# 001_data_preprocessing

September 3, 2022

## 1 Data preprocessing

[ ]:
```
#%cd ..
```

### 1.1 Dependencies and helper functions

[ ]:
```
# Standard imports
import numpy as np
import pandas as pd
import os
import zarr
import ipywidgets as widgets
import matplotlib.pyplot as plt
from umap import UMAP
from sklearn.cluster import KMeans
from skimage import segmentation, feature, future
from sklearn.ensemble import RandomForestClassifier
from functools import partial
import matplotlib.patches as patches
from skimage.segmentation import morphological_chan_vese
import cv2
import imutils
from scipy.signal import savgol_filter
import datetime
from scipy.interpolate import interp1d
import matplotlib.dates as mdates
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
import pickle
import calmap

# Prettier plots
import seaborn as sns
sns.set(font='Palatino',
        rc={
 'axes.axisbelow': False,
 'axes.edgecolor': 'k',
 'axes.facecolor': 'None',
```

```
 'axes.grid' : False,
 'axes.spines.right': False,
 'axes.spines.top': False,
 'figure.facecolor': 'white',
 'lines.solid_capstyle': 'round',
 'patch.edgecolor': 'w',
 'patch.force_edgecolor': True,
 'xtick.bottom': True,
 'xtick.direction': 'out',
 'xtick.top': False,
 'ytick.direction': 'out',
 'ytick.left': True,
 'ytick.right': False})

# Vectorial plot
import matplotlib_inline.backend_inline as backend_inline
backend_inline.set_matplotlib_formats('svg')

## 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)

def loadindex(index):
    try:
        return img[index][:]
    except Exception as e:
        print(e)

## Visualization method
def visualize_data(array_data, array_segments = None, array_times = None,␣
 ↪cmap='crest'):
    # Widget slider to browse the data
    index = widgets.IntSlider(
        value=5, min=0, max=array_data.shape[0] - 1, step=1, description="Index"
    )

    # Other widget slider to browse the channels
    channel = widgets.IntSlider(
```

```
        value=5, min=0, max=array_data.shape[3] - 1, step=1,␣
↪description="Channel"
    )

    # Checkbox to display RGB (override the channel)
    display_RGB = widgets.Checkbox(description="Display RGB", value=False)

    ui = widgets.HBox([index, channel, display_RGB])

    # Widget interaction function
    def anim(index_value, channel_value, display_RGB_value):
        fig = plt.figure(figsize=(10,8))
        if display_RGB_value:
            plt.imshow( array_data[index_value, :, :, (3,2,1)].swapaxes(0,1).
↪swapaxes(1, 2))
        else:
            plt.imshow(array_data[index_value, :, :, channel_value], cmap =␣
↪cmap)
        if array_segments is not None:
            if np.sum(array_segments[index_value])>0:
                plt.contour(array_segments[index_value], [0.5], colors='r')
        if array_times is not None:
            plt.title('Acquisition time: ' + str(array_times[index_value]))
        else:
            plt.title('Acquisition time: ' +␣
↪str(df['beginposition'][index_value]))
        plt.axis('off')
        return

    # Link widget and function
    out = widgets.interactive_output(anim, {"index_value": index,␣
↪'channel_value': channel, 'display_RGB_value': display_RGB})

    # Display result
    return ui, out
```

### 1.2 Loading metadata dataframe

[ ]:
```
# Load all dataframes in a list
l_df = []
for file in os.listdir(r"./data"):
    if file.endswith(".pickle"):
        #df_temp = pd.read_pickle("Data engineering/" + file)
        with open(r"./data/" + file, "rb") as fh:
            df_temp = pickle.load(fh)
        df_temp['year'] = int(file.split("_")[0])
```

```
        l_df.append(df_temp)

# Merge all dataframes
df = pd.concat(l_df, axis=0)

# Remove useless columns
df = df[['datatakesensingstart', 'beginposition', 'endposition',␣
 ↪'ingestiondate', 'processinglevel', 'platformname', 'size','year']]

# Sort by date
df = df.sort_values(by="beginposition")

# Delete initial list of dataframes
del l_df
```

[ ]:
```
def display_table_summary():
    display(df)
```

### 1.3 Loading arrays into memory

[ ]:
```
# Load all arrays in memory
with zarr.open(r"./data/" + str(df.year.iloc[0]) + '.zarr', mode = 'r') as img:
    shape = np.shape(img[df.index[0]][:])

# Crop images as they're too large for ML on a laptop
shape = ((551-150), (751-150), shape[0]-1)

array_data = np.zeros(dtype=np.int16, shape=(len(df.year),␣
 ↪shape[0],shape[1],shape[2]))
for i, (index, year) in enumerate(zip(df.index, df.year)):
    with zarr.open(r"./data/" + str(year) + '.zarr', mode = 'r') as img:
        try:
            pro = img[index][:]
            pro = np.delete(pro, 9, axis = 0)
            pro = np.swapaxes(pro, 0, 1)
            pro = np.swapaxes(pro, 1, 2)

            # Crop images as they're too large for ML on a laptop
            pro = pro[150:551,150:751,:]

            array_data[i] = pro
        except Exception as e:
            print(year, e)
```

[ ]:
```
# Normalize by the 90th percentile for RGB channels
percentile = np.percentile(array_data[:,:,:,(1,2,3)], 90)
```

```
for ax in [1,2,3]:
    array_data[:,:,:,ax] = (array_data[:,:,:,ax].astype(np.float64) /␣
↪percentile * 255).astype(np.int16)
    # Cap values above 255
    array_data[:,:,:,ax] = np.clip(array_data[:,:,:,ax], 0, 255)
```

[ ]:
```
def visualize_all_products():
    ui, out = visualize_data(array_data, array_times = df['beginposition'])
    display(ui, out)
```

[ ]:
```
def calendar_all_products():
    dates = df['beginposition']
    events = pd.Series(1, index=dates)

    ca = calmap.calendarplot(events, monthticks=3, daylabels='MTWTFSS',
                         dayticks=[0, 2, 4, 6], cmap='YlGn',
                         fillcolor='grey', linewidth=0,
                         yearlabel_kws={'fontsize': 20},
                         fig_kws=dict(figsize=(28, 8)))
    plt.show()
```

### 1.4 Filtering out products with clouds and black images

[ ]:
```
l_mean = [np.mean(x.flatten()) for x in array_data[:, :, :, :]]
# Compute mean value of the image and check outliers
lb, hb = 500, 1750

def plot_average_image_value():
    # Create figure and axes
    fig, ax = plt.subplots(1, figsize = (10,5))
    fig.patch.set_facecolor('white')
    plt.hist(l_mean, bins=100)
    plt.ylim(0, 50)
    plt.xlim(0, 6100)
    plt.xlabel("Image mean value")
    plt.ylabel("Frequency")

    # Create two rectangle patches to show discarded data and add them to the␣
↪plot
    rect = patches.Rectangle((0, 0), lb, 50, alpha = 0.3, facecolor="red")
    rect2 = patches.Rectangle((hb, 0), 6100-hb, 50, alpha = 0.3,␣
↪facecolor="red")
    ax.add_patch(rect)
    ax.add_patch(rect2)
    plt.title('Average image value (across all channels) distribution')
    plt.show()
```

[ ]:
```
# Discard images out of selected threshold
l_idx_to_keep = [idx for idx, m in enumerate(l_mean) if m > lb and m < hb]
array_data_cropped = array_data[l_idx_to_keep, :, :, :]
array_times = [df['beginposition'][index_value] for index_value in␣
 ↪l_idx_to_keep]
```

[ ]:
```
def visualize_selected_products():
    ui, out = visualize_data(array_data_cropped, array_times = array_times)
    display(ui, out)
```

[ ]:
```
def calendar_selected_products():
    dates = array_times
    events = pd.Series(1, index=dates)

    ca = calmap.calendarplot(events, monthticks=3, daylabels='MTWTFSS',
                         dayticks=[0, 2, 4, 6], cmap='YlGn',
                         fillcolor='grey', linewidth=0,
                         yearlabel_kws={'fontsize': 20},
                         fig_kws=dict(figsize=(28, 8)))
    plt.show()
```

[ ]:
```
np.save('./temp/array_data_cropped.npy', array_data_cropped)

with open('./temp/array_times.pickle', 'wb') as handle:
    pickle.dump(array_times, handle, protocol=pickle.HIGHEST_PROTOCOL)
```