Snow cover classifier using satellite images

Project work on Data Mining M
Computer Engineering Master Program
University of Bologna
Prof. Claudio Sartori
claudio.sartori@unibo.it

Student
Davide Tazzioli
davide.tazzioli@studio.unibo.it

Abstract

This paper reports the work and the consequent consideration on a satellite images classifier, specified on the recognition and the monitoring of land snow cove. Satellite images are from Sentinel 2 Satellite by ESA, made available on the Sentinel Hub Platform. For the images storage and pre-processing the library eo-learn is used. The classifier is trained and tested with different classifier, trying to obtain the best result in terms of efficiency, scalability, and precision and accuracy of the results. Land snow cover of an area and snow cover variation are important indicator for different environment aspect, used for useful consideration and prevision. With the work reported in this paper, a new algorithm is tested using images from the new European space program for the land monitoring. The resulting classification is finally visualized on the map of the area.

Keywords: Classification, Image Classification, Land Cover Classification, Visualization

1 Content

New technologies on airborne ground monitoring are offering lot of new possibilities, even thanks to the development and cost reduction of satellite technologies. A large-scale observation of the earth's coverage with more and more detail of the elements and specific areas is offered, and data are made almost immediately available thanks to the new communication technologies. The potential of data collection tools, however, must be supported by equally efficient information processing, capable of managing large amounts of data, knowing how to extrapolate relevant information and then delivering it in a timely manner for proliferating use. Furthermore, a large variety of information from different sensors allows us to obtain higher-level information that takes into account not only the spatial location, but also the temporal evolution. In this project we intend to apply machine learning algorithms for the automatic classification of the soil, specific for the detection of snow-covered areas. That information, stored over long periods of time, can be used for statistical consideration or represent the data for a higher elaboration, which can lead to significant insight: the combination of precipitation, temperature and snow coverage data, for instance, could forecast abnormal events on the flow of rivers; information for the correct management of drinking water reserves, starts from the information about the snow trend during the winter; also agriculture can benefit of this information (*High Resolution Snow and Ice Monitoring*, n.d.).

1.1 Sentinel-Hub Project

Sentinel-2 is a space mission by ESA; it is part of the Copernicus program, which intends to monitor the green areas of the planet and provide support in the management of natural disasters. It consists of two identical satellites, Sentinel-2A and Sentinel-2B (Wikipedia Contributors, 2022) It offers satellite images of the entire earth's surface in the areas between the 84S and 84N meridians; their multi-spectrum sensors offer information from visible light to infrared. The resolution of the data allows a ground detail of 10, 20 and 60 meters depending on the spectrum band. The data is freely accessible from the portal offered by the European space agency.

1.2 EO-LEARN Project

eo-learn is a open-source project developed to seamlessly access and process spatio-temporal image sequences acquired by any satellite. Packages are written in Python and available on github: sharing and collaboration is encouraged thank to the web site with instruction and examples of projects. Lot of tasks and utilities are available and tested, and continuously improved by the community: cloud masking, image co-registration, feature extraction, classification, etc. Eo-learn library acts as a bridge between Earth observation/Remote sensing field and Python ecosystem for data science, machine learning and visualization; it uses NumPy arrays to store and handle remote sensing data (Eo-learn Contributors, 2018).

1.3 Machine Learning for soil classification

Research on the observation of the earth's soil with surveying instruments installed on satellites, began in the second half of the 9th century as a part of space exploration by America and Russia. (Lubej, 2018) Nowadays, sensors are specific and accurate, and the work on the data collector software, allows us to have values that are mostly not influenced by the atmosphere variations. In particular, Sentinel 2 satellite acquires data in a period of observation between 17 and 32 minutes, creating a single interpolated map of the area. Bands of the Multispectral sensor are referred to the visible light and to the infrared (*MultiSpectral Instrument (MSI) Overview*, n.d.):

Table 1: Band specifics captured by multyspectral instruments on *Sentinel II* Satellites

Sentinel-2 bands	**usage**	**Central wavelength**	**Bandwidth**	**Spatial resolution**
`Band 01`	*Coastal aerosol*	442.2 *nm*	21 *nm*	60 *m*
`Band 02`	*Blue*	492.1 *nm*	66 *nm*	10 *m*
`Band 03`	*Green*	559.0 *nm*	36 *nm*	10 *m*
`Band 04`	*Red*	664.9 *nm*	31 *nm*	10 *m*
`Band 05`	*Vegetation red edge*	703.8 *nm*	16 *nm*	20 *m*
`Band 06`	*Vegetation red edge*	739.1 *nm*	15 *nm*	20 *m*
`Band 07`	*Vegetation red edge*	779.7 *nm*	20 *nm*	20 *m*
`Band 08`	*NIR*	832.9 *nm*	106 *nm*	10 *m*
`Band 08A`	*Narrow NIR*	864.0 *nm*	22 *nm*	20 *m*
`Band 09`	*Water vapour*	943.2 *nm*	21 *nm*	60 *m*
`Band 10`	*SWIR – Cirrus*	1376.9 *nm*	30 *nm*	60 *m*
`Band 11`	*SWIR*	1610.4 *nm*	94 *nm*	20 *m*
`Band 12`	*SWIR*	2185.7 *nm*	185 *nm*	20 *m*

Over the year some parameters have been studied using earth knowledge, observation on the ground and machine learning. Among the most used parameters:

- **NDVI** *(Normalized difference vegetation index)*: Live green plants absorb solar radiation during the photosynthesis process; at the same time leaf cells re-emit solar radiation in the near-infrared spectral region. Analyzing the re-emitted infrared and near-infrared spectral region you can conclude information about the presence of vegetation in each area. The index is calculated normalizing the difference between the spectral reflectance measurements acquired in the red (visible) and near-infrared regions.
- **NDWI** *(Normalized Difference Water Index)*: it is derived from the *Near-Infrared* (NIR) and *Short-Wave Infrared* (SWIR) channels. The SWIR reflectance reflects changes in both the vegetation water content and the spongy mesophyll structure in vegetation canopies, while the NIR reflectance is affected by leaf internal structure and leaf dry matter content but not by water content. The combination of the NIR with the SWIR removes variations induced by leaf internal structure and leaf dry matter content, improving the accuracy in retrieving the vegetation water content.
- **NDBI** *(Normalized Difference Built-Up Index)*: This index highlights urban areas where there is typically a higher reflectance in the *shortwave-infrared* (SWIR) region, compared to the *near-infrared* (NIR) region.
- **NDSI** *(Normalized-Difference Snow Index)*: NDSI is a measure of the relative magnitude of the reflectance difference between *visible* (green) and *shortwave infrared* (SWIR).

The last index is often used to classify snow coverage from satellite images, just applying different thresholds. *NASA portal for earth observation* (*Earth observatory*, n.d.) offers maps of the whole globe on different aspects registered by the sensors on their satellites. The snow monitoring portal, indeed, offers maps of the only NDSI pure index, and maps of a simple classification based on this index. The threshold value between *snow* and *not-snow* depends by the algorithm. The ground-truth values used in this project to train the classification model and to test it, are made starting from NDSI index obtained by NASA satellite, then manually merged and validated with values obtained from ground station widespread in northern Italy mountains. Snow coverage automatic classification of the soil is a field already studied also using satellite images; high level information are usually

obtained by the manually work of earth scientists also on the data from NASA and ESA. Automatic classification can be useful to obtain data on the whole globe and have fast statistic information. Higher level usage of those information still requires expert intervention. The project aim is not only the evaluation of the results, but also having consideration on the machine learning algorithm: for this reason, NDSI index will not be used for the model training, but only the bands used to calculate it.

2 Land Snow Cover Classification from Satellite Images

2.1 Tools and Environment

The project is developed using the programming language Python, compiled by the software Python 3.10; code is organised in different parts, and result are obtained working with the Jupiter Notebook Environment. Data are elaborated by EO-Learn data structure, that are based on Nunmpy Arrays, while the machine learining tools are made available by SciKit-Learn libraries. Data Visualization, finally, uses MatPlotLib functions and the library Bokeh for graphs and interaction.
Libraries needed to the classification software are imported at the beginning of the code:

- *Built-in methods*: `os, datetime, itertools, MultiValueEnum (from aenum)`
- *Python basic for data handle and visualization*: `numpy, geopandas, pyplot (from matplotlib), ListedColormap and boundaryNorm (from matplotlib.color) Polygon (from shapely.geometry)`
- *Machine learning tools*: `datasets (from sklearn), train_test_split and GridSearchCV (from sklearn.model_sel classification_report (from sklearn.metrics) SVC, Perceptron, MLPClassifier, DecisionTreeClassifier, KNeighborsClassifier (from sklearn. ..)`
- *Tools for raster image management*: `fiona, rasterio, pyproj, re, gdal (from osgeo)`
- *eo_learn libraries*: `EOTask, EOPatch, EOWorkflow, linearly_connect_tasks, FeatureType, OverwritePermissio LoadTask, SaveTask, EOExecutor, MergeFeatureTask, SentinelHubInputTask, VectorImportTask, ExportToTiffTask VectorToRasterTask, ErosionTask, LinearInterpolationTask, SimpleFilterTask, NormalizedDifferenceIndexTask, BlockSamplingTask`
- *Sentinel Hub libraries*: `UtmZoneSplitter, DataCollection, UtmGridSplitter, OsmSplitter`

2.2 Obtain Satellite data

Satellite images are available on the Sentinel-Hub portal after registration; data are automatically collect by a client previusly configurated. The area of interest of this project is located in the north of Italy on the Appennini's mountain, with a focus on the Bologna and Modena's district: a restricted area of interest meets the limitation imposed by Sentinel Hub portal on the download of data.

2.2.1 Definition of the area of interest

Data on the Italian district boundaries are available freely on the portal *openpolis* (*geoJson-italy*, 2022). To verify data and contextualize the work, general information are elaborated and visualizated.

```
DATA_FOLDER = os.path.join(".", "example_data")

# Load geojson file
province = gpd.read_file(os.path.join(DATA_FOLDER, "province.geojson"))
interest_area = province[(province['prov_name']=='Modena') |
    (province['prov_name']=='Bologna')]
interest_area = interest_area.dissolve()

# Plot the area of interest
interest_area.plot();

# Get the area's shape in polygon format and print the size
interest_area_shape = interest_area.geometry.values[0]
interest_area_width = (interest_area_shape.bounds[2] - interest_area_shape.bounds[0])*100
interest_area_height = (interest_area_shape.bounds[3] - interest_area_shape.bounds[1])*100

print(f"Dimension of the area is {interest_area_width:.2f} x {interest_area_height:.2f}
    km2")
```

```
Dimension of the area is 137.03 x 90.07 km2
```

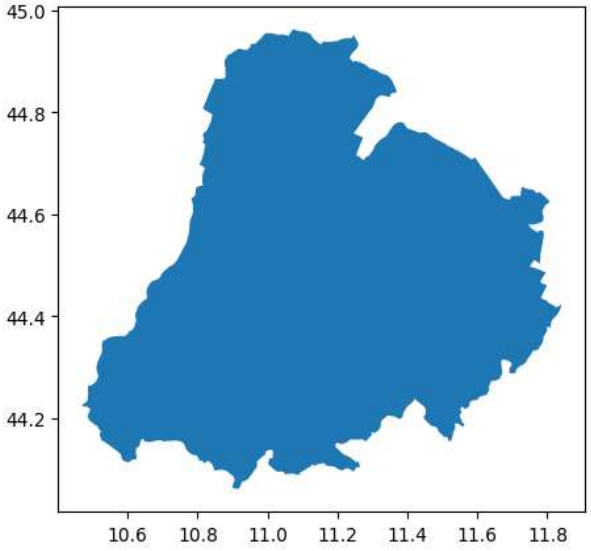

Figure 1: Bologna and Modena Provinces

The identified area is mostly interested by plain land: a further resizing will avoid the download of data referred to areas where snow is uncommon. A Buffer of 500 m is added to the political boundaries.

```
clip_shape = Polygon([(-57.35962388414055, -0.5221310661900134), (-57.35962388414055,
    44.7), (79.6701812734866, 44.7), (79.6701812734866, -0.5221310661900134)])
interest_area = interest_area.geometry.clip(clip_shape)

# Add a buffer of 500m to the final selected area and plot it
interest_area.buffer(500);
interest_area.plot()

# Get the area's shape in polygon format and print the size
interest_area_shape = interest_area.values[0]
interest_area_bounds = interest_area_shape.bounds
interest_area_width = (interest_area_bounds[2] - interest_area_bounds[0])*100
interest_area_height = (interest_area_bounds[3] - interest_area_bounds[1])*100

print(f"Dimension of the area after splitting to a smaller place is
    {interest_area_width:.2f} x {interest_area_height:.2f} km2")
```

```
Dimension of the area after splitting to a smaller place is 137.03 x 63.77 km2
```

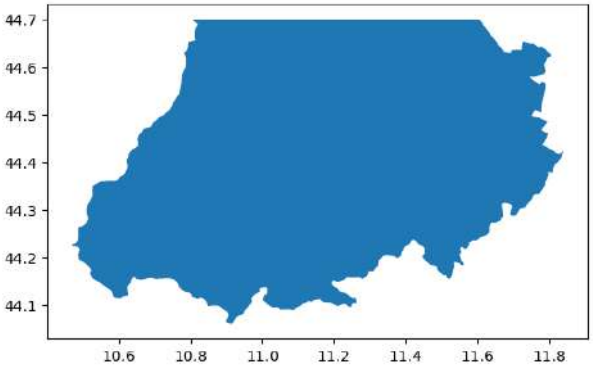

Figure 2: Bologna and Modena Provinces reduced with a focus on the Appennini's Mountains

A eo-learn framework allows to split the Area Of Interest *(AOI)* into smaller patches, so they can be processed with limited computational resources. Boundaries map is then split in smaller and regular boxes thanks to

Geopandas and *shapely* tools: as indicated on eo-learn documentation, the splitting choice depends on the amount of available resources, so the pipeline can be executed on a high-end scientific machine, as well as on a laptop. The output of this step is a list of bounding boxes covering the AOI.

```
# Create a splitter to obtain a list of bboxes with 5km sides
bbox_splitter = OsmSplitter([interest_area_shape], CRS.WGS84, zoom_level=11)
bbox_list = np.array(bbox_splitter.get_bbox_list())
info_list = np.array(bbox_splitter.get_info_list())

# Prepare info of selected EOPatches
geometry = [Polygon(bbox.get_polygon()) for bbox in bbox_list]
idxs = [i for i, p in enumerate(bbox_list)]
idxs_x = [info["index_x"] for info in info_list]
idxs_y = [info["index_y"] for info in info_list]

bbox_gdf = gpd.GeoDataFrame({"index": idxs, "index_x": idxs_x, "index_y": idxs_y},
    crs=interest_area.crs, geometry=geometry)

# Display bboxes over country
fig, ax = plt.subplots(figsize=(30, 30))
ax.set_title("Interest area split", fontsize=25)
interest_area.plot(ax=ax, facecolor="w", edgecolor="b", alpha=0.5)
bbox_gdf.plot(ax=ax, facecolor="w", edgecolor="r", alpha=0.5, aspect=1)

for index, bbox in zip(idxs, bbox_list):
    geo = bbox.geometry
    ax.text(geo.centroid.x, geo.centroid.y, index, ha="center", va="center")
```

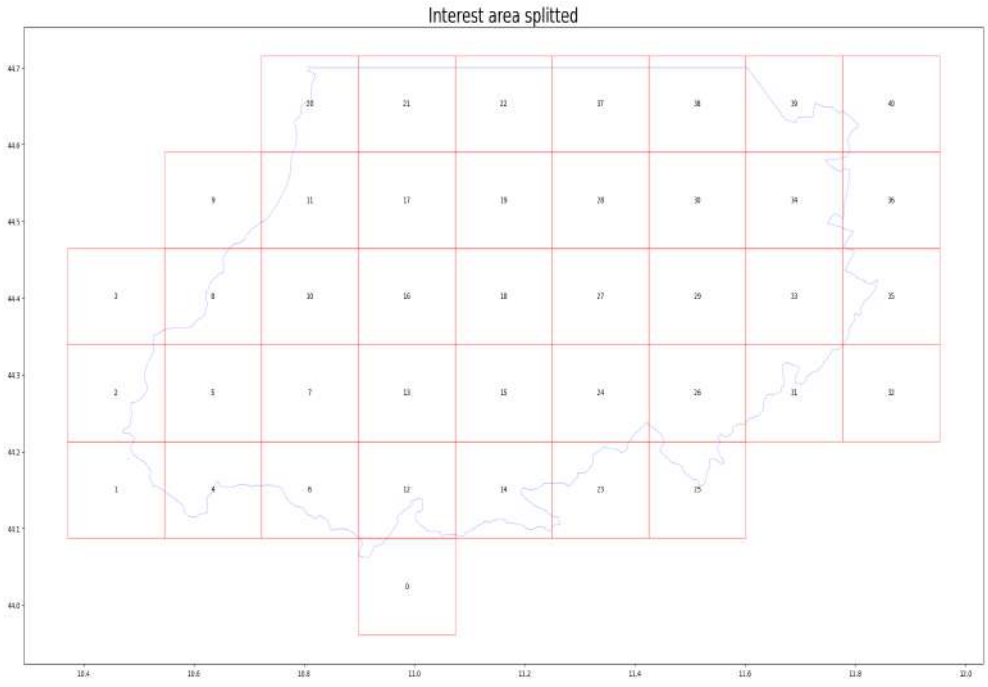

Figure 3: Work area split in boxes

For a better visualization, boxes outside a regular shape are ignored: the focus is manually setted on a rectangular area centered on the previously defined one.

```
# Manually select a rectangular area where focus the attention.
focus_list=[9, 11, 17, 19, 28, 30, 8, 10, 16, 18, 27, 29, 5, 7, 13, 15, 24, 26, 4, 6, 12,
    14, 23, 25]
```

2.2.2 Prepare Sentinel-hub request

With the bounding boxes of the empty patches in place, eo-learn enables the automatic download of Sentinel image data. In addition to data, eo-learn makes it possible to seamlessly access cloud masks and cloud probabilities: the Python package *s2cloudless* provides automated cloud detection in Sentinel-2 L1C imagery, based on a single-scene pixel-based classification. (*Sentinel2 cloud-detector*, 2022)

```
# Define the classes who build a functional mask for valid and invalid pixels on the
    satellite images
# @author https://github.com/sentinel-hub/eo-learn-examples/tree/main/land-cover-map

class SentinelHubValidDataTask(EOTask):
    """
    Combine Sen2Cor's classification map with 'IS_DATA' to define a 'VALID_DATA_SH' mask
    The SentinelHub's cloud mask is asumed to be found in eopatch.mask['CLM']
    """

    def __init__(self, output_feature):
        self.output_feature = output_feature

    def execute(self, eopatch):
        eopatch[self.output_feature] = eopatch.mask["IS_DATA"].astype(bool) &
            (~eopatch.mask["CLM"].astype(bool))
        return eopatch


class AddValidCountTask(EOTask):
    """
    The task counts number of valid observations in time-series and stores the results in
        the timeless mask.
    """

    def __init__(self, count_what, feature_name):
        self.what = count_what
        self.name = feature_name

    def execute(self, eopatch):
        eopatch[FeatureType.MASK_TIMELESS, self.name] =
            np.count_nonzero(eopatch.mask[self.what], axis=0)
        return eopatch
```

The request for S2 bands *B02, B03, B04, B06, B08, B11* and *B12*. This Bands allows us to calculate new useful features *NDVI* and *NDWI*. *NDSI* is calculated but not used in the project. The time period chosen is the *winter 2019-2020*, from the First of November to the First of May.

```
# Add a request for S2 bands adding the pre-built filter of cloudy scenes;
# The s2cloudless masks and probabilities are requested via additional data.
# @author https://github.com/sentinel-hub/eo-learn-examples/tree/main/land-cover-map
band_names = ["B02", "B03", "B04", "B06", "B08", "B11", "B12"]
add_data = SentinelHubInputTask(
    bands_feature=(FeatureType.DATA, "BANDS"),
    bands=band_names,
    resolution=50,
    maxcc=0.8,
    time_difference=datetime.timedelta(minutes=120),
    data_collection=DataCollection.SENTINEL2_L1C,
    additional_data=[(FeatureType.MASK, "dataMask", "IS_DATA"), (FeatureType.MASK, "CLM"),
        (FeatureType.DATA, "CLP")],
    max_threads=5,
)

# CALCULATING NEW FEATURES
# NDVI: (B08 - B04)/(B08 + B04)
# NDWI: (B03 - B08)/(B03 + B08)
```

```
# NDSI: (B04 - B06)/(B04 + B06)

ndvi = NormalizedDifferenceIndexTask(
    (FeatureType.DATA, "BANDS"), (FeatureType.DATA, "NDVI"), [band_names.index("B08"),
        band_names.index("B04")]
)
ndwi = NormalizedDifferenceIndexTask(
    (FeatureType.DATA, "BANDS"), (FeatureType.DATA, "NDWI"), [band_names.index("B03"),
        band_names.index("B08")]
)
ndsi = NormalizedDifferenceIndexTask(
    (FeatureType.DATA, "BANDS"), (FeatureType.DATA, "NDSI"), [band_names.index("B04"),
        band_names.index("B06")]
)

# DEFINE EO-LEAND TASKS AND THE WORFLOW:
# VALIDITY MASK: Validate pixels using SentinelHub's cloud detection mask and region of
    acquisition
add_sh_validmask = SentinelHubValidDataTask((FeatureType.MASK, "IS_VALID"))
# COUNTING VALID PIXELS: Count the number of valid observations per pixel using valid data
    mask
add_valid_count = AddValidCountTask("IS_VALID", "VALID_COUNT")
# SAVING TO OUTPUT (if needed)
save = SaveTask(EOPATCH_FOLDER, overwrite_permission=OverwritePermission.OVERWRITE_PATCH)

# WORKFLOW
workflow_nodes = linearly_connect_tasks(
    add_data, ndvi, ndwi, ndbi, ndsi, add_sh_validmask, add_valid_count, save
)
workflow = EOWorkflow(workflow_nodes)

# EXECUTE EO-LEARN WORKFLOW
%%time
time_interval = ["2019-11-01", "2020-05-31"]
input_node = workflow_nodes[0]
save_node = workflow_nodes[-1]
execution_args = []
for idx, bbox in enumerate(bbox_list):
    execution_args.append(
        {
            input_node: {"bbox": bbox, "time_interval": time_interval},
            save_node: {"eopatch_folder": f"eopatch_{idx}"},
        }
    )
executor = EOExecutor(workflow, execution_args, save_logs=True)
executor.run()

executor.make_report()

failed_ids = executor.get_failed_executions()
if failed_ids:
    raise RuntimeError(
        f"Execution failed EOPatches with IDs:\n{failed_ids}\n"
        f"For more info check report at {executor.get_report_path()}"
    )
```

2.2.3 Visualize the data

Data are visualized to contestualize the work. Firt of all the *RGB* image where the AOI land is shown.

```
fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(20, 15))

date = datetime.datetime(2020, 2, 14)
```

```
for i, patchID in enumerate(focus_list):
    eopatch_path = os.path.join(EOPATCH_FOLDER, f"eopatch_{patchID}")
    eopatch = EOPatch.load(eopatch_path, lazy_loading=True)

    dates = np.array([timestamp.replace(tzinfo=None) for timestamp in eopatch.timestamp])
    closest_date_id = np.argsort(abs(date - dates))[0]

    ax = axs[i // 6][i % 6]
    ax.imshow(np.clip(eopatch.data["BANDS"][closest_date_id][..., [2, 1, 0]] * 3.5, 0, 1))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect("auto")
    del eopatch

fig.subplots_adjust(wspace=0, hspace=0)
```

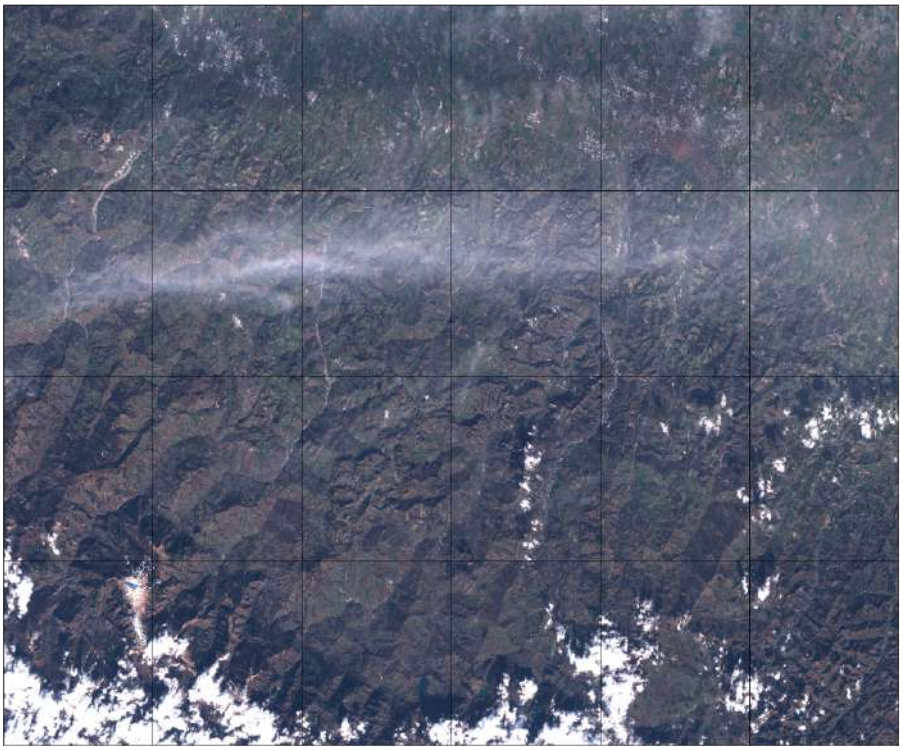

Figure 4: RGB image of the Area of Intrest

Automatic cloud cleaning is applied as a eo-learn task: pixels considered as a cloud are labeled as *INVALID*. To appreciate effects of cloud cleaning, let's analyze the mean of the *NDVI* index: it represents the mean of the amount of vegetation of the area; we expect a linear trend over the time period, just lower during the winter.

```
eID = 16
eopatch = EOPatch.load(os.path.join(EOPATCH_FOLDER, f"eopatch_{i}"), lazy_loading=True)

ndvi = eopatch.data["NDVI"]
mask = eopatch.mask["IS_VALID"]
time = np.array(eopatch.timestamp)
t, w, h, _ = ndvi.shape

ndvi_clean = ndvi.copy()
ndvi_clean[~mask] = np.nan # Set values of invalid pixels to NaN's

# Calculate means, remove NaN's from means
ndvi_mean = np.nanmean(ndvi.reshape(t, w * h), axis=1)
```

```
ndvi_mean_clean = np.nanmean(ndvi_clean.reshape(t, w * h), axis=1)
time_clean = time[~np.isnan(ndvi_mean_clean)]
ndvi_mean_clean = ndvi_mean_clean[~np.isnan(ndvi_mean_clean)]

fig = plt.figure(figsize=(20, 5))
plt.plot(time_clean, ndvi_mean_clean, "s-", label="Mean NDVI with cloud cleaning")
plt.plot(time, ndvi_mean, "o-", label="Mean NDVI without cloud cleaning")
plt.xlabel("Time", fontsize=15)
plt.ylabel("Mean NDVI over patch", fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.legend(loc=2, prop={"size": 15});
```

```
ndvi_mean_clean = np.nanmean(ndvi_clean.reshape(t, w * h), axis=1)
```

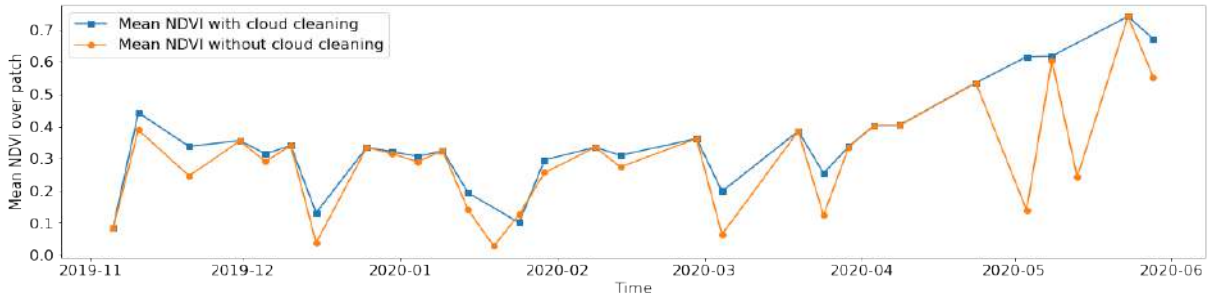

Figure 5: NDVI mean value over boxes applied to original images and to cloud-cleaned images

Without cloud cleaning, values are highly variable from month to month, while this trend is a little corrected with the cloud cleaning.

It is relevant and interesting to visualize the NDVI map for the vegetation and the NDSI map for the snow:

```
fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(20, 15))

for i, patchID in enumerate(focus_list):
    eopatch_path = os.path.join(EOPATCH_FOLDER, f"eopatch_{patchID}")
    eopatch = EOPatch.load(eopatch_path, lazy_loading=True)
    ndvi = eopatch.data["NDVI"]         # NDSI
    mask = eopatch.mask["IS_VALID"]
    ndvi[~mask] = np.nan
    ndvi_mean = np.nanmean(ndvi, axis=0).squeeze()

    ax = axs[i // 6][i % 6]
    im = ax.imshow(ndvi_mean, vmin=0, vmax=0.8, cmap=plt.get_cmap("YlGn"))
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect("auto")
    del eopatch

fig.subplots_adjust(wspace=0, hspace=0)

cb = fig.colorbar(im, ax=axs.ravel().tolist(), orientation="horizontal", pad=0.01,
     aspect=100)
cb.ax.tick_params(labelsize=20)
plt.show()
```

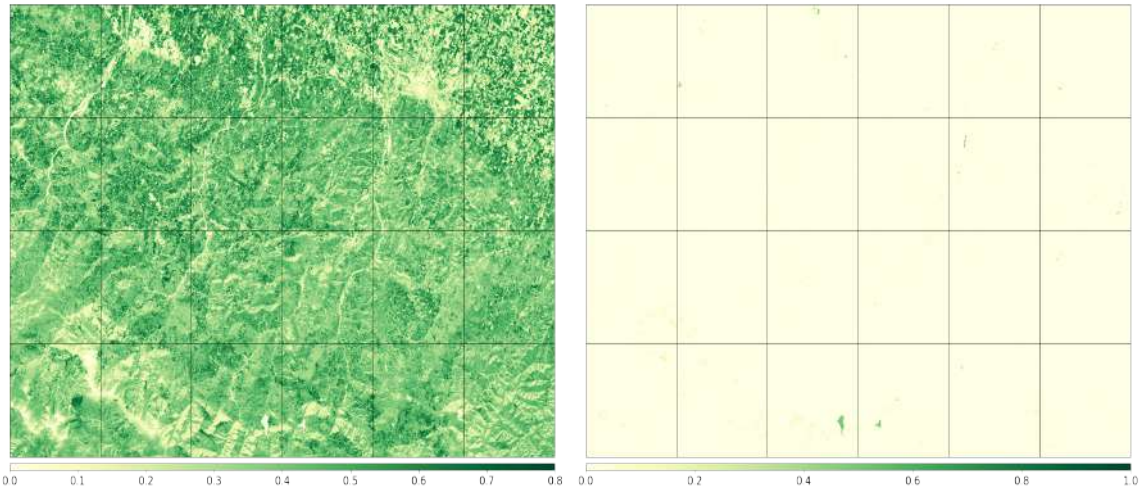

Figure 6: NDVI (left) and NDSI (right) maps

Finally the cloud probability map, calculated using values defined as cloud by the *s2cloudless* software.

```
# cloud probability
fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(20, 15))

for i, patchID in enumerate(focus_list):
    eopatch_path = os.path.join(EOPATCH_FOLDER, f"eopatch_{patchID}")
    eopatch = EOPatch.load(eopatch_path, lazy_loading=True)
    clp = eopatch.data["CLP"].astype(float) / 255
    mask = eopatch.mask["IS_VALID"]
    clp[~mask] = np.nan
    clp_mean = np.nanmean(clp, axis=0).squeeze()

    ax = axs[i // 6][i % 6]
    im = ax.imshow(clp_mean, vmin=0.0, vmax=0.3, cmap=plt.cm.inferno)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect("auto")
    del eopatch

fig.subplots_adjust(wspace=0, hspace=0)

cb = fig.colorbar(im, ax=axs.ravel().tolist(), orientation="horizontal", pad=0.01,
     aspect=100)
cb.ax.tick_params(labelsize=20)
plt.show()
```

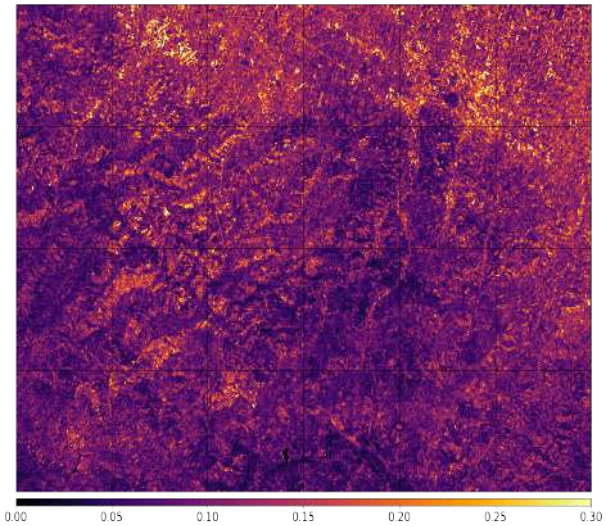

Figure 7: Cloud probability map

2.3 Obtain ground truth Data

Ground truth data are obtained by the validation work of NASA maps (Magnani, 2022): from NASA satellite data NDSI map is calculated thought a validation work using ground station values, a specific threshold is calculated to distinguish snow areas, iced cloud and areas without snow.
The ground truth maps are referred to the whole Emilia Romagna, and saved as *geotiff* images; the reference system is different from the one used to obtain data from Sentinel Hub portal, and definition is lower.

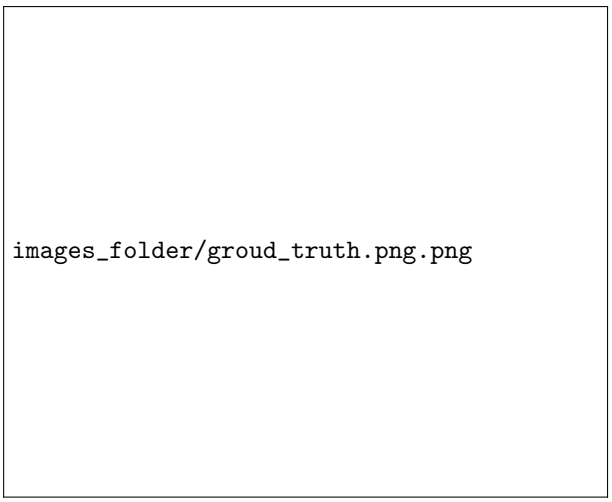

Figure 8: Ground truth map of Emilia Romagna

First of all, we define utility functions to work with the maps: coordinates are translate to the reference system *EPSG:32632 WGS:84*.

```
def translateCoordinate(geometry):
    proj = pyproj.Transformer.from_crs(4326, 32632, always_xy=True)
    shape_points = []
    for xx, yy in geometry.exterior.coords:
        shape_points.append(proj.transform(xx, yy))
    return MultiPolygon([Polygon(shape_points)])
```

Another funcion applies a mask to Geotiff images and defines the new image dimension.

```
NoDataValue = -9999.0
def applyMask(raster_file, shape_mask, size_x, size_y):
```

```
    with rasterio.open(raster_file) as src:
        out_image, out_transform = mask(src, shape_mask, crop=True, nodata=None)
        out_meta = src.meta.copy()

    out_meta.update({"driver": "GTiff",
                     "height": size_y,
                     "width": size_x,
                     "transform": out_transform})

    return out_image, out_meta
```

The funcions are now applied to files: we will obtain a map for every boxes previusly defined, for every day in the time window of interest. To obtain a pixel-to-pixel correspondence between the ground truth map and the eo-learn bands, pixels of real data maps are manually replicated for an area of 8x8 pixels. A filler pad is added when dimensions do not correspond: it will be left out during the training phase.

```
path_file = os.path('ground_truth_data')
path_out = os.path('output')

# File elaboration

for filename in <real_data_file_list>: ##

    output_path = os.path.join(path_out, str("real_data_" +
        filename.rsplit(".",6)[1].rsplit("A", 1)[1]))
    os.makedirs(output_path, exist_ok=True)

    for box in focus_list:
        shape_mask = translateCoordinate(bbox_gdf[bbox_gdf['index']==box].geometry.item()) #
            translate real data coordinates
        sample_path = os.path.join('eopatches_sampled', f"eopatch_{box}")
        bbox_meta = EOPatch.load(sample_path, lazy_loading=True).meta_info              #
            load box coordinates

        raster_img, raster_img_meta = applyMask(filename, shape_mask.geoms,
            bbox_meta['size_x'], bbox_meta['size_y'])

        pad_width = int((bbox_meta['size_x'] - (raster_img.shape[1]*8)-1)/2)            #
            padding
        pad_height = int((bbox_meta['size_y'] - (raster_img.shape[2]*8)-1)/2)

        raster_out = np.full((1, bbox_meta['size_x'], bbox_meta['size_y']), -1)         # -1
            = NO DATA

        for index, value in np.ndenumerate(raster_img):
            if value == 1:           # snow
                for c in range(0, 8):
                    for r in range (0, 8):
                        raster_out[(index[0], pad_width+(index[1]*8)+c,
                            pad_height+(index[2]*8)+r)]=1

            elif vale == -1:      # NO_DATA (CLOUD)
                for c in range(0, 8):
                    for r in range (0, 8):
                        raster_out[(index[0], pad_width+(index[1]*8)+c,
                            pad_height+(index[2]*8)+r)]=-1 # NO DATA
                                                    # we dont't use this classification for
                                                        cloud
            elif value == 0:      # LAND
                for c in range(0, 8):
                    for r in range (0, 8):
                        raster_out[(index[0], pad_width+(index[1]*8)+c,
                            pad_height+(index[2]*8)+r)]=0 #land
```

```
            with rasterio.open(os.path.join(output_path, str('bbox_'+str(box)+'.tiff')), "w",
                **raster_img_meta) as dest:
                dest.write(raster_out)

    print("Done")
```

The new map of real values is added to the eo-learn data structures.

```
    # Add the real data to the eo_patches
    # -1 : NO DATA
    # 0 : LAND
    # 1 : SNOW
    # 2 : ICE CLOUD

    for box in focus_list:
        sample_path = os.path.join(EOPATCH_FOLDER, f"eopatch_{box}")
        eopatch = EOPatch.load(sample_path, lazy_loading=True)
        eopatch.data['LABEL'] = np.full((len(eopatch.timestamp), eopatch.meta_info['size_y'],
            eopatch.meta_info['size_x'], 1), 0)
        eopatch.mask_timeless['REAL_DATA_SNOW_COUNT'] = np.full((len(eopatch.timestamp), 1, 1),
            0)
        for day_index, day in enumerate(eopatch.timestamp):
            day_of_year = day.timetuple().tm_yday

            if day_of_year>106:
                continue

            year = day.year
            raster_folder = str('real_data_'+str(year)+str(day_of_year).zfill(3))
            with rasterio.open(os.path.join('output', os.path.join(raster_folder,
                str('bbox_'+str(box)+'.tiff'))), 'r') as src:
                labels = src.read()
            eopatch.data['LABEL'][day_index] = np.transpose(labels, (1, 2, 0))
            eopatch.mask_timeless['REAL_DATA_SNOW_COUNT'][day_index] =
                [np.count_nonzero(labels==1)]

        eopatch.save(os.path.join(NEW_EOPATCH_FOLDER, str('eopatch_'+str(box))),
            overwrite_permission=OverwritePermission.OVERWRITE_PATCH)
```

Groud truth data maps contain a cloud classification; we will then compare the classification with the one obtained with the eo-learn instruments. A difference is expected because the low definition of the gound truth data maps. We can keep only the oe-learn classification because is more defined.

```
for box in focus_list:
    sample_path = os.path.join(NEW_EOPATCH_FOLDER, f"eopatch_{box}")
    eopatch = EOPatch.load(sample_path, lazy_loading=True)
    #eopatch.data['LABEL']
    count = 0
    for index, value in np.ndenumerate(eopatch.mask["CLM"]):
        if value == 1:
            eopatch.data['LABEL'][index] = 2
            #eopatch.data['LABEL'][index] = 2
            count += 1
    eopatch.save(os.path.join(NEW_EOPATCH_FOLDER, str('eopatch_'+str(box))),
        overwrite_permission=OverwritePermission.OVERWRITE_PATCH)
print("cloud pixels: ", count)
```

```
    # LOAD EXISTING EOPATCHES
    load = LoadTask(NEW_EOPATCH_FOLDER)

    # FEATURE CONCATENATION
    concatenate = MergeFeatureTask({FeatureType.DATA: [ 'BANDS', 'NDBI', 'NDVI', 'NDWI']},
```

```
        (FeatureType.DATA, "FEATURES"))

    # SAVING
    save = SaveTask(EOPATCH_SAMPLES_FOLDER,
        overwrite_permission=OverwritePermission.OVERWRITE_PATCH)

    # Define the workflow
    workflow_nodes = linearly_connect_tasks(load, concatenate, save)
    workflow = EOWorkflow(workflow_nodes)

    %%time # EXECUTION

            [ ... ]
```

Sampling data can help training algoritm to be more efficient; in the work contest of this project, particularly, most of data are cloud or land, with just a little percent of snow, centered in the Appennini's mountain. A first data cleaning consists of deleteing all the parts of maps in all the time spots where there is no snow: the effect of this operation is to obtain a smaller dataset, where snow, cloud and land pixels are more balanced.

```
count_land = 0
count_snow = 0
count_cloud = 0
for box in focus_list:
    sample_path = os.path.join(EOPATCH_SAMPLES_FOLDER, f"eopatch_{box}")
    eopatch = EOPatch.load(sample_path, lazy_loading=True)
    count_land += np.count_nonzero(eopatch.data['LABEL']==0)
    count_snow += np.count_nonzero(eopatch.data['LABEL']==1)
    count_cloud += np.count_nonzero(eopatch.data['LABEL']==2)

valid_pixels = count_snow+count_cloud+count_land
print("Snow Pixel are the {:.2f}% of the total of valid
    pixels".format(count_snow*100/valid_pixels))
print("Land Pixel are the {:.2f}% of the total of valid
    pixels".format(count_land*100/valid_pixels))
print("Cloud Pixel are the {:.2f}% of the total of valid
    pixels".format(count_cloud*100/valid_pixels))
```

```
    Snow Pixel are the 0.14% of the total of valid pixels
    Land Pixel are the 69.59% of the total of valid pixels
    Cloud Pixel are the 30.27% of the total of valid pixels
```

```
    # Keep only days with at least 500 snow pixels
    for box in focus_list:
        sample_path = os.path.join(EOPATCH_SAMPLES_FOLDER, f"eopatch_{box}")
        eopatch = EOPatch.load(sample_path, lazy_loading=True)

        for snow_pixels_index in reversed(range(0,
            len(eopatch.mask_timeless['REAL_DATA_SNOW_COUNT']))):
            if eopatch.mask_timeless['REAL_DATA_SNOW_COUNT'][snow_pixels_index][0][0] < 500:
                eopatch.data['FEATURES'] = np.delete(eopatch.data['FEATURES'],
                    snow_pixels_index, 0)
                eopatch.data['LABEL'] = np.delete(eopatch.data['LABEL'], snow_pixels_index, 0)
        eopatch.save(os.path.join(EOPATCH_TRAINING_FOLDER, str('eopatch_'+str(box))),
            overwrite_permission=OverwritePermission.OVERWRITE_PATCH)
```

```
    count_land = 0
    count_snow = 0
    count_cloud = 0
    for box in focus_list:
        sample_path = os.path.join(EOPATCH_TRAINING_FOLDER, f"eopatch_{box}")
        eopatch = EOPatch.load(sample_path, lazy_loading=True)
```

```
    count_land += np.count_nonzero(eopatch.data['LABEL']==0)
    count_snow += np.count_nonzero(eopatch.data['LABEL']==1)
    count_cloud += np.count_nonzero(eopatch.data['LABEL']==2)

valid_pixels = count_snow+count_cloud+count_land
print("After optimization, snow pixel are the {:.2f}% of the total of valid
    pixels".format(count_snow*100/valid_pixels))
print("After optimization, land pixel are the {:.2f}% of the total of valid
    pixels".format(count_land*100/valid_pixels))
print("After optimization, cloud pixel are the {:.2f}% of the total of valid
    pixels".format(count_cloud*100/valid_pixels))
```

```
After optimization, snow pixel are the 9.94% of the total of valid pixels
After optimization, land pixel are the 73.83% of the total of valid pixels
After optimization, cloud pixel are the 16.23% of the total of valid pixels
```

A further sampling with no criteria on the selected images.

```
# LOAD EXISTING EOPATCHES
load_2 = LoadTask(EOPATCH_TRAINING_FOLDER)

# SPATIAL SAMPLIG OF 60% OF DATA
spatial_sampling = BlockSamplingTask([(FeatureType.DATA, "FEATURES", "FEATURES_SAMPLED"),
    (FeatureType.DATA, "LABEL", "LABEL_SAMPLED")], 0.6)

#SAVING
save_2 = SaveTask(EOPATCH_TRAINING_FOLDER,
    overwrite_permission=OverwritePermission.OVERWRITE_PATCH)

# define a new workflow
workflow_nodes_2 = linearly_connect_tasks(load_2, spatial_sampling, save_2)
workflow_2 = EOWorkflow(workflow_nodes_2)

%%time     # EXECUTION
       [ ... ]
```

2.4 Training the classification Algoritm

2.4.1 Prepare traning and test dataset

Data previusly prepared are now splittend in two dataset, for train and test the model.

```
# Load sampled eopatches
sampled_eopatches = []

for i in focus_list:
    sample_path = os.path.join(EOPATCH_TRAINING_FOLDER, f"eopatch_{i}")
    sampled_eopatches.append(EOPatch.load(sample_path, lazy_loading=True))

    # Set the features and the labels for train and test sets
    features = []
    labels = []
    num_data = 0
    for eopatch in sampled_eopatches:
        for timestamp in range(0, len(eopatch.data['FEATURES_SAMPLED'])):
            num_data+=1
            features.append(eopatch.data["FEATURES_SAMPLED"][timestamp])
            labels.append(eopatch.data['LABEL_SAMPLED'][timestamp])

    X = np.concatenate(features)
    y = np.concatenate(labels)

print(X.shape)
```

```
print(y.shape)
```

```
(992866, 1, 10)
(992866, 1, 1)
```

```
X = X.reshape(X.shape[0], X.shape[-1])
y= y.reshape(y.shape[0], y.shape[-1])
print('Shape of features dataset after reshaping: \t' + str(X.shape))
print('Shape of labels dataset after reshaping: \t' + str(y.shape))
```

```
Shape of features dataset after reshaping:  (992866, 10)
Shape of labels dataset after reshaping:  (992866, 1)
```

Any of the satellite images from Sentinel 2 have missing data; labels contain a filling pad, previously added, containing the value *-1*: data have to be cleaned from invalid entries before the training phase.

```
rows_with_NaN = np.shape(X[np.isnan(X).any(axis=1), :])[0]
labels_NaN = np.shape(y[y==-1])[0]

print("There are {:.0f} rows containing NaN value in the features
    dataset".format(rows_with_NaN))
print("There are {:.0f} rows containing -1 in labels in the training
    dataset".format(labels_NaN))
```

```
There are 274186 rows containing NaN value in the features dataset
There are 177957 rows containing -1 in labels in the training dataset
```

```
# Remove rows containing NaN values
ds_complete_ = np.append(X, y, 1)
ds_complete_ = ds_complete_[~np.isnan(ds_complete_).any(axis=1), :]
ds_complete_ = ds_complete_[ds_complete_[:, -1]!=-1]

X = np.delete(ds_complete_, -1, 1)
y = ds_complete_[:, -1]

rows_with_NaN = np.shape(X[np.isnan(X).any(axis=1), :])[0]
labels_NaN = np.shape(y[y==-1])[0]

print("After cleaning, there are {:.0f} rows containing NaN value in the features
    dataset".format(rows_with_NaN))
print("After cleaning, there are {:.0f} rows containing -1 in label of the training
    dataset".format(labels_NaN))
```

```
After cleaning, there are 0 rows containing NaN value in the features dataset
After cleaning, there are 0 rows containing -1 in label of the training dataset
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=1/5.0,random_state=8)
print('The training dataset contains {:.2f}% of total entries.
    '.format(X_train.shape[0]*100/(X_train.shape[0]+X_test.shape[0])))
```

```
The training dataset contains 80.00% of total entries.
```

2.4.2 Train and Test the model

The optimal choice of a classifier and the correct parameters heavily depends on the application and the specific case of interest. In order to optimise these choices, several experiments are now performed, trying to find the best classifier and tune the parameters to obtain the best results. The choice of the best classifier will be done manually, exploring the precision and f1-score: a useful function prints the results and the scikit-learn classification report.

```
model_lbls = [
        'dt',  # Decisio Tree
        'nb',  # Gaussian Naive Bayes
        'lp'   # Linear Perceptron
        'svc'  # Support Vector
        'rndf' # Random Forest
        ]

# Set the parameters to be explored by the grid for each classifier
tuned_param_dt = [{'max_depth': list(range(1, 12))}]
tuned_param_nb = [{'var_smoothing': [10, 1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-07,
    1e-8, 1e-9, 1e-10]}]
tuned_param_lp = [{'early_stopping': [True]}]
tuned_param_svc = [{'kernel': ['rbf'],
        'gamma': ['scale'],
        'C': [1, 10],
        },
        {'kernel': ['linear'],
         'C': [1, 10, 100],
        },
       ]
tuned_param_rndf = [{'max_depth': list(range(1, 25))}]

# set the models to be fitted specifying name, estimator and parameter structure
models = {
    'dt': {'name': 'Decision Tree    ',
        'estimator': DecisionTreeClassifier(),
        'param': tuned_param_dt,
        },
    'nb': {'name': 'Gaussian Naive Bayes',
        'estimator': GaussianNB(),
        'param': tuned_param_nb
        },
    'lp': {'name': 'Linear Perceptron ',
        'estimator': Perceptron(),
        'param': tuned_param_lp,
        },
    'svc':{'name': 'Support Vector',
       'estimator': SVC(),
       'param': tuned_param_svc
        },
    'knn':{'name': 'K Nearest Neighbor ',
       'estimator': KNeighborsClassifier(),
       'param': tuned_param_knn
        },
    'rndf':{'name': 'Random Forest Classifier',
          'estimator': RandomForestClassifier(),
          'param': tuned_param_rndf,
},

}

# scores to be explored
scores = [
        'precision',
        ]
```

```
'''Print a detailed report

Parameters
----------
model:
```

```
        istance of fitted estimator
    file:
        file where to print informations

    Outupt
    ------

'''
'''
        Code modified from Data Mining M 2020-2021 course material
        Professor Claudio Sartori - University of Bologna
        https://www.unibo.it/en/teaching/course-unit-catalogue/course-unit/2020/391683

'''
def print_results(model, file = None):
    print("Best parameters set found on train set:")
    print()
    # if best is linear there is no gamma parameter
    print(model.best_params_)
    print()
    print("Grid scores on train set:")
    print()
    means = model.cv_results_['mean_test_score']
    stds = model.cv_results_['std_test_score']
    params = model.cv_results_['params']
    for mean, std, params_tuple in zip(means, stds, params):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params_tuple))
    print()
    print("Detailed classification report for the best parameter set:")
    print()
    print("The model is trained on the full train set.")
    print("The scores are computed on the full test set.")
    print()
    y_true, y_pred = y_test, model.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
    if file is not None:
        file.write("Best parameters set found on train set:\n")
        file.write('\n')
        file.write(str(model.best_params_)+'\n')
        file.write('\n')
        file.write("Grid scores on train set:\n")
        file.write('\n')
        for mean, std, params_tuple in zip(means, stds, params):
            file.write("%0.3f (+/-%0.03f) for %r"
                  % (mean, std * 2, params_tuple)+'\n')
        file.write('\n')
        file.write("Detailed classification report for the best parameter set:\n")
        file.write('\n')
        file.write("The model is trained on the full train set.\n")
        file.write("The scores are computed on the full test set.\n")
        file.write('\n')
        file.write(classification_report(y_true, y_pred)+'\n')
        file.write('\n')
```

```
    '''
        Code from Data Mining M 2020-2021 course material
        Professor Claudio Sartori - University of Bologna
        https://www.unibo.it/en/teaching/course-unit-catalogue/course-unit/2020/391683

    '''
results_short = {}
```

```
for score in scores:
    print('='*40)
    print("# Tuning hyper-parameters for %s" % score)
    print()

    #'%s_macro' % score ## is a string formatting expression
    # the parameter after % is substituted in the string placeholder %s
    for m in model_lbls:
        print('-'*40)
        print("Trying model {}".format(models[m]['name']))
        start = datetime.datetime.now()
        clf = GridSearchCV(models[m]['estimator'], models[m]['param'], cv=5,
                           scoring='%s_macro' % score,
#                            iid = False,
                           return_train_score = False,
                           n_jobs = 2, # this allows using multi-cores
                           )
        clf.fit(X_train, y_train)
        with open(os.path.join(RESULTS_FOLDER, "classification_results2.txt"), 'w+') as
            classification_results_output:
            classification_results_output.write('='*40+'\n')
            classification_results_output.write("# Tuning hyper-parameters for %s \n" % score)
            classification_results_output.write('\n')
            print_results(clf, classification_results_output)
            results_short[m] = clf.best_score_
            execution_time = datetime.datetime.now() - start
            print()
            print('execution time : '+ str(execution_time))
            print()
            classification_results_output.write('\n')
            classification_results_output.write('execution time : '+ str(execution_time)+'\n')
            classification_results_output.write('\n')

    with open(os.path.join(RESULTS_FOLDER, "classification_results2.txt"), 'a+') as
        classification_results_output:
        print("Summary of results for {}".format(score))
        print("Estimator")
        classification_results_output.write("Summary of results for " + str(score)+'\n')
        classification_results_output.write('Estimator \n')

        for m in results_short.keys():
            print("{}\t - score: {:5.2f}%".format(models[m]['name'], results_short[m]*100))
            classification_results_output.write("{}\t - score:
                {:5.2f}%".format(models[m]['name'], results_short[m]*100)+'\n')
```

```
========================================
# Tuning hyper-parameters for precision

----------------------------------------
Trying model Decision Tree

Best parameters set found on train set:

{'max_depth': 9}

Grid scores on train set:

0.242 (+/-0.000) for {'max_depth': 1}
0.496 (+/-0.003) for {'max_depth': 2}
0.752 (+/-0.003) for {'max_depth': 3}
0.745 (+/-0.009) for {'max_depth': 4}
0.763 (+/-0.017) for {'max_depth': 5}
```

```
0.776 (+/-0.004) for {'max_depth': 6}
0.780 (+/-0.011) for {'max_depth': 7}
0.783 (+/-0.005) for {'max_depth': 8}
0.790 (+/-0.008) for {'max_depth': 9}
0.789 (+/-0.007) for {'max_depth': 10}
0.788 (+/-0.006) for {'max_depth': 11}
0.784 (+/-0.005) for {'max_depth': 12}
0.779 (+/-0.005) for {'max_depth': 13}
0.773 (+/-0.004) for {'max_depth': 14}
0.766 (+/-0.006) for {'max_depth': 15}
0.761 (+/-0.006) for {'max_depth': 16}
0.754 (+/-0.006) for {'max_depth': 17}
0.749 (+/-0.006) for {'max_depth': 18}
0.743 (+/-0.007) for {'max_depth': 19}
0.738 (+/-0.009) for {'max_depth': 20}
0.733 (+/-0.007) for {'max_depth': 21}
0.730 (+/-0.008) for {'max_depth': 22}
0.727 (+/-0.006) for {'max_depth': 23}
0.725 (+/-0.007) for {'max_depth': 24}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

         0.0       0.90      0.94      0.92     54990
         1.0       0.57      0.34      0.43      6458
         2.0       0.86      0.85      0.86     13929

    accuracy                           0.87     75377
   macro avg       0.78      0.71      0.73     75377
weighted avg       0.86      0.87      0.86     75377



execution time : 0:01:06.321023

----------------------------------------
Trying model Gaussian Naive Bayes

Best parameters set found on train set:

{'var_smoothing': 1}

Grid scores on train set:

0.448 (+/-0.023) for {'var_smoothing': 10}
0.754 (+/-0.005) for {'var_smoothing': 1}
0.740 (+/-0.003) for {'var_smoothing': 0.1}
0.730 (+/-0.004) for {'var_smoothing': 0.01}
0.727 (+/-0.004) for {'var_smoothing': 0.001}
0.726 (+/-0.004) for {'var_smoothing': 0.0001}
0.726 (+/-0.003) for {'var_smoothing': 1e-05}
0.726 (+/-0.003) for {'var_smoothing': 1e-06}
0.726 (+/-0.003) for {'var_smoothing': 1e-07}
0.726 (+/-0.003) for {'var_smoothing': 1e-08}
0.726 (+/-0.003) for {'var_smoothing': 1e-09}
0.726 (+/-0.003) for {'var_smoothing': 1e-10}

Detailed classification report for the best parameter set:
```

```
The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

         0.0       0.81      0.96      0.88     54990
         1.0       0.51      0.33      0.40      6458
         2.0       0.94      0.41      0.57     13929

    accuracy                           0.80     75377
   macro avg       0.75      0.57      0.62     75377
weighted avg       0.81      0.80      0.78     75377



execution time : 0:02:02.358903

----------------------------------------
Trying model Linear Perceptron

Best parameters set found on train set:

{'early_stopping': True}

Grid scores on train set:

0.767 (+/-0.062) for {'early_stopping': True}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

         0.0       0.84      0.97      0.90     54990
         1.0       0.52      0.39      0.44      6458
         2.0       0.98      0.51      0.67     13929

    accuracy                           0.83     75377
   macro avg       0.78      0.62      0.67     75377
weighted avg       0.84      0.83      0.82     75377



execution time : 0:02:37.514822

----------------------------------------
Trying model Random Forest Classifier

Best parameters set found on train set:

{'max_depth': 17}

Grid scores on train set:

0.587 (+/-0.002) for {'max_depth': 1}
0.578 (+/-0.002) for {'max_depth': 2}
0.574 (+/-0.003) for {'max_depth': 3}
0.784 (+/-0.003) for {'max_depth': 4}
0.788 (+/-0.002) for {'max_depth': 5}
```

```
0.792 (+/-0.003) for {'max_depth': 6}
0.796 (+/-0.002) for {'max_depth': 7}
0.800 (+/-0.003) for {'max_depth': 8}
0.804 (+/-0.002) for {'max_depth': 9}
0.806 (+/-0.002) for {'max_depth': 10}
0.810 (+/-0.003) for {'max_depth': 11}
0.813 (+/-0.003) for {'max_depth': 12}
0.815 (+/-0.003) for {'max_depth': 13}
0.816 (+/-0.003) for {'max_depth': 14}
0.817 (+/-0.005) for {'max_depth': 15}
0.817 (+/-0.005) for {'max_depth': 16}
0.817 (+/-0.004) for {'max_depth': 17}
0.817 (+/-0.003) for {'max_depth': 18}
0.817 (+/-0.004) for {'max_depth': 19}
0.816 (+/-0.005) for {'max_depth': 20}
0.815 (+/-0.005) for {'max_depth': 21}
0.814 (+/-0.005) for {'max_depth': 22}
0.813 (+/-0.003) for {'max_depth': 23}
0.814 (+/-0.006) for {'max_depth': 24}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

         0.0       0.90      0.96      0.93     54990
         1.0       0.64      0.35      0.46      6458
         2.0       0.91      0.88      0.90     13929

    accuracy                           0.89     75377
   macro avg       0.82      0.73      0.76     75377
weighted avg       0.88      0.89      0.88     75377



execution time : 0:41:40.845284

----------------------------------------
Trying model Support Vector
Best parameters set found on train set:

{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}

Grid scores on train set:

0.807 (+/-0.003) for {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
0.815 (+/-0.004) for {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
0.815 (+/-0.007) for {'C': 1, 'kernel': 'linear'}
0.811 (+/-0.002) for {'C': 10, 'kernel': 'linear'}
0.809 (+/-0.003) for {'C': 100, 'kernel': 'linear'}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

         0.0       0.90      0.96      0.93     36525
         1.0       0.64      0.30      0.41      4226
```

```
         2.0       0.90      0.88      0.89      9433

    accuracy                           0.89     50184
   macro avg       0.81      0.71      0.74     50184
weighted avg       0.88      0.89      0.88     50184


execution time : 1:37:29.341459

Summary of results for precision
Estimator
Decision Tree         - score: 79.04%
Gaussian Naive Bayes  - score: 75.42%
Linear Perceptron     - score: 76.66%
Random Forest Classifier  - score: 81.75%
Support Vector        - score: 81.54%
```

The obtained results are almost equivalents for the best parameters in each classifies; among them, RandomForest Classifier was chosen for the better performances.

```
model = RandomForestClassifier(max_depth = 17)
model.fit(X_train, y_train)
```

2.5 Validate the model

In order to undestand the behavior of the seleced model, and to increase the value of the classifier, an in depth analyzes is applied to the classification results.

```
# Predict the test labels
y_test_predicted = model.predict(X_test)
class_labels = np.unique(y_test)
class_names = ['land', 'snow', 'ice cloud']
mask = np.in1d(y_test_predicted, y_test)
predictions = y_test_predicted[mask]
true_labels = y_test[mask]

# Extract and display metrics
f1_scores = metrics.f1_score(true_labels, predictions, labels=class_labels, average=None)
avg_f1_score = metrics.f1_score(true_labels, predictions, average="weighted")
recall = metrics.recall_score(true_labels, predictions, labels=class_labels, average=None)
precision = metrics.precision_score(true_labels, predictions, labels=class_labels,
    average=None)
accuracy = metrics.accuracy_score(true_labels, predictions)

print("Classification accuracy {:.1f}%".format(100 * accuracy))
print("Classification F1-score {:.1f}%".format(100 * avg_f1_score))
print()
print("          Class            =  F1  | Recall | Precision")
print("       ------------------------------------------------")
for idx, lulctype in enumerate([class_names[index] for index in range(0,
    len(class_labels))]):
    line_data = (lulctype, f1_scores[idx] * 100, recall[idx] * 100, precision[idx] * 100)
    print("       * {0:20s} = {1:2.1f} | {2:2.1f} | {3:2.1f}".format(*line_data))
```

```
Classification accuracy 89.2%
Classification F1-score 88.3%

          Class              =  F1  | Recall | Precision
       ------------------------------------------------
       * land                 = 92.9 |  95.6  | 90.4
       * snow                 = 45.5 |  35.6  | 63.0
       * ice cloud            = 89.6 |  88.3  | 91.0
```

Confusion matrices are a very expresive instrument about the model performance. They show the number of correctly and incorrectly predicted labels for each label from the reference map or vice versa. This shows if the classifier has a bias towards wrongly classifying certain types of land cover.

```
# Define the plotting function
def plot_confusion_matrix(
    confusion_matrix,
    classes,
    normalize=False,
    title="Confusion matrix",
    cmap=plt.cm.Blues,
    ylabel="True label",
    xlabel="Predicted label",
    filename=None,
):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
np.set_printoptions(precision=2, suppress=True)

if normalize:
    normalisation_factor = confusion_matrix.sum(axis=1)[:, np.newaxis] + np.finfo(float).eps
    confusion_matrix = confusion_matrix.astype("float") / normalisation_factor

plt.imshow(confusion_matrix, interpolation="nearest", cmap=cmap, vmin=0, vmax=1)
plt.title(title, fontsize=20)
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=90, fontsize=20)
plt.yticks(tick_marks, classes, fontsize=20)

fmt = ".2f" if normalize else "d"
threshold = confusion_matrix.max() / 2.0
for i, j in itertools.product(range(confusion_matrix.shape[0]),
    range(confusion_matrix.shape[1])):
    plt.text(
        j,
        i,
        format(confusion_matrix[i, j], fmt),
        horizontalalignment="center",
        color="white" if confusion_matrix[i, j] > threshold else "black",
        fontsize=12,
    )

plt.tight_layout()
plt.ylabel(ylabel, fontsize=20)
plt.xlabel(xlabel, fontsize=20)

fig = plt.figure(figsize=(20, 20))

plt.subplot(1, 2, 1)
confusion_matrix_gbm = metrics.confusion_matrix(true_labels, predictions)
plot_confusion_matrix(
    confusion_matrix_gbm,
    classes=[name for idx, name in enumerate(class_names) if idx in class_labels],
    normalize=True,
    ylabel="Truth (LAND COVER)",
    xlabel="Predicted (GBM)",
    title="Confusion matrix",
)

plt.subplot(1, 2, 2)
confusion_matrix_gbm = metrics.confusion_matrix(predictions, true_labels)
plot_confusion_matrix(
```

```
    confusion_matrix_gbm,
    classes=[name for idx, name in enumerate(class_names) if idx in class_labels],
    normalize=True,
    xlabel="Truth (LAND COVER)",
    ylabel="Predicted (GBM)",
    title="Transposed Confusion matrix",
)

plt.tight_layout()
```

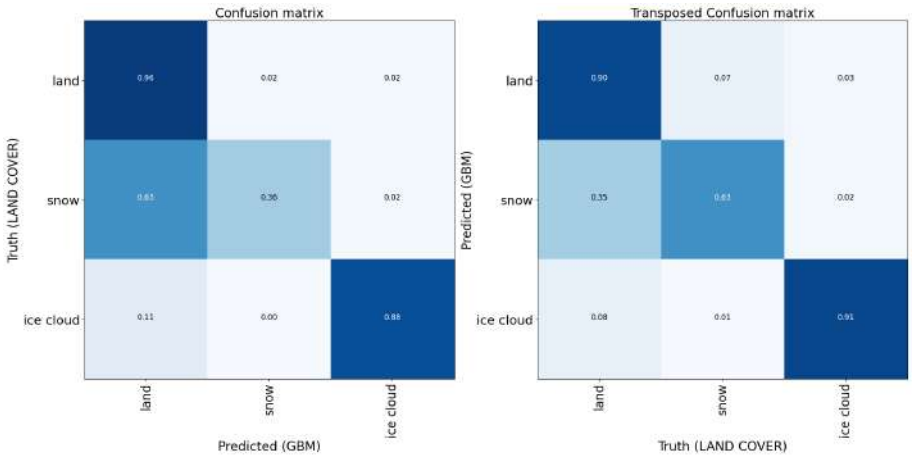

Figure 9: Confusion Matrix for the model

The ROC curve shows the diagnostic ability of a classifier as its discrimination threshold is varied: indeed the threshold for the prediction of a certain label can be manipulated. The x-axis shows the false positive rate (we want it to be small), and the y-axis shows the true positive rate (we want it to be large) at different thresholds. Good classifier performance is characterised by a curve, which has a large integral value, also known as the area under curve (AUC). In the specific case of this project, the snow curve has the worst trend, symptom of a not optimal classifier for this value. The snow classifictaion, probably, resents of an under-representation in the the dataset.

```
scores_test = model.predict_proba(X_test)
labels_binarized = preprocessing.label_binarize(y_test, classes=class_labels)

fpr, tpr, roc_auc = {}, {}, {}

for idx, lbl in enumerate(class_labels):
    fpr[idx], tpr[idx], _ = metrics.roc_curve(labels_binarized[:, -1], scores_test[:, -1])
    roc_auc[idx] = metrics.auc(fpr[idx], tpr[idx])
```

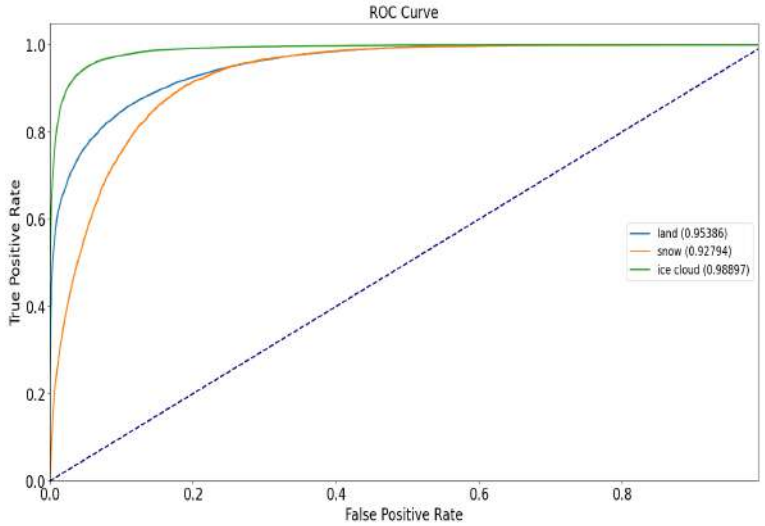

Figure 10: ROC curve

Working with maps and images, the best validation method is probably visualization. For each pixel the defined model has to be applied, in order to obtain a new map that defines areas with snow, land and cloud. The following function computes the map and adds it to the eo-learn dataset.

```
for box in focus_list:
    sample_path = os.path.join(EOPATCH_SAMPLES_FOLDER, f"eopatch_{box}")
    eopatch = EOPatch.load(sample_path, lazy_loading=True)

    input_data = np.concatenate([eopatch.data['FEATURES']])
    t, w, h, l = input_data.shape
    input_data = input_data.reshape(t * w * h, l)
    y_predicted = model.predict(input_data)

    eopatch.data['LABEL_PREDICTED'] = y_predicted.reshape(t, w, h, 1)

    eopatch.save(os.path.join(RESULTS_FOLDER, str('eopatch_'+str(box))),
        overwrite_permission=OverwritePermission.OVERWRITE_PATCH)
```

The map of a generic winter day is visualized with the following code lines: the color map *SNOW_CLOUD_MAP* defines land areas as *green*, snow areas as *white* and cloud area as *red*.

```
class SNOW_CLOUD_MAP(MultiValueEnum):
    """Enum class containing basic LULC types"""
    NO_SNOW = "Land", 0, "#0DAE26"
    SNOW_AND_ICE = "Snow and Ice", 1, "#ffffff"
    CLOUD = "cloud", 2, '#ff0000'
    @property
    def id(self):
        return self.values[1]
    @property
    def color(self):
        return self.values[2]

# Reference colormap things
snow_cloud_cmap = ListedColormap([x.color for x in SNOW_CLOUD_MAP], name="snow_cloud_cmap")
snow_cloud_norm = BoundaryNorm([x - 0.5 for x in range(len(SNOW_CLOUD_MAP) + 1)],
    snow_cloud_cmap.N)

################################################################################################################

fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(20, 15))
```

```
date = datetime.datetime(2020, 1, 29)

for i, patchID in enumerate(focus_list):
    eopatch_path = os.path.join(RESULTS_FOLDER, f"eopatch_{patchID}")
    eopatch = EOPatch.load(eopatch_path, lazy_loading=True)

    dates = np.array([timestamp.replace(tzinfo=None) for timestamp in eopatch.timestamp])
    closest_date_id = np.argsort(abs(date - dates))[0]

    ax = axs[i // 6][i % 6]
    ax.imshow(np.clip(eopatch.data["BANDS"][closest_date_id][..., [2, 1, 0]] * 3.5, 0, 1))
    ax.imshow(eopatch.data["LABEL_PREDICTED"][closest_date_id].astype(float), cmap=
        snow_cloud_cmap)

    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect("auto")
    del eopatch

fig.subplots_adjust(wspace=0, hspace=0)
```

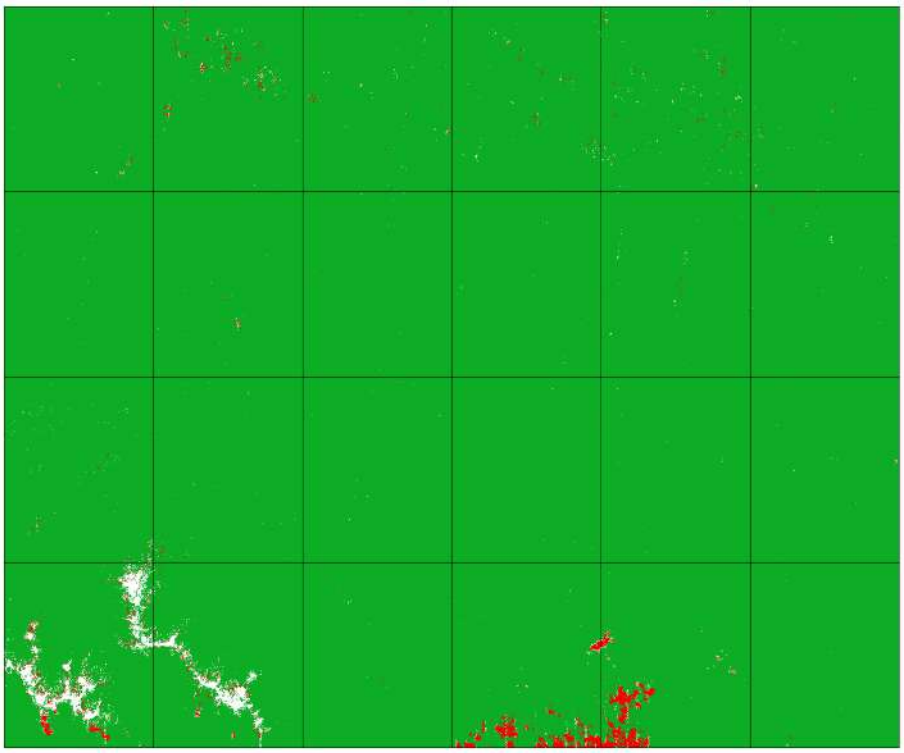

Figure 11: Snow (white) and Cloud (red) prediction

Even more significant could be a comparison between official snow cover data with automatically classified data.

```
fig = plt.figure(figsize=(20, 20))

idx = np.random.choice(focus_list)
inspect_size = 100

eopatch = EOPatch.load(os.path.join(RESULTS_FOLDER, f"eopatch_{idx}"), lazy_loading=True)

date = datetime.datetime(2020, 1, 29)
dates = np.array([timestamp.replace(tzinfo=None) for timestamp in eopatch.timestamp])
```

```
closest_date_id = np.argsort(abs(date - dates))[0]

t, y, x, p = eopatch.data['LABEL_PREDICTED'].shape

w_min = np.random.choice(range(y - inspect_size))
w_max = w_min + inspect_size
h_min = np.random.choice(range(x - inspect_size))
h_max = h_min + inspect_size

ax = plt.subplot(2, 2, 1)
plt.imshow(eopatch.data['LABEL_PREDICTED'][closest_date_id][w_min:w_max, h_min:h_max],
    cmap=lulc_cmap, norm=lulc_norm)
plt.xticks([])
plt.yticks([])
ax.set_aspect("auto")
plt.title("Ground Truth", fontsize=20)

ax = plt.subplot(2, 2, 2)
plt.imshow(eopatch.data['LABEL'][closest_date_id][w_min:w_max, h_min:h_max],
    cmap=lulc_cmap, norm=lulc_norm)
plt.xticks([])
plt.yticks([])
ax.set_aspect("auto")
plt.title("Prediction", fontsize=20)

ax = plt.subplot(2, 2, 3)
mask = eopatch.data['LABEL_PREDICTED'][closest_date_id] !=
    eopatch.data['LABEL'][closest_date_id]
plt.imshow(mask[w_min:w_max, h_min:h_max], cmap="gray")
plt.xticks([])
plt.yticks([])
ax.set_aspect("auto")
plt.title("Difference", fontsize=20)

ax = plt.subplot(2, 2, 4)
image = np.clip(eopatch.data["FEATURES"][closest_date_id][..., [2, 1, 0]] * 3.5, 0, 1)
plt.imshow(image[w_min:w_max, h_min:h_max])
plt.xticks([])
plt.yticks([])
ax.set_aspect("auto")
plt.title("True Color", fontsize=20)

fig.subplots_adjust(wspace=0.1, hspace=0.1)
```

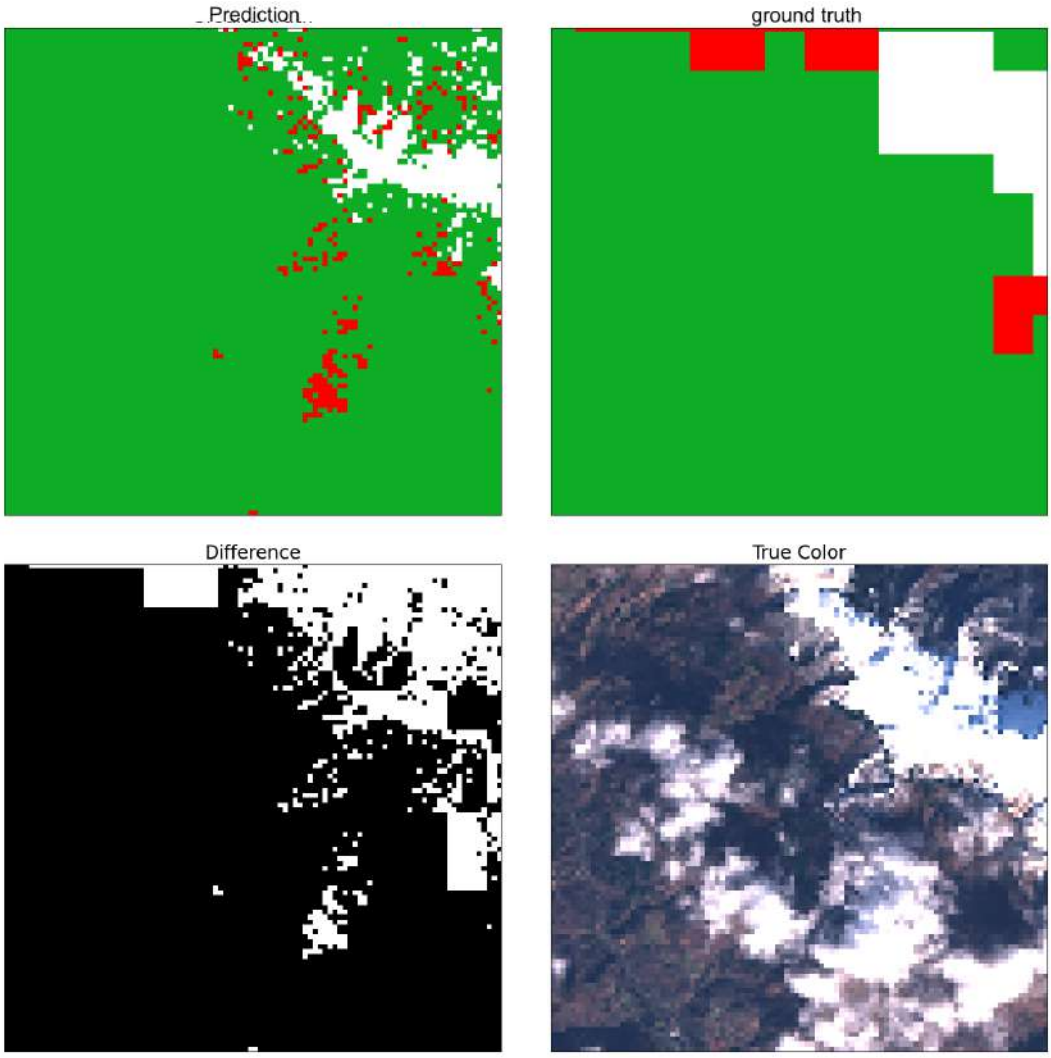

Figure 12: A comparison between ground truth data and predicted data.

2.6 Visualization

Data visualization is an important part of data science: it can help the data interpretation, make data more undestandable, allowing a view on different levels, and integrate human knowledge to slim data insight. In the specific case of the project, working with maps and geographical images, data visualization is an integrated part of the classification process, needed for reading the results.
The consultation screen will contains all the relevant information to have a clear reference in space and in time of the snow coverage of the AOI.

- **A bar graph** shows the amount of snow among the time, to have a rapid view on periods with more snow coverage;
- **A map** shows the details of snow coverage for a single day;
- **Interactive controls** allows the selection of a specific day for the detailed visualization on the map; different modality of visualization are allowed, with the possibility to view the snow coverage, the cloud coverage and both the coverages.

The following functions create a dictionary that relates the amount of snow for each day of the year.

```
day_list = []
for box in focus_list:
    sample_path = os.path.join(RESULTS_FOLDER, f"eopatch_{box}")
    eopatch = EOPatch.load(sample_path, lazy_loading=True)
    day_list.append(eopatch.timestamp)
    del eopatch
day_list = np.concatenate(day_list);
```

```
snow_amount = {}
for day in day_list:

    daily_amount = 0
    for box in focus_list:
        sample_path = os.path.join(RESULTS_FOLDER, f"eopatch_{box}")
        eopatch = EOPatch.load(sample_path, lazy_loading=True)

        dates = np.array([timestamp.replace(tzinfo=None) for timestamp in eopatch.timestamp])
        closest_date_id = np.argsort(abs(day - dates))[0]

        daily_amount +=
            eopatch.mask_timeless['REAL_DATA_SNOW_COUNT'][closest_date_id][0][0].astype(float)
    snow_amount.update({day.strftime("%x") : daily_amount})
```

New color maps are defined, to meet the visualization modality specifics.

```
class SNOW_MAP(MultiValueEnum):
    """Enum class containing basic LULC types"""
    NO_SNOW = "Land", 0, "#0DAE26"
    SNOW_AND_ICE = "Snow and Ice", 1, "#ffffff"
    CLOUD = "cloud", 2, '#0DAE25'
    @property
    def id(self):
        return self.values[1]
    @property
    def color(self):
        return self.values[2]

# Reference colormap things
snow_cmap = ListedColormap([x.color for x in SNOW_MAP], name="snow_cmap")
snow_norm = BoundaryNorm([x - 0.5 for x in range(len(SNOW_MAP) + 1)], snow_cmap.N)
```

```
class CLOUD_MAP(MultiValueEnum):
    NO_SNOW = "Land", 0, "#0DAE26"
    SNOW_AND_ICE = "Snow and Ice", 1, "#0DAE25"
    CLOUD = "cloud", 2, '#ff0000'
    @property
           [ ... ]
class SNOW_CLOUD_MAP(MultiValueEnum):

    NO_SNOW = "Land", 0, "#0DAE26"
    SNOW_AND_ICE = "Snow and Ice", 1, "#ffffff"
    CLOUD = "cloud", 2, '#ff0000'
    @property
           [ ... ]

cloud_cmap = ListedColormap([x.color for x in CLOUD_MAP], name="cloud_cmap")
cloud_norm = BoundaryNorm([x - 0.5 for x in range(len(CLOUD_MAP) + 1)], cloud_cmap.N)
snow_cloud_cmap = ListedColormap([x.color for x in SNOW_CLOUD_MAP], name="snow_cloud_cmap")
snow_cloud_norm = BoundaryNorm([x - 0.5 for x in range(len(SNOW_CLOUD_MAP) + 1)],
    snow_cloud_cmap.N)
```

To make the Visualization application interactive, the bokeh package was chosen: bokeh is an open source, flexible and interactive library, that easily allows to create graphs and controls (*Bokeh at a glance - Bokeh documentation*, n.d.).

```
from tkinter import colorchooser
from ipywidgets import interact
import matplotlib.image as mpimg
from bokeh.io import output_notebook, show, push_notebook
from bokeh.models.widgets import Div
```

```
from bokeh.application import Application
from bokeh.application.handlers import FunctionHandler

os.makedirs(os.path.join(RESULTS_FOLDER, 'images'), exist_ok=True)
output_notebook()

desc = Div(text=open("description.html").read(), sizing_mode="stretch_width")

# Create Column Data Source that will be used by the plot
source = ColumnDataSource(data=dict(date=[d for d in snow_amount.keys()], amount = [v for v
    in snow_amount.values()], colors = ['green' for i in range(len(snow_amount.keys()))]))
x = [k for k in snow_amount.keys()]

p = figure(x_range=FactorRange(*x), plot_height=300, plot_width=1000, title="",
    toolbar_location=None, tools="")
p.xaxis.major_label_orientation = "vertical"
p.vbar(x='date', top='amount', width=0.9, source=source, color = 'colors')

image = Div(text='', sizing_mode="stretch_width")

def update(day, modality):
    for index, day_selected in enumerate(day_list):
        if day_selected.strftime("%x") == day:
            break

    fig, axs = plt.subplots(nrows=4, ncols=6, figsize=(15, 10));

    for i, patchID in enumerate(focus_list):
        eopatch_path = os.path.join(RESULTS_FOLDER, f"eopatch_{patchID}")
        eopatch = EOPatch.load(eopatch_path, lazy_loading=True)

        dates = np.array([timestamp.replace(tzinfo=None) for timestamp in eopatch.timestamp])
        closest_date_id = np.argsort(abs(day_selected - dates))[0]

        ax = axs[i // 6][i % 6]
        if modality == 'snow':
            ax.imshow(eopatch.data["LABEL_PREDICTED"][closest_date_id].astype(float), cmap=
                snow_cmap);
        elif modality == 'cloud':
            ax.imshow(eopatch.data["LABEL_PREDICTED"][closest_date_id].astype(float), cmap=
                cloud_cmap);
        elif modality == 'snow&cloud':
            ax.imshow(eopatch.data["LABEL_PREDICTED"][closest_date_id].astype(float), cmap=
                snow_cloud_cmap);

        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_aspect("auto")
        del eopatch
    fig.subplots_adjust(wspace=0, hspace=0)
    fig.savefig(os.path.join(RESULTS_FOLDER, os.path.join('images', 'temp_image.png')))

    p.title.text = str(str(modality) + 'map for the day: '+ str(day))
    color_selected = ['green' for i in range(len(snow_amount.keys()))]
    color_selected[index] = 'red'
    source.data = dict(
        date=[d for d in snow_amount.keys()],
        amount = [v for v in snow_amount.values()],
        colors = color_selected
    )

    image = Div(text=open(os.path.join(RESULTS_FOLDER, os.path.join('images',
        'temp_image.png')), encoding='utf8', errors='ignore').read(),
```

```
        sizing_mode="stretch_width")
    push_notebook()

show(column(desc, p, image), notebook_handle=True)
interact(update, day = [day.strftime("%x") for day in day_list], modality = ['snow',
    'cloud', 'snow&cloud'])
```

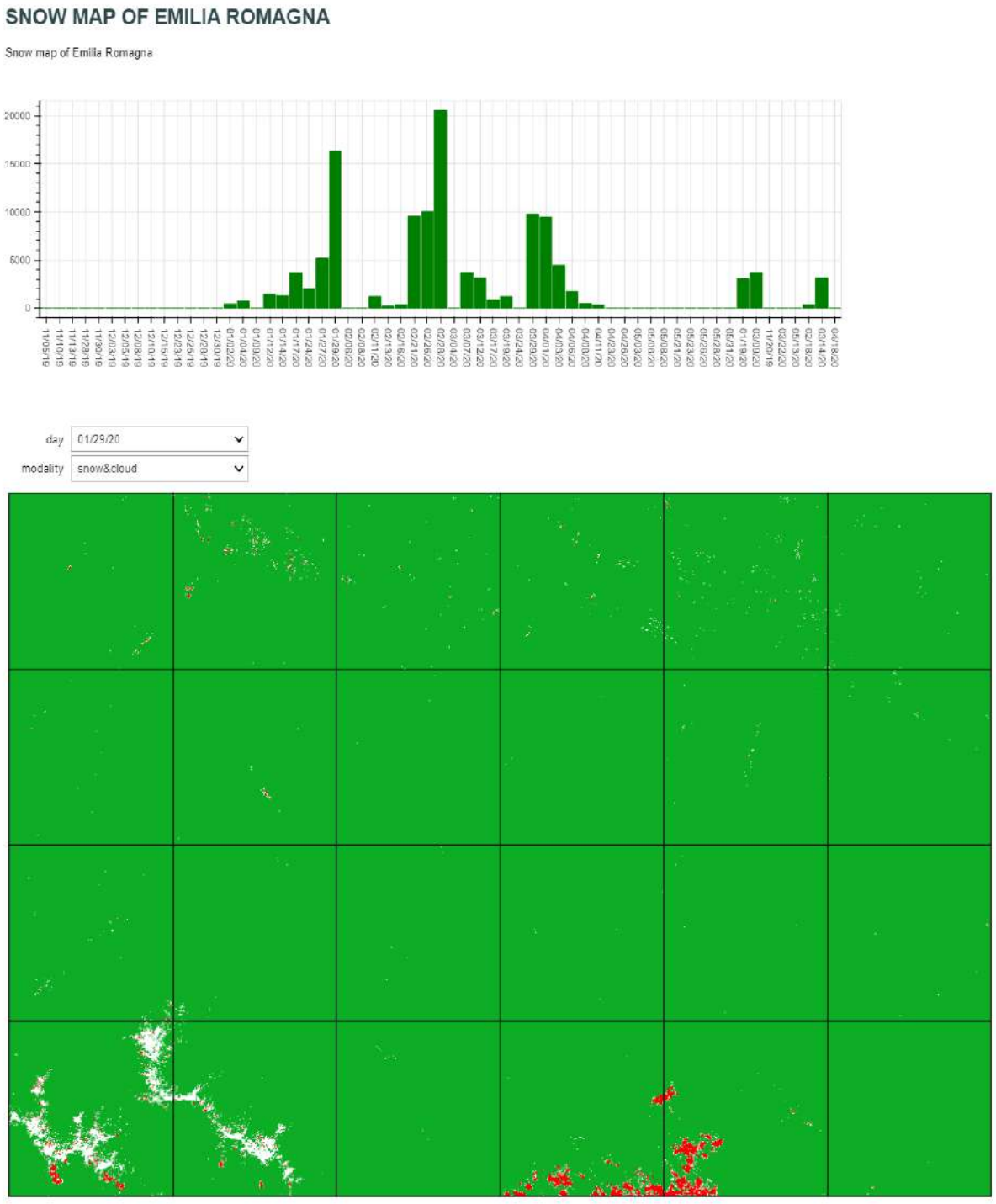

Figure 13: Interactive application for prediction visualization

2.7 Conclusion

The project described did not fully lead to the expected results due to the low quality and low definition of the ground truth data. Furthermore, in the chosen area of interest, the Appennini's Mountain in the north of Italy, snow happens to be scarcely represented. More variated data could be obtained considering a larger temporal scale, but data access on the sentinel hub portal is limited. However, besides a less than optimal precision and accuracy, the algorithm works and data are accessible intuitively and clearly. The use of el-learn instrument has been put into practice, integrated with a variety of package for machine learing and data visualization.

2.8 Future project

Besides the low quality of the results, snow cover monitoring remains a usefull and full of potential activity. The machine learining approach on data satellite images, allows a low cost global monitoring on lare time intervals: the potential insight we can obtain are multiple, with a huge social impact. This project, in particular, can be easily improved and made more powerful using a larger dataset, with a more balanced data of snow and not-snow in space and time. The requested amount of data requires an higher computational power, and a larger data exchange: for a better performance and for avoinding the overhead for data downloading, the application with the machine learining training phase should be ported on a cloud application. Finally, a more interactive data visualization application can be easily obtained, eventualy integrating a web client to automatically obtain data from cloud.

References

Bokeh at a glance - bokeh documentation. (n.d.). https://bokeh.org.

Earth observatory. (n.d.). https://earthobservatory.nasa.gov.

Eo-learn Contributors. (2018). *Introduction – eo-learn documentation.* https://eo-learn.readthedocs.io/en/latest/index.html#introduction.

geojson-italy. (2022). GitHub. Retrieved from hhttps://github.com/openpolis/geojson-italy

High resolution snow and ice monitoring. (n.d.). https://land.copernicus.eu/pan-european/biophysical-parameters/high-resolution-snow-and-ice-monitoring.

Lubej, M. (2018). Land cover classification with eo-learn: Part 1. *Sentinel Hub Blog*.

Magnani, C. (2022). Sviluppo e validazione di mappe di copertura nevosa per lo studio di variabilità del manto nevoso in emilia romagna nel periodo 2000-2020.

Multispectral instrument (msi) overview. (n.d.). https://sentinels.copernicus.eu/web/sentinel/technical-guides/sentinel-2-msi/msi-instrument.

Sentinel2 cloud-detector. (2022). GitHub. Retrieved from https://github.com/sentinel-hub/sentinel2-cloud-detector

Wikipedia Contributors. (2022). *Sentinel-2 — Wikipedia, the free encyclopedia.* https://en.wikipedia.org/w/index.php?title=Sentinel-2&oldid=1105470108. ([Online; accessed 19-September-2022])