Captura de imágenes satelitales Sentinel-2

Se ha realizado un estudio de las diferentes alternativas para la descarga automática de imágenes satelitales. En concreto, se han realizado pruebas con 3 librerías: SentinelHub y SentinelSat de Python y rsat de R.

Existen seis familias de satélites Sentinel, desarrolladas por la Agencia Espacial Europea (ESA) y enmarcado dentro del programa Copernicus de la Comisión Europea (CE), cuyo objetivo es la observación de la Tierra.

Entre todas las familias nos quedamos con la pareja de satélites Sentinel-2, ya que proporciona imágenes ópticas terrestres de alta resolución y las bandas espectrales (rojo, verde y azul) para componer imágenes en color natural (imágenes RGB). Además, están capturadas con el objetivo de ayudar en los servicios de emergencia, tal y como se detalla en el apartado 3.2.4.2 del [Documento de requisitos de la misión Sentinel-2](https://esamultimedia.esa.int/docs/GMES/Sentinel-2_MRD.pdf).

Existen otras misiones satelitales, entre las cuales destacan MODIS desarrollado por la NASA y Landsat desarrollado por la NASA y el Servicio Geológico de los Estados Unidos (USGS).

Los datos de Sentinel-2 tienen bandas espectrales muy similares a Landsat-8. Ambos satélites capturan las bandas rojo, verde y azul, pero en el caso de Sentinel-2, la resolución espacial es mejor, siendo de 10 metros respecto a los 30 metros de resolución que ofrecen las bandas RGB de Landsat-8. En cuanto a los satélites MODIS, estos capturan las bandas RBG con una resolución de 500 metros, bastante peor que en Sentinel y Landsat.

Pasos seguidos en cada librería utilizada para la obtención de imágenes satelitales:

1. Iniciar sesión en el servicio web que proporciona las imágenes
2. Definir región y tiempo de interés
3. Realizar la búsqueda de las imágenes
4. Descargar las imágenes
5. Mostrar imágenes descargadas

## Librería SentinelSat

SentinelSat es una librería de Python que facilita la búsqueda y la descarga de las imágenes de satélite Sentinel del repositorio Copernicus Open Access Hub.

### Iniciar sesión

En primer lugar, cargamos las librerías que vamos a utilizar y configuramos las credenciales de la API.

from sentinelsat import SentinelAPI, placename\_to\_wkt  
from shapely.geometry import Polygon  
from datetime import date  
import geopandas as gpd  
from matplotlib import pyplot as plt  
  
api = SentinelAPI('user','pass','https://apihub.copernicus.eu/apihub')

### Definir región y tiempo de interés

Generamos un BoundingBox para delimitar la zona de interés. Empleamos el objeto “Polygon” para generar el rectángulo correspondiente al BoundingBox. También definimos el rango de fechas en el que queremos realizar la búsqueda.

lat\_min = 42.53  
lon\_min = -1.98  
lat\_max = 42.54  
lon\_max = -1.97  
poly = Polygon([(lon\_min, lat\_max), (lon\_max, lat\_max),

(lon\_max, lat\_min), (lon\_min, lat\_min)])

date\_range = (date(2022, 7, 1), date(2022, 7, 30))

### Búsqueda de imágenes

A partir de la región y tiempo de interés, filtramos los productos publicados en SciHub. También se define un rango de nubosidad, que a lo máximo será del 50%, y un límite de 6 productos.

footprint = poly.wkt  
bbox\_products = api.query(footprint,  
 date = date\_range,  
 platformname='Sentinel-2',  
 cloudcoverpercentage=(0, 50),  
 limit = 6)

Mostramos los productos filtrados. Tenemos 6 productos que corresponden a la misma región, pero de fechas distintas.

Podemos comprobar que todos los productos corresponden a la misma región, observando que todos tienen la misma geometría.

products\_gdf = api.to\_geodataframe(bbox\_products)  
blankIndex=[''] \* len(products\_gdf)  
products\_gdf.index=blankIndex  
pd.set\_option('display.max\_colwidth', 120)   
products\_gdf[['generationdate', 'geometry']]

|  |  |
| --- | --- |
| generationdate | geometry |
| 2022-07-27 12:39:49 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |
| 2022-07-27 11:44:39 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |
| 2022-07-17 12:35:55 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |
| 2022-07-17 11:46:15 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |
| 2022-07-12 21:31:38 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |
| 2022-07-12 14:39:48 | MULTIPOLYGON (((-1.66699 42.35632, -1.64554 43.34483, -3.00024 43.35286, -3.00024 42.36407, -1.66699 42.35632))) |

### 

### Descarga de imágenes

Descargamos de imágenes mediante el comando ‘download\_all\_quicklooks’, las cuales se almacenan en la ruta especificada.

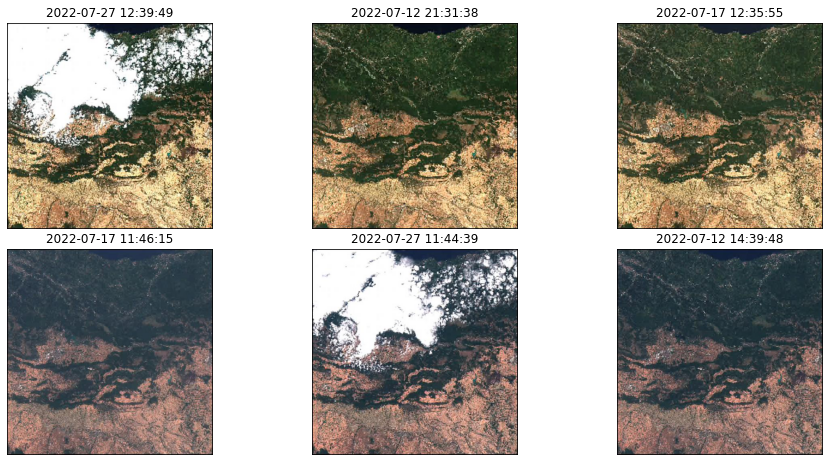
api.download\_all\_quicklooks(bbox\_products, 'images/bbox')

### 

### Mostrar de imágenes descargadas

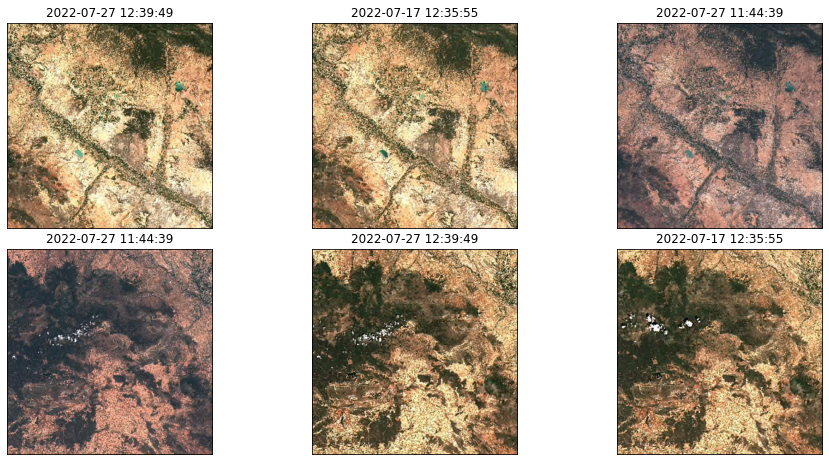
Mostramos las imágenes descargadas, correspondientes a los 6 productos de SciHub filtrados.

def mostrarImagenes(products: gpd.GeoDataFrame, path: str):  
  
 indexes = pd.Series([0, 1, 2, 3, 4, 5])  
 products = products.set\_index(indexes)  
 f, axarr = plt.subplots(2,3, figsize=(15, 15))  
  
 for index, row in products.iterrows():  
 i = index % 2  
 j = index - (3 \* int(index > 2))  
 img = plt.imread(path + row['title'] + ".jpeg")  
 axarr[i,j].set\_title(row['generationdate'])  
 axarr[i,j].axis('off')  
 axarr[i,j].imshow(img)  
   
 plt.subplots\_adjust(left=0.1,   
 bottom=0.1,   
 right=0.9,   
 top=0.5,   
 wspace=0.2,   
 hspace=0.1)  
  
 plt.show()  
  
mostrarImagenes(products\_gdf, "images/bbox/")



Repetimos el proceso, seleccionando una región de interés con el nombre del municipio. En este caso hemos hecho la prueba con Tudela.

tudela\_products = api.query(placename\_to\_wkt("Tudela")[0],  
 date = date\_range,  
 platformname='Sentinel-2',  
 cloudcoverpercentage=(0, 50),  
 limit = 6)  
  
tudela\_gdf = api.to\_geodataframe(tudela\_products)  
api.download\_all\_quicklooks(tudela\_products, 'images/tudela/')  
mostrarImagenes(tudela\_gdf, "images/tudela/")



## Librería SentinelHub

SentinelHub es una plataforma SIG (Sistema de Información Geográfica) basada en la nube para la distribución, gestión y análisis de imágenes satelitales. Podemos acceder a los servicios de SentinelHub a través de la librería sentinelhub de Python.

### Iniciar sesión

En primer lugar, cargamos las librerías que vamos a utilizar y configuramos las credenciales de la API.

from sentinelhub import (  
 SHConfig,  
 CRS,  
 BBox,  
 DataCollection,  
 MimeType,  
 SentinelHubDownloadClient,  
 SentinelHubRequest,  
 bbox\_to\_dimensions,  
 WmsRequest,  
 CustomUrlParam  
)  
import datetime  
import matplotlib.pyplot as plt  
import numpy as np  
  
config = SHConfig()  
config.instance\_id = '70265a40-8c2b-4751-86d9-217b24acd2e6'  
config.sh\_client\_id = 'ef80979a-159a-4758-8cfc-8034ec77943f'  
config.sh\_client\_secret = 'ZF,95^NS[l5w+Hf3FbJ\_}HcMWduP&bFzDekDhXt\_'

### Definir región y tiempo de interés

Generamos un BoundingBox para delimitar la zona de interés. Para ello, creamos la clase BoundingBox formada por un objeto BBox propio de SentinelHub y por las dimensiones de dicho objeto. Creamos una nueva instancia de la clase mediante las coordenadas geográficas de dos puntos (esquina superior izquierda e inferior derecha) del BoundingBox, el Sistema de Referencia de Coordenadas (CRS) y la resolución espacial de la imagen. También definimos el rango de fechas en el que queremos realizar la búsqueda.

class BoundingBox():  
 def \_\_init\_\_(self, coords, crs, resolution):  
 self.bbox = BBox(bbox = coords, crs = crs)  
 self.size = bbox\_to\_dimensions(bbox = self.bbox, resolution = resolution)

bbox\_coords = [-1.98, 42.54, -1.97, 42.53]  
bbox = BoundingBox(bbox\_coords, CRS.WGS84, resolution = 10)  
date\_range = ("2022-07-01","2022-07-30")

### Búsqueda de imágenes

Además de la región y tiempo de interés, para realizar la búsqueda de las imágenes, es necesario crear un script de evaluación de las bandas espectrales. En nuestro caso, queremos obtener el color natural, por lo tanto, trabajamos sólo con las bandas rojo, verde y azul.

evalscript\_true\_color = """  
 //VERSION=3  
  
 function setup() {  
 return {  
 input: [{  
 bands: ["B02", "B03", "B04"]  
 }],  
 output: {  
 bands: 3  
 }  
 };  
 }  
  
 function evaluatePixel(sample) {  
 return [sample.B04, sample.B03, sample.B02];  
 }  
"""

Con todo ya definido, podemos proceder a la búsqueda de las imágenes, creando una lista de peticiones a la API de SentinelHub.

def CreateSentinelRequest(evalscript: str, time\_interval: tuple, bb: BoundingBox):  
 request = SentinelHubRequest(  
 evalscript = evalscript,  
 input\_data = [  
 SentinelHubRequest.input\_data(  
 data\_collection = DataCollection.SENTINEL2\_L1C,  
 time\_interval = time\_interval,  
 )  
 ],  
 responses = [  
 SentinelHubRequest.output\_response("default", MimeType.PNG),  
 ],  
 bbox = bb.bbox,  
 size = bb.size,  
 config = config,  
 )  
 return request  
  
def GetSentinelRequest(evalscript: str, slots: list, bounding\_box: BoundingBox):   
  
 # create a list of requests  
 list\_of\_requests = [CreateSentinelRequest(evalscript, slot, bounding\_box) for slot in slots]  
 list\_of\_requests = [request.download\_list[0] for request in list\_of\_requests]  
  
 return list\_of\_requests  
  
requests = GetSentinelRequest (  
 script = evalscript\_true\_color,  
 dates = date\_range,  
 bounding\_box = bbox  
)

### Descarga de imágenes

Efectuamos las peticiones a través de múltiples hilos a partir del listado de peticiones obtenido anteriormente.

def GetSentinelImages(list\_of\_requests: list):   
 # download data with multiple threads  
 images = SentinelHubDownloadClient(config=config).download(list\_of\_requests, max\_threads=5)  
 return images  
  
images = GetSentinelImages(requests)

### 

### Mostrar imágenes descargadas

def ShowImages(images, slots, bbox\_size):  
   
 ncols = 4  
 nrows = 3  
 aspect\_ratio = bbox\_size[0] / bbox\_size[1]  
 subplot\_kw = {"xticks": [], "yticks": [], "frame\_on": False}  
  
 fig, axs = plt.subplots(ncols=ncols, nrows=nrows, figsize=(5 \* ncols \* aspect\_ratio, 5 \* nrows), subplot\_kw=subplot\_kw)  
  
 for idx, image in enumerate(images):  
 ax = axs[idx // ncols][idx % ncols]  
 ax.imshow(np.clip(image \* 2.5 / 255, 0, 1))  
 ax.set\_title(f"{slots[idx][0]} - {slots[idx][1]}", fontsize=10)  
  
 plt.tight\_layout()  
  
ShowImages(images)

Existe una forma directa de descargar de las imágenes. Hacemos la prueba de descargar imágenes recientes del cauce del río Ebro a su paso por Tudela.

tudela\_coords = [-1.61, 42.07, -1.58, 42.05]  
tudela\_bbox = BoundingBox(tudela\_coords, CRS.WGS84, 10)  
  
wms\_request = WmsRequest(  
 data\_collection = DataCollection.SENTINEL2\_L1C,  
 layer = "NATURAL-COLOR",  
 bbox = tudela\_bbox.bbox,  
 time = "latest",  
 width = tudela\_bbox.size[0],  
 height = tudela\_bbox.size[1],  
 custom\_url\_params = {CustomUrlParam.SHOWLOGO: False},  
 image\_format = MimeType.PNG,  
 config = config  
)  
  
print(wms\_request.get\_url\_list())  
wms\_img = wms\_request.get\_data()

from PIL import Image  
im = Image.fromarray(wms\_img[-1])  
im.save("C:/i3/EMERAL/Sentinel/savedimage.png")  
  
def plot\_image(image, factor=1):  
 plt.subplots(nrows=1, ncols=1, figsize=(15, 7))  
  
 if np.issubdtype(image.dtype, np.floating):  
 plt.imshow(np.minimum(image \* factor, 1))  
 else:  
 plt.imshow(image)  
   
plot\_image(wms\_img[-1])



## Librería rsat

rsat es una librería de R desarrollada por el grupo de investigación de estadística espacial de la Universidad Pública de Navarra. Permite la descarga de imágenes satelitales de Landsat, MODIS y Sentinel para una región y hora de interés. Para ello, rsat se conecta con distintas APIs open source de imágenes satelitales, en concreto accede a los siguientes servicios web:

* USGS es la única agencia científica del Departamento del Interior de los Estados Unidos.
* EarthData es un repositorio de conjuntos de datos de observación de la Tierra de la NASA.
* SciHub es un servicio web que da acceso al centro de datos científicos de Copernicus.

### Iniciar sesión

En primer lugar cargamos las librerías necesarias y configuramos las credenciales de la API. Si no se especifica la API de la que obtener las imágenes, rsat obtendrá las imágenes de las tres APIs anteriores.

library(rsat)  
library(sf)  
  
set\_credentials("rsat.package", "UpnaSSG.2021")

### Definir región y tiempo de interés (roi y toi)

Generamos un BoundingBox que incluya la zona de interés. Las coordenadas máximas hacen referencia a la esquina superior derecha y las mínimas a la inferior izquierda. También definimos el sistema de referencia de coordenadas, en este caso utilizamos el CRS más habitual (WGS 84 EPSG:4326).

# Región de interés definida mediante BoundingBox  
roi <- st\_sf(st\_as\_sfc(st\_bbox(c(  
 xmin = -1.98,  
 xmax = -1.97,  
 ymin = 42.53,  
 ymax = 42.54  
), crs = 4326)))

# Fecha de interés, mes de Julio de 2021  
toi <- seq(as.Date("2021-07-01"), as.Date("2021-07-31"),1)

### Búsqueda de imágenes

Se centra la búsqueda en imágenes Sentinel, debido a su mayor resolución para el área y tiempo definidos.

search.records <- rsat\_search(region = roi,  
 product = c("S2MSI2A"),  
 dates = toi, lvl=2)

## Attached 6 new entries of product S2MSI2A to your rtoi.

# Se obtiene una variable de tipo records  
class(search.records)

# Número de imágenes encontradas  
length(search.records)

## [1] 6

### 

rtoi (region and time of interest) es un objeto diseñado para gestionar regiones que consisten en varios mosaicos. El objeto se compone de partes:

* Localización de la base de datos: Ruta dónde se almacenan las imágenes Sentinel descargadas (db.path).
* Localización de las regiones: Ruta dónde se almacenan los archivos rtoi (rtoi.path).
* El objeto del lenguaje R.

base\_path <- tempdir()  
db.path <- file.path(base\_path, "DATABASE")  
rtoi.path <- file.path(base\_path, "regions")  
# Creamos los direcctorios si no existen  
if (!dir.exists(db.path)) { dir.create(db.path) }  
if (!dir.exists(rtoi.path)) { dir.create(rtoi.path) }

Creamos un objeto rtoi con la información anterior.

rtoi <- new\_rtoi("rtoi", roi, rtoi.path, db.path)

Como hemos visto anteriormente el objeto rtoi se almacena en un archivo, por lo tanto, podemos leer ese archivo para recuperar el objeto.

rtoi <- read\_rtoi(file.path(rtoi.path,"rtoi"))  
plot(rtoi, "preview")

Imagen que contiene Calendario

Descripción generada automáticamente

Tenemos 6 imágenes, por lo tanto, vemos que no se recoge una imagen por cada día. Mostramos un calendario con los días en los que se han recogido imágenes de Sentinel-02 durante el mes de Julio de 2021.

plot(rtoi, "dates")

Tabla

Descripción generada automáticamente

### Descarga de imágenes

Se procede a la descarga de imágenes mediante el comando ‘rsat\_download’, las cuales se almacenan en la ruta especificada para la bbdd (db.path).

rsat\_download(rtoi)

Después de la descarga, se crean las imágenes mosaico con las bandas RGB de la mejor resolución posible (10 metros).

rsat\_mosaic(rtoi,overwrite=TRUE,bfilter=c("B02\_10m","B03\_10m","B04\_10m"))

Creamos la función RGB para generar una capa ráster con las bandas de rojo, verde y azul de todas las imágenes descargadas.

RGB = function(green, red, blue){  
 return(c(red, green, blue))  
}  
rsat\_derive(rtoi, product = "S2MSI2A", variable = "rgb", fun = RGB)

Podemos ver que el objeto rtoi contine tanto las bandas de cada color con resolución de 10 metros, como la unión de todas ellas en la variable “rgb”.

rsat\_list\_data(rtoi)

## satellite product stage variable  
## 1 Sentinel-2 S2MSI2A variables rgb  
## 2 Sentinel-2 S2MSI2A mosaic B03\_10m  
## 3 Sentinel-2 S2MSI2A mosaic B02\_10m  
## 4 Sentinel-2 S2MSI2A mosaic B04\_10m

### Mostrar de imágenes descargadas

Seleccionamos la capa ráster que contiene todas las bandas de color de las imágenes Sentinel-02 descargadas y la mostramos. Podemos ver que está delimitada por el BoundingBox definido inicialmente.

suppressWarnings(rgb <- rsat\_get\_SpatRaster(rtoi, "S2MSI2A", "rgb"))

plot(rgb, "view")

Escala de tiempo

Descripción generada automáticamente

## Comparación

La librería SentinelSat no ofrece la posibilidad de descargar imágenes a partir de un BoundingBox, sino que este se define para descargar los productos que se solapan con él