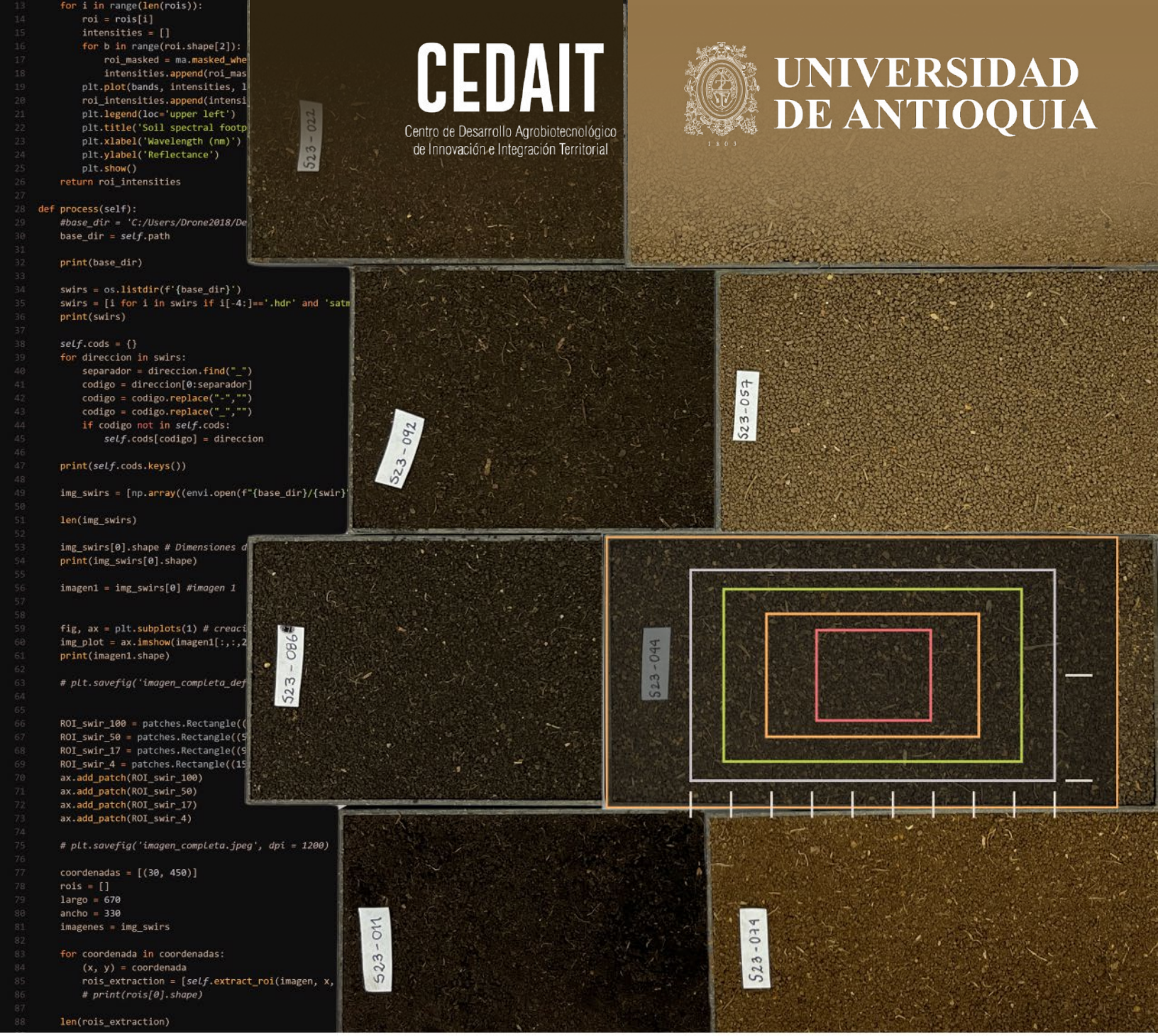


Manual para el procesamiento y comparación de imágenes hiperespectrales

Mateo Vargas-Zapata - Mario Fernando Cerón-Muñoz
Marisol Medina-Sierra - Luis Fernando Galeano-Vasco

Manual para el procesamiento y comparación de imágenes hiperespectrales

Mateo Vargas-Zapata

Mario Fernando Cerón-Muñoz

Marisol Medina-Sierra

Luis Fernando Galeano-Vasco

Esta publicación contó con el apoyo financiero del proyecto "Desarrollo y establecimiento del Centro de Desarrollo Agrobiotecnológico de Innovación e Integración Territorial, el Carmen de Viboral, Antioquia, Occidente (CEDAIT)", Componente Sistema Experto; con recursos del Sistema General de Regalías y la Gobernación de Antioquia.

Editorial UTP

Colección Trabajos de Investigación

2024

Manual para el procesamiento y comparación de imágenes hiperespectrales

Mateo Vargas-Zapata[1]

Mario Fernando Cerón-Muñoz[2]

Marisol Medina-Sierra [3]

Luis Fernando Galeano-Vasco[4]

[1] Zoot. MSc, Grupo de Investigación en Agrociencias, biodiversidad y territorio (GAMMA), Facultad de Ciencias Agrarias, Universidad de Antioquia, mateo.vargasz@udea.edu.co, https://orcid.org/0000-0003-4140-088X

[2] Zoot. MSc, DrSc, Grupo de investigación en Agrociencias, biodiversidad y territorio (GAMMA), Facultad de Ciencias Agrarias, Universidad de Antioquia, mario.ceron@udea.edu.co, https://orcid.org/0000-0002-7233-6625

[3] Ing. Agron., MSc, DrSc, Grupo de investigación en Agrociencias, biodiversidad y territorio (GAMMA), Facultad de Ciencias Agrarias, Universidad de Antioquia, marisol.medina@udea.edu.co, https://orcid.org/0000-0003-1929-8305

[4] Zoot, MSc, DrSc, Grupo de investigación en Agrociencias, biodiversidad y territorio (GAMMA), Facultad de Ciencias Agrarias, Universidad de Antioquia, luis.galeano@udea.edu.co, https://orcid.org/0000-0002-6842-3945

Manual para el procesamiento y comparación de imágenes hiperespectrales / Mateo Vargas Zapata y otros. -- Pereira : Universidad Tecnológica de Pereira, 2024.
59 páginas. -- (Colección Trabajos de Investigación).

e-ISBN: 978-958-722-932-5

1. Procesamiento de imágenes hiperespectrales 2. Lenguajes de programación 3. Innovaciones tecnológicas 4. Programación en Python 5. Análisis estadístico

CDD. 620.0028

Manual para el procesamiento y comparación de imágenes hiperespectrales

eISBN: 978-958-722-932-5

Universidad Tecnológica de Pereira
Vicerrectoría de Investigaciones, Innovación y Extensión
Editorial Universidad Tecnológica de Pereira
Pereira, Colombia

Imágen de cubierta: Selene Orozco Medina Ing. Mec.

Coordinador editorial:
Luis Miguel Vargas Valencia
luismvargas@utp.edu.co
Teléfono (606) 313 7381
Edificio 9, Biblioteca Central Jorge Roa Martínez
Cra. 27 No. 10-02 Los Álamos, Pereira, Colombia
www.utp.edu.co

Montaje y producción
Tomás Flórez Calle
Universidad Tecnológica de Pereira

Pereira, Risaralda, Colombia.

Esta publicación fue generada gracias al desarrollo de las actividades de formación académica en pregrado y posgrado. El contenido fue elaborado en R-project con Sweave y LaTeX.

Contenido

Agradecimientos

Los autores agradecen el apoyo financiero para la realización de las actividades, escritura y publicación, al proyecto con código BPIN 2016000100060 “Desarrollo y establecimiento del Centro de Desarrollo Agrobiotecnológico de Innovación e Integración Territorial, el Carmen de Viboral, Antioquia, Occidente (CEDAIT)”, Componente Sistema Experto; financiado por el Sistema General de Regalías y la Gobernación de Antioquia, liderado por la Universidad de Antioquia y la Universidad Católica de Oriente.

Prefacio

El procesamiento de imágenes es una etapa necesaria para extraer la valiosa información contenida en ellas, especialmente en el caso de imágenes hiperespectrales. Esta información puede estar relacionada con las características de los objetos a través de análisis estadísticos, los cuales suelen ser generalmente complejos.

Este manual es un complemento de lo expuesto en el libro *Captura, gestión, extracción de imágenes hiperespectrales y desarrollo de modelos para la predicción de parámetros fisicoquímicos de suelos*[1], y aborda detalladamente el procesamiento de imágenes de suelo de predios dedicados a la producción agropecuaria, mediante ejemplos prácticos en los lenguajes de programación Python y R-project.

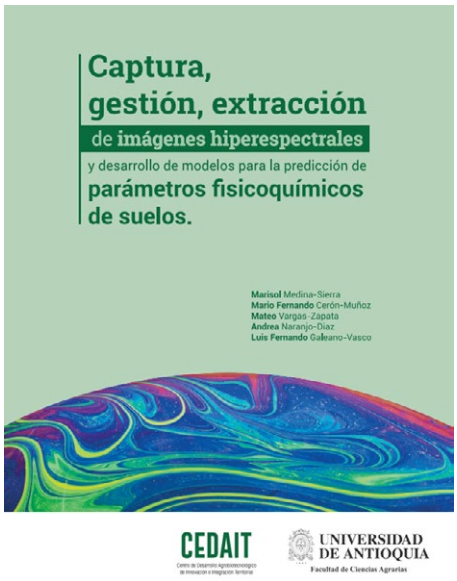
Está dirigido a estudiantes, investigadores y profesionales que deseen adquirir habilidades en el procesamiento de imágenes hiperespectrales (en especial procedentes de las cámaras hipe-

1 Marisol Medina-Sierra et al. (2023). Libro publicado por la Editorial Universidad Tecnológica de Pereira con corrección de estilo / Diseño y diagramación de la Facultad de Comunicaciones y Filología de la Universidad de Antioquia https://hdl.handle.net/11059/14891.

respectrales de la línea HySpex) y aplicarlas en sus ámbitos de trabajo o estudio. Se asume que los lectores tienen conocimientos básicos en programación y análisis de imágenes, pero no es necesario tener experiencia previa en el procesamiento de imágenes hiperespectrales.

Esperamos que sea de utilidad para aquellos interesados en el procesamiento de imágenes hiperespectrales y les proporcione las herramientas necesarias para aplicar esta técnica en sus proyectos y análisis de datos.

Obra previa *Captura, gestión, extracción de imágenes hiperespectrales y desarrollo de modelos para la predicción de parámetros fisicoquímicos de suelos*



Como se indica en el prefacio de este libro, "contiene información importante para el manejo de cámaras hiperespectrales, procesamiento de imágenes, procesamiento de bases de datos, análisis estadísticos y presentación de resultados" y es el producto de las investigaciones realizadas por el grupo GAMMA de la Universidad de Antioquia, en el marco del proyecto "Sistema Experto de Información Agropecuaria" del Centro de Desarrollo Agrobiotecnológico de Innovación e Integración Territorial-CEDAIT.

El libro inicia con una descripción detallada de las bases conceptuales sobre la teoría del espectro electromagnético de objetos expuestos a la luz y define "una imagen hiperespectral como un conjunto de píxeles con diferentes longitudes de ondas asociadas a regiones del espectro electromagnético, organizadas en bandas, desde las básicas RGB (rojo, verde y azul), y otras que están en la región del infrarrojo. Una muestra o un objeto irradiado por una fuente de emisión de luz absorberá o reflejará esta luz, dependiendo de su composición, la cual es detectada por el sensor en forma de imagen. Estas imágenes son procesadas para generar información, para después analizarlas con

un modelo estadístico previamente estudiado para la calibración de las cámaras y tipo de muestras. En todo este proceso, se deben tener protocolos que parten desde la preparación de las muestras, calibración de equipos, hasta el almacenamiento de información y el análisis de datos.

El libro detalla aspectos relevantes del análisis de suelos con cámaras hiperespectrales para conocer sus características físico-químicas y hace una descripción de los programas computacionales Python y R-project utilizados desde: la captura de información, análisis gráficos para detectar anomalías o inconsistencias de la información, transformaciones de los datos, métricas de evaluación (criterios de calidad del modelo) y desarrollo de modelos de aprendizaje de máquina.

Este tipo de modelos permiten analizar variables que no cumplen con los supuestos estadísticos, utilizando procesos de entrenamiento, prueba y validación. Entre ellos se encuentran modelos que reducen la alta dimensionalidad y multicolinealidad entre covariables, como la regresión por componentes principales y por mínimos cuadrados parciales; los modelos de árboles de clasificación; las redes neuronales artificiales; entre otras.

Se culmina con un ejercicio de aplicación en R-project para análisis de datos, partiendo de la transformación de los datos, la graficación y la aplicación del modelo estadístico mínimos cuadrados parciales para la materia orgánica contrastando una base de datos de espectros y otra base de química húmeda del suelo.

Como valoración global de la obra, podemos decir que se describe ordenadamente las ideas, desde los conceptos tóricos y la aplicación de la técnica y la estadística para que el lector pueda entender y aplicar el análisis de imágenes hiperespectrales. Sin embargo, consideramos que, a pesar de tener un ejercicio de análisis, se requieren mayores detalles de este proceso, por esto, los autores consideramos relevante aportar a los lectores, más información.

Los autores

1 CAPÍTULO UNO

Procesamiento de imágenes hiperespectrales y extracción de regiones de interés en lenguaje Python

Las bases de datos las encuentra en este enlace

Instalación de librerías necesarias

Para realizar la manipulación de las imágenes en el lenguaje Python (Van Rossum, 2009), es necesario realizar la importación de librerías que permitan acceder a la información de manera más sencilla. Por lo tanto, se utilizarán las librerías *numpy* (Harris *et al.,* 2020) y *pandas* (McKinney *et al.,* 2010) para el manejo de datos numéricos y bases de datos; *spectral* (Boggs, 2014), para el procesamiento y manipulación de información espectral; *spectralPy*, módulo de Python para procesar datos de imágenes hiperespectrales que tiene funciones para leer, visualizar, manipular y clasificar imágenes hiperespectrales; *matplotlib* (Hunter, 2007) para la creación y visualización de imágenes; y la librería *os* para la lectura de datos dentro de carpetas. La importación de librerías generalmente se realiza mediante el uso de 'alias' que ayuda con la codificación de forma organizada y resumida. Para instalar las librerías**,** se deberá utilizar el comando *!pip install***,** para las que no estén incluidas por defecto, así:

```
!pip install SpectralPy
!pip install spectral
!pip install os
!pip install matplotlib
```

La importación de las librerías se hace mediante el comando *import*:

```
from spectral import imshow, view_cube
import spectral.io.envi as envi
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib
import pandas as pd
import numpy.ma as ma
import os
```

Lectura de las rutas donde se encuentran las imágenes

Una vez importadas las librerías, se crea una base de direcciones que corresponderá a la carpeta donde se encuentran las imágenes a procesar. En este caso, se realizará con imágenes SWIR, pero el flujo de programación también será especificado para las imágenes VNIR.

Mediante una lista de comprensión, se eliminarán los archivos que tengan una extensión diferente a .img, de esta manera se puede estar seguro de que todos los archivos existentes dentro de la base de direcciones serán imágenes hiperespectrales y no archivos complementarios:

```
base_dir = 'C:/Users/Drone2018/Desktop/SWIR'
swir = os.listdir(f'{base_dir}')
swir = [i for i in swir if i[-4:]=='.hdr' and 'satmask' not in i ]
print(swir)
```

```
['S21-0001_SWIR_384_SN12004_10616us_2021-06-03T115117_raw_rad_ref_float32.hdr' 'S21-
0002_SWIR_384_SN12004_10616us_2021-06-03T120458_raw_rad_ref_float32.hdr', 'S21-
0003_SWIR_384_SN12004_10616us_2021-06-03T121600_raw_rad_ref_float32.hdr', 'S21-
0004_SWIR_384_SN12004_10616us_2021-06-03T122758_raw_rad_ref_float32.hdr', 'S21-
0005_SWIR_384_SN12004_10616us_2021-06-03T123832_raw_rad_ref_float32.hdr']
```

Depuración de códigos

El siguiente paso corresponderá a la depuración y creación de códigos existentes dentro de nuestra base de direcciones <<base_dir>>. Para llevar a cabo este proceso, se creará una variable tipo diccionario con la cual se almacenarán los códigos existentes dentro de la carpeta y que más adelante serán utilizados para almacenar los espectros extraídos.

```
cods = {}
for direccion in swir:
    separador = direccion.find("_")
    codigo = direccion[0:separador]
    codigo = codigo.replace("-","")
    codigo = codigo.replace("_","")
    if codigo not in cods:
        cods[codigo] = direccion
print(cods.keys())
```

```
dict_keys(['S210001', 'S210002', 'S210003', 'S210004', 'S210005'])
```

La función del código ejecutado anteriormente hace que, para cada dirección dentro de nuestra base de direcciones, se extraigan todos los códigos sin la presencia de guiones y sin almacenar los repetidos.

Apertura de imágenes SWIR

El tiempo que conlleva la apertura de imágenes dependerá de la cantidad y el peso individual de cada archivo. Las imágenes SWIR, al tener menor resolución que las VNIR, se pueden leer en grupos más grandes, dependiendo de la capacidad computacional del equipo utilizado para la ejecución del código. El siguiente paso para realizar la lectura de las imágenes, será realizar la apertura de cada archivo, mediante una función iterable ***for*** dentro de una *lista de comprensión*. El resultado final será una lista compuesta de matrices multidimensionales (<<*img_swirs*>>).

```
img_swirs = [np.array((envi.open(f"{base_dir}/{swir}", f"{base_dir}/{swir[:-4]}.img")).load()) for swir in swirs]
```

Para acceder a cada elemento dentro de *img_swirs*, se utilizará el índice en el que está almacenada la imagen, en este caso el lugar número 1 (*img_swirs[0]*). Como la visualización de tensores (imágenes) con dimensiones altas es compleja dentro de la consola de Python, se utilizará el comando *.shape* para conocer las dimensiones de uno de los elementos. En este caso tenemos 1216 píxeles en el eje Y, 384 en el eje X y 288 en el eje Z.

```
img_swirs[0].shape # Dimensiones del primer elemento dentro de img_swirs
```

```
(1216, 384, 288)
```

Adicionalmente, para comprobar que se hayan leído los archivos correctamente, podemos utilizar la función *len()* para conocer la longitud o la cantidad de elementos dentro de *img_swirs*:

```
len(img_swirs) # Longitud o cantidad de elementos dentro de img_swirs
```

```
5
```

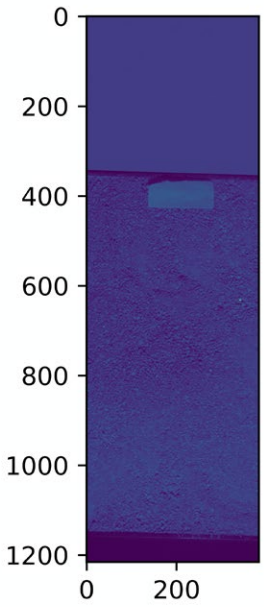
Plot de una banda o capa individual

Para visualizar las imágenes de manera más clara, se recomienda realizar la graficación de una banda o capa a la vez. Por lo tanto, para crear un *plot,* se utiliza la imagen número 1 procedente de la lista *img_swirs*. En este caso, se definen los ejes y se grafica la banda número 200 por medio de las funciones *plt.subplots()* e *imshow()*. Si se desea guardar la imagen, se utiliza la función *plt.savefig(),* en la cual se debe especificar el nombre que llevará el archivo y la resolución en *dpi* (Figura 1):

```
imagen1 = img_swirs[0] #imagen 1
fig, ax = plt.subplots(1) # creación de ejes
img_plot =ax.imshow(imagen1[:,:,200]) # banda 200
plt.savefig('imagen_completa.jpeg', dpi = 1200)
```

Figura 1

Graficación de la banda número 200 por medio de plt.subplots e imshow.



Fuente: elaboración propia generada en Phyton

Plot de ROIs dentro de una imagen

Para añadir los posibles tamaños de ROI, se utilizará la función *patches.Rectangle()*, con la que se podrá graficar rectángulos encima de la imagen anterior. Dentro de esta función se deberán especificar las coordenadas en *X* y en *Y*, además de las dimensiones deseadas. Por ejemplo, si se desea graficar la ROI del 100%, se deberán especificar las coordenadas (X = 30, Y = 450) con unas dimensiones de 330 de ancho x 670 largo píxeles, un ancho de línea (*linewidth*), un color de borde (*edgecolor*) y un relleno, en ese caso transparente (*facecolor = "none"*). Para graficar las ROI creadas, se utilizará la función *ax.add_patch(),* como se aprecia en la Figura 2.

```
imagen1 = img_swirs[0] #imagen 1
fig, ax = plt.subplots(1) # creación de ejes
img_plot = ax.imshow(imagen1[:,:,200])#Banda 200
print(imagen1.shape)
```

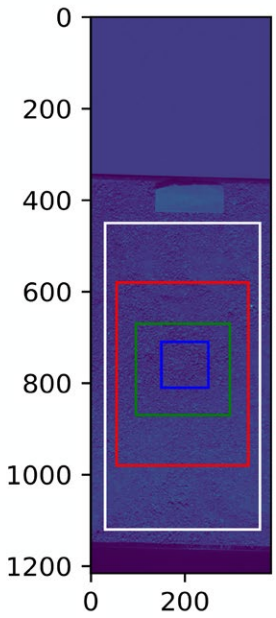
```
ROI_swir_100 = patches.Rectangle((30, 450),330, 670, linewidth=1, edgecolor='w',
facecolor="none")        #Coordenadas 100 en X y 600 en Y, con un tamaño de 200x200
pix
ROI_swir_50 = patches.Rectangle((55, 580),280,400 , linewidth=1, edgecolor='r',
facecolor="none")
ROI_swir_17 = patches.Rectangle((96, 670),200, 200, linewidth=1, edgecolor='g',
facecolor="none")
ROI_swir_4 = patches.Rectangle((150, 710),100, 100, linewidth=1, edgecolor='b',
facecolor="none")

ax.add_patch(ROI_swir_100)
ax.add_patch(ROI_swir_50)
ax.add_patch(ROI_swir_17)
ax.add_patch(ROI_swir_4)

plt.savefig('roi_total.jpeg', dpi = 1200)
```

Figura 2

Graficación de las ROI dentro de una imagen SWIR, mediante la función patches.Rectangle().



Fuente: elaboración propia generada en Phyton

Por otra parte, se debe tener en cuenta que las imágenes procedentes de la cámara VNIR tienen unas dimensiones diferentes al tener mayor resolución espacial (3289 x 1024 pix) y menor resolución espectral (88 bandas), comparadas con las SWIR que tienen una resolución espacial de

1216 x 384 pix y una resolución espectral de 288 bandas. Por lo tanto, las coordenadas para graficar y extraer las ROI difieren, como se indica en la siguiente programación, cuyo resultado se aprecia en la Figura 3.

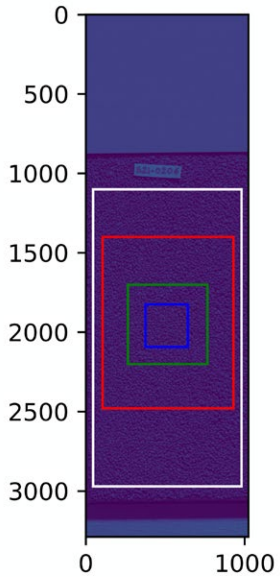
```
imagen_vnirs1 = img_vnirs[0] #imagen 1
fig, ax = plt.subplots(1) # creación de ejes
img_plot = ax.imshow(imagen1[:,:,80])#Banda 80
print(imagen_vnirs1.shape)

ROI_vnir_100 = patches.Rectangle((30, 450),330, 670, linewidth=1, edgecolor='w',
facecolor="none")        #Coordenadas 100 en X y 600 en Y, con un tamaño de 200x200
pix
ROI_vnir_50 = patches.Rectangle((55, 580),280,400 , linewidth=1, edgecolor='r',
facecolor="none")
ROI_vnir_17 = patches.Rectangle((96, 670),200, 200, linewidth=1, edgecolor='g',
facecolor="none")
ROI_vnir_4 = patches.Rectangle((150, 710),100, 100, linewidth=1, edgecolor='b',
facecolor="none")

ax.add_patch(ROI_vnir_100)
ax.add_patch(ROI_vnir_50)
ax.add_patch(ROI_vnir_17)
ax.add_patch(ROI_vnir_4)
```

Figura 3

Graficación de las ROI dentro de una imagen VNIR, mediante la función patches.Rectangle().



Fuente: elaboración propia generada en Phyton

Hay que tener en cuenta que las coordenadas y dimensiones de las imágenes pueden variar de acuerdo con el tipo de procesamiento y ubicación original de las muestras dentro de la banda transportadora, por lo tanto, se recomienda graficar todas las muestras y cerciorarse de que las dimensiones y coordenadas seleccionadas sean las correctas.

Importación de un archivo en extensión .csv con las bandas de interés

Una vez definidas las ROI junto con sus coordenadas, se procede a importar la base de datos que contiene las bandas del rango del espectro, en este caso SWIR. No obstante, el mismo proceso es válido para importar las bandas del rango VNIR.

```
bands_swir = np.genfromtxt('C:/Users/Drone2018/Desktop/BANDAS/SWIR.csv',
delimiter=',')
len(bands_swir) #la longitud debe coincidir con la resolución espectral de la imagen
```

```
288
```

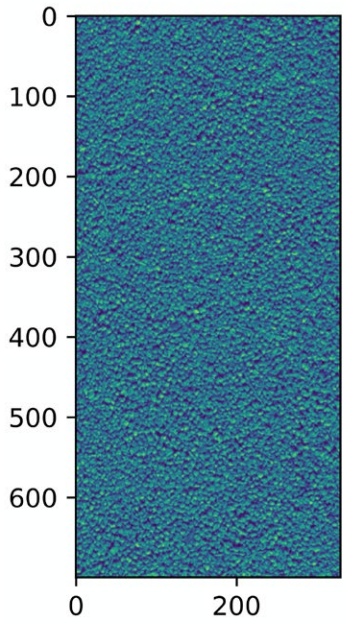
Función y definición de parámetros para extraer la ROI

Para realizar la extracción de la ROI, se debe definir una función en este caso se llamará ***extract_roi()***, que estará compuesta de 5 elementos: Una variable tipo matriz ***(arr)***, una coordenada ***x***, una coordenada ***y***, un ancho ***w***, y un largo ***h***, las cuales servirán para indicar qué porción de la imagen será extraída:

```
def extract_roi(arr, x, y, w, h):
    roi = arr[y:y+h, x:x+w, :]
    return roi
```

Figura 4

Graficación de una ROI individual.



Fuente: elaboración propia generada en Phyton

```
    (x, y) = coordenada
  rois_extraction = [extract_roi(imagen, x, y, ancho, largo) for imagen in imágenes]
print(rois_extraction[0].shape) # dimensiones de una ROI
```

Definición de la función *calcule_reflectance()* para obtener la reflectancia de una ROI

Para continuar con el proceso de extracción de ROIs, se debe acceder a los valores que componen cada ROI, teniendo en cuenta que cada una es una matriz de dimensiones Y = 670; X =330; Z = 288. Por lo tanto, se deben promediar todos los valores de los píxeles en cada una de las matrices que componen la dimensión *Z*. Para lograrlo, se debe construir una función que calcule la reflectancia de las ROI que se le entregue. En este caso nuestra función se llamará ***calcule_reflectance()***, la cual estará compuesta por 4 elementos ***(rois, bands, maskmin, maskmax)*** que corresponden a la lista de las imágenes ROI, las bandas para las cuales se calculará el promedio, el nivel de mínimo de reflectancia aceptado y el nivel máximo de reflectancia aceptado:

```
def calcule_reflectance(rois, bands, maskmin, maskmax):
  roi_intensities = []
  for i in range(len(rois)):
      roi = rois[i]
      intensities = []
      for b in range(roi.shape[2]):
          roi_masked = ma.masked_where(np.logical_or(roi[:, :,b] <= maskmin,roi[:, :,b]
          >= maskmax),roi[:, :,b])
          intensities.append(roi_masked.mean())
      plt.plot(bands, intensities, label='{}'.format(list(cods.keys())[i]))
      roi_intensities.append(intensities)
      plt.legend(loc='upper left')
      plt.title('Soil spectral footprint\n Mean in ROI Area')
      plt.xlabel('Wavelength (nm)')
      plt.ylabel('Reflectance')
      plt.show()
return roi_intensities
```

En primera instancia se indica que para cada elemento que compone la lista *rois* una variable asumirá una ROI en cada iteración y para esta, se calcularán los promedios de cada matriz enmascarada (de acuerdo con los valores definidos para *maskmin* y *maskmax*) que componen la dimensión Z en un rango del 1:288 *(range(len(rois)))*. Adicionalmente, se incluirán gráficas de cada firma espectral (por medio del comando *plt.plot(), plt.legend(), plt.title(), plt.xlabel(), plt.ylabel()* y *plt.show()*) cada vez que se ejecute la función, así, al finalizar el proceso, la función entregará los espectros promedios para cada banda y la gráfica de la firma espectral.

Exportación de archivos extraídos en formato .csv

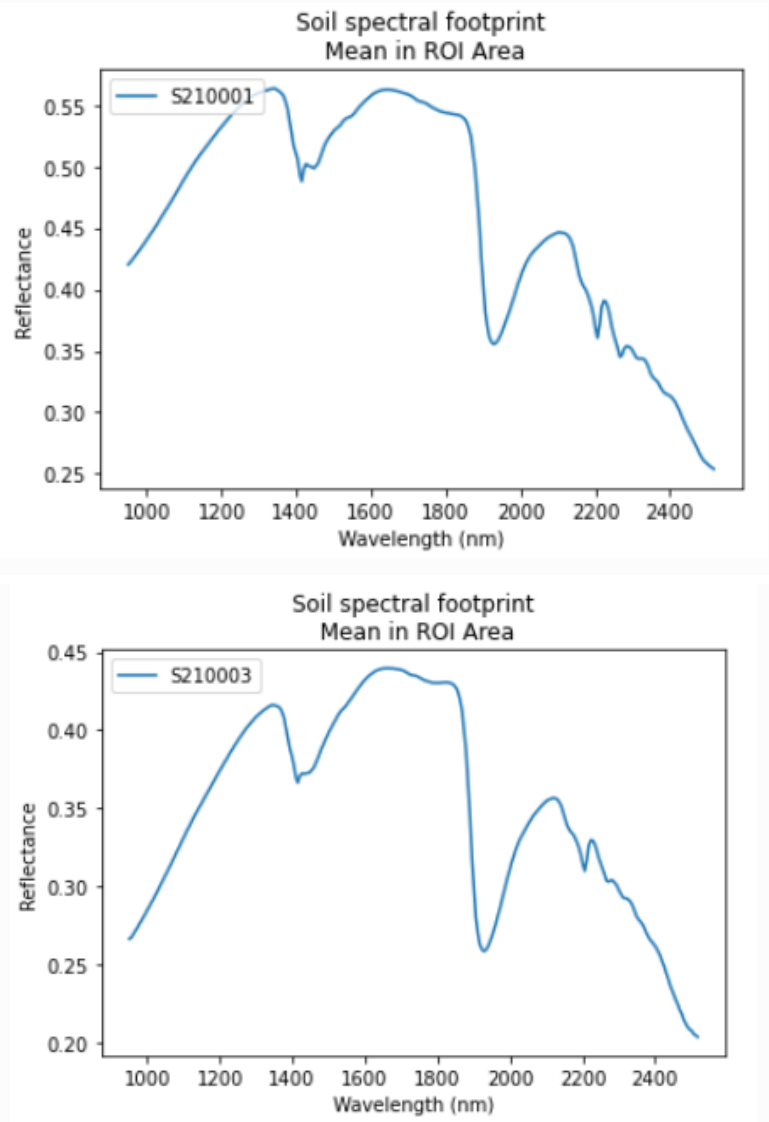
Para finalizar el proceso de extracción de ROIs, se ejecutará la función *calcule_reflectance()* que tendrá como datos de ingreso las ROIs (*rois_extraction*), las bandas importadas (*bands_swir*), el nivel de enmascaramiento mínimo (*maskmin)* y máximo (*maskmax*) deseado. Para obtener este resultado en forma de base de datos, la función *calcule_reflectance()* debe ir dentro de la función *pd.DataFrame()*, acompañado de las columnas, que en este caso serán

las bandas importadas, y el índice, que en este caso serán los códigos (*cods.keys()*) de cada muestra (Figura 5). La exportación de la base de datos se hará en forma de *csv* mediante la función *to_csv*, dónde se especificará la dirección donde se quiere guardar y el nombre del archivo.

```
swir_data = pd.DataFrame(np.array(calcule_reflectance(rois_extraction, bands_swir, 0.1, 0.9)), columns=bands_swir, index= list(cods.keys()))
swir_data.to_csv('C:/Users/Drone2018/Desktop/SWIR_CSV_REF/ensayo.csv')
```

Figura 5

*Ejecución de la función **calcule_reflectance()** y el uso de la función .to_csv para guardar la base de datos creada en formato .csv*



Fuente: elaboración propia generada en Phyton

2 CAPÍTULO DOS

Comparación de regiones de interés en lenguaje R-project

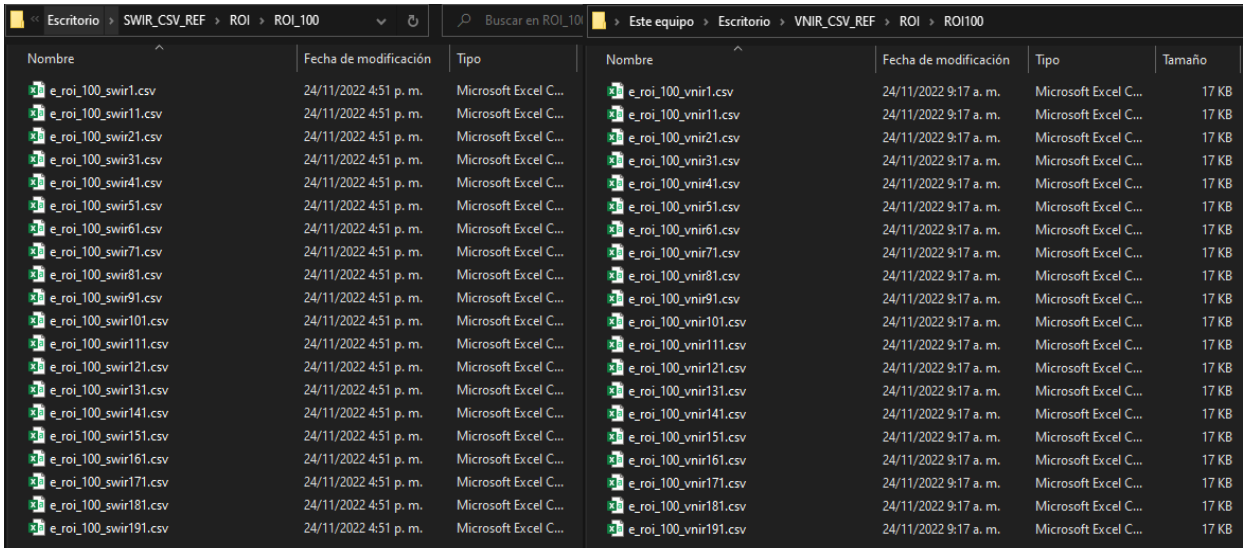
Las bases de datos las encuentra en este enlace

Ubicación de los archivos en extensión .csv

Para ejecutar la comparación entre diferentes regiones de interés (ROI) en el lenguaje R (R Core Team 2020; RStudio 2021), utilizaremos 3 ROI (100%, 50% y 4%) para imágenes VNIR y SWIR extraídas mediante las técnicas expuestas en el capítulo anterior. En específico, para la ejecución de este ejemplo, se utilizarán 20 archivos compuestos de 10 muestras para cada rango ROI en ambos rangos espectrales (VNIR y SWIR) como se puede ver en la Figura 6.

Figura 6

Ubicación de los archivos en formato .csv referentes al 100% de la ROI en imágenes VNIR y SWIR



Fuente: Elaboración propia

Librerías necesarias para el procesamiento de información espectral

Una vez identificadas las direcciones y archivos que se utilizarán, es necesario hacer uso de librerías que permitan la concatenación de las bases de datos, la lectura y graficación de información espectral en el ambiente de desarrollo integrado RStudio. En caso de no tenerlas instaladas, se deberá hacer uso del comando *install.packages("")* especificando el nombre de la librería a instalar dentro de comillas.

```
library(readxl)
library(tidyverse)
library(spectrolab)
library(prospectr)
library(fs)
```

Concatenación de bases de datos VNIR y SWIR

Para realizar la concatenación de las bases de datos, se debe partir de la carpeta o ubicación de los archivos que se desean juntar, en este caso se utilizará la función *dir_ls()* para fijar las diferentes direcciones:

```
v100<-fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\VNIR_CSV_REF\\ROI\\ROI100")
v50<-fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\VNIR_CSV_REF\\ROI\\ROI50")
v4<-fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\VNIR_CSV_REF\\ROI\\ROI4")
```

Una vez fijadas las direcciones, se deben almacenar dentro de listas vacías, este procedimiento se logra al utilizar la función *for* junto con la función *read.csv()* para cada uno de los tamaños de ROI.

```
for (i in seq_along(v100)){    #Para vnir 100%
  ldir100[[i]] <- read.csv(
      file = v100[[i]])}
for (i in seq_along(v50)){    #Para vnir 50%
   ldir50[[i]] <- read.csv(
      file = v50[[i]])}
for (i in seq_along(v4)){    #Para vnir 4%
```

```
  ldir4[[i]] <- read.csv(
    file = v4[[i]])}
v100 # direcciones para ROIs vnir 100%
```

```
C:/Users/Drone2018/Desktop/VNIR_CSV_REF/ROI/ROI100/roi100_vnir_01-02-2022-2.csv
C:/Users/Drone2018/Desktop/VNIR_CSV_REF/ROI/ROI100/roi100_vnir_01-02-2022.csv
C:/Users/Drone2018/Desktop/VNIR_CSV_REF/ROI/ROI100/roi100_vnir_01_03-02-2022.csv
C:/Users/Drone2018/Desktop/VNIR_CSV_REF/ROI/ROI100/roi100_vnir_03-02-2022.csv
```

```
head(ldir100[[1]]) # Base de datos número 1 almacenada en la lista ldir100
```

```
X   X485.0   X490.0   X495.0   X501.0
1  S210265 0.1804463 0.1568751  0.1555133
2  S210268 0.1530118 0.1363289  0.1333611
3  S210270 0.1351079 0.1228770  0.1232563
4  S210558 0.1485641  0.1302492  0.1283510
5  S210559 0.1581648 0.1466283  0.1465500
6  S210560 0.1492138 0.1325415  0.1311880
```

Una vez obtenidas las listas de las direcciones de las bases de datos, se procede a crear variables tipo *data frame* vacías, las cuales servirán para concatenar y almacenar los datos existentes en cada una de las listas, esto podrá ser realizado mediante la función *for* y *rbind()*:

```
f1 <- data.frame()
f2 <- data.frame()
f3 <- data.frame()

for (i in ldir100) {f1 <- rbind(f1,i)}
for (i in ldir50) {f2 <- rbind(f2,i)}
for (i in ldir4) {f3 <- rbind(f3,i)}
vnir100 <- f1
vnir50 <- f2
vnir4 <- f3
```

Una vez creada las bases de datos, se procede a crear una variable correspondiente al tamaño de ROI, esto puede realizarse mediante la librería *tidyverse* (Wickham *et al*., 2019) y la posibilidad de utilizar las tuberías o <<*pipes*>> (%>%), la función *mutate* (para la creación de variables dentro de bases de datos) y *relocate* (para la reubicación de variables dentro de bases de datos), así:

```
vnir100 <-  vnir100 %>%
          unique() %>%
          mutate(roi = "100%") %>%
          relocate(X,roi)
        vnir50 <-  vnir50 %>%
          unique() %>%
          mutate(roi = "50%")%>%
          relocate(X,roi)
        vnir4 <-  vnir4 %>%
          unique() %>%
          mutate(roi = "4%")%>%
          relocate(X,roi)
        head(vnir100) #Estructura de la base de datos
```

```
     X    roi  X485.0    X490.0    X495.0    X501.0
1 S210265 100% 0.1804463 0.1568751 0.1555133 0.1577146
2 S210268 100% 0.1530118 0.1363289 0.1333611 0.1329362
3 S210270 100% 0.1351079 0.1228770 0.1232563 0.1229806
4 S210558 100% 0.1485641 0.1302492 0.1283510 0.1297426
5 S210559 100% 0.1581648 0.1466283 0.1465500 0.1492375
6 S210560 100% 0.1492138 0.1325415 0.1311880 0.1326857
```

Una vez creada esta variable para todos los tamaños de ROI, se procede a juntar las bases de datos mediante la función *bind_rois()*, así:

```
vnir_rois <- bind_rows(vnir100, vnir50, vnir4)
```

En cuanto a la creación de bases de datos para imágenes SWIR, se debe seguir el mismo procedimiento explicado para las imágenes VNIR, solamente cambiando el nombre de las variables como se indica a continuación:

```
s100<-fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\SWIR_CSV_REF\\ROI\\ROI_100")
s50 <-fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\SWIR_CSV_REF\\ROI\\ROI_50")
s4 <- fs::dir_ls("C:\\Users\\Drone2018\\Desktop\\SWIR_CSV_REF\\ROI\\ROI_4")

sldir100 <- list()
sldir50 <- list()
sldir4 <- list()

for (i in seq_along(s100)){    #Para swir 100%
      sldir100[[i]] <- read.csv(
            file = s100[[i]])}
```

```
for (i in seq_along(s50)){    #Para swir 50%
      sldir50[[i]] <- read.csv(
            file = s50[[i]])}

for (i in seq_along(s4)){    #Para swir 4%
      sldir4[[i]] <- read.csv(
            file = s4[[i]])}
s1 <- data.frame()
s2 <- data.frame()
s3 <- data.frame()

for (i in sldir100) {s1 <- rbind(s1,i)}
for (i in sldir50) {s2 <- rbind(s2,i)}
for (i in sldir4) {s3 <- rbind(s3,i)}

swir100 <- s1
swir50 <- s2
swir4 <- s3
```

Para la creación de la variable “roi” y concatenación de las bases de datos para las imágenes SWIR usamos la siguiente programación:

```
swir100 <-  swir100 %>%
 unique() %>%
 mutate(roi = "100%") %>%
 relocate(X,roi)

swir50 <-  swir50 %>%
 unique() %>%
 mutate(roi = "50%")%>%
 relocate(X,roi)

swir4 <-  swir4 %>%
 unique() %>%
 mutate(roi = "4%")%>%
 relocate(X,roi)

swir_rois <- bind_rows(swir100, swir50, swir4)
```

Graficación de las firmas espectrales

Para poder graficar los espectros, es necesario que las variables contengan un nombre numérico correspondiente al ancho o longitud de onda, no obstante, cuando los archivos son importados, el nombre de cada variable está acompañado de la letra "X", por lo que el nombre de la variable será reconocido como una variable tipo carácter.

```
names(vnir_rois)
```

```
 [1] "X"     "roi"   "X485.0" "X490.0" "X495.0"
 [6] "X501.0" "X506.0" "X512.0" "X517.0" "X523.0"
[11] "X528.0" "X533.0" "X539.0" "X544.0" "X550.0"
[16] "X555.0" "X560.0" "X566.0" "X571.0" "X577.0"
[21] "X582.0" "X587.0" "X593.0" "X598.0" "X604.0"
[26] "X609.0" "X614.0" "X620.0" "X625.0" "X631.0"
```

Es necesario renombrar las variables con el uso de la función *names()* donde se especificará cuáles columnas se desean renombrar y la función *substr()* con la que se extraerán los dígitos numéricos que componen los nombres de las variables. En esta porción de código, se especifica que se cambiarán los nombres desde la base de datos *vnir_rois* desde la columna 3 hasta el número total de columnas (dim(vnir_rois)[2]) por los números presentes en el mismo rango, es decir, desde el 2do hasta el 4to carácter. (*substr(names(vnir_rois)[3:dim(vnir_rois)[2]],2,4)*).

```
names(vnir_rois)[3:dim(vnir_rois)[2]] <- s
ubstr(names(vnir_rois)[3:dim(vnir_rois)[2]],2,4)
names(vnir_rois)
```

```
 [1] "X"   "roi" "485" "490" "495" "501" "506" "512"
 [9] "517" "523" "528" "533" "539" "544" "550" "555"
[17] "560" "566" "571" "577" "582" "587" "593" "598"
[25] "604" "609" "614" "620" "625" "631" "636" "641"
```

Este procedimiento también es necesario para las variables que componen las bases de datos *swir_rois* en 2 pasos, ya que, el número de caracteres que componen sus nombres es diferente en las primeras bandas.

```
        names(swir_rois)[3:11] <- substr(names(swir_rois)[3:11],2,4)
        names(swir_rois)[12:dim(swir_rois)[2]] <-
substr(names(swir_rois)[12:dim(swir_rois)[2]],2,5)
        names(swir_rois)
```

```
  [1] "X"    "roi"  "951"  "957"  "962"
  [6] "967"  "973"  "978"  "984"  "989"
 [11] "995"  "1000" "1006" "1011" "1017"
 [16] "1022" "1028" "1033" "1038" "1044"
 [21] "1049" "1055" "1060" "1066" "1071"
 [26] "1077" "1082" "1088" "1093" "1098"
```

Adicionalmente, las bases de datos se deben agrupar por tamaño de ROI y promediar los valores de reflectancia. Para lograrlo, se utilizarán las funciones *group_by()* y *summarise_at()* como se indica a continuación:

```
        vnir_groups <- vnir_rois %>%
          group_by(roi) %>%
          summarise_at(vars(3:89), list(mean))

        vnir_groups <- vnir_groups[c(1,2,3),]

        vnir_groups
```

```
# A tibble: 3 x 88
  roi   `490` `495` `501` `506` `512` `517`
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 100%  0.137 0.136 0.137 0.139 0.141 0.144
2 4%    0.137 0.136 0.137 0.139 0.141 0.143
3 50%   0.137 0.136 0.137 0.139 0.141 0.143
```

```
        swir_groups <- swir_rois %>%
          group_by(roi) %>%
          summarise_at(vars(3:289), list(mean))

        swir_groups <- swir_groups[c(1,2,3),]
```

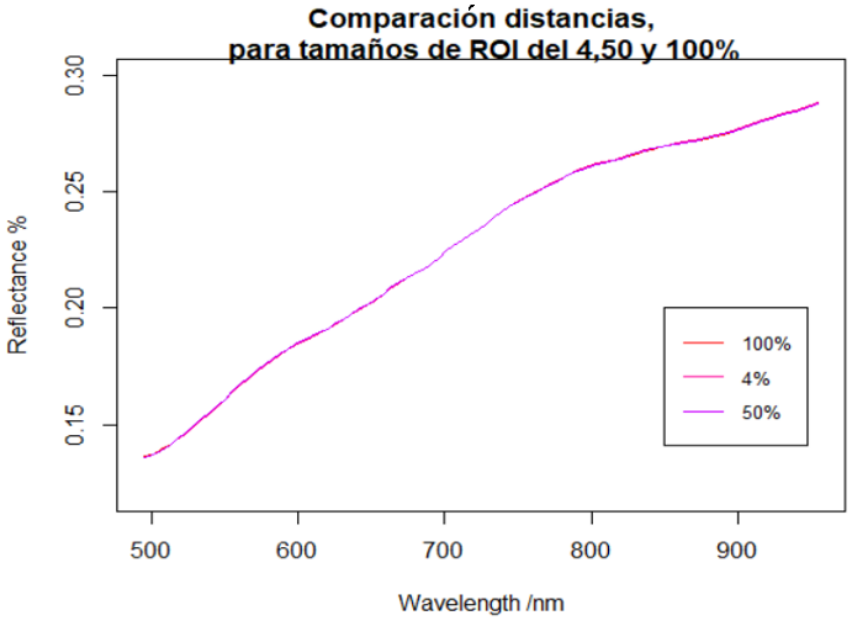
```
# A tibble: 3 x 288
  roi   `957` `962` `967` `973` `978` `984`
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 100%  0.316 0.317 0.319 0.320 0.322 0.323
2 4%    0.317 0.318 0.319 0.321 0.322 0.324
3 50%   0.316 0.317 0.319 0.320 0.322 0.323
```

Una vez agrupados y promediados los datos, se procede a graficar las firmas espectrales de la base de datos *vnir_groups* mediante la función *matplot(),* como se indica en la Figura 7. Los dos primeros parámetros de esta función se componen de la matriz de datos y su transpuesta, además se deben proveer las etiquetas para los ejes *X* y *Y*.

```
matplot(x =colnames(as.matrix(vnir_groups[,3:88])), y =t(as.matrix(vnir_groups[,3:88])),
        xlab = "Wavelength /nm",
        ylab = "Reflectance %",
        ylim = c(0.12, 0.3),
        type = "l",
        lty = 1,
        main = paste("Comparacion distancias,\n","para tamaños de ROI del 4,50 y 100%"),
        col = rainbow(11, s = 1, v = 1, start = 0, end = 1,
                      alpha=1, rev = T))
legend(850, 0.2, legend=substring(vnir_groups$roi,1,4),
       col= rainbow(11, s = 1, v = 1, start = 0, end = 1,
                    alpha=1, rev = T),lty = 1, cex=0.8)
```

Figura 7

Firmas espectrales para la región VNIR por medio de la función matplot().



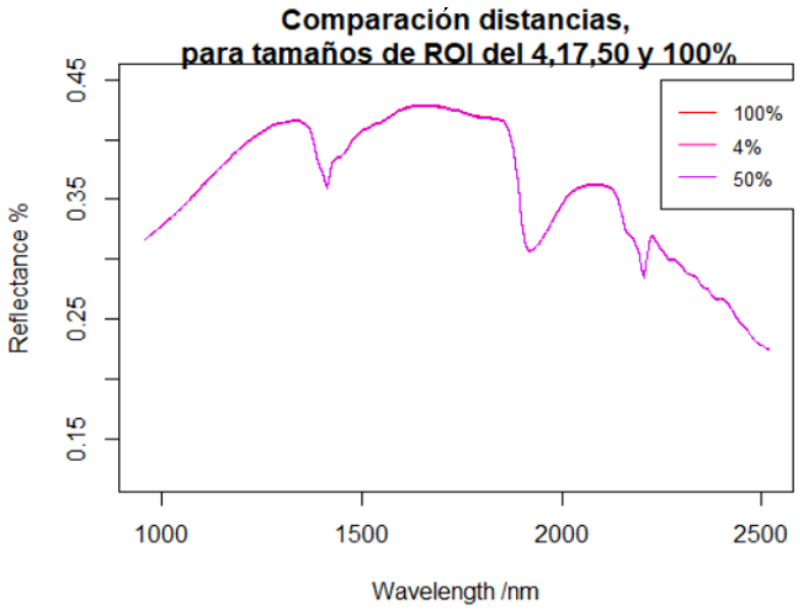
Fuente: elaboración propia generada en R-project

De igual manera, se grafican los espectros que componen la base de datos *swir_groups* (Figura 8), con la siguiente programación:

```
matplot(x =colnames(as.matrix(swir_groups[,2:288])), y =t(as.matrix(swir_groups[,2:288])),
        xlab = "Wavelength /nm",
        ylab = "Reflectance %",
        ylim = c(0.12, 0.45),
        type = "l",
        lty = 1,
        main = paste("Comparación distancias,\n","para tamaños de ROI del 4,17,50 y 100%"),
        col = rainbow(11, s = 1, v = 1, start = 0, end = 1,
                      alpha=1, rev = T))
legend(2250, 0.45, legend=substring(swir_groups$roi,1,4),
       col= rainbow(11, s = 1, v = 1, start = 0, end = 1,
                    alpha=1, rev = T),lty = 1, cex=0.8)
```

Figura 8

Firmas espectrales para la región SWIR por medio de la función matplot().



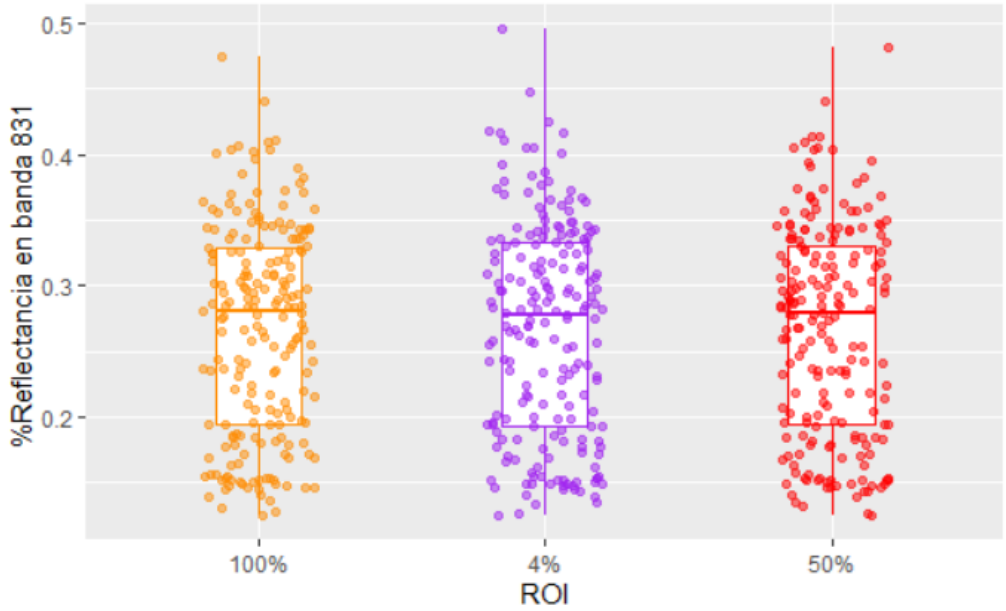
Fuente: elaboración propia generada en R-project

Por otra parte, es posible realizar la comparación de las bandas de manera individual mediante un gráfico de cajas o *boxplot*. En este caso, se realizará mediante la función *ggplot* (creación de gráficos) y los complementos *geom_boxplot* (presentación de datos en forma de boxplot) y *geom_jitter* (ubicación de los puntos sobre el gráfico) como se muestra en la Figura 9 para *vnir_rois* y en la Figura 10 para *swir_rois*, con las siguientes programaciones:

```
ggplot(data = vnir_rois, aes(x = roi, y = `831`)) +
  geom_boxplot(aes(color = roi), width = 0.3, show.legend = F)+
  geom_jitter(aes(color = roi), alpha = 0.5, show.legend = FALSE, position =
                position_jitter(width = 0.2, seed = 0)) +
  scale_color_manual(values = c("darkorange","purple","red")) +
  labs(x = "ROI",
       y = "%Reflectancia en banda 831 ")
```

Figura 9

Distribución de los niveles de reflectancia presentes en la banda 831 para los tamaños 100%, 50% y 4% para ROIs pertenecientes a la región VNIR.

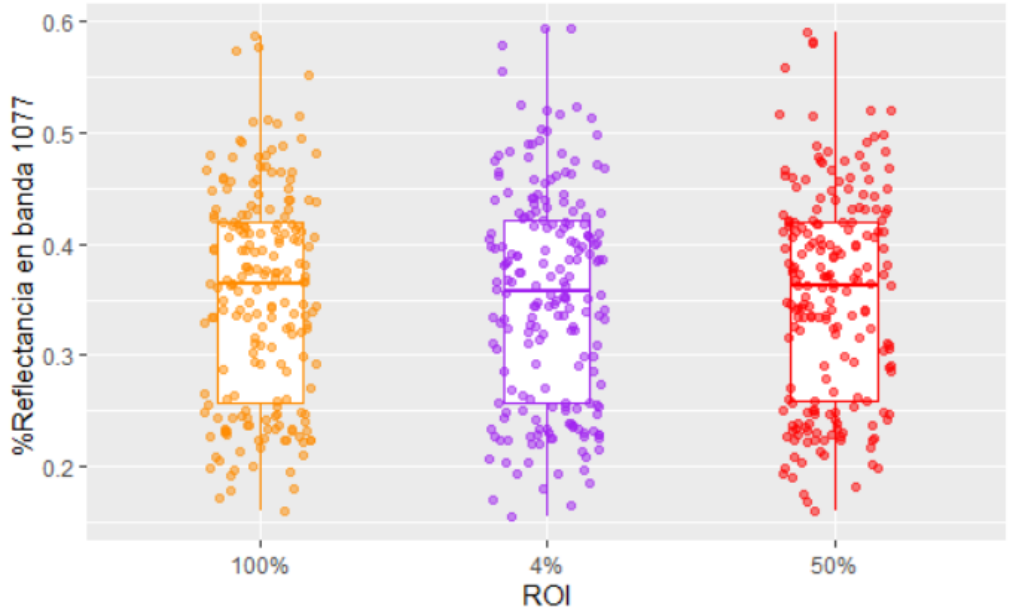


Fuente: elaboración propia generada en R-project

```
ggplot(data = swir_rois, aes(x = roi, y =`1077`)) +
  geom_boxplot(aes(color = roi), width = 0.3, show.legend = F)+
  geom_jitter(aes(color = roi), alpha = 0.5, show.legend = FALSE, position =
                position_jitter(width = 0.2, seed = 0)) +
  scale_color_manual(values = c("darkorange","purple","red")) +
  labs(x = "ROI",
       y = "%Reflectancia en banda 1077 ")
```

Figura 10

Distribución de los niveles de reflectancia presentes en la banda 1077 para los tamaños 100%, 50% y 4% para ROIs pertenecientes a la región SWIR.



Fuente: elaboración propia generada en R-project

Como las diferencias no se perciben a simple vista, la comparación de firmas espectrales debe realizarse por medio de pruebas de distancia por medio de la librería *resemble* (Ramírez-López *et al*., 2022), en este caso se realizará la prueba euclidiana dónde los valores que se alejan de 0 denotan mayor diferencia (Wadoux *et al*., 2021).

```
dist_eu <- f_diss(Xr = as.matrix(vnir_groups[2:dim(vnir_groups)[2]]),
Xu = as.matrix(vnir_groups[2:dim(vnir_groups)[2]]),
diss_method = "euclid",
center = TRUE, scale = TRUE)

dist_eu <- as.data.frame(dist_eu)
colnames(dist_eu) <- as.vector(vnir_groups$roi)
rownames(dist_eu) <- as.vector(vnir_groups$roi)

dist_eu
```

```
          100%       50%       4%
100%  0.0000000 0.4430175 1.968261
50%   0.4430175 0.000000 1.8519403
4%    1.9682613 1.851940 0.0000000
```

Una vez identificadas las diferencias o distancias, podemos observar que entre las ROI al 100 y al 50% existe poca diferencia (0.44), pero con respecto al tamaño más pequeño (4%), el valor es de 1.9. Lo que muestra que, aunque gráficamente no se note la diferencia, esta puede ser percibida aplicando las pruebas de distancia. Sin embargo, es recomendable realizar las comparaciones finales mediante la aplicación de algoritmos de Machine Learning.

3 CAPÍTULO TRES

Comparación de regiones de interés mediante la implementación de modelos de Machine Learning

Las bases de datos las encuentra en este enlace

Librerías necesarias para la comparación de regiones de interés (ROI) mediante modelos de regresión parcial por mínimos cuadrados parciales (PLSR)

La comparación se realizará mediante la construcción de modelos para la predicción de carbono orgánico (CO) en un conjunto de datos que corresponderá al 80% de la base de datos total. Para ejecutar este procedimiento, es necesario realizar la importación de las librerías *pls* (Mevick y Wehrens, 2007), *Metrics* (Hammer y Frasco, 2018), *spectrolab* (Meireles *et al.,* 2017) *y Prospectr* (Stevens y Ramírez-López, 2022), que permitirán la ejecución de los modelos PLSR, la evaluación de los resultados, la graficación de los espectros y la aplicación de transformaciones sobre las firmas espectrales brutas respectivamente:

```
library(pls)
library(tidyverse)
library(Metrics)
library(spectrolab)
library(prospectr)
library(resemble)
```

Concatenación de bases de datos

A diferencia de los capítulos anteriores, para ejecutar la programación que compone este capítulo, se utilizará una base de datos conjunta (VNIR - SWIR), es decir, que contenga la información espectral de los rangos VNIR (485 - 955 nm) y SWIR (951 - 2517 nm), ya que es necesario para identificar datos *outliers* de forma más sencilla. Para lograrlo, se utilizará la función *cbind()* para combinar bases de datos que tengan el mismo número de observaciones mediante la adición de columnas. Adicionalmente, se utilizarán las funciones *names()* y *substr()* para cambiar los nombres de las variables como se abordó en el Capítulo 2:

```
vnir_rois <- read.csv("D:\\BIGDATA\\MANUAL\\vnir_rois_co.csv")
names(vnir_rois)[5:dim(vnir_rois)[2]] <-
  substr(names(vnir_rois)[5:dim(vnir_rois)[2]], 2,4)

swir_rois <- read.csv("D:\\BIGDATA\\MANUAL\\swir_rois_co.csv")
names(swir_rois)[5:13] <-
  substr(names(swir_rois)[5:13], 2,4)

names(swir_rois)[14:dim(swir_rois)[2]] <-
  substr(names(swir_rois)[14:dim(swir_rois)[2]], 2,5)
#Se combinan las bases de datos omitiendo la banda 951
vnir_swir_rois <- cbind(vnir_rois, swir_rois[,5:dim(swir_rois)[2]])

names(vnir_swir_rois)
```

```
 [1] "X"   "roi" "485" "490" "495" "501" "506" "512"
 [9] "517" "523" "528" "533" "539" "544" "550" "555"
[17] "560" "566" "571" "577" "582" "587" "593" "598"
[25] "604" "609" "614" "620" "625" "631" "636" "641"
[33] "647" "652" "658" "663" "669" "674" "679" "685"
[41] "690" "696" "701" "706" "712" "717" "723" "728"
[49] "733" "739" "744" "750" "755" "760" "766" "771"
```

En este caso, omitimos *la banda 951* de la base de datos *swir_rois,* ya que al ser unida con la base de datos vnir_rois, crea una zona de traslape en la firma espectral.

Detección de *outliers*

En diferentes técnicas de espectroscopía, es común utilizar más de un sensor para capturar la información de una muestra, por lo tanto, es necesario realizar un análisis minucioso en la zona de empalme existente entre los diferentes sensores, la cual se puede subsanar eliminando bandas repetidas entre sensores o que no encajen en el orden numérico de las bandas de interés. Para que esto sea posible, se grafican los espectros de una manera alterna. En este caso, se debe transformar la base de datos a un objeto tipo *spectra* mediante el comando *as_spectra()* de la librería *spectrolab* (Meireles *et al*., 2017). Cuando se obtiene un objeto *spectra,* es posible conocer un resumen invocando el nombre de esta directamente en la consola.

```
bd_sp <- as_spectra(vnir_swir_rois[5:dim(vnir_swir_rois)[2]], name_idx = 1)
bd_sp
```

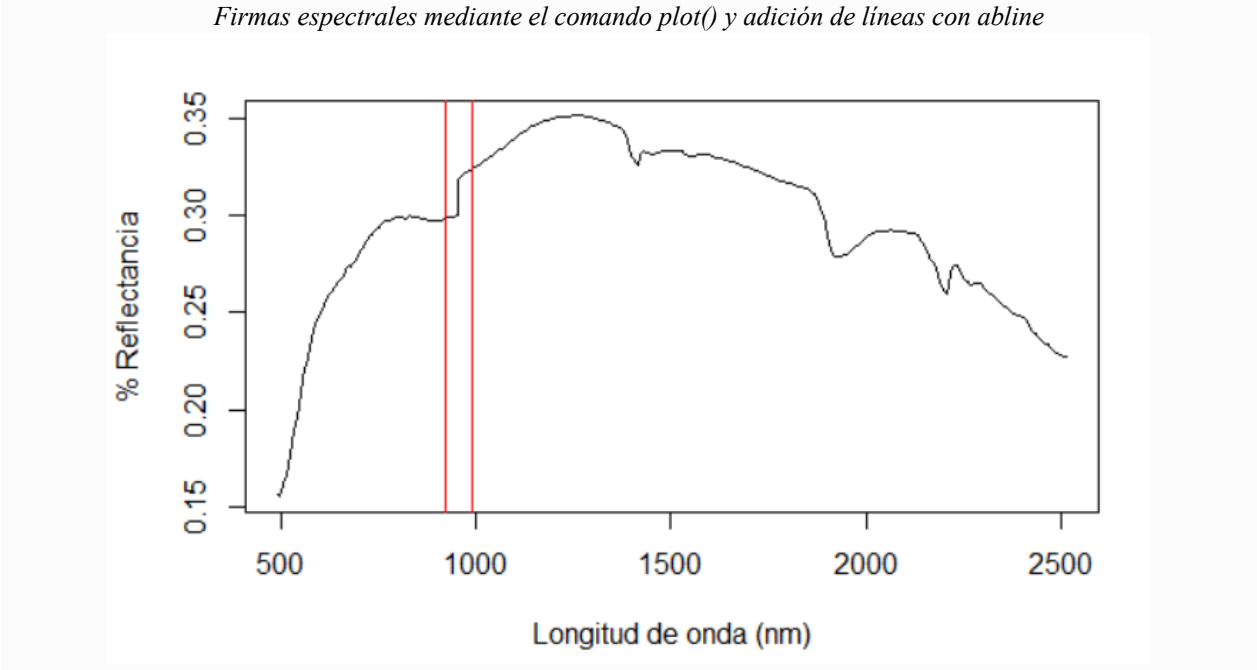
```
spectra object
number of samples: 600
bands: 495 to 2517 (373 bands)
metadata: none
```

Una de las ventajas de transformar la base de datos a un objeto *spectra,* es que se puede graficar de forma sencilla mediante el comando básico *plot()* y parámetros como *abline* (Figura 11). No obstante, es recomendable utilizar con pocas firmas espectrales al mismo tiempo, ya que utilizar numerosas muestras dificulta la capacidad de visualización y análisis individual.

```
#Se grafica la firma espectral número 1
plot(bd_sp[5],ylab = "% Reflectancia", xlab = "Longitud de onda (nm)")
abline(v=920, col = "red")
abline(v=990, col = "red")
```

Figura 11

Firmas espectrales mediante el comando plot() y adición de líneas con abline



Fuente: elaboración propia generada en R-project

Detección de *outliers* mediante pruebas de distancia

Una estrategia para la detección de *outliers* reportada en la literatura, consiste en detectar muestras que pueden ser potenciales valores perdidos u *outliers*, por medio de la distancia de Mahalanobis. En este caso, se ejecutará según lo proponen Wadoux *et al.* (2021) haciendo uso de la librería *resemble*. Adicionalmente, se debe seleccionar la cantidad de variabilidad explicada por los componentes principales (0.99 en este caso), transformar los espectros brutos en absorbancia y aplicar la transformación variable normal estándar (SNV). Una vez transformados los espectros, se debe calcular los componentes principales (PC) por medio del comando *pc_projection* y convertir los resultados a un objeto tipo matriz de una sola fila.

```
# Se elige la cantidad de variabilidad explicada
maxexplvar <- 0.99
#Se transforma a absorbancia
abs_m01 <- log10(1/vnir_swir_rois[5:dim(vnir_swir_rois)[2]])
#Se transforma a snv
ma <- as.matrix(abs_m01)
abs_m01$snv <- standardNormalVariate(X=ma)
snv_m01 <- as.data.frame(abs_m01$snv)
# Se calculan los componentes ppales
pcspectraA <- pc_projection(Xr = as.matrix(snv_m01[,3:375]),
pcSelection = list("cumvar", maxexplvar),
center = TRUE, scale = FALSE)
pcspectraA
# se calcula el promedio de los scores de los pca
pcspectraACentre <- colMeans(pcspectraA$scores)
# se convierte el vector obtenido en matriz
pcspectraACentre <- t(as.matrix(pcspectraACentre))
print(pcspectraACentre)
```

```
              pc_1          pc_2          pc_3          pc_4          pc_5
[1,] 1.058413e-16 2.318747e-17 -6.033946e-18 -7.850202e-17 -2.714842e-17
```

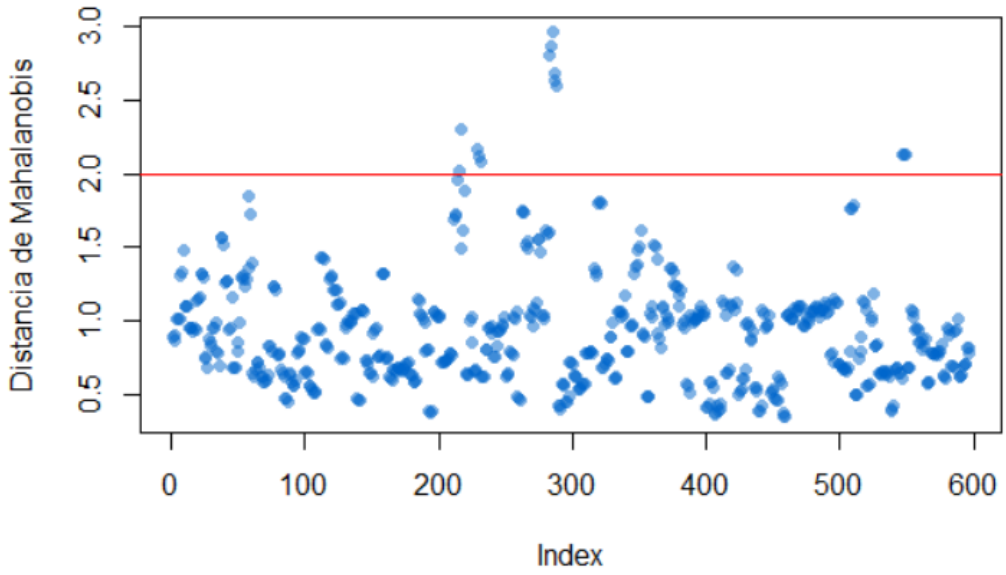
Para calcular la distancia de Mahalanobis, se utilizará el comando *f_diss()* utilizando los scores y la matriz creados anteriormente. Adicionalmente, se podrán graficar las distancias mediante el comando *plot()* como se indica en la Figura 12, con la siguiente programación:

```
wmahald <- f_diss(Xr = pcspectraA$scores,
Xu = pcspectraACentre,
diss_method = "mahalanobis",
center = FALSE, scale = FALSE)

plot(wmahald,
pch = 16,
col = rgb(red = 0, green = 0.4, blue = 0.8, alpha = 0.5),
ylab = "Distancia de Mahalanobis")
abline(h =2, col = "red")
```

Figura 12

Distancia de Mahalanobis mediante f_diss() y plot()



Fuente: elaboración propia generada en R-project

El siguiente paso será identificar y descartar las muestras que presentaron una distancia mayor a 2. Esto se puede realizar con la función *which(),* con la que se puede detectar el índice o lugar donde se ubican las distancias de interés.

```
indxOutM <- which(wmahald > 2)
length(indxOutM)
```

```
14
```

Para descartar las muestras, se deberá identificar el código específico que acompaña los índices que componen la variable *indxOut.* En este caso, se hará uso de una función iterativa *for* para almacenar los códigos dentro de una variable tipo vector.

```
cods_maha <- c()
for (i in 1:length(indxOutM)) {
  cods_maha <- append(cods_maha, vnir_swir_rois$X[indxOutM[i]])}
cods_maha
```

```
```

```
[1]  "S211754" "S211754" "S211772" "S211772" "S211772" "S211753"
[7]  "S211753" "S211753" "S211759" "S211759" "S211759" "S212082"
[13] "S212082" "S212082"
```

Para eliminar los códigos identificados, se hace uso de la función *%notin%* que se crea de forma sencilla, y también, de la función *filter()* perteneciente a la librería *dplyr* disponible por medio de *tidyverse*.

```
`%notin%` <- Negate(`%in%`)

vnir_swir_rois <- vnir_swir_rois %>%
  filter(X %notin% cods_maha)
dim(vnir_swir_rois)
```

```
[1] 582 380
```

```
table(vnir_swir_rois$roi) # Cantidad de muestras por cada tamaño de ROI
```

```
100%  4%  50%
194  194  194
```

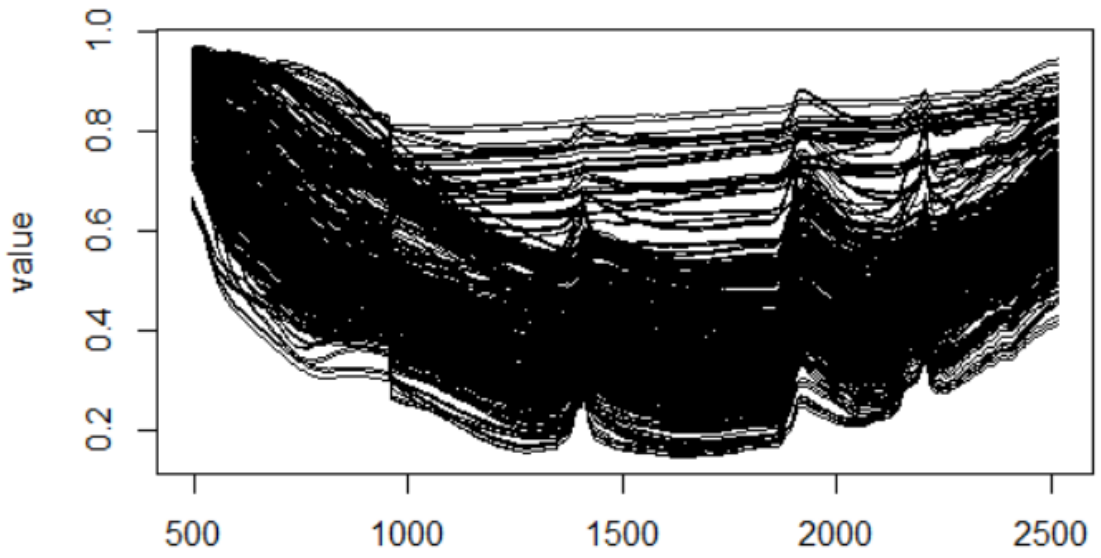
Transformación y graficación de espectros

Una vez finalizado el proceso de detección de muestras anómalas, se realiza la comparación con diferentes tamaños de regiones de interés (ROI). Inicialmente se comenzará con la transformación y graficación (Figura 13) de los espectros brutos a absorbancia.

```
roi_abs <- log10(1/vnir_swir_rois[5:dim(vnir_swir_rois)[2]])#transformacion de
espectros
roi_abs <- roi_abs %>%
  mutate( X = vnir_swir_rois$X, co = vnir_swir_rois$CO, roi = vnir_swir_rois$roi )
%>%
  relocate(X,roi,co)
plot(as_spectra(roi_abs[4:dim(roi_abs)[2]]))
```

Figura 13

Transformación y graficación de espectros brutos a absorbancia



Fuente: elaboración propia generada en R-project

Construcción de listas compuestas de bases de datos para operaciones iterativas

Por otra parte, es necesario construir bases de datos individuales para cada una de las regiones de interés que se desean probar y almacenarlas dentro de un objeto tipo lista:

```
ROI100 <-  roi_abs %>% filter(roi == "100%") %>% select(3:dim(roi_abs)[2])
ROI50 <- roi_abs %>% filter(roi == "50%") %>% select(3:dim(roi_abs)[2])
ROI4 <-roi_abs %>% filter(roi == "4%") %>% select(3:dim(roi_abs)[2])

bds_roi <- list(ROI100,ROI50,ROI4) #lista de bases de datos
bds_roi_t <- c("ROI100","ROI50","ROI4") #títulos en forma de vector

dim(ROI100)
```

```
[1] 194 377
```

Ejecución iterativa de modelos PLSR

Una vez obtenida la lista compuesta de las bases de datos de interés, se realizará la ejecución iterativa de un modelo PLSR para cada tamaño de ROI mediante la función *pls* de la librería *pls* (Mevick y Wehrens, 2007). Inicialmente es necesario construir objetos tipo data.frame sin ningún contenido, con el objetivo de que se puedan utilizar para almacenar los resultados de cada uno de los modelos.

```
mse <- data.frame()
rmse <- data.frame()
comps <- data.frame()
r2 <- data.frame()
rpd <- data.frame()
resultados <- data.frame()
```

Al tener almacenadas las bases de datos en una lista, es posible realizar procesos iterativos para construir los modelos de interés mediante la utilización del comando *for*, también la utilización de una semilla mediante el comando *set.seed()* para que los resultados sean siempre los mismos.

```
for (b in bds_roi) {
  set.seed(123)
```

Hay que tener en cuenta que en el código escrito dentro de la función for debe ser ejecutado al mismo tiempo para que se haga efectivo. Es decir, se debe ejecutar todo lo que está contenido dentro de las llaves {}. En el presente ejemplo se separó, para explicar cada paso.

La comparación de los modelos se realizará utilizando el 80% de la base de datos, es decir, solo en un conjunto de entrenamiento. La separación del conjunto de datos puede ser realizada mediante la creación de un índice aleatorio para separar las muestras de interés:

```
index <- sample(nrow(b),0.8 * nrow(b))
 sh_train_roi <- b[index,]
```

Una vez construido el conjunto de entrenamiento, se procede a aplicar el algoritmo PLSR (regresión de mínimos cuadrados parciales por sus siglas en inglés *Partial Least Squares Regression*), especificando que se utilizará la técnica CV dentro de los hiper parámetros que constituyen la ejecución del algoritmo. Para este ejemplo se utilizarán 10 PC y el método "*oscorepls*" con la finalidad de detectar el número óptimo de PC para la predicción de las características de interés:

```
pls_roi <- pls::plsr(formula = co ~ ., ncomp = 10, data = sh_train_roi, scale. = F,
  validation = "CV", method = "oscorespls")
```

Para almacenar los resultados de los modelos, se calculan las métricas R^2 (Coeficiente de determinación), RMSE (cuadrado medio del error, de las siglas en inglés *Root-Mean-Square Error*) en conjuntos de entrenamiento y validación*:*

```
msep_train <- pls::MSEP(pls_roi, newdata = sh_train_roi)
rsep_train <- pls::RMSEP(pls_roi, newdata = sh_train_roi)
r2_train <- pls::R2(pls_roi, newdata = sh_train_roi)
```

Para calcular la desviación residual de la predicción (RPD), es necesario realizar las predicciones con el modelo seleccionado mediante la función *predict()*. La métrica RPD será calculada utilizando la función *RPD* de la librería *chillR* (Luedeling *et al*., 2023).

```
pred_train_roi <- predict(pls_roi, ncomp = which.min(msep_train$val)-1, newdata =
sh_train_roi)
rpd_train <- chillR::RPD(pred_train_roi, sh_train_roi$co, na.rm = F)
```

Los resultados de entrenamiento serán almacenados en las bases de datos vacías creadas antes de ejecutar el for, así:

```
mse <- rbind(mse,min(msep_train$val))
comps <- rbind(comps,(which.min(msep_train$val)-1))
rmse <- rbind(rmse,min(rsep_train$val))
r2 <- rbind(r2,max(r2_train$val))
rpd <- rbind(rpd,rpd_train)
```

Para finalizar la función *for*, se concatenan los resultados en la base de datos vacía *resultados* y se nombran los elementos dentro de esta.

```
resultados <- cbind(mse, rmse,  r2,  rpd, comps)
} # Cierre de la función for
pls_compar <- cbind(bds_roi_t,resultados)

names(pls_compar) <- c("ROI","MSE", "RMSE","R2","RPD","COMP")

pls_compar
```

	ROI	MSE	RMSE	R2	RPD	COMP
1	ROI100	1.047832	1.023637	0.8648401	2.728862	10
2	ROI50	1.065055	1.032015	0.8626185	2.706708	10
3	ROI4	1.122083	1.059284	0.8552625	2.637029	10

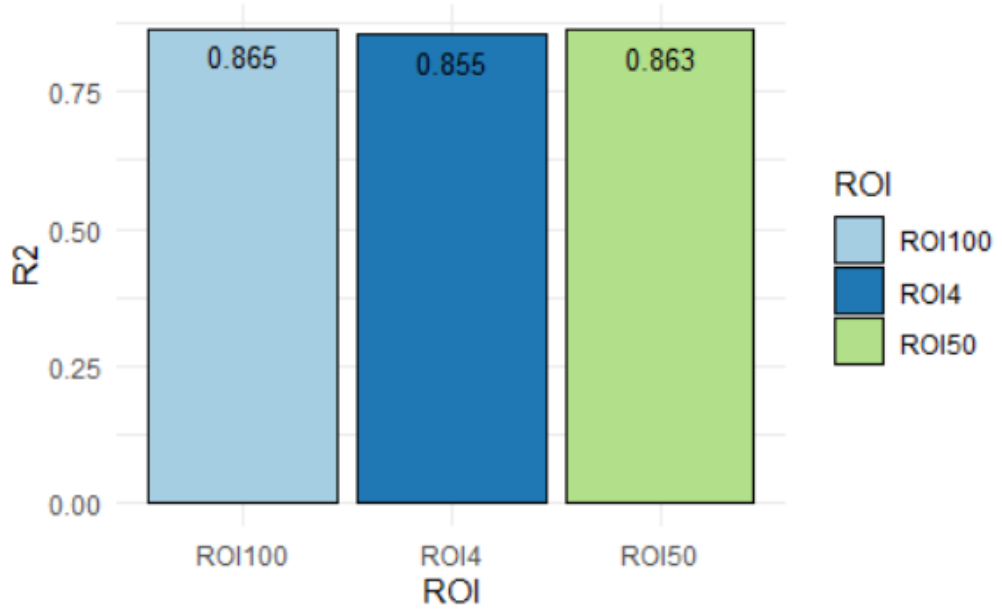
Para tener mayor claridad en la comparación de los modelos, se pueden utilizar gráficos de barra mediante la función *ggplot()* y el comando *geom_bar()* pertenecientes a la librería *ggplot,* disponibles dentro de la librería *tidyverse* (Wickham *et al*., 2019) (Figura 14). Utilizando el coeficiente de determinación, se busca comparar la variabilidad explicada en cada conjunto.

```
graph <- pls_compar %>%
  select(c("ROI", "R2")) %>%
  mutate(R2 = round(R2, 3)) %>%
  ggplot(aes(y = ROI, x = R2, fill = ROI)) +
  geom_bar(stat = "identity", color="black", position=position_dodge())+
  theme_minimal()+
  geom_text(aes(y=ROI, label=R2), vjust=1.6,
            color="BLACK", size=3.5, position=position_dodge(0.9))+
  scale_fill_brewer(palette="Paired")+
    coord_flip()

graph
```

Figura 14

Coeficientes de determinación (R^2) para modelos con diferente tamaño del área de interés (ROI).



Fuente: elaboración propia generada en R-project

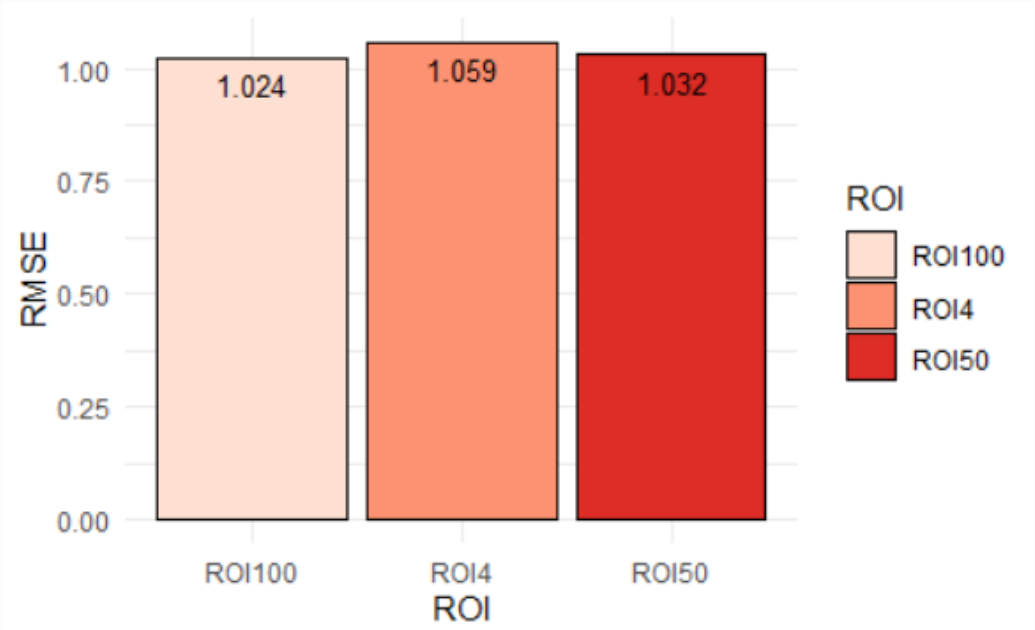
De igual manera se pueden comparar las métricas RMSE y RPD, las cuales podrán verse en las Figuras 15 y 16, con la siguiente programación:

```
graph2 <- pls_compar %>%
  select(c("ROI", "RMSE")) %>%
  mutate(RMSE = round(RMSE, 3)) %>%
  ggplot(aes(y = ROI, x = RMSE, fill = ROI)) +
  geom_bar(stat = "identity", color="black", position=position_dodge())+
  theme_minimal()+
  geom_text(aes(y=ROI, label=RMSE), vjust=1.6,
            color="BLACK", size=3.5, position=position_dodge(0.9))+
  scale_fill_brewer(palette="Reds")+
    coord_flip()

graph2
```

Figura 15

Cuadrado medio del error (RMSE) para modelos con diferente tamaño del área de interés (ROI)



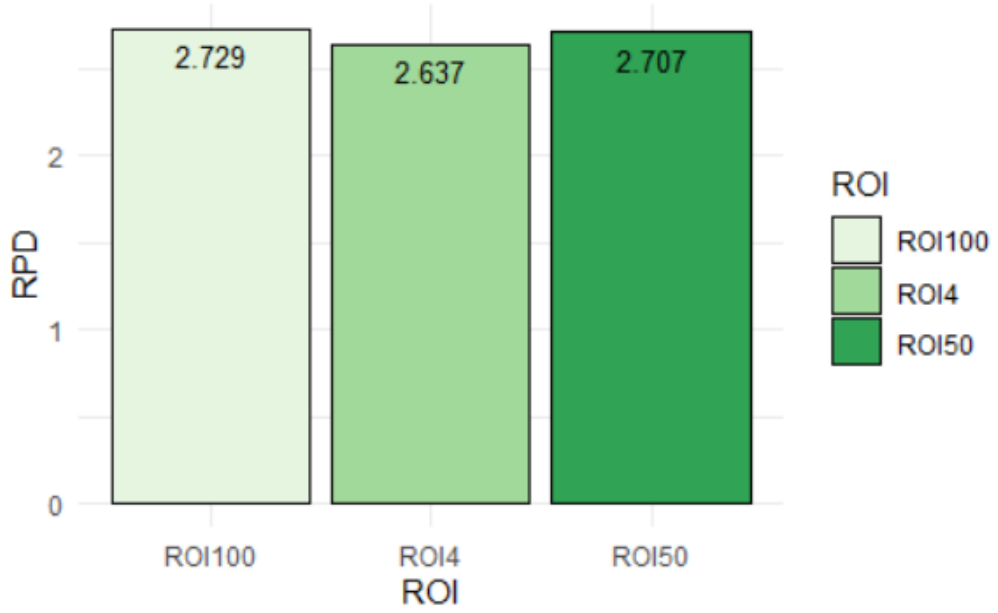
Fuente: elaboración propia generada en R-project

```
graph3 <- pls_compar %>%
  select(c("ROI", "RPD")) %>%
  mutate(RPD = round(RPD, 3)) %>%
  ggplot(aes(y = ROI, x = RPD, fill = ROI)) +
  geom_bar(stat = "identity", color="black", position=position_dodge())+
  theme_minimal()+
  geom_text(aes(y=ROI, label=RPD), vjust=1.6,
            color="BLACK", size=3.5, position=position_dodge(0.9))+
  scale_fill_brewer(palette="Greens")+
    coord_flip()

graph3
```

Figura 16

Desviación residual de la predicción (RPD) para modelos con diferente tamaño del área de interés (ROI)



Fuente: elaboración propia generada en R-project

De acuerdo con los resultados, se puede ver que existen diferencias mínimas en las métricas de evaluación, lo que quiere decir que el tamaño de ROI no tiene un efecto muy representativo. Por lo tanto, su selección debería hacerse de acuerdo con la capacidad de cómputo disponible al momento de realizar el procesamiento inicial de las imágenes.

Referencias bibliográficas

Boggs, T. (2014). *Spectral Python (SPy)*. https://www.spectralpython.net/index.html.

Hamner, B., y Frasco, M. (2018). *Metrics: Evaluation Metrics for Machine Learning, R package version 0.1.4,* https://CRAN.R-project.org/package=Metrics.

Harris, C.R., Millman, K.J., y van der Walt, S.J. (2020). *Array programming with NumPy*. Nature 585, 357–362. DOI: 10.1038/s41586-020-2649-2.

Hunter, J.D. (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95.

Luedeling, E., Caspersen, L., y Fernández, E. (2023). *chillR: Statistical Methods for Phenology Analysis in Temperate Fruit Trees*. R package version 0.74.1, https://CRAN.R-project.org/package=chillR.

McKinney, W., y others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

Meireles, J., Schweiger, A., y Cavender-Bares, J. (2017). spectrolab: Class and Methods for Spectral Data in R. doi:10.5281/zenodo.3934575, R package version 0.0.18.

Mevik, B.H., y Wehrens, R. (2007). El paquete pls: componente principal y regresión de mínimos cuadrados parciales en R. Journal of Statistical Software, 18 (2), 1–23. https://doi.org/10.18637/jss.v018.i02.

R Core Team, R. (2020). *A language and environment for statistical computing. R Foundation for Statistical Computing*, https://www.R-project.org/

Ramirez-Lopez, L., Stevens, A., Viscarra Rossel, R., Lobsey, C., Wadoux, A., y Breure, T. (2022). Resemble: Regression and similarity evaluation for memory-based learning in spectral chemometrics. R package Vignette R package version 2.2.1.

Rstudio Team, RStudio. (2021). *Integrated Development for R. RStudio,* Inc., http://www.rstudio.com/

Stevens, A., y Ramírez-López, L. (2022). *An introduction to the prospect package. R package Vignette R* package version 0.2.6.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley. CA: CreateSpace

Wadoux, A.M.C., Malone, B., Minasny, B., Fajardo, M., y McBratney, A.B. (2021). Soil Spectral Inference with R: Analyzing Digital Soil Spectra Using the R Programming Environment. *Springer Nature*. https://doi.org/10.1007/978-3-030-64896-1.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T., Miller, E., Bache, S., Müller, K., Ooms, J., Robinson, D., Seidel, D., Spinu, V., y Yutani, H. (2019). Welcome to the tidy verse. Journal of Open-Source Software, 4(43), 1686.

Autores

Mateo Vargas Zapata (Medellin, Antioquia, Colombia, 1993)

Magister en Ciencias Animales y Zootecnista de la Universidad de Antioquia. Sus publicaciones incluyen: *Captura, gestión, extracción de imágenes hiperespectrales y desarrollo de modelos para la predicción de parámetros fisicoquímicos de suelos* (2023); *Algoritmos de aprendizaje de máquina para la predicción de propiedades fisicoquímicas del suelo mediante información espectral: una revisión sistemática* (2022); *Plant growth evaluation of Cajanus cajan (L.) Millsp., Canavalia ensiformis (L.) DC. and Cratylia argentea (Desvaux) O. Kuntze., in soils degraded by sand and gravel extraction* (2017). *Reconocimiento Cum Laude* (2023).

Forma parte del grupo de investigación Grupo de Investigación Agrociencias Biodiversidad y Desarrollo GAMMA de la Universidad de Antioquia.

mateo.vargasz@udea.edu.co

Mario Fernando Cerón Muñoz (Pasto, Nariño, Colombia, 1972)

Doctor en Producción Animal de la Universidad Estadual Paulista (Brasil), y Zootecnista de la Universidad de Nariño (Colombia). Es profesor Titular de la Universidad de Antioquia

Ha sido reconocido como Doctor Honoris Causa en Andragogía y Galardón a la Excelencia Educativa por la Organización Internacional para la Inclusión y Calidad de la Educación OIICE (Chile 2023); Doctor Honoris Causa en Educación Inclusiva y Galardón a la Excelencia Educativa por la Organización Internacional para la Inclusión y Calidad de la Educación OIICE (México, 2022); y en los Premios en la Universidad de Antioquia: Extensión 2023, Excelencia docente 2017 e investigación 2010 (Medalla Francisco José de Caldas)

Sus publicaciones incluyen: Modelos Lineales para evaluaciones genéticas en animales (2024); Modelación aplicada a las ciencias animales: *Generalidades de R- Project* (2013); *Modelos Aplicados a las Ciencias Animales: Genética cuantitativa* (2008); *Modelos Aplicados a las Ciencias Animales: Evaluaciones genéticas* (2012); *Juzgamiento, clasificación y selección de ganado bubalino* (2011); Tres (3) capítulos del libro *Innovación en la Investigación Agropecuaria* (2022); *Manejo de información zootécnica en hatos lecheros. Evaluación genética de ganado Holstein de Antioquia* (2014).

Forma parte del Grupo de Investigación Agrociencias Biodiversidad y Desarrollo GAMMA de la Universidad de Antioquia.

mario.ceron@udea.edu.co

Marisol Medina Sierra (Medellín, Antioquia, Colombia, 1968)

Doctora en Ciencias Ambientales de la Universidad de Salamanca (España); Ingeniera Agrónoma de la Universidad Nacional de Colombia, Sede Medellín. Es Profesora Titular de la Universidad de Antioquia.

Sus publicaciones incluyen: *Captura, gestión, extracción de imágenes hiperespectrales y desarrollo de modelos para la predicción de parámetros fisicoquímicos de suelos* (2023); *Laboratorios Territoriales como experiencia innovadora en el acompañamiento técnico, económico y social a familias productoras de cacao* (2022); *Predicción de la Composición Química de Alimentos de Uso Animal Mediante NIRS. Aplicaciones e interpretación práctica* (2018); Dos capítulos del libro *Innovación en la Investigación Agropecuaria* (2022). Ha publicado artículos en revistas especializadas nacionales e internacionales de su especialidad.

Forma parte del Grupo de Investigación Agrociencias Biodiversidad y Desarrollo GAMMA de la Universidad de Antioquia.

marisol.medina@udea.edu.co

Luis Fernando Galeano Vasco (Ebéjico, Antioquia, Colombia, 1976)

Doctor en Ciencias Animales y Zootecnista de la Universidad de Antioquia, Colombia. Es Profesor asistente de la Facultad de Ciencias Agrarias de la Universidad de Antioquia. Sus publicaciones incluyen: Modelación aplicada a las ciencias animales: *Diseño experimental, con implementación del programa R- Project* (2013) y *Modelación aplicada a las ciencias animales: Generalidades de R- Project* (2013).

Forma parte del Grupo de Investigación Agrociencias Biodiversidad y Desarrollo GAMMA de la Universidad de Antioquia.

luis.galeano@udea.edu.co

El procesamiento de imágenes hiperespectrales es una técnica que permite extraer información valiosa de imágenes que contienen una gran cantidad de información multidimensional. Esta técnica se utiliza en diversas aplicaciones, como la agricultura, la medicina, la geología y meteorología, entre otras. En este manual se proporcionan ejemplos prácticos en los lenguajes de programación Python y R-project, que son ampliamente utilizados en el ámbito de la ciencia de datos y el análisis de imágenes.

Este manual está dirigido a estudiantes, investigadores y profesionales que deseen adquirir habilidades en el procesamiento de imágenes hiperespectrales (en especial procedentes de las cámaras hiperespectrales de la línea HySpex ®) y aplicarlas en sus ámbitos de trabajo o estudio. Se asume que los lectores tienen conocimientos básicos en programación y análisis de imágenes, pero no es necesario tener experiencia previa en el procesamiento de imágenes hiperespectrales.

Hyperspectral image processing is a technique that allows extracting valuable information from images containing a large amount of multidimensional information. This technique is used in various applications such as agriculture, medicine, geology, and meteorology, among others. This manual provides practical examples in Python and R-project programming languages, which are widely used in the fields of data science and image analysis.

This handbook is intended for students, researchers, and professionals who wish to acquire skills in hyperspectral image processing (especially those from HySpex® hyperspectral cameras) and apply them in their fields of work or study. It is assumed that readers have a basic knowledge of programming and image analysis, but previous experience in hyperspectral image processing is not necessary

eISBN:978-958-722-932-5

CEDAIT
Centro de Desarrollo Agrobiotecnológico de Innovación e Integración Territorial

UNIVERSIDAD DE ANTIOQUIA

Editorial UTP