read("muestras_poligonos.gpkg")

# Sólo un detalle. La función desarrollada a continuación trabaja con objetos de clase "sp",
mientras que la muestra fue cargada
# como un objeto "sf". Hay que transformarlo.
muestras = as(muestras, 'Spatial')

# Ver el mapa de las muestras
plot(muestras)
```

```r
# Definir cuál es la variable que se utilizará para clasificar.
muestras$class = as.factor(muestras$class) ; responseCol <- "class"

# Crear una base de datos paralela sobre la cual se realizará la clasificación.
pixeles = data.frame(matrix(vector(), nrow = 0, ncol = length(names(img)) + 1))

# Función que superpone las muestras encima de la imagen satelital.
for (i in 1:length(unique(muestras[[responseCol]]))){
  category <- unique(muestras[[responseCol]])[i]
  categorymap <- muestras[muestras[[responseCol]] == category,]
  dataSet <- extract(img, categorymap)
  dataSet <- dataSet[!unlist(lapply(dataSet, is.null))]
  if(is(muestras, "SpatialPointsDataFrame")){
    dataSet <- cbind(dataSet, class = as.numeric(category))
    pixeles <- rbind(pixeles, dataSet)
  }
  if(is(muestras, "SpatialPolygonsDataFrame")){
    dataSet <- lapply(dataSet, function(x){cbind(x, class = as.numeric(rep(category,
nrow(x))))})
    df <- do.call("rbind", dataSet)
    pixeles <- rbind(pixeles, df)
  }
}

# Definir cantidad de píxeles dentro de los polígonos de muestra que se utilizarán para el
análisis. Más pixeles, más precision, pero mayor tiempo de procesamiento. A modo de ejemplo,
se utilizan sólo 1000 píxeles de los 33.589 disponibles.
n <- 1000

# TOmar una muestra en base a la cantidad de píxeles definidos anteriormente.
spixeles <- pixeles[sample(1:nrow(pixeles), n ), ]

# Entrenar un modelo para clasificar los píxeles dejados fuera de la muestra y evaluar el
desempeño.
modFit_rf <- train(as.factor(class) ~ ., method = "rf", data = spixeles)
modFit_rf

# Utilizar el modelo anterior para estimar toda la imagen satelital.
beginCluster()
preds_rf <- clusterR(img, raster::predict, args = list(model = modFit_rf))
endCluster()

# Mapear resultados
maxpixels =  1e+06
m <- leaflet() %>%
  addProviderTiles('Esri.WorldImagery') %>%
  addRasterImage(preds_rf , opacity = 0.6) %>%
  addScaleBar(position = "topright") %>%
  addMiniMap( position = "bottomleft")
m

# Exportar Raster con la estimación
writeRaster(preds_rf, "estimacion.tif")
```