000_data_downloading

September 3, 2022

1 Data downloading

[ ]:

```
# %cd ..
```

[1]:

```
import hda
from pathlib import Path
from hda import Client
import os

from sentinelsat import SentinelAPI, read_geojson, geojson_to_wkt
from datetime import date

from shapely import wkt
import geopandas as gpd
import rasterio as rio
from pyproj import Proj
from pyproj import Transformer
from rasterio.mask import mask
from shapely.geometry import Polygon
from rasterio.warp import calculate_default_transform, reproject
import zipfile
import numpy as np
from skimage.transform import resize
import shutil
import pickle
import zarr
from glob import glob

User = ''
Password = ''
Token = ''
```

1.1 Search of products

[3]:

```
# Define range of interest
```

[8]:

```
api = SentinelAPI(User, Password,'https://apihub.copernicus.eu/apihub')
first_date = date(2017, 1, 1)
last_date = date(2017, 12, 31)

footprint = "POLYGON((-70.9649357878204 -33.81244707234685,-70.96591676385808␣
 ↪-33.89260670554516,-70.84251675089554 -33.89488656955273,-70.84506715448308␣
 ↪-33.81375110356652,-70.9649357878204 -33.81244707234685))"

products = api.query(footprint,
                     date=(first_date, last_date),
                         platformname='Sentinel-2',
                     processinglevel='Level-1C')#,
                     #cloudcoverpercentage=(0, 100))

print(f'{len(products)} products found')
```

```
4 products found
```

1.2 Extract area of interest in original CRS

[3]:

```
initial_geometry = [wkt.loads(footprint)]
```

[2]:

```
c = Client(url='https://wekeo-broker.apps.mercator.dpi.wekeo.eu/databroker',
           user = User,
           password = Password,
           token=Token, debug=False, quiet=True);
```

[5]:

```
# Methods
def area2ts(p):
    n = p['filename'].split('.')[0]
    datstrip = p['datastripidentifier'].split('_')[8][1:]
    granuleid = p['granuleidentifier'].split('_')



    identifier = p['identifier']
    request = { "datasetId": "EO:ESA:DAT:SENTINEL-2:MSI",
               "stringInputValues": [{"name": "productIdentifier", "value":␣
 ↪identifier}]}

    matches = c.search(request)
    title = p['title']
    matches.download()

    filepath = title + '.zip'
    with zipfile.ZipFile(filepath,"r") as zip_ref:
        zip_ref.extractall("./data")
```

```python
    os.remove(filepath)


    source_crs = "EPSG:4326"

    fullband = []

    granule_folder = glob("./data/{}.SAFE/GRANULE/*/".format(n), recursive = True)[0]

    for band in ['B01','B02','B03','B04','B05','B06','B07','B08','B09','B10','B11','B12','B8A']:
        with rio.open('{}/IMG_DATA/{}_{}_{}.jp2'.format(granule_folder, n.split('_')[5], n.split('_')[2], band)) as img:
            #print(img.meta)
            # Error when loading the jp2 in mac or windows. I can't take the crs from the image.
            target_crs = 'EPSG:32719'#img.crs.to_string()
            x, y = initial_geometry[0].exterior.coords.xy

            aoi = list(zip(x, y))
            transformer = Transformer.from_crs(source_crs, target_crs)

            new_coords = []
            for co in aoi:
                t = transformer.transform(co[1],co[0])
                new_coords.append(t)

            aoi = [Polygon(new_coords)]

            clipped, transform = mask(img, aoi, crop=True)
            metadata = img.meta.copy()

            metadata.update({"transform": transform,
                         "height": clipped.shape[1],
                         "width": clipped.shape[2]#,
                            #'driver': 'GTiff'
                            })

            with rio.open('./temp/{}_{}.tif'.format(n, band), 'w', **metadata) as dst:
                dst.write(clipped)


            with rio.open('./temp/{}_{}.tif'.format(n,band)) as r:
                fullband.append(r.read())
            os.remove('./temp/{}_{}.tif'.format(n,band))
            os.remove('./temp/{}_{}.tif.aux.xml'.format(n,band))


    fullband_resized = []
    max_shape = tuple(np.max([np.shape(np.squeeze(band)) for band in␣
↪fullband],axis=0))


    for img in fullband:
        if img.shape != max_shape:
            image_resized = resize(np.squeeze(img), max_shape,␣
↪anti_aliasing=False, preserve_range=True)
            fullband_resized.append(image_resized)

    shutil.rmtree('./data/{}.SAFE'.format(n))
    return np.array(fullband_resized, dtype=np.int16)

def load_and_append_zarr(array, filename='output.zarr'):
    if os.path.isdir(filename):
        z = zarr.open(filename, mode='a')
        z.append(array[np.newaxis])
        zarr.save(filename, z)
    else:
        z.save(filename, array[np.newaxis])

def load_and_expand_zarr(array, key, filename='output.zarr', debug=False):
    if os.path.isdir(filename):
        z = zarr.open(filename) #, mode='a')
        z[key] = array
        #zarr.save(filename, z)
    else:
        zarr.save(filename, **{key: array})
```

[ ]:

```
# #TOA 2 LAC
# import ee
# ee.Authenticate()
```

[ ]:

```
# %%time
# from SIAC import SIAC_S2 #conda install lightgbm #https://github.com/
↪multiply-org/atmospheric_correction
# global_dem = '/vsicurl/https://gws-access.jasmin.ac.uk/public/nceo_ard/DEM_V3/
↪global_dem.vrt'
# cams_dir = '/vsicurl/https://gws-access.jasmin.ac.uk/public/nceo_ard/cams/'
# SIAC_S2('S2B_MSIL1C_20181225T143749_N0207_R096_T19HCC_20181225T175914.SAFE',␣
↪global_dem = global_dem, cams_dir=cams_dir)
```

1.3 Download of selected products

[17]:

```
## Testing parallel loading of ZARR
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor

def paral(func, lista, N, threads=True, processes=False):
    if processes:
        with ProcessPoolExecutor(max_workers=N) as executor:
            results = executor.map(func, lista)
        return list(results)
    elif threads:
        with ThreadPoolExecutor(max_workers=N) as executor:
            results = executor.map(func, lista)
        return list(results)
```

[7]:

```
outputZarr = './data/2017.zarr'
failedProducts = []
for p in products:
    try:
        if os.path.isdir(outputZarr):
            if p not in list(zarr.load(outputZarr)):
                print('{} downloading new dataset'.format(p))
                image = area2ts(products[p])
                load_and_expand_zarr(image, p, filename=outputZarr)
            else:
                print('{} already downloaded and processed'.format(p))
        else:
            print('{} downloading new initial dataset'.format(p))
            image = area2ts(products[p])
            load_and_expand_zarr(image, p, filename=outputZarr)


    except Exception as e:
        if  os.path.isfile('{}.zip'.format(products[p]['title'])):
            os.remove('{}.zip'.format(products[p]['title']))

        if  os.path.isdir('data/{}.SAFE'.format(products[p]['title'])):
            shutil.rmtree('data/{}.SAFE'.format(products[p]['title']))

        failedProducts.append(p)
        print('{} error. Ignoring dataset'.format(p))
        print(e)
```

```
470e2932-6334-4b3d-a272-be23c63a2d1b already downloaded and processed
92d7472f-8a5b-4196-a1f5-1dd36bbdd6e1 already downloaded and processed
e543432e-946f-4ada-83ed-eb59baf18149 already downloaded and processed
15a5e8db-fac1-47e4-81df-9a09ab02b408 already downloaded and processed
```

[ ]:

```
# ## output of failed
# with open('{}_failed.txt'.format(outputZarr.split('.')[0]), 'w') as fp:
#     for item in np.unique(failedProducts):
#         # write each item on a new line
#         fp.write("%s\n" % item)
```

[ ]:

```
# ## do dataframe with the information. Basically matches to dataframe
# entries = list(zarr.load(outputZarr))
# df = api.to_dataframe(products)
# df = df.loc[entries]
# df.to_pickle('{}_df.pickle'.format(outputZarr.split('.')[0]))
# df
```

[ ]:

```
# os.system("zip -r {}.zip {} ".format(outputZarr.split('.')[0], outputZarr)) #>/dev/null 2>&1
# #os.system("unzip {}.zip".format(outputZarr.split('.')[0]))
```