ARSS Project — Project for ”Sensing and Radar Technologies” and ”Project Course”

Luca Zanolo
luca.zanolo@studenti.unitn.it
University of Trento
Trento, Italy

***Abstract*—This project is part of the ”Sensing and Radar Technologies” and ”Project Course” modules, contributing 12 CFU to the Master’s program in Artificial Intelligence Systems. The primary objective is to reimplement an existing pipeline for processing Sentinel-2 Level-2A (L2A) data, part of the European Space Agency’s Climate Change Initiative (CCI) High-Resolution Land Cover (HRLC) project. Key processing steps include refining cloud and shadow masks, restoring cloud-affected areas, generating advanced features and optimizing Support Vector Machine (SVM) based classification. This report details the reimplemented workflow, the methodological approaches adopted and the results of the experiments.**

I. Introduction

This project is part of the ”Sensing and Radar Technologies” and ”Project Course” modules, contributing a total of 12 CFU to the Master’s program in Artificial Intelligence Systems. The focus is on analyzing and processing multispectral remote sensing data to create land cover maps for applications related to climate change.

The project focuses on the reimplementation of a portion of an existing pipeline for processing multispectral data. This pipeline is part of the European Space Agency’s Climate Change Initiative (CCI) High-Resolution Land Cover (HRLC) project. The red blocks in Figure 1 indicate the specific components of the pipeline involved in this project. The figure is adapted from Chapter 2 of the ESA CCI HRLC Algorithm Theoretical Basis Document (ATBD), version 4.0 [1].

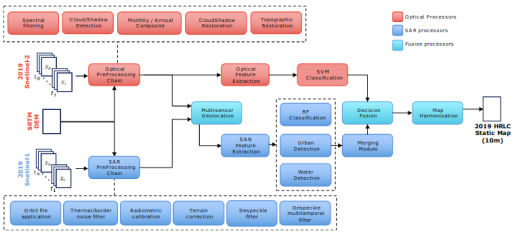


Fig. 1: Block diagram representing the original HRLC processing chain for the production of static HRLC maps [1]. The red rectangles indicate the segments of the pipeline targeted in this project.

The initial pipeline consisted of seven main components, each corresponding to a processing step:

- **Image selection**: filters and selects input images based on specific quality criteria.
- **Spectral filtering**: uses spectral filters to enhance data quality.
- **Cloud/Shadow detection**: enhances the quality of the Sen2Cor cloud and shadow masks available with the L2 products.
- **Monthly/Annual composites generation**: creates composite images over specific periods to represent consistent surface conditions.
- **Cloud and topographic restoration**: addresses missing data due to cloud cover through restoration techniques.
- **Features extraction, dataset generation and Support Vector Machine (SVM) training**: extract features from composites and Gray-Level Co-occurrence Matrix (GLCM) and relate them to available annotated points creating a dataset. Use the dataset to train a Support Vector Machine (SVM) classifier.
- **SVM classification**: applies the trained SVM classifier to produce final land cover maps.

The ultimate aim of the CCI HRLC project is to produce accurate land cover maps at a spatial resolution of 10 meters and the original pipeline was based on GDAL for handling initial multispectral Sentinel-2 Level-2A (L2A) products [2] and subsequent data.

The reimplementation transitioned from GDAL to `xarray` for efficient handling of multi-dimensional datasets, with `dask` implicitly utilized for parallel computation [3], [4]. The logic and methodology embedded within the existing pipeline were preserved and adapted to the new implementation. The operational flow was restructured, omitting the image selection and optical spectral filtering steps to ensure that all available images were utilized. This approach reduced artifacts along map borders resulting from varying image selections per tile. Although this change increased the data volume to process, the impact was mitigated by the adoption of the new libraries.

The experimental part of the project focused firstly on generating previously unused features to try to enhance the quality of SVM predictions. The generated datasets, composed of composites and features, were enriched using techniques involving the final land cover maps from the old pipeline to add new samples. Then, feature selection involved two approaches: a multicollinearity matrix analysis and recursive feature elimination with cross-validation (`RFECV`) and for feature reduction, Principal Component Analysis (PCA) was considered. Finally, the SVMs were trained using both the

original pipeline's configuration and an alternative binary approach, training one binary SVM for each class. This allowed for evaluating and comparing the calibration curves of the final classifiers.

The resulting pipeline thus consists of seven redefined steps:

- **Masks Refinement**: enhances Sen2Cor cloud and shadow masks, as in the original pipeline.
- **Cloud Restoration**: restores externally generated composites to minimize missing values. As in *Cloud and Topographic Restoration* step of the existing pipeline.
- **Features Generation**: creates GLCM features as in the existing pipeline, along with new spectral and spatial features.
- **Dataset Generation**: supports the creation of various datasets based on composites and generated features.
- **SVM Pipeline**: enables experimentation with different SVM configurations, including grid search for systematic parameter testing. This component and the `dataset and features generation` components, correspond to the `features extraction`, `dataset generation and SVM training` step of the existing pipeline.
- **LC Maps Generation**: uses trained SVMs to generate and store land cover maps. This corresponds to `SVM Classification` step of the existing pipeline.

The `monthly/annual composites generation` step is not part of this reimplemented pipeline. It was reimplemented independently and the generated composites were shared to be used for this project, from the `features generation` step onwards.

A. Repository

The entire project has been developed as a standard Python package, leveraging Docker. This setup ensures that the code can be tested and executed within a controlled environment and on different machines. All project-related materials are available inside the repository on `Bitbucket` and detailed instructions for setting up the environment and executing each step of the pipeline are provided in the project's README document. For a complete overview of the code structure and usage, refer to the repository, starting from README.

B. Development and Testing Environment

The development and testing of this project were conducted on a remote machine provided by the Remote Sensing Laboratory (RSLab) at the University of Trento. The specifications of the system are as follows:

- **Processor**: AMD Ryzen 7 5800X, 8 cores, 16 threads
- **RAM**: 67 GB
- **GPU**: NVIDIA Quadro P6000, 24,576 MiB
- **Operating System**: Ubuntu 22.04.3 LTS

C. Xarray and Dask

This section provides a brief presentation to the two main tools adopted in the new pipeline for data management.

`xarray` is an open-source Python package designed for handling multi-dimensional labeled data. It extends the capabilities of the `pandas` library to N-dimensional datasets by introducing labels in the form of dimensions, coordinates and attributes. This approach facilitates handling complex datasets. Moreover, `xarray` integrates seamlessly with `dask`, enabling parallel computation on datasets that are too large to fit into memory [3].

`Dask` is an open-source Python library that enables parallel and distributed computing. It extends existing Python libraries like `NumPy` and `pandas` to efficiently handle larger-than-memory datasets and supports distributed computation. Dask provides high-level abstractions for arrays, dataframes and other data structures, making it an effective tool for scalable data analysis [4].

An important feature of `dask`, leveraged in this project, is *lazy evaluation*. This means that computations are not performed immediately but are instead added as tasks to a tasks graph and executed when necessary. This approach enables the construction of complex graphs without immediate execution, optimizing memory usage and reducing unnecessary computations. When combined with `xarray`, `Dask`'s lazy evaluation allows efficient processing of large multi-dimensional datasets by loading and computing only the necessary data chunks when required.

The final important feature to highlight is the chunking capability provided by both libraries. Chunking enables the division of data into smaller, manageable blocks, allowing for efficient parallel processing. Only the necessary chunks are loaded into memory at a given time, which makes it possible to work with datasets that exceed available memory.

D. Sentinel-2 L2A Products and Sen2Cor Mask

The Sentinel-2 Level-2A (L2A) products, which are the initial data in input to both the existing and reimplemented pipelines, consist of Bottom of Atmosphere (BOA) reflectance images derived from Level-1C Top of Atmosphere (TOA) data. The L2A products include essential information for land cover analysis, such as surface reflectance corrected for atmospheric effects and other useful data like cloud and shadow masks [2].

Atmospheric correction for L2A products is performed using the Sen2Cor processor. Sen2Cor applies the Sentinel-2 Atmospheric Correction (S2AC) algorithm, which is based on the LIBRADTRAN radiative transfer model. This model generates a Lookup Table (LUT) containing sensor-specific functions that account for various atmospheric conditions, solar geometry and ground elevations. The LUT is used to invert the radiative transfer equation and compute BOA reflectance efficiently. Additionally, S2AC employs Lambert's reflectance law and can correct topographic effects using a Digital Elevation Model (DEM). It assumes a constant viewing angle per tile and can interpolate solar angles across the scene [1]. Sen2Cor also provides cloud and shadow masks, which are crucial for identifying and excluding cloud-covered or shadowed pixels.

E. Report Structure

The next section of this report outlines the baseline scripts from which the project originated, providing context for the reimplemented components. The following sections then describe each element of the new pipeline in detail, presenting each step in order, from mask refinement to the final land cover map generation.

II. Existing Pipeline - Overview of Pipeline Components

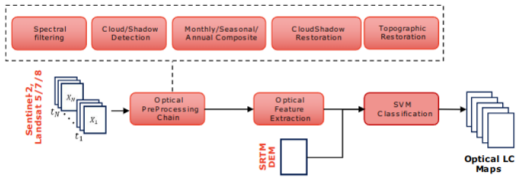


Fig. 2: Optical data processing chain for the prototype production of both the static and the historical HR LC maps [1].

As anticipated, the original pipeline, designed as part of the CCI HRLC project, aimed to process multispectral Sentinel-2 Level-2A (L2A) products to produce high-resolution land cover maps. The pipeline was primarily implemented using GDAL for data handling and consisted of seven key steps, each corresponding to specific scripts and processing objectives.

A. Image Selection

The original pipeline included an image selection step aimed at filtering Sentinel-2 L2A images based on cloud cover percentage and temporal range. Images with cloud cover below a specific threshold were selected to be used in the pipeline, while temporal filtering ensured coverage consistency within the requested periods.

However, this approach potentially led to inconsistencies along the borders of adjacent patches. Since images for each of them could be selected from different time steps, artifacts could arise due to temporal discrepancies in land cover representation. To mitigate this issue, the reimplemented pipeline removes this step, opting instead to process all available images. This change reduces the risk of introducing border artifacts between adjacent patches while potentially increases the amount of data in input to manage.

B. Spectral Filtering

The second step of the pipeline, spectral filtering, aimed to detect and remove outliers present in optical images. For each spectral band, reflectance values lower than the 0.001 quantile and higher than the 0.999 quantile were discarded. This approach mitigates the impact of extreme values, which could result from sensor noise or atmospheric effects [1].

All images considered for this step had cloud coverage below 40%. Additionally, to minimize the influence of clouds and shadows, pixels identified by the Sen2Cor mask were excluded from the evaluation [1].

Formally, for a reflectance band x, the filtered reflectance $x_{filtered}$ is defined as:

$$x_{filtered} = \begin{cases} x_{min}, & \text{if } x < x_{min}, \\ x_{max}, & \text{if } x > x_{max}, \\ x, & \text{otherwise} \end{cases} \tag{1}$$

where x_{min} and x_{max} correspond to the 0.001 and 0.999 quantiles of the reflectance distribution, respectively.

In the reimplementation, this step was removed to simplify processing and to avoid potential inconsistencies introduced by the filtering process.

C. Cloud / Shadow Detection

The cloud detection step aimed to refine the initial Sen2Cor cloud and shadow masks by reducing misclassifications and enhancing overall mask accuracy. The refinement process consisted of two key phases: background modeling and clustering-based mask enhancement [1].

1) Background Modeling: Seasonal background images were generated by analyzing temporal series of images. For each season, the process involved calculating the 25th percentile of the blue band reflectance values across all available images within a given patch. Only pixels not flagged as clouds or shadows by the initial Sen2Cor masks were considered for this percentile calculation, ensuring that the background model represented clear-sky observations as accurately as possible [1].

2) Cloud Mask Refinement: The refinement process utilized the K-Means clustering algorithm from Scikit-learn to enhance the cloud mask. The procedure began by computing the absolute difference between the blue band of the current image and the corresponding seasonal background image:

$$D = |I_{blue} - B_{blue}| \tag{2}$$

where I_{blue} is the blue band of the current image, and B_{blue} is the background image.

This difference image D was then segmented into three clusters using the K-Means algorithm. The cluster with the smallest mean difference, closest to the mean reflectance of the existing cloud mask pixels, was selected and merged with the initial mask to improve cloud detection accuracy [1].

Figure 3, from [1], show the flowchart of the operations described above.

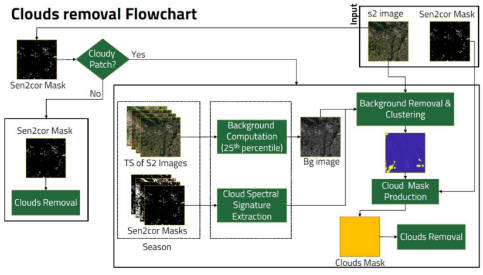


Fig. 3: Flowchart of the Sen2cor cloud mask improvement [1].

3) Shadow Mask Refinement: The shadow mask refinement focused on enhancing the detection of shadowed pixels using spectral information from the Near-Infrared (NIR) and Short-Wave Infrared (SWIR) bands. The Cloud Shadow Index (CSI) was calculated as follows [1], [5]:

$$CSI = \frac{NIR + SWIR}{2} \tag{3}$$

To refine shadow detection, thresholds were defined for both the CSI and the blue band:

$$T_{CSI} = \min(CSI) + 0.5(\overline{CSI} - \min(CSI)) \tag{4}$$

$$T_{Blue} = \min(Blue) + 0.25(\overline{Blue} - \min(Blue)) \tag{5}$$

where $\overline{CSI}$ and $\overline{Blue}$ represent the mean values of the CSI and blue band, respectively.

Pixels satisfying both threshold conditions were incorporated into the refined shadow mask. To further enhance mask quality and reduce noise, the resulting shadow mask was smoothed using a median filter.

Figure 4, from [1], show the flowchart of the operations described above.

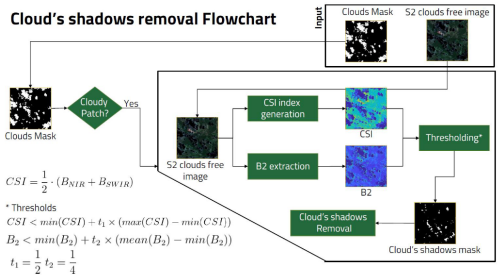


Fig. 4: Flowchart of the Sen2cor cloud shadow mask improvement and removal [1].

D. Monthly / Annual Composites Generation

As previously noted, this component was not reimplemented within the new pipeline, but rather developed externally. Anyway, since the composites generated by this external module were used during testing, a breakdown of the composite generation logic from the original pipeline is included here for completeness.

The objective of this component is to generate cloud-free Sentinel-2 composites by aggregating multiple Sentinel-2 L2A products over a monthly basis.

Satellite land monitoring is often challenged by atmospheric disturbances, particularly cloud and shadow contamination. Generating composites over a period helps mitigate these effects by integrating multiple observations of the same location and selecting representative values. The core functionality of this component is to synthesize a single image per temporal window by leveraging multiple cloud-free observations.

The composite images are generated using a statistical aggregation method applied pixel-wise across multiple observations. The chosen approach is the **median composite method**, which computes the median reflectance value for each spectral band across the available cloud-free observations.

The median is an estimator of central tendency, making it particularly suitable for handling noisy or contaminated observations. Unlike the mean, which can be skewed by

extreme outliers, the median is resilient to changes caused by residual clouds, shadows or atmospheric variations.

In practical terms, the composite generation involves the following key steps:

- **Masking invalid observations**: each Sentinel-2 image is accompanied by cloud and shadow masks. These masks, refined in the previous processing step, are used to filter out pixels affected by cloud or shadow contamination.
- **Stacking valid observations**: for each pixel location, all cloud-free reflectance values over the given temporal window are collected.
- **Computing the median**: The median reflectance is calculated for each spectral band separately.

This process ensures that the final composite retains as much spatial detail as possible while minimizing atmospheric disturbances. The final output includes:

- A multi-band composite image where each pixel value is derived using the median compositing method.
- A **cloud persistence mask**, which records how many times each pixel was flagged as cloudy or shadowed during the compositing period. This provides additional insight into the atmospheric conditions of the analyzed region.

E. Cloud and Topographic Restoration

The cloud restoration step aimed to address gaps in composite images caused by cloud and shadow coverage. The approach utilized temporal information and topographic correction to fill missing data and enhance image quality [1].

1) Temporal Restoration: Missing pixels were restored using observations from temporally adjacent images. If only one adjacent image (either before or after) was available, its pixel values were directly used for restoration. When both before and after images were present, the missing pixel values were estimated as the mean of the corresponding pixels from the two images. This approach aimed to minimize temporal inconsistencies and preserve surface characteristics. The only case in which a pixels remains missing is if in both the adjacent images that pixels are missed.

2) Topographic Shadow Reconstruction: Following temporal restoration, the process involved correcting topographic shadows using slope information from a Digital Elevation Model (DEM). The correction methodology comprised the following steps:

- **Slope calculation**: the slope is derived from the DEM.
- **Shadow Detection**: Pixels were identified as shadowed based on their reflectance values in specific spectral bands. Shadows tend to have lower reflectance values, particularly in the Near-Infrared (NIR) and Short-Wave Infrared (SWIR) bands. The CSI was computed as in Equation 3 and the two threshold to differentiate shadowed pixels with the Equations 4 and 5.
  A pixel was classified as shadowed if its CSI and blue band values were below these thresholds. Slope information was also used to distinguish shadowed areas from water bodies, ensuring more accurate shadow detection [1].
- **Statistical matching**: shadowed pixels typically exhibit lower reflectance due to limited illumination. However, the underlying surface can still provide a weak signal. The correction aimed to adjust shadowed pixel values to match the spectral characteristics of surrounding shadow-free areas. This was achieved by applying the following equation:
  $$y_j = \frac{S_f}{S_s}(x_j - M_s) + M_f \tag{6}$$
  where y_j is the restored pixel value, S_s and M_s are the standard deviation and mean of the shadowed area, respectively, and S_f and M_f are the standard deviation and mean of the shadow-free surrounding area [1].
  This transformation scales and shifts the shadowed pixel values to align them statistically with the non-shadowed areas, reducing the disparity introduced by shadowing effects.
- **Smoother interpolation**: to further enhance the continuity of the restoration, an inpainting technique based on the fast marching method was employed. This approach interpolates pixel values by propagating information from surrounding regions, ensuring smoother transitions [1] [6].

F. GLCM Computation and SVM Training

This stage of the original pipeline involved extracting textural features using the GLCM, generating the dataset and training a SVM classifier for land cover classification [1].

1) GLCM Computation: GLCM is a statistical method used to analyze the spatial relationship of pixels, capturing textural properties within an image. The computation process was structured as follows:

- **Input and preprocessing**: the input images were normalized to a reflectance range of 0 to 254 to facilitate the GLCM computation and to any missing or invalid data point was assigned a no-data value.
- **Sliding window approach**: a sliding window of size 7×7 pixels was applied across the image to compute GLCM properties locally. For each window, six GLCM properties were computed:
  - *Contrast*
  - *Dissimilarity*
  - *Homogeneity*
  - *Energy*
  - *Correlation*
  - *Angular Second Moment (ASM)*

The resulting feature maps were resized to match the original image dimensions using bivariate spline interpolation to ensure spatial alignment and the generated feature layers were saved as GeoTIFF files for subsequent processing step.

2) SVM Training: The SVM classifier was trained using both spectral and textural features. The training process consisted of the following steps:

- **Data preparation**: training samples were extracted from labeled shapefiles and corresponding image patches. Each sample included spectral bands and computed GLCM features.
- **Normalization**: features were normalized using either min-max scaling or standardization. For min-max scaling, the normalization formula was:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{7}$$

  where x_{min} and x_{max} represent the minimum and maximum observed values, respectively.
  For standard normalization, the formula used was:

$$x_{norm} = \frac{x - \mu}{\sigma} \tag{8}$$

  where μ and σ are the mean and standard deviation of the feature, respectively.
- **Grid search for hyperparameters**: a grid search was conducted to optimize the SVM's hyperparameters. The search space included values for the regularization parameter C and the kernel coefficient γ, specifically:
  - $C \in [100, 1000]$
  - $\gamma \in [0.001, 10]$

  The evaluation follow a three-fold cross-validation approach for each combination, with the best performing parameters selected for the final model.
- **Model training**: the SVM model was trained using the optimal parameters. Then, the trained SVM model along with normalization parameters and class information was saved.

G. Classification

The final classification step involved applying the trained SVM model to generate land cover maps. The process utilized spectral and textural features and involved the following steps:

1) Model Loading: The SVM model, along with the associated normalization parameters and class information, were loaded from files. This ensured consistency with the training process.

2) Tile-based Processing: The classification was conducted tile by tile. For each tile, the corresponding features were processed in patches to manage memory usage efficiently. Each patch was classified and the resulting probabilities were stored in a multi-dimensional probability map.

3) Probability Mapping: The classifier produced probability maps for each class. The probabilities were scaled to fit an 8-bit integer range for efficient storage, using the transformation:

$$p_{scaled} = 254 \times p + 1 \tag{9}$$

where p is the original probability. Zero pixels were denoted by setting to zero the probabilities of all classes for the affected pixels.

4) Land cover map Generation: The final classification maps were saved as GeoTIFF files, ensuring geospatial consistency by applying the original images' georeferencing information.

The classification labels used are shown in Figure 5.

HRLC CLASSES			
CODE	DESCRIPTION		
0	No data		
10	Tree cover evergreen broadleaf		
20	Tree cover evergreen needleleaf		
30	Tree cover deciduous broadleaf		
40	Tree cover deciduous needleleaf		
50	Shrub cover evergreen		
60	Shrub cover deciduous		
70	Grasslands		
80	Croplands		
90	Woody vegetation aquatic or regularly flooded		
100	Grassland vegetation aquatic or regularly flooded		
110	Lichens and mosses		
120	Bare areas		
130	Built-up		
140	**Open water**	141	Open water seasonal
		142	Open water permanent
150	Permanent snow and/or ice		

Fig. 5: Final high resolution HR Land Cover classification legend defined during the HRLC project activity [1].

III. Reimplemented Pipeline - Introduction

This section covers all the reimplemented steps, presenting the modifications made to the previous workflow. As anticipated in Section I, the reimplemented steps include Masks Refinement, Cloud Restoration, Features Generation, Dataset Generation, SVM Pipeline and LC Maps Generation.

Before proceeding into the detailed descriptions of each pipeline step, the subsequent sections introduce the organization of the work and the sources of performance metrics. Additionally, the Digital Elevation Model (DEM) utilized in some pipeline steps is introduced.

A. General Approach

As described in Section I-A, the development is based on the usage of Docker, which provides an environment with all the necessary Python packages for executing the code. To support more interactive and user-friendly development, Python notebooks were also employed. These notebooks were used as a test environment, and once the code was appropriately validated, it was integrated into scripts for deployment as a Python package within the Docker container.

Although the notebooks do not run within the Docker environment, they can be executed by creating a virtual environment using the provided setup script. These notebooks are an integral part of the project as they enable direct control over each internal operation of the processing steps. They also allow for the generation and inspection of various outputs. For instance, when datasets are generated, interactive maps visualize the spatial distribution of sampling points. Most of the materials subsequently presented originate from these notebooks.

Similar to the existing pipeline, the reimplemented pipeline includes multiple configurable parameters for each step. These parameters range from path-like variables, indicating input and output locations, to specific values regulating the behavior of processing components. The parameters for the reimplemented pipeline are organized into `.yaml` files, with one dedicated file for each step.

In the following sections, each step of the pipeline will be detailed, including a dedicated explanation of the parameters that govern the processing workflow.

The following parameters are common to each component and will not be presented in subsequent sections. These parameters are:

- **Logging directory** (`log_dir`): the path to the directory in which the logging files are stored.
- **Logging level** (`log_level`): level of the stdout handler.
- **Verbose** (`verbose`): regulate the quantity of the output presented in console or log file.

The next section will briefly introduce the Digital Elevation Model and the method used to retrieve such data, then each component of the pipeline will be presented.

B. Digital Elevation Model (DEM)

The DEM was used by the existing pipeline for certain operations, as shown in Figure 1 and Figure 2. The reimplemented pipeline used the DEM image for the `Cloud Restoration` step and also to extract features, such as slope and aspect.

The DEM utilized was obtained through the browser service offered by the Visioterra project, accessible at https://visioterra.fr/web/Digital-Elevation-Model-for-Sentinel-2-1935?lang=en [7]. This service provides DEM data for specific Sentinel-2 tiles as downloadable `.tif` files.

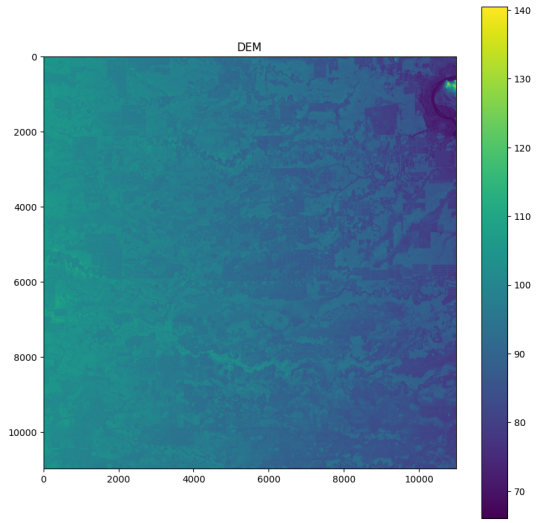


Fig. 6: Digital Elevation Model image for tile 21KUQ of 2022 with a resolution of 10 meters. Downloaded from [7].

From the DEM, two essential terrain characteristics were derived: **slope** and **aspect**.

- **Slope**: this represents the steepness or degree of incline of the terrain. The slope for each pixel is computed by measuring the rate of elevation change in both horizontal (`x`) and vertical (`y`) directions. Mathematically, it is calculated as follows:

$$S = \arctan\left(\sqrt{\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2}\right) \tag{10}$$

  where $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ represent the rate of elevation change in the `x` and `y` directions, respectively. These partial derivatives, in other words, represent the gradients of the terrain in their respective directions. The combination of these gradients determines the overall slope at each point. The arctangent function converts this gradient magnitude into an angle, measured in degrees, representing the steepness of the slope.
- **Aspect**: this indicates the compass direction that the slope faces. The aspect is computed using the arctangent of the gradients and adjusted to ensure the result falls within a 0 to 360-degree range, where 0° represents North, 90° East, 180° South, and 270° West. The formula used is:

$$A = (450 - \arctan\left(\frac{\partial z}{\partial y}, -\frac{\partial z}{\partial x}\right)) \mod 360 \tag{11}$$

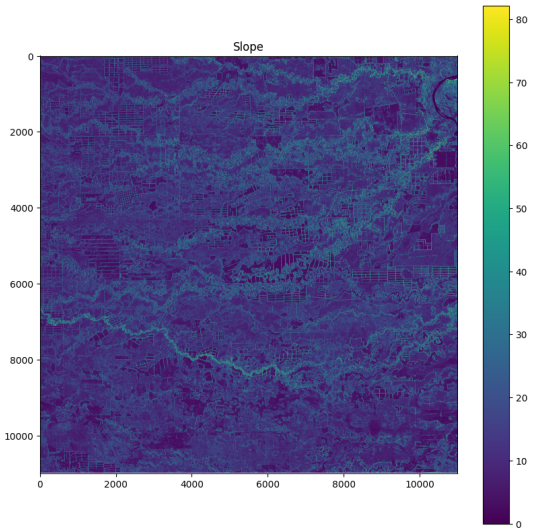


Fig. 7: Slope image derived from DEM in Figure 6.

The expression $(450 - [angle]) \mod 360$ adjusts the angle to ensure it aligns with compass directions

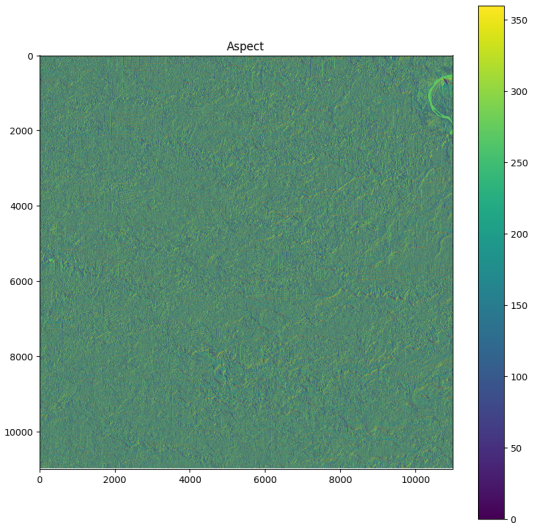


Fig. 8: Aspect image derived from DEM in Figure 6.

IV. Reimplemented Pipeline - Masks Refinement

The first component of the reimplemented pipeline processes Sentinel-2 L2A products to extract and refine Sen2cor cloud and shadow masks. This step corresponds to the **Cloud / Shadow Detection** component of the original pipeline, presented in Section II-C.

A. Parameters

This section details the parameters utilized in this component. The first group of parameters is used to manage I/O for the component.

- **Sentinel-2 products directory** (`input_path`): specifies the directory containing the Sentinel-2 L2A products as `.tif` files.
- **Output path for refined masks** (`output_path`): indicates the directory where the outputs generated by this component are stored.

The following parameters instead enable filtering of the input files by specific acquisition characteristics:

- **Sensor type** (`sensor`): filters input files by the specified Sentinel-2 sensor type.
- **Tile identifier** (`tile_id`): filters data by specific Sentinel-2 tile identifiers.
- **Acquisition year** (`year`): restricts data to those acquired in the specified year.

Finally, additional parameters regulate preprocessing and mask refinement operations:

- **Product resolutions to interpolate** (`resolutions`): specifies the resolutions of the Sentinel-2 products to be interpolated for processing (can be 10m, 20m and 60m).
- **Scene Classification Layer (SCL) mask values to isolate** (`mask_definitions`): defines which SCL values will be used to generate binary masks, assigning 1 for specified values and 0 otherwise.
- **Cloud coverage threshold** (`cloud_threshold`): specifies the upper limit of cloud coverage under which no refinement is performed for both cloud and shadow masks.
- **Image brightness threshold** (`brightness_threshold`): determines the minimum average brightness below which shadow mask refinement is not performed.

B. Data Loading and Preprocessing

The refinement process begins with a structured loading of Sentinel-2 data. This process leverages xarray and consists of the following steps.

- **File Filtering**: The initial phase expands input file paths specified in `input_path`, accommodating wildcard patterns. Files are then filtered based on `year`, `tile` and `sensor` to ensure that only needed data is included.
- **Dataset Loading**: The datasets are loaded and concatenated based on shared coordinates between the specified L2A products in input. Lazy evaluation ensures that data is processed only when required and not immediately.

The products are loaded from different resolutions based on the labels in `resolutions` parameter.

- **Dataset Filtering**: When the dataset is lazy loaded, non-essential auxiliary bands are excluded, retaining only the necessary spectral bands. The bands used for analysis are B02, B03, B04, B08 and B11, presented with the other available bands in Figure 9.

Name	Description	Resolution
B01	Coastal aerosol, 442.7 nm (S2A), 442.3 nm (S2B)	60m
B02	Blue, 492.4 nm (S2A), 492.1 nm (S2B)	10m
B03	Green, 559.8 nm (S2A), 559.0 nm (S2B)	10m
B04	Red, 664.6 nm (S2A), 665.0 nm (S2B)	10m
B05	Vegetation red edge, 704.1 nm (S2A), 703.8 nm (S2B)	20m
B06	Vegetation red edge, 740.5 nm (S2A), 739.1 nm (S2B)	20m
B07	Vegetation red edge, 782.8 nm (S2A), 779.7 nm (S2B)	20m
B08	NIR, 832.8 nm (S2A), 833.0 nm (S2B)	10m
B8A	Narrow NIR, 864.7 nm (S2A), 864.0 nm (S2B)	20m
B09	Water vapour, 945.1 nm (S2A), 943.2 nm (S2B)	60m
B11	SWIR, 1613.7 nm (S2A), 1610.4 nm (S2B)	20m
B12	SWIR, 2202.4 nm (S2A), 2185.7 nm (S2B)	20m

Fig. 9: Spectral bands of Sentinel-2 L2A product [2].

- **Masks from SCL**: The SCL mask of the L2A products is employed to derive binary masks for cloud and shadow coverage, for each image at each time step. If 10m resolution data is available, masks from lower resolutions (20m or 60m) are interpolated using nearest-neighbor methods to ensure spatial consistency. For each time step, binary masks are generated based on SCL values: 8, 9 and 10 for clouds (representing medium to high probability clouds and cirrus) and 3 for shadows (cloud shadows); these values are specified in `masks_definitions`. Masks are then concatenated along time and mask type dimensions.
  Figure 10 from [2] shows the legend and other information about the SCL mask of Sentinel-2 L2A products.

SCL	Scene classification mask (unitless)	DN	UINT8	0 - No data 1 - Saturated / Defective 2 - Dark Area Pixels 3 - Cloud Shadows 4 - Vegetation 5 - Bare Soils 6 - Water 7 - Clouds low probability / Unclassified 8 - Clouds medium probability 9 - Clouds high probability 10 - Cirrus 11 - Snow / Ice

Fig. 10: Scene Classification Layer details [2], used for refining cloud and shadow masks.

- **Dataset Consolidation**: The refined masks and band data are combined into a single xarray dataset structure as variables, sharing dimensions and coordinates and facilitating efficient processing in subsequent steps. An example of the final lazy-loaded dataset structure is shown in Figure 11. The dataset shown consists of five dimensions, with coordinates specifying values along these dimensions. Additional coordinates such as `file_name` and `season_id` are directly linked to the time dimension. Data variables contain the dataset content, which consists of Dask's arrays. Indexes facilitate efficient lookup and attributes store metadata.

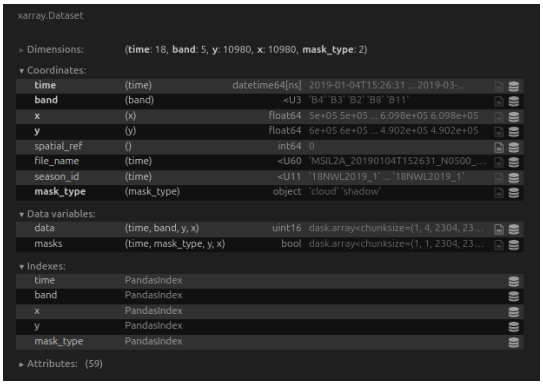


Fig. 11: Structure of the `xarray` dataset used in the masks refinement step.

C. Background Image

After the dataset is loaded, the next processing step consists of generating seasonal background images based on the input L2A products. These backgrounds are essential for refining Sen2cor cloud masks.

The computation of the background follows a logic identical to that presented in Paragraph II-C1 of the existing pipeline. However, the reimplementation leverages `xarray`.

Specifically, the new approach utilizes `xarray`'s `apply_ufunc` method, which maps an input custom function across dataset variables, enabling parallel computation for each data chunk. The advantage of this function is that it allows the use of the exact same Numpy procedure as the existing pipeline, reimplemented within that function. This avoids manual management of the data in patches since the chunking strategy implicitly used by `xarray` mimics the patch subdivision behaviour of the existing pipeline. Each chunk of data retains the entire time dimension while splitting the data spatially over the `x` and `y` dimensions. This means that multiple spatial chunks are processed independently, each contributing to the construction of a seasonal background.

The final output consists of a dataset containing a background image for each season, with each background assembled from the collective contributions of the processed chunks.

1) Percentile Computation Details: The percentile computation involves multiple steps, explained in the following.

- **Mask Application**: Each pixel in the image is evaluated against the cloud and shadow masks. Pixels flagged as cloud or shadow are masked by setting them to `NaN`. This ensures that only valid pixel values are used in the subsequent computation.
- **Data Reshaping**: The masked data is reshaped into a two-dimensional array where one axis represents the spatial dimension (flattened `y` and `x`), and the other represents the temporal dimension.
- **Group Processing**: The dataset identifies unique counts of valid observations for each pixel, allowing grouping based on data availability.

- **Percentile Calculation**: For each group, valid pixel values are identified and the percentile is computed using the Numpy `percentile` function with 'midpoint' as the interpolation method.
- **Data Reshaping**: The resulting percentile values are reshaped back to their original spatial configuration (`y` and `x`) to maintain their original shape.
- **Background Assembly**: The computed background for each season is related with appropriate spatial coordinates and season identifiers. These backgrounds are finally concatenated along the `season_id` dimension to form the final xarray dataset of backgrounds.

D. Cloud and Shadow Masks Refinement

The refinement of Sen2cor cloud and shadow masks is the final operation within the masks refinement component of the reimplemented pipeline. Cloud mask processing is conducted prior to shadow mask refinement, as the enhanced cloud masks are subsequently used in place of the original Sen2cor masks during shadow mask adjustment. The core logic in both refinement procedures replicates the approach from the original pipeline but has been adapted to leverage `xarray` for parallel and lazy data processing.

1) Refinement Workflow: The refinement process is structured into several stages, employing `xarray's apply_ufunc` for parallel computation. The workflow operates season by season, grouping datasets based on the `season_id` coordinate. Within each group, the following steps are performed:

- **Background Loading**: The corresponding background image is loaded into memory, resolving the Dask graph and computing the real values of the background. This approach is justified as the background image is reused multiple times to refine each mask within the current season and occupies a manageable amount of memory (approximately 500 MB).
- **Mask Refinement**: For each acquisition time within the current group (one image at a time), the Sen2cor cloud mask is refined first, immediately followed by the refinement of the Sen2cor shadow mask. Both refinement processes are executed using `apply_ufunc`, facilitating parallel processing across spatial chunks. As each image corresponds to a single time step, chunking is applied solely across spatial dimensions (`x` and `y`). The refinement functions used within `apply_ufunc` replicate the exact logic of the original pipeline presented in Section II-C, handling data internally as Numpy arrays.
- **Data Storage**: The refined masks are saved to disk using `rioxarray's to_raster` method, thereby triggering the computation and storage of the enhanced cloud and shadow masks.

This approach reimplements the existing pipeline's refinement logic while utilizing `xarray` and `dask` to try to enhance processing efficiency and scalability, while trying to ensure that the refinement process is consistent with the original methodology.

E. Performance Evaluation

This section presents results, method and external resources used to evaluate the masks produced by the current component and the same masks produced by the existing pipeline, presented in Section II-C.

1) CloudSEN12+: The evaluation method is based on the CloudSEN12+ dataset, which is a collection of over 50,000 image patches from Sentinel-2 acquisitions distributed across all continents except Antarctica. The dataset is divided into two main collections, **p509** and **p2000**. These numbers correspond to the image patch sizes of 509x509 and 2000x2000 pixels, respectively; both with a 10-meter spatial resolution [8]. Figure 12 represents the CloudSEN12+ dataset organization.

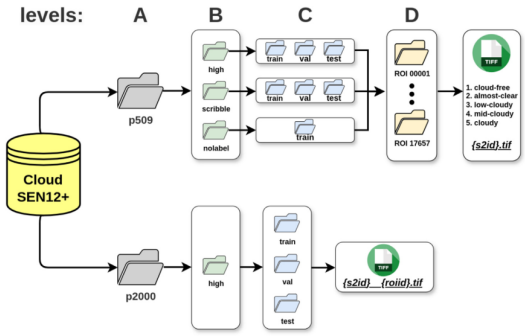


Fig. 12: Figure and caption from [8]. The CloudSEN12+ dataset is structured hierarchically, with the top level (A) dividing the dataset into two main categories: **p509** and **p2000** image patches, represented by gray folders. Moving to the next level (B), the images are further organized based on the label type, with each label type having a different folder. Within each label type, an additional level (C) groups the images based on a block of random data splitting, represented by blue folders. Moreover, within the p509 category, there is an additional division based on geographic location, highlighted by yellow folders (D). Each yellow folder contains a set of five distinct images with cloud cover ranging from 0% (cloud-free) to near 100% (cloudy).

Each patch includes multiple data layers, such as Sentinel-2 reflectance data, cloud and shadow masks and quality annotations. Importantly, CloudSEN12+ provides expert-labeled annotations for cloud and cloud shadow regions, which serve as ground truth for algorithm evaluation [8]. The labeling system adopted in CloudSEN12+ is represented in Table I.

In Figure 12, folders labelled as `high` indicate that each pixel within the images in that folder is associated with a cloud semantic category described in Table I. Folders with `scribble` label instead contain patches that cover only a small percentage of pixels with annotations (less than 5%). Finally, the folders with `nolabel` label do not provide annotations [8].

2) Data Used: For the purposes of this project, only patches in the **p2000** folder of Figure 12 are used. Since the scope was not the training of a model but the evaluation of already

TABLE I: Table adapted from [8]. Cloud semantic categories considered in CloudSEN12+.

Code	Class	Description
0	Clear	Pixels without cloud and cloud shadow contamination.
1	Thick Cloud	Opaque clouds that block all the reflected light from the Earth's surface.
2	Thin Cloud	Semitransparent clouds that alter the surface spectral signal but still allow recognition of the background.
3	Cloud Shadow	Dark pixels where light is occluded by thick or thin clouds.

available masks, data from the `train`, `test` and `val` folders are used without distinction.

The current evaluation leverages 2000x2000 patches from CloudSEN12+, aligning them to the area of the tile considered using the metadata available with each sample. Figure 13 shows the content of a sample with an example of annotated cloud and shadow masks used during the evaluation procedure.

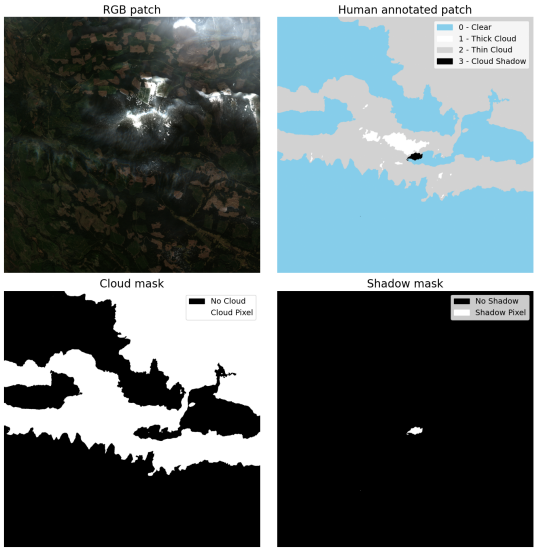


Fig. 13: This figure shows part of the content of a sample from CloudSEN12. The meaning of the colors is described in the legend of each image.

Specifically, 19 Sentinel-2 L2A products from tile 18NWL (17 GB, covering January to March) and 37 from tile 10UED (37 GB, covering October to December) are used for the evaluation of the time and resources used. For quality assessment, since CloudSEN12+ provides 4 patches for tile 10UED (dated 2019-10-06) and 4 for tile 18NWL (dated 2019-02-23), only the cloud and shadow masks dated 2019-10-06 (10UED) and 2019-02-23 (18NWL) from both the reimplemented and the existing pipeline are used for the mask quality assessment. Table II summarizes these selected specific masks.

3) Evaluation Method: The evaluation process consisted of the following steps, repeated for each patch of a specific tile available in CloudSEN12+.

TABLE II: Summary of masks used for evaluating `Masks refinement` component of the reimplemented pipeline

Tile	Mask Type	Date	Pipeline
T18NWL	Sen2Cor Cloud	2019-02-23	Existing
T18NWL	Sen2Cor Shadow	2019-02-23	Existing
T18NWL	Refined Cloud	2019-02-23	Existing
T18NWL	Refined Shadow	2019-02-23	Existing
T18NWL	Refined Cloud	2019-02-23	Reimplemented
T18NWL	Refined Shadow	2019-02-23	Reimplemented
T10UED	Sen2Cor Cloud	2019-10-06	Existing
T10UED	Sen2Cor Shadow	2019-10-06	Existing
T10UED	Refined Cloud	2019-10-06	Existing
T10UED	Refined Shadow	2019-10-06	Existing
T10UED	Refined Cloud	2019-10-06	Reimplemented
T10UED	Refined Shadow	2019-10-06	Reimplemented

- **Data Retrieval**: For each CloudSEN12+ patch, the corresponding data were downloaded. From these, the annotated cloud and shadow masks, each covering a 2000x2000 pixel region, were extracted. Binary ground truth masks were then generated by identifying the correct labels according to Table I, distinguishing between cloud (thick and thin) and shadow classes.
- **Mask Alignment and Cropping**: Using the metadata associated with each CloudSEN12+ patch (such as spatial bounds and coordinate reference system (CRS)), corresponding regions were extracted from the full 10980x10980 Sentinel-2 L2A masks. This included masks generated by:
  - Existing pipeline (refined cloud and shadow masks).
  - Reimplemented pipeline (refined cloud and shadow masks).
  - Original Sen2Cor masks (cloud and shadow).

  The extracted patches were spatially aligned with the ground truth annotations, ensuring that evaluations were performed over matching regions.
- **Flattening and Concatenation**: Each extracted patch and the corresponding masks were flattened into a one-dimensional array. These flattened arrays were then concatenated with the correspondant global array.

The above steps were repeated for each patch available in the dataset. The resulting global flattened arrays contained all values necessary for evaluation. These were then passed to the `classification_report` function from the `scikit-learn` library. This function provided the standard evaluation metrics (precision, recall, F1-score and support) for each class (cloud, no cloud, shadow and no shadow). Evaluations were conducted separately for cloud and shadow masks across both tiles (18NWL and 10UED) and for both the existing and reimplemented pipelines.

4) Quality of the Refined Masks: The evaluation procedure generates classification reports for each available cloud and shadow mask by comparing them against expert-labeled annotations from the CloudSEN12+ dataset. To enhance readability, the raw classification reports have been summarized in Tables III, IV, V and VI. These tables report the key evaluation metrics (precision, recall and F1-score) for both the existing

and reimplemented pipeline components, in comparison to the original Sen2Cor masks.

The reported values refer exclusively to the detection quality of the target class (cloud or shadow), and are computed over binary classification tasks. For a more complete view of the evaluation pipeline, including the handling of CloudSEN12+ alignment and patch extraction, refer to the notebook `evaluate_methods`.

TABLE III: Cloud Mask Evaluation Metrics for Tile 10UED

Pipeline	Mask Type	Precision	Recall	F1-Score
Existing	Sen2Cor Cloud	0.89	0.57	0.69
	Refined Cloud	0.43	0.61	0.51
Reimplemented	Sen2Cor Cloud	0.89	0.57	0.69
	Refined Cloud	0.58	0.61	0.60

TABLE IV: Shadow Mask Evaluation Metrics for Tile 10UED

Pipeline	Mask Type	Precision	Recall	F1-Score
Existing	Sen2Cor Shadow	0.00	0.00	0.00
	Refined Shadow	0.00	0.00	0.00
Reimplemented	Sen2Cor Shadow	0.00	0.00	0.00
	Refined Shadow	0.00	0.00	0.00

TABLE V: Cloud Mask Evaluation Metrics for Tile 18NWL

Pipeline	Mask Type	Precision	Recall	F1-Score
Existing	Sen2Cor Cloud	0.55	0.89	0.68
	Refined Cloud	0.51	0.90	0.65
Reimplemented	Sen2Cor Cloud	0.55	0.89	0.68
	Refined Cloud	0.55	0.89	0.68

TABLE VI: Shadow Mask Evaluation Metrics for Tile 18NWL

Pipeline	Mask Type	Precision	Recall	F1-Score
Existing	Sen2Cor Shadow	0.76	0.07	0.13
	Refined Shadow	0.76	0.07	0.13
Reimplemented	Sen2Cor Shadow	0.76	0.07	0.13
	Refined Shadow	0.72	0.08	0.14

The metrics highlight several trends regarding the refinement quality of both cloud and shadow masks:

- For tile **10UED**, the refined cloud masks from the reimplemented pipeline show improved precision and F1-score compared to those of the existing pipeline, while recall remains consistent.
- For tile **18NWL**, the refined cloud mask from the reimplemented pipeline shows almost no improvement compared to the original Sen2Cor mask. This differs from the behavior of the existing component, which demonstrates a modest improvement in recall.
- Shadow detection remains consistent with the existing pipeline. Across both tiles, the refined shadow masks yield poor performance in both pipelines, with nearly identical results to the original Sen2Cor masks. For tile 18NWL, the reimplemented pipeline shows a slight improvement in F1-score and recall, while precision is marginally lower compared to the existing pipeline.
- The ground truth data derived from Sen2Cor, used to evaluate the shadow masks in Table IV, shows that only 4587 shadow pixels are present across all available patches. This is a very small number compared to the total of 16 million pixels analyzed. As a result, both the Sen2Cor and the refined masks consistently fail to detect any of these pixels in either pipeline, leading to zero values across all metrics related to the shadow class. In particular, for tile 10UED, there are four patches, and only patch number 2 contains the 4587 annotated shadow pixels. This issue can be inspected in the `evaluate_methods` notebook, specifically in the output produced before the classification reports, where summary statistics for the ground truth, Sen2Cor and refined masks are provided.

These results suggest that despite preserving the original refinement logic, the reimplemented component exhibits slightly different behavior, especially in the cloud mask generation. The key factor contributing to this divergence is the difference in the seasonal background images created during refinement.

Figures 14 and 15 illustrate a comparison of the seasonal backgrounds, Sen2Cor and refined masks produced by both pipeline versions. These comparisons highlight visual differences in the background quality and the resulting segmentation of cloudy and shadowed areas. The percentage values below each mask image represent the pixel-wise cloud or shadow coverage.

The refinement strategy is particularly sensitive to the differences between the current image and the background image used during clustering. In the case of tile 18NWL, the background produced by the reimplemented pipeline is smoother and less affected by residual cloud patterns. This fact may also reduce the contrast necessary to distinguish cloud-contaminated areas, leading to a refined mask that fails to improve recall over the original Sen2Cor mask, since it doesn't detect other cloud pixels. Conversely, for tile 10UED, characterized by lower cloud coverage, the reimplemented component's background leads to better performance, reflected in higher precision and F1-score.

Overall, while the reimplementation preserves the original algorithmic logic, subtle differences introduced by the use of alternative libraries (such as `xarray` and `dask`) may influence the refinement results, preventing a perfect match with the existing pipeline.

5) Time and Resource Usage: This last section presents a comparative analysis of the time and resources used by the reimplemented and the existing pipeline for mask refinement. As described in Section IV-E2, the evaluation was conducted using Sentinel-2 L2A products from two different tiles: 10UED and 18NWL. Figures 16 and 17 provide the performance reports for both tiles.

Both the reimplemented and existing components were executed three times under identical configurations to ensure more reliable results. The evaluation reports include execution statistics such as average and maximum memory consumption, as well as total execution time. These statistics are available as overall averages for each pipeline version and as separate measurements for each individual execution.

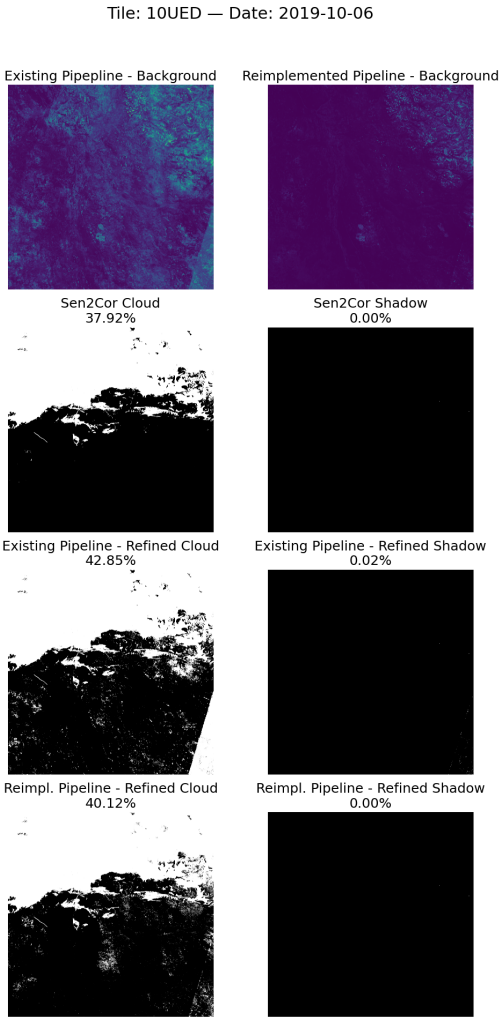


Fig. 14: Comparison of cloud and shadow masks for tile 10UED (2019-10-06). The first row shows the background images computed from the blue bands. The following rows display, in order: the Sen2Cor masks, the refined masks produced by the existing pipeline and the refined masks generated by the reimplemented pipeline. In all images except those in the first row, white pixels indicate the presence of clouds or shadows, while black pixels indicate their absence.

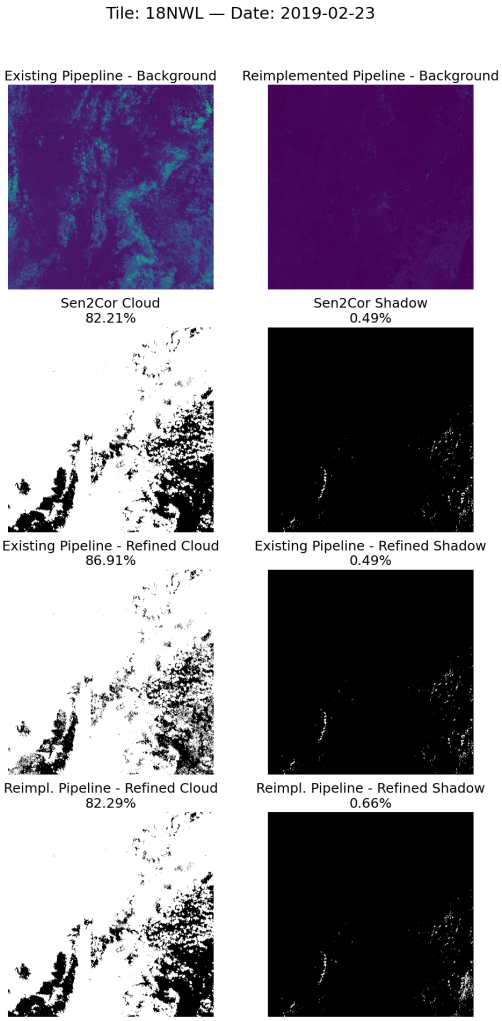


Fig. 15: Comparison of cloud and shadow masks for tile 18NWL (2019-02-23). The first row shows the background images computed from the blue bands. The following rows display, in order: the Sen2Cor masks, the refined masks produced by the existing pipeline and the refined masks generated by the reimplemented pipeline. In all images except those in the first row, white pixels indicate the presence of clouds or shadows, while black pixels indicate their absence.

The results indicate that for both tested tiles, the reimplemented component achieves a moderate reduction in execution time compared to the existing version. However, this improvement comes at the cost of increased memory usage.

- **Execution Time**: the reimplemented component consistently completed processing in slightly less time than the existing component. The difference in execution time is more pronounced for tile 18NWL, where the reimplemented version runs approximately 24% faster.
- **Memory Consumption**: the reimplemented pipeline consumes nearly twice the amount of memory as the existing component. This increase is expected due to the use of `xarray` and `dask` and also due to the removal of the `Image selection` step.

Overall, while the reimplemented version provides efficiency gains in execution time, its higher memory requirements should be taken into account when deploying the pipeline in environments with constrained resources.

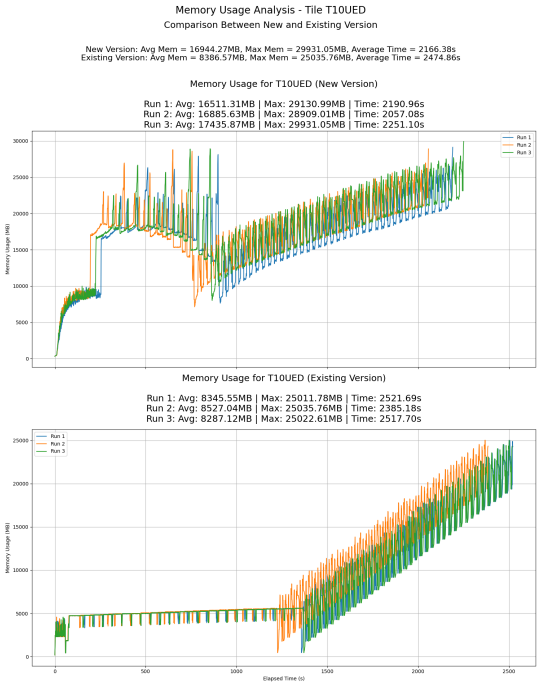


Fig. 16: Memory usage analysis of the masks refinement component for tile 10UED, comparing the reimplemented and existing pipeline versions. The report provides statistics on average and maximum memory consumption and execution time for three separate runs.

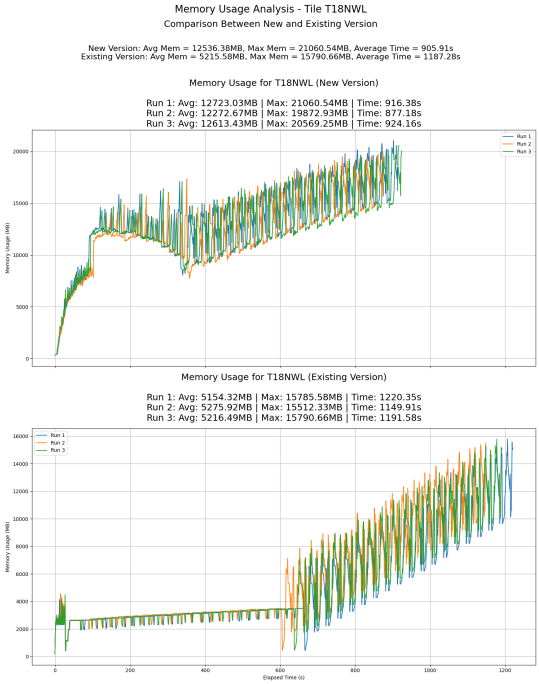


Fig. 17: Memory usage analysis of the masks refinement component for tile 18NWL, comparing the reimplemented and existing pipeline versions. The report provides statistics on average and maximum memory consumption and execution time for three separate runs.

V. Reimplemented Pipeline - Cloud Restoration

The cloud restoration component of the pipeline relies on externally computed composites. These composites represent tile 21KUQ and are obtained by aggregating available Sentinel-2 L2A products on a monthly basis. They are generated using the methods explained in Section II-D. The objective of this component is to process the selected composites to restore missing pixels, which may be absent due to cloud cover, shadows or data unavailability. In the composites, missing pixels are explicitly indicated with a zero value, simplifying their identification and facilitating the restoration process.

This component uses an external DEM along with the derived slope and aspect information, as detailed in Section III-B.

A. Parameters

The first group of parameters define the selection criteria for input data:

- **Composites year** (`composites_year`): filters the input composites to include only those corresponding to the specified year.
- **DEM year** (`dems_year`): selects the DEM corresponding to the specified year.
- **Tile identifier** (`tile_id`): ensures that only composites from the specified tile are processed.

The last group instead define the directories used for loading and saving data:

- **Composites directory** (`composites_path`): directory containing the input composites.
- **DEMs directory** (`dems_path`): directory containing the available DEMs used for topographic correction.
- **Output path for restored composites** (`output_path`): destination directory where the restored composites will be saved.

B. Composite Restoration

The composite restoration logic follows the same methodology described in Section II-E. It consists of two sequential operations:

1) **Cloud Restoration**: Interpolates missing values over the temporal dimension by considering the same pixel in adjacent time steps and filling gaps with the mean of these two observations.
2) **Topographic Shadow Correction**: Utilizes slope information derived from the DEM to correct shadows caused by terrain elevation variations.

While the logic remains unchanged from the existing pipeline, the reimplementation fully leverages `xarray`. The core steps are structured as follows:

*1) **Dataset loading**:* An `xarray` dataset of composites is lazily laoded and merged with the DEM related information. The resulting initial dataset is show in Figure 18 below.

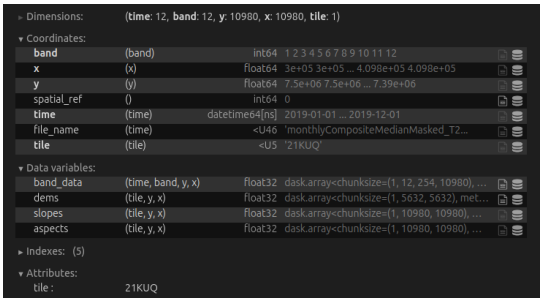


Fig. 18: Initial `xarray` dataset for cloud restoration operations.

*2) **Cloud Restoration**:* The cloud restoration operation is performed lazily on the `xarray` composites dataset, grouping the data by spectral band. Since the interpolation depends on the temporal dimension but not on the spatial dimensions (`x` and `y`), the dataset is chunked along the spatial dimensions while keeping the time dimension intact. The interpolation function is applied to each chunk using the `xarray.groupby.map()` method, ensuring independent computation for each spectral band. Finally, the restored cloud composite is concatenated as a new variable within the original dataset, sharing all existing dimensions.

*3) **Iterative Processing and Shadow Correction**:* Once the cloud restoration step is defined, the component enters a processing loop that iterates over each time step of the dataset. At the beginning of each iteration:

- The previously lazy computed cloud-restored composite is saved to disk, triggering its computation.
- This restored composite is then used as input for the topographic shadow correction.

Since the cloud-restored composite is now a fully computed `numpy ndarray`, the topographic shadow correction is applied using the same Numpy based logic as in the existing pipeline. This correction modifies the pixel values in shadowed regions based on the slope information from the DEM. After applying the topographic shadow correction, the final composite for the current time step is integrated with attributes from the original `xarray` dataset and saved to disk.

4) Summary of the Process: The workflow repeats for each composite in input directory, producing two output composites per time step:

- **Cloud-Restored Composite**: A version of the composite where missing values due to clouds are interpolated using temporal data.
- **Cloud and Shadow Restored Composite**: A final corrected composite where topographic shadows have also been mitigated using DEM information.

The images presented in Figure 19 were generated for tile `21KUQ` using the composite from `2019-05`, and track the progressive reduction of missing pixels. The figure shows the input composite with the corresponding mask of missing pixels (zeros), followed by the results after the cloud restoration and topographic shadow correction operations.

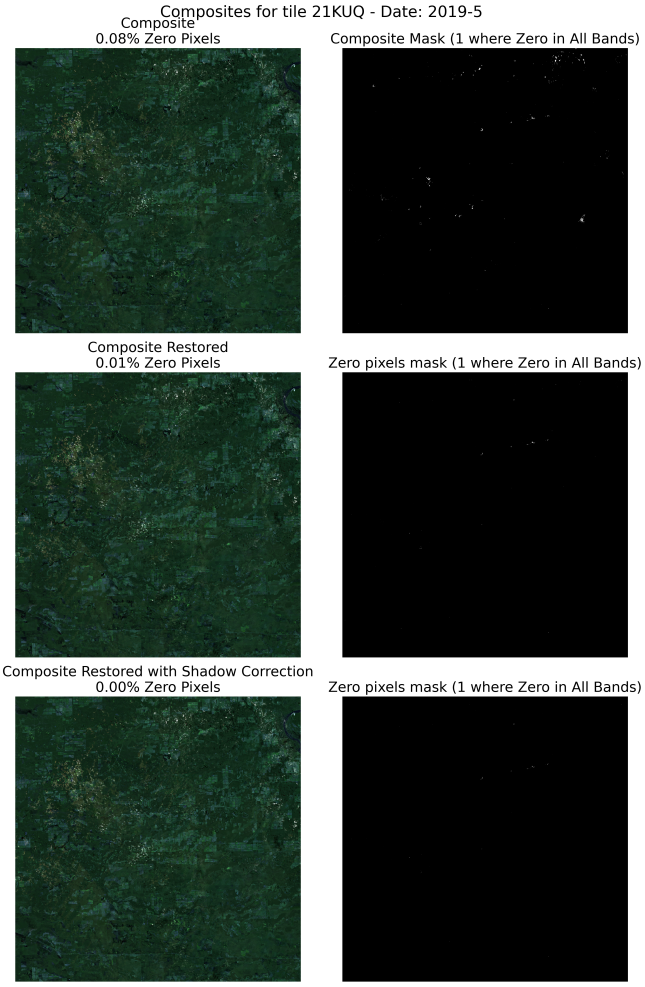


Fig. 19: Visual report of the cloud restoration process for tile 21KUQ (May 2019). Top row: original composite and zero-pixel mask; middle row: after temporal cloud restoration; bottom row: after topographic shadow correction. The percentage of remaining missing pixels is reported at each stage.

As illustrated in this example, the restoration process successfully reduces the percentage of missing pixels from `0.08%` in the original composite to `0.00%` after applying both cloud and shadow correction steps.

VI. Reimplemented Pipeline - Features Generation

The feature generation component processes the restored composites and DEM data to produce a set of features used in building the dataset for SVM training and evaluation. While it replicates the logic of the existing pipeline for GLCM feature computation (as detailed in Section II-F), it extends the original approach by introducing additional spectral and spatial indices to enrich the feature representation.

In the original pipeline, this step is not implemented as an explicit module, but rather embedded within the dataset construction and SVM evaluation phase. In the reimplemented pipeline, it has been modularized into a dedicated component to allow finer control over feature generation and to emphasize its importance and extensibility in the overall classification process.

A. Parameters

The first group of parameters allows specification of filtering values for the restored composites and DEM data.

- **Date used to generate features** (`features_date`): specifies the year-month combination used to select the relevant composite.
- **DEM year** (`dems_year`): indicates the year of the DEM to be used.
- **Tile identifier** (`tile_id`): indicates the tile of interest.

Then there are parameters related to the input and output destinations .

- **Composites directory** (`composites_path`): directory with input composites.
- **DEMs directory** (`dems_path`): directory with the available DEMs.
- **Output path for generated features** (`output_path`): directory in which the generated features will be stored.

The last group of parameters regulates the features computation process.

- **GLCM computation flag** (`compute_GLCM`): enables or disables GLCM feature computation. If disabled, only the newly added features are computed.
- **GLCM band selection** (`band_to_use`): band index used for GLCM computation. The red band (band 1) is used, consistent with the existing pipeline.
- **Window size** (`window_size`): set to 7, matching the original implementation.
- **Batch size** (`batch_size`): controls the size of parallelized computation blocks for the GLCM.
- **GLCM distance** (`glcm_distance`): set to 1 as in the existing pipeline.
- **GLCM angle** (`glcm_angle`): set to 0 as in the existing pipeline.

B. Generated Features

The initial feature set includes the GLCM texture features described in Section II-F, along with topographic indicators derived from DEM data: elevation (directly from the DEM), slope, and aspect (see Section III-B). Together, these account for a total of 9 base features—six GLCM texture features and three terrain-related descriptors.

All features, both base and additional, are computed from data corresponding to a specific month of the year, selected using the `features_date` parameter. Feature outputs are organized in subfolders named according to this date.

In addition to the base features, the reimplemented pipeline introduces a set of spectral and spatial indices. These newly added features are designed to capture land surface characteristics such as vegetation health, moisture content, soil exposure, urban development and water presence. Each index is computed using specific combinations of Sentinel-2 bands and supports the discrimination of the HRLC land cover classes shown in Figure 20 and listed again below for convenience.

HRLC CLASSES			
CODE	DESCRIPTION		
0	No data		
10	Tree cover evergreen broadleaf		
20	Tree cover evergreen needleleaf		
30	Tree cover deciduous broadleaf		
40	Tree cover deciduous needleleaf		
50	Shrub cover evergreen		
60	Shrub cover deciduous		
70	Grasslands		
80	Croplands		
90	Woody vegetation aquatic or regularly flooded		
100	Grassland vegetation aquatic or regularly flooded		
110	Lichens and mosses		
120	Bare areas		
130	Built-up		
140	Open water	141	Open water seasonal
		142	Open water permanent
150	Permanent snow and/or ice		

Fig. 20: Final high resolution HR Land Cover classification legend defined during the HRLC project activity [1].

All index formulas include a small constant ε, set to `1e-40`, to prevent division by zero during computation. It is also important to note that the band indices of the composites follow a custom enumeration based on their generation process, which differs from the standard Sentinel-2 L2A band identifiers. Table VII presents the correspondence between the standard Sentinel-2 band names and the composite band indices used in feature extraction.

A final note concerns the preprocessing of reflectance values from the composites. Prior to computing any index, each band is scaled by dividing its values by 10000 and then clipped to the (0,1) range. This normalization ensures that reflectance values remain within physically meaningful bounds.

The following list presents the complete set of spectral and spatial indices supported by the reimplemented pipeline.

Sentinel-2 Band ID	Description	Composite Band Index
B01	Coastal aerosol	11
B02	Blue	3
B03	Green	2
B04	Red	1
B05	Red-edge (704.5 nm)	5
B06	Red-edge (740.5 nm)	6
B07	Red-edge (783.0 nm)	7
B08	Near-Infrared (NIR)	4
B8A	Narrow NIR	8
B09	Water vapour	12
B11	Short-Wave Infrared 1 (SWIR1, 1613.7 nm)	9
B12	Short-Wave Infrared 2 (SWIR2, 2202.4 nm)	10

TABLE VII: Mapping of Sentinel-2 band IDs to composite band indices.

- **NDVI (Normalized Difference Vegetation Index)**: This index quantifies vegetation greenness using the red and near-infrared reflectance. It is effective in identifying healthy vegetation and distinguishing it from bare or built-up surfaces. In the context of HRLC classes, NDVI is particularly useful for detecting *tree cover*, *shrub cover*, *grasslands* and *croplands* [9].
  Bands used: Red (B04, band 1), NIR (B08, band 4)
  Formula:
  $$\text{NDVI} = \frac{NIR - Red}{NIR + Red + \varepsilon}$$
- **GNDVI (Green Normalized Difference Vegetation Index)**: A chlorophyll-sensitive variant of NDVI, GNDVI replaces the red band with green to better capture subtle changes in plant health and stress. It is suited for distinguishing early stress in *croplands*, *grasslands* and other herbaceous vegetation classes [10].
  Bands used: Green (B03, band 2), NIR (B08, band 4)
  Formula:
  $$\text{GNDVI} = \frac{NIR - Green}{NIR + Green + \varepsilon}$$
- **NDVI705 (Red-edge NDVI)**: NDVI705 enhances chlorophyll sensitivity by using the red-edge instead of red reflectance, avoiding saturation in dense vegetation. It helps discriminate between *woody vegetation*, *forest types* and other photosynthetically active land covers like *grasslands* or *croplands* [11].
  Bands used: Red-edge (B05, band 5), NIR (B08, band 4)
  Formula:
  $$\text{NDVI705} = \frac{NIR - Red\_Edge}{NIR + Red\_Edge + \varepsilon}$$
- **NDYI (Normalized Difference Yellow Index)**: NDYI highlights yellowing vegetation, which is often a sign of stress. This makes it valuable for detecting *grasslands* and *croplands*, and for distinguishing degraded herbaceous vegetation from bare or non-vegetated areas [12].
  Bands used: Green (B03, band 2), Blue (B02, band 3)
  Formula:
  $$\text{NDYI} = \frac{Green - Blue}{Green + Blue + \varepsilon}$$
- **EVI2 (Enhanced Vegetation Index 2)**: EVI2 is a simplified version of the traditional EVI, designed to enhance vegetation monitoring in areas with high biomass. Unlike EVI, it does not rely on the blue band, making it more stable under atmospheric disturbances. EVI2 is especially effective in detecting *croplands*, *shrubs*, and *dense forest* where NDVI may saturate [13].
  Bands used: Red (B04, band 1), NIR (B08, band 4)
  Formula:
  $$\text{EVI2} = 2.5 \cdot \frac{NIR - Red}{NIR + Red + 1 + \varepsilon}$$
- **GLI (Green Leaf Index)**: GLI emphasizes green reflectance and suppresses red and blue reflectance to enhance detection of actively photosynthesizing vegetation. It is particularly effective in highlighting *grasslands*, *tree cover* and *shrub cover*, especially in areas where vegetation dominates over soil or built-up backgrounds [14].
  Bands used: Green (B03, band 2), Red (B04, band 1), Blue (B02, band 3)
  Formula:
  $$\text{GLI} = \frac{2 \cdot Green - Red - Blue}{2 \cdot Green + Red + Blue + \varepsilon}$$
- **SAVI (Soil-Adjusted Vegetation Index)**: SAVI is a modification of NDVI that minimizes the influence of soil brightness in areas with sparse vegetation. It is particularly effective in distinguishing *bare areas*, *grasslands*, and *transitional zones* where soil exposure can distort typical vegetation indices [15].
  Bands used: Red (B04, band 1), Near-Infrared (B08, band 4)
  Formula:
  $$\text{SAVI} = \left(\frac{NIR - Red}{NIR + Red + L + \varepsilon} \right) \cdot (1 + L)$$
  where $L = 0.5$ is the soil brightness correction factor.
- **NDMI (Normalized Difference Moisture Index)**: NDMI is designed to monitor vegetation water content and detect drought stress. It distinguishes vegetation types based on moisture availability, making it relevant for classifying *woody vegetation*, *croplands* and *grasslands* [16].
  Bands used: Near-Infrared (B08, band 4), Short-Wave Infrared 1 (SWIR1, B11, band 9)
  Formula:
  $$\text{NDMI} = \frac{NIR - SWIR1}{NIR + SWIR1 + \varepsilon}$$
- **NDLI (Normalized Difference Lignin Index)**: NDLI helps highlight areas with higher lignin content, which is typical of woody vegetation like trees and shrubs. It uses the absorption behavior of short-wave infrared bands to

separate *woody vegetation*, *shrubs* and *forest types* from other land cover classes [17].
Bands used: SWIR1 (B11, band 9), SWIR2 (B12, band 10)
Formula:

$$NDLI = \frac{\log(1/\text{SWIR2}) - \log(1/\text{SWIR1})}{\log(1/\text{SWIR2}) + \log(1/\text{SWIR1}) + \varepsilon}$$

- **NDWI (Normalized Difference Water Index)**: NDWI is used to identify *open water* bodies such as rivers, lakes and wetlands. It is based on green and near-infrared reflectance, where water typically absorbs NIR and reflects green, allowing clear distinction from vegetation and soil. This index should help detect water-related surfaces in the HRLC classification [18].
  Bands used: Green (B03, band 2), NIR (B08, band 4)
  Formula:

$$\text{NDWI} = \frac{Green - NIR}{Green + NIR + \varepsilon}$$

- **NDBI (Normalized Difference Built-up Index)**: NDBI is designed to detect *urban* and *built-up areas*, which usually have higher reflectance in SWIR and lower in NIR. It complements vegetation and water indices by focusing on human-made surfaces [19].
  Bands used: SWIR1 (B11, band 9), NIR (B08, band 4)
  Formula:

$$NDBI = \frac{SWIR1 - NIR}{SWIR1 + NIR + \varepsilon}$$

- **NDSI (Normalized Difference Snow Index)**: NDSI is used to detect *snow* and *ice*, helping to distinguish them from clouds or bright soil. It uses green and SWIR1 reflectance to exploit the high reflectance of snow in the visible spectrum and its strong absorption in the short-wave infrared [20].
  Bands used: Green (B03, band 2), SWIR1 (B11, band 9)
  Formula:

$$\text{NDSI} = \frac{Green - SWIR1}{Green + SWIR1 + \varepsilon}$$

- **BSI (Bare Soil Index)**: BSI is used to detect *bare soil* by combining bands that respond to soil and vegetation differently. It helps distinguish bare land from vegetation and built-up surfaces [21].
  Bands used: SWIR1 (B11, band 9), Red (B04, band 1), NIR (B08, band 4), Blue (B02, band 3)
  Formula:

$$BSI = \frac{(SWIR1 + Red) - (NIR + Blue)}{(SWIR1 + Red) + (NIR + Blue) + \varepsilon}$$

- **Sobel Edge**: This filter is applied to the NDVI layer to emphasize sudden changes in vegetation, which are often found at field edges or roads. It highlights *landscape boundaries* that might be important for class separation [22].
  Input: NDVI layer
  Method: Gradient magnitude computed using Sobel operator on NDVI values

These features were selected to provide diverse and complementary information that could improves class separability across land cover types.

In the next page, in Figure 21, is presented the report generated with the GLCM features and all the other presented above, computed using the data of January 2019.

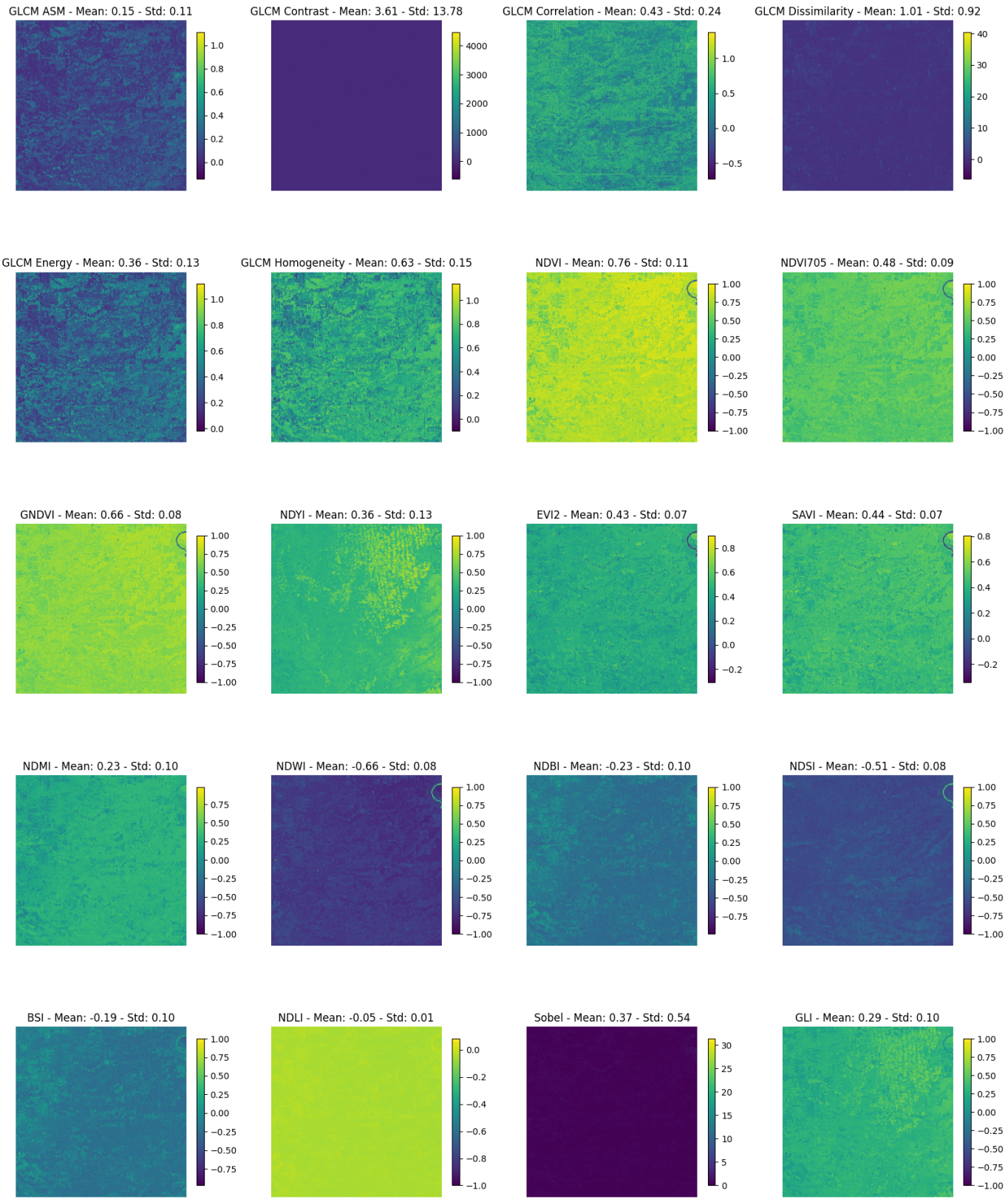


Fig. 21: A report with all the new 2D generated features. Above each image are indicated the name of the feature and its mean and standard deviation.

VII. Reimplemented Pipeline - Dataset Generation

The dataset generation component in the reimplemented pipeline is derived from the dataset preparation logic originally embedded within the `GLCM computation and SVM training` step, described in Section II-F. This new module provides a reproducible and configurable way to generate annotated datasets for land cover classification using Sentinel-2 composites and other features.

More specifically, the generation process uses restored Sentinel-2 composites (see Section V), topographic features such as elevation, slope and aspect (see Section III-B) and spectral and spatial features computed in the preceding processing step (see Section VI).

These features are then associated with the point from one of the two provided shapefiles:

- `amazonia_static_area_2019_photointerpreted_UniTN`
- `amazonia_static_ER10400-10439_2019_merged_UniTN-PoliMi`

Each file contains georeferenced ground truth points annotated with land cover class labels and serves as the primary reference for supervised training and validation. Optionally, additional points can be sampled from a pre-existing LC map generated by the original pipeline.

A. Parameters

The following parameters are used to configure this component. The first group of parameters are used for filtering inputs.

- **Composites year** (`composites_year`): Filter input composites accordingly to the specified year.
- **DEM year** (`dems_year`): year for filtering DEM products.
- **Features folder (date)** `features_date`: folder name of the features computed in the previous pipeline step (see Section VI).
- **Tile identifier** (`tile_id`): Filter products for the specified tile.

The following parameters contain filesystem paths to required input sources:

- **Composites directory** (`composites_path`): directory containing the composite images.
- **DEMs directory** (`dems_path`): folder with DEMs.
- **Features directory** (`features_path`): path to the directory containing all features folders generated with previous step.
- **Keyhole Markup Language (KML) file path** (`kml_path`): file used to match tiles to spatial extents.
- **Labels file path** (`labels_path`): `.csv` file containing class label mappings.
- **Shapefile with annotated points path** (`points_dataset_path`): path to shapefile containing annotated ground truth points.
- **Existing pipeline land cover map path** (`lc_map_path`): old pipeline's LC classification map path.
- **Output directory for dataset** (`output_path`): output location for the final dataset.
- **Output directory for reports** (`report_path`): directory where diagnostics info will be stored.

The last parameters are used to control dataset generation strategy.

- **Dataset type** `dataset_type`: one of `std`, `enhanced` or `fullLC`, defining the strategy for point selection.
- **Samples per class** `samples_per_class`: maximum number of samples per class (only for `fullLC` mode).
- **Use erosion operation** `apply_erosion`: if `True`, applies erosion to LC map before selecting training points (used in `enhanced` and `fullLC`).
- **Enhancement type** `enhanced_type`: used only in `enhanced` mode, sets how many LC map points per class to include (`avg` or `max`).

B. Overview

This section introduces the general workflow of the dataset generation component. The following sections will then provide a more detailed breakdown of each element involved in the process, including the specific operations that can be applied and the different types of datasets produced, along with their respective class distributions.

From a high-level perspective, the process is divided into two main phases:

- **Phase 1 – Sample collection and index mapping:**
  - Annotated points are read from shapefiles and, if needed, extracted from an LC map (`lc_map_path`).
  - Composites and other features data are lazily read into a single `xarray.Dataset`.
  - LC map samples are filtered by class, and if requested (through `apply_erosion`), a morphological erosion operation is applied to remove border pixels and improve sample purity.
  - Coordinates (latitude, longitude) of all points are converted into pixel indices that correspond to feature dataset `x` and `y` dimensions. This transformation ensures efficient feature extraction in subsequent phases.
- **Phase 2 – Feature extraction and dataset construction:**
  - At each pixel index, feature values are extracted and associated with the corresponding class label.
  - The resulting dataset is saved as a `.csv` file. Diagnostic plots and interactive HTML maps are also generated to visualize the spatial distribution of points and class labels. For shapefile-based samples, a consistency check is performed to verify the accuracy of pixel-to-geocoordinate conversion, generating reports that can be inspected.

`GeoPandas` is employed for spatial manipulation and coordinate handling of points data, while `xarray` is used to efficiently load and manage the multidimensional feature datasets.

C. KML File and Tile Geometry

This section and the next introduce essential building blocks of the dataset generation pipeline, following a bottom-up approach. The current section focuses on tile geometry handling through the use of a `.kml` file, while the next details the erosion operation used to refine LC map sampling. A subsequent section will describe how these components are used across the different dataset generation strategies.

The employed KML file encodes the Sentinel-2 tiling system defined by the Military Grid Reference System (MGRS). The file was obtained from [23] and contains the full set of Sentinel-2 grid tiles with related geometry. Figure 22 shows a view of the grid as provided by ESA.

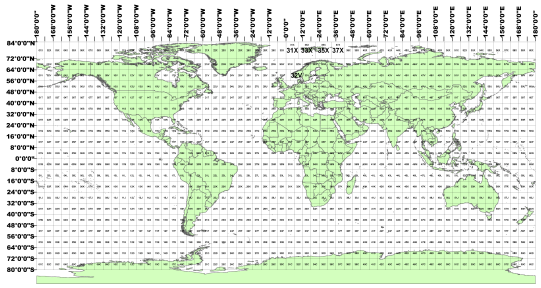


Fig. 22: Figure from [23]. The Military Grid Reference System showing the 6° × 8° grid cells used by Sentinel-2.

During execution, the file is read as a `GeoPandas GeoDataFrame`. The geometry of the tile specified by `tile_id` is extracted and used to spatially filter the annotated points from the input shapefiles. This ensures that only points located within the current tile are used in the dataset generation process.

D. Erosion Operation on LC Map

When using the LC map as a source of training points, an optional morphological erosion operation can be applied to reduce the influence of class boundary uncertainty and mislabeling. This preprocessing step is enabled by setting `apply_erosion` to `True` and is supported by both the `enhanced` and `fullLC` dataset types.

The erosion is applied separately for each land cover class. For a given class, a binary mask is generated from the LC map. This mask is then eroded using a (2,2) elliptical element. The resulting eroded masks identify the pixels of each class that can be chosen for the dataset.

E. Coordinate Transformation and Index Mapping

Before extracting feature values from datasets at point locations, all spatial data must share a common coordinate reference system (CRS). In this pipeline, all features and the LC map are already georeferenced using the UTM Zone

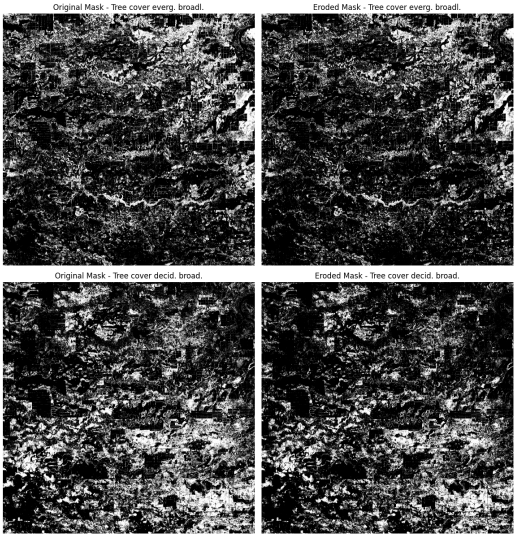


Fig. 23: Slice of the erosion report for the operation applied to masks derived from the 2019 LC map for tile 21KUQ. On the left, the original binary masks extracted per class; on the right, the corresponding eroded versions. White pixels represent the areas belonging to the target class, while black pixels indicate areas that do not, for this example the target class is `Tree cover evergreen broadleaf`.

21S projection (EPSG:32721). To ensure alignment, the input shapefiles, which may use a different CRS, are reprojected to this common system using GeoPandas' `to_crs()` method. No CRS transformation is applied to the features datasets.

Once the spatial alignment is ensured, each point's geometry is converted to pixel indices corresponding to the features dataset images grid. These indices are used for efficient access to feature values via `xarray` operations.

a) Pixel Index Mapping via Affine Transform:: Given the aligned CRS, each point's projected coordinates (x, y) are mapped to 2D indices (i, j) using a simplified affine transformation.

Let:

- $x_{\text{origin}}, y_{\text{origin}}$: coordinates of the upper-left corner of the image, derived from the dataset's `GeoTransform` metadata or from the first elements of the `xarray x` and `y` coordinates.
- $\Delta x, \Delta y$: pixel spacing (10 meters for Sentinel-2). These values are computed as differences between consecutive coordinate values: $\Delta x = |x_1 - x_0|$, $\Delta y = |y_1 - y_0|$.
- x, y: UTM-projected coordinates of the input point.

Then, the pixel indices are computed as:

$$i = \frac{x - x_{\text{origin}}}{\Delta x}, \qquad j = \frac{y_{\text{origin}} - y}{\Delta y}$$

The values are rounded to the nearest integer to determine the closest indexes. This index pair (i, j) enables direct feature extraction using `xarray.isel()`, allowing fast lookup of values across all bands and layers.

b) Conversion Accuracy Verification:: To validate the pixel index mapping strategy, the inverse affine transformation was used to reconstruct projected coordinates from the computed pixel indices. Specifically, for each annotated point, its corresponding pixel indices (i, j) were used to recompute the original UTM coordinates using the affine parameters extracted from the dataset's `GeoTransform` metadata:

$$x = x_{\text{origin}} + (i + 0.5) \cdot \Delta x, \qquad y = y_{\text{origin}} - (j + 0.5) \cdot \Delta y$$

This reverse operation centers the point within each pixel by adding 0.5 to the indices.

The recalculated coordinates were then compared to the original geometries from the input shapefile.

Figure 24 and Table VIII summarize the accuracy. The histogram shows that most points differ by less than 1 meter, with a few reaching up to 5 meters. The table reports the average and extreme differences along both axes, confirming pixel level minimal deviations.

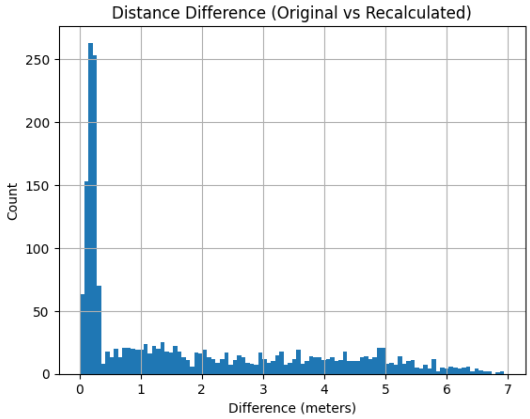


Fig. 24: Histogram of distances (in meters) between original and recalculated coordinates.

Metric	Value (meters)
Mean X difference	0.045
Min / Max X difference	–4.99 / 4.99
Mean Y difference	–0.118
Min / Max Y difference	–4.99 / 4.99

TABLE VIII: Coordinate difference statistics for UniTN dataset (standard mode).

Additionally, Figure 25 provides a static visual inspection by overlaying the original and recalculated points on the same spatial extent. An interactive version of this figure, generated using Plotly and exported in HTML format, is also generated by the component to allow dynamic exploration of the point alignment.

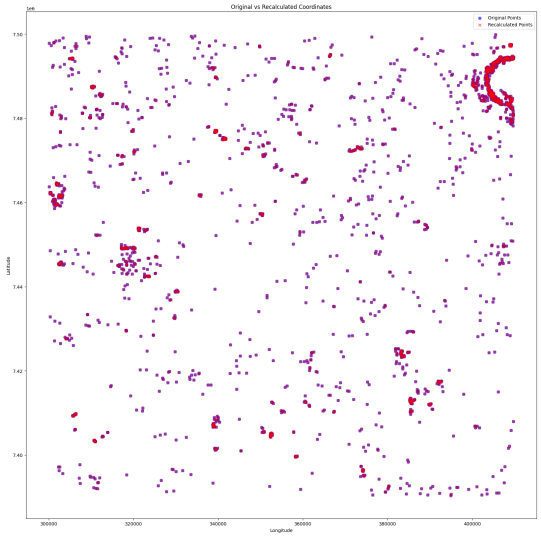


Fig. 25: Overlay of original shapefile coordinates (blue) and recalculated coordinates (red) after pixel index mapping.

The coordinate verification results confirm the reliability of the pixel index mapping process. With average differences well below 0.15 meters for both latitude and longitude, and maximum deviations not exceeding 5 meters, the transformation accuracy remains within the 10-meter spatial resolution of Sentinel-2 imagery. This precision ensures that feature values are extracted from the correct spatial locations, supporting the validity of the dataset construction strategy.

F. Dataset Types

This section describes the dataset generation strategies supported by the reimplemented pipeline. These are configured using the `dataset_type` parameter and differ in how training points are selected and balanced, depending on whether they originate from shapefiles or from a LC map. The three available strategies are described below.

1) Standard Dataset: When `dataset_type` is set to `std`, the dataset is built exclusively from ground truth points provided in the annotated shapefiles. These points are filtered spatially using the tile geometry extracted from the KML file and then matched to the corresponding feature values using pixel-based indexing.

Figures 26 and 27 show the class distributions obtained using the two available shapefiles. As can be seen, the datasets suffer from class imbalance.

To address these imbalances, two additional dataset types are introduced.

2) Enhanced Dataset: The enhanced dataset augments the shapefile training points with additional samples selected from the LC map generated by the original pipeline. This allows rebalancing of classes while preserving the spatial reference of known labeled points. This mode is activated using `dataset_type = enhanced`, and the number of LC derived samples is controlled by the `enhanced_type` parameter, which supports two modes:

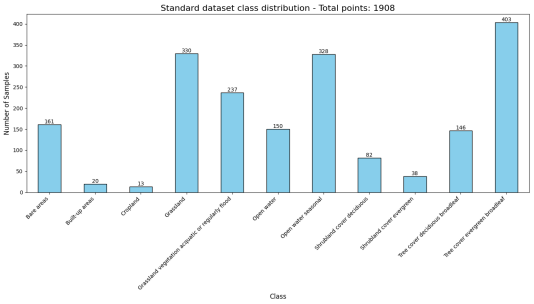


Fig. 26: Class distribution for standard dataset from UniTN shapefile.

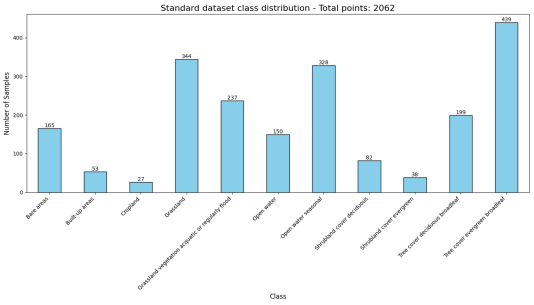


Fig. 27: Class distribution for standard dataset from merged UniTN-PoliMi shapefile.

- **Average enhancement:** The average number of samples across all classes in the shapefile is calculated. Classes with fewer samples than this average are filled using LC-derived samples.
- **Maximum enhancement:** The largest class size is used as a target, and all other classes are augmented up to this number using LC-derived samples.

The LC derived points are selected through a systematic sampling strategy. For each class, the corresponding binary mask (eroded if specified, see Section VII-D) is scanned to identify valid candidate pixels. Points already taken by shapefiles are excluded from selection and the sampling is performed by uniformly selecting one pixel every k pixels, where k is calculated as:

$$k = \left\lfloor \frac{\text{Total Available Points}}{\text{Requested Samples}} \right\rfloor$$

If a class has fewer available pixels than the required number, all valid pixels are used.

Figures below show the class distributions after applying average and maximum enhancement on the two shapefiles:

3) FullLC Dataset: This mode builds the training set using only samples derived from the LC map, fully excluding shapefile points from the training process. But the validation and test splits rely exclusively on the ground truth shapefile annotations.

This strategy is enabled using `dataset_type = fullLC`, and the number of samples per class can be controlled using the `samples_per_class` parameter.

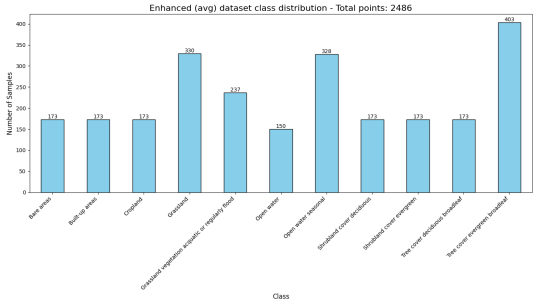


Fig. 28: Enhanced (average) dataset from UniTN shapefile (tile 21KUQ).

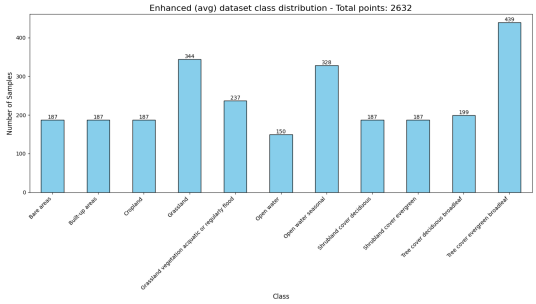


Fig. 29: Enhanced (average) dataset from merged UniTN-PoliMi shapefile (tile 21KUQ).

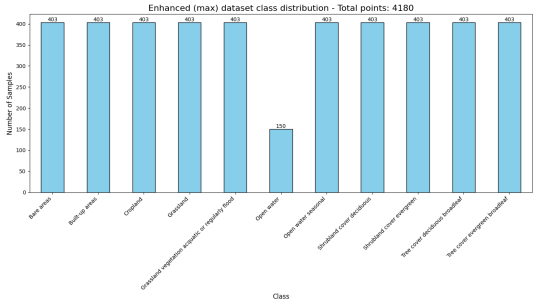


Fig. 30: Enhanced (maximum) dataset from UniTN shapefile (tile 21KUQ).

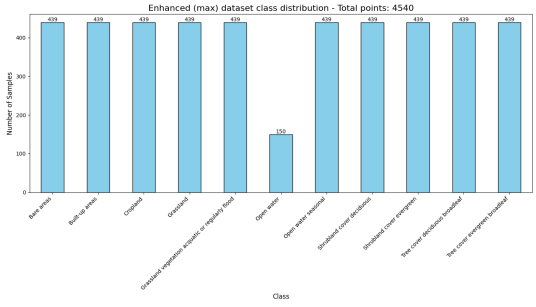


Fig. 31: Enhanced (maximum) dataset from merged UniTN-PoliMi shapefile (tile 21KUQ).

The training samples are selected using the same systematic sampling procedure described above, with exclusion of test/validation points which are directly taken from shapefile. The final dataset includes a `split` column with values `train`, `val`, or `test`, used by the next component of the pipeline.

Figure 32 and 33 illustrates the class distribution of the training and test/calibration sets in this mode.

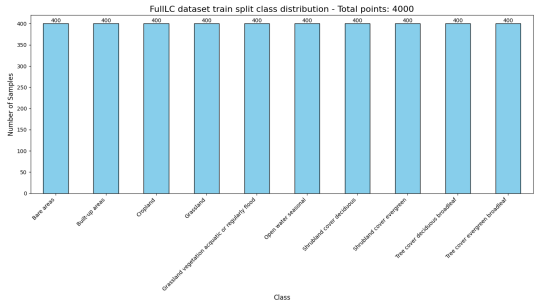


Fig. 32: Training set class distribution for FullLC dataset (tile 21KUQ).

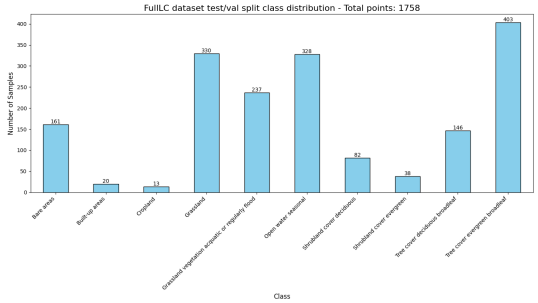


Fig. 33: Test/Val set class distribution for FullLC dataset (tile 21KUQ).

G. Imported Datasets

The reimplemented pipeline also supports the direct use of datasets previously generated by the existing pipeline. This is managed through two additional parameters not mentioned earlier: `convert_sav_dataset` and `sav_dataset_path`. If this mechanism is enabled, no strictly new dataset is generated.

When `convert_sav_dataset` is set to `True`, the pipeline will load a precomputed dataset stored in `.sav` format from the location specified by `sav_dataset_path`. This dataset, along with the corresponding labels, is then automatically converted to a `.csv` file that is compatible with the reimplemented pipeline. The conversion includes renaming and structuring the features, as well as mapping the ground truth indices to land cover codes and descriptive labels, allowing integration with the subsequent SVM training components.

VIII. Reimplemented Pipeline – SVM Training and Evaluation

This section describes the pipeline used to train and evaluate SVM classifiers for land cover classification, using the datasets generated in the previous component. The pipeline is designed to allow the configuration of different strategies for preprocessing, feature selection and model training.

The overall workflow can be summarized as follows:

- **Component initialization:** Load the dataset and prepare the experiment folder for storing models, plots and reports.
- **Dataset Splitting:** Split the input dataset into three subsets: training, test and calibration. The strategy differs slightly depending on the dataset type (standard or fullLC).
- **Feature Preprocessing:** Normalize the features using either `StandardScaler` or `MinMaxScaler` from `scikit-learn`.
- **Feature Selection (Optional):** select a subset of features using multicollinearity analysis or Recursive Feature Elimination with Cross-Validation (`RFECV`). A third option (`Null` strategy) disables this step, retaining all features as-is.
- **Feature Reduction (Optional):** If enabled, apply PCA to reduce dimensionality while retaining most of the variance.
- **Multiclass SVM Training:** Train a single multiclass SVM model using `sklearn`'s `SVC` in a One-vs-Rest (OvR) configuration, internally managed by `sklearn`.
- **Binary SVM Training:** For each class, train a binary SVM by treating that class as positive and all others as negative. In this case the OvR approach is manually implemented.
- **Model Packaging and Integration:** The trained classifiers are stored within a custom Python class that wraps the models and exposes a unified interface compatible with `scikit-learn`'s `predict()` and `predict_proba()` methods. This class simplifies the usage of all SVM models during inference and also for other tasks.
- **Calibration Curve Generation:** Probability calibration curves are generated to evaluate the reliability of the predicted probabilities. These plots compare predicted confidence scores against true observed frequencies, helping to assess model calibration and over/under confidence in classification.

During the pipeline execution, a variety of diagnostic reports and plots are produced to facilitate inspection and interpretation of the training process and model behavior.

This pipeline can also be executed in a grid search configuration, in which the entire training workflow is repeated for different combinations of parameters. The next section describes the parameters that control each part of this procedure, followed by a step-by-step explanation of each phase and mechanism in detail.

A. Parameters

This section outlines all parameters used to control this component behavior. These include input/output paths, preprocessing options and feature selection settings.

The following parameters are used to define paths for the dataset, labels and output directories:

- **Dataset path** (`dataset_path`): Path to the `.csv` dataset generated by the previous pipeline component.
- **Labels .csv path** (`labels_path`): Path to the `.csv` file containing the mapping between class IDs and their corresponding labels.
- **Output folder path** (`output_path`): Directory where all outputs such as models and charts will be stored.

The next group of parameters is used to configure the preprocessing of the feature values:

- **Non features columns** (`non_features_columns`): List of columns that should be excluded from the feature set (e.g., `x`, `y`, `ground_truth_index`, `ground_truth_label`, `split`).
- **Preprocessing type for band features** (`band_features_preprocessing`): Method used to normalize the spectral band features. Options are `standard` or `minmax`.
- **Preprocessing type for non-band features** (`other_features_preprocessing`): Method used to normalize all other features (non-band). Options are `standard` or `minmax`.
- **Features excluded from preprocessing** (`excluded_from_preprocessing`): List of specific feature names to exclude from preprocessing.

These parameters define which groups of features are included and how they are selected:

- **Use band features** (`use_band_features`): If `True`, the features derived from composites spectral bands are included.
- **Use GLCM features** (`use_glcm_features`): If `True`, GLCM texture features are included.
- **Use other features** (`use_other_features`): If `True`, additional features such as spectral indices and terrain information are included.
- **Fixed features** (`fixed_features`): List of specific features to always include, regardless of the group flags.
- **Drop constant features** (`drop_constant_features`): If `True`, features with constant values are automatically removed.

The last group of parameters configures feature selection and dimensionality reduction:

- **Features selection strategy** (`features_selection_strategy`): Strategy used to select features. Options are `RFECV`, `multicollinearity_analysis` or `None`.
- **Use bands in features selection** (`fss_use_bands`): Indicates whether the band features should be included in the selection process.
- **Multicollinearity Analysis correlation threshold** (`ma_correlation_threshold`): Correlation threshold used in the multicollinearity analysis. Features above this threshold became candidates for elimination.
- **Apply PCA** (`apply_pca`): If `True`, applies PCA after feature selection to reduce dimensionality.
- **PCA variance threshold** (`pca_variance_threshold`): this threshold influences the number of components kept by PCA analysis.
- **Use bands in PCA** (`pca_use_bands`): similarly to `fss_use_bands` this parameter specifies if bands must be involved in PCA analysis.
- **PCA components selection strategy**

B. Component Initialization and Dataset Splitting

The pipeline starts by setting up the environment and loading the input dataset. A new output directory is created for each run to store models, visualizations and evaluation results. This initialization also involves extracting metadata such as the dataset type (e.g., `std`, `enhanced`, `fullLC`) and setting up internal paths used in subsequent stages.

After initialization, the dataset is split into three subsets: training, test and calibration. The splitting strategy depends on the dataset type and ensures that the class distributions are preserved in each subset by using stratified sampling.

If the dataset type is `fullLC`, the dataset already contains a `split` column that explicitly defines which samples belong to training or test sets. In this case, samples labeled as `train` are used for training and those labeled as `test` are further split into test and calibration subsets using a 50/50 ratio.

For `std` and `enhanced` dataset types, which do not include predefined splits, a two-stage stratified split is applied:

1) The dataset is first split into 70% training and 30% temporary test data.
2) The temporary test data is then split equally into 15% test and 15% calibration sets.

Optionally, class distribution histograms are generated for each subset and stored in the experiment directory. These plots help verify that the split maintains class balance and can be visually inspected as part of the experiment logs.

Figures 34, 35 and 36 show an example of the class distributions for the training, test and calibration sets, respectively.

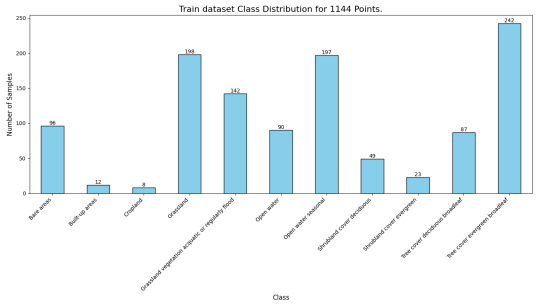


Fig. 34: Class distribution in the training set.

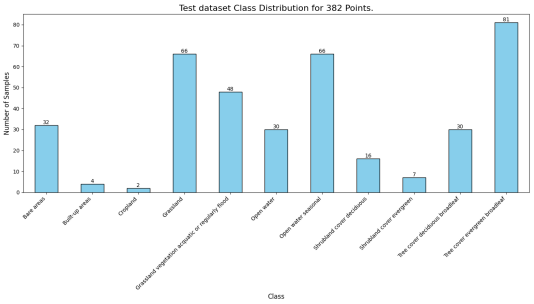


Fig. 35: Class distribution in the test set.

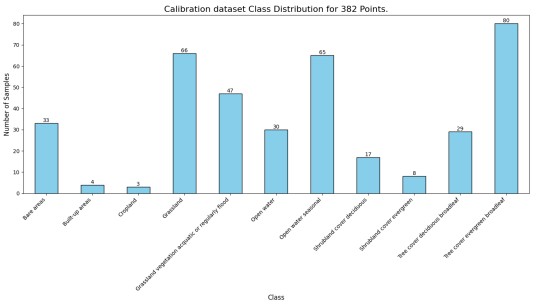


Fig. 36: Class distribution in the calibration set.

The dataset is split as the very first step because all subsequent operations, such as feature preprocessing and selection, are initially performed exclusively on the training subset. The same transformations are then applied to the test and calibration subsets using the parameters derived from the training set.

This design choice was made to prevent *data leakage* [24], a situation where information from the test or calibration sets influences the training process. If the dataset were processed as a whole before splitting, statistics used during normalization or feature selection (e.g., means, variances, correlations) would be influenced by data that the model is not supposed to observe during training; and this would lead to inconsistent evaluation results.

C. Feature Preprocessing

After splitting the dataset, the next step involves normalizing the feature values to ensure they are on comparable scales. This is especially important for algorithms like SVMs, which are sensitive to the range and distribution of input features. In this pipeline, different preprocessing strategies can be applied to spectral features and to the remaining features. Two separate parameters control the normalization method:`band_features_preprocessing` and `other_features_preprocessing`.

Accepted strategies include `standard` (z-score normalization), `minmax` (scaling to [0, 1] using the global minimum and maximum values, as implemented by scikit-learn's `MinMaxScaler`), and `minmax_q`, a quantile-based variant used in the existing pipeline. The `minmax_q` method rescales each feature to the [0, 1] range based on its lower and upper quantiles (defaulting to the 0.1% and 99.9% quantiles, respectively), potentially offering greater robustness to outliers and skewed distributions. Features that should not be normalized can be explicitly excluded using the `non_features_columns` and `excluded_from_preprocessing` parameters.

Internally, each feature is reshaped into a column vector and passed through the appropriate scaler. For reproducibility and later reuse, the transformation parameters for each feature are saved in a metadata dictionary. In the case of min-max normalization, any values falling slightly outside the [0, 1] interval are clipped to remain within bounds, coherently with the existing pipeline. If verbose mode is enabled, the pipeline also logs statistical summaries before and after normalization, which helps to monitor and debug the preprocessing step.

D. Feature Selection

This phase aims to reduce the number of input features by selecting a subset of them, to try to improve model generalization. The reimplemented pipeline supports two automatic feature selection strategies: `RFECV` and Multicollinearity Analysis.

a) Recursive Feature Elimination with Cross Validation (RFECV):: `RFECV` is a wrapper-based supervised feature selection technique that iteratively eliminates the least relevant features using internal model feedback and cross-validation. The method used in this pipeline is based on the implementation provided by `scikit-learn`'s `RFECV` class, which follows the approach introduced in [25].

At each iteration, a base estimator is trained on a progressively smaller subset of features. Features are ranked according to the estimator's internal importance metric, the absolute weights in the case of linear models. The least important feature is removed and the model is retrained with the reduced feature set. After each elimination step, the estimator's performance is evaluated using cross-validation, and the F1-macro score is recorded. The process continues until all features have been ranked and the optimal number of features is determined by selecting the subset that yields the highest average cross-validation score.

In this pipeline, `RFECV` is applied to the training set using a linear SVM (`LinearSVC`) as the base estimator. This choice allows the ranking of features to be derived from the absolute magnitude of the learned model coefficients. Cross-validation is performed using stratified folds to maintain class distribution and the scoring metric is set to `f1_macro`.

The final subset of selected features is then applied to the test and calibration datasets, ensuring that no information from these sets influences the selection process. The pipeline also generates a series of visual reports specifically for the `RFECV`-based feature selection process.

- **RFECV cross-validation performance curve:** this plot shows the variation in F1-macro score across recursive feature elimination steps. For each step, a subset of features is used to train a model and the cross-validation score is computed. The red vertical dashed line indicates

the number of selected features at which the maximum validation score is achieved. In Figure 40, this optimal point corresponds to 41 features out of the initial set. This chart is one of the most useful outputs, as it directly reflects the trade-off between model complexity and generalization ability.

- **RFECV ranking of all features:** this horizontal bar plot displays the ranking assigned to each input feature by the RFECV process. The vertical axis lists all features (including those discarded), while the horizontal axis indicates their rank (inverted for visual clarity: lower rank means higher relevance). Features with a ranking value of 1 are retained, while those with higher rankings were progressively removed. Figure 41 shows the final RFECV ranking of the original features.
  Figures 40 and 41 are reported in the next page for a better visualization.
- **Comparison of selected and removed features:** this simple bar chart summarizes the overall reduction achieved by RFECV, showing the total number of selected versus discarded features. Figure 37 presents the results for the same dataset and confirms that 41 features were retained while over 120 were removed, significantly reducing dimensionality.

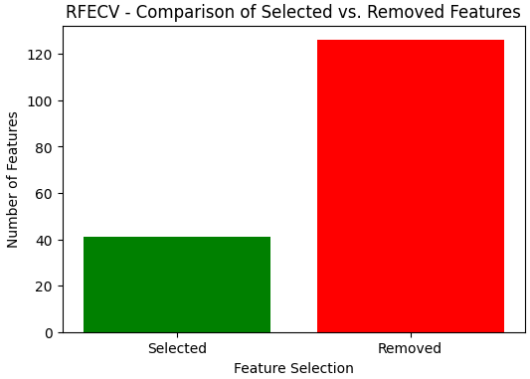


Fig. 37: Comparison of selected and removed features after RFECV. In this experiment, 41 features were selected, while 126 were discarded. The features are from January 2019.

b) Multicollinearity Analysis: As an alternative to RFECV, the pipeline supports a custom strategy for multicollinearity analysis, aimed at removing features that are highly redundant due to statistical dependencies. This method is particularly useful to avoid model-based selection as in RFECV.

The implemented procedure consists of three main steps:

- **Correlation detection:** The correlation matrix is computed across all candidate features, quantifying the linear relationships between pairs. Any pair with an absolute correlation coefficient greater than a user-defined threshold (0.9 in the reimplemented pipeline) is flagged as redundant. This step identifies groups of features that likely have overlapping information. An example of the correlation matrix before filtering is shown in Figure 38, based on the UniTN standard dataset (tile 21KUQ).

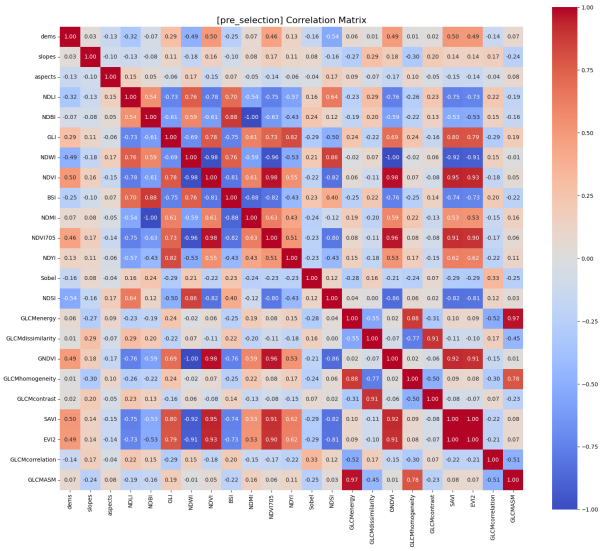


Fig. 38: Correlation matrix before multicollinearity analysis (UniTN standard dataset, tile 21KUQ). Strong dependencies appear as high values close to 1 or -1. In this example, spectral band features are omitted to reduce the number of rows and columns, providing a clearer visualization.

- **Importance estimation:** To decide which feature to discard in each correlated pair, a relevance score is computed using mutual information (MI) with respect to the target labels. MI is a non-parametric metric that captures nonlinear dependencies, estimated using the mutual_info_classif function from scikit-learn, which is based on k-nearest neighbors density estimation [26]. The resulting MI scores are then combined with the mean absolute correlation of each feature to define a hybrid selection metric:

$$\text{combined\_metric} = \frac{\text{mutual\_information}}{1 + |\text{mean\_correlation}|}$$

This metric is designed to rank features that are both informative and independent. High values indicate features with strong relevance to the target and weak average correlation with others, which are ideal candidates to be kept. Low values, on the other hand, identify features that are either less informative or redundant, and are likely to be removed. The mutual information scores prior to filtering are shown in Figure 39.

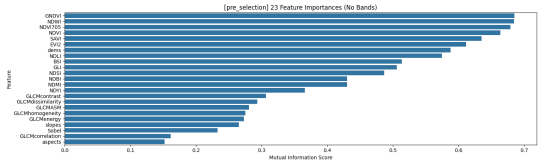


Fig. 39: Mutual information scores before multicollinearity analysis (UniTN standard dataset, tile 21KUQ).

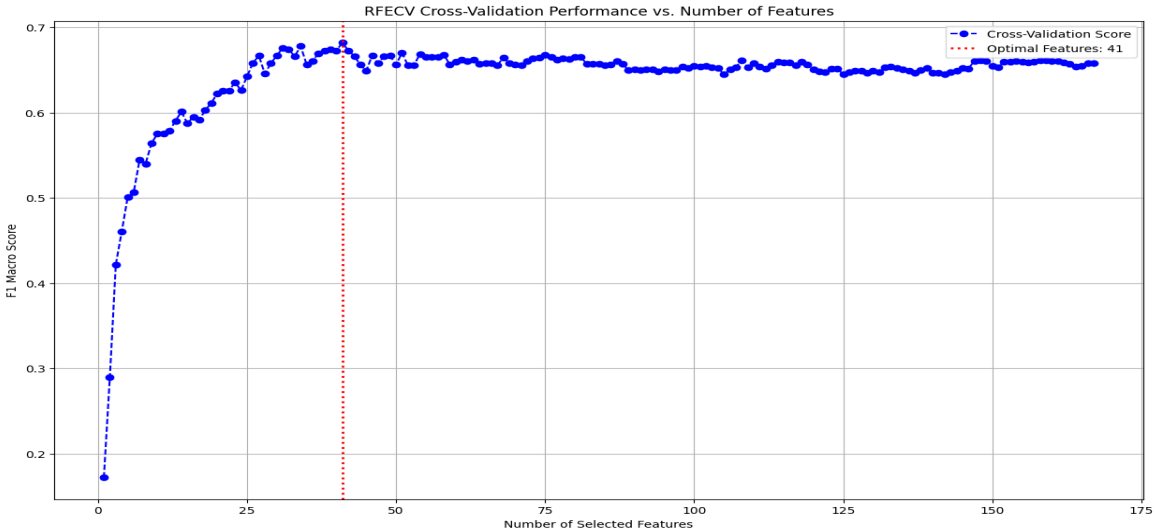


Fig. 40: RFECV performance curve for the UniTN standard dataset of tile 21KUQ, with features computed using January 2019 data. The F1-macro score is plotted against the number of selected features. The red dashed line indicates the optimal number of features (41) selected by the algorithm.

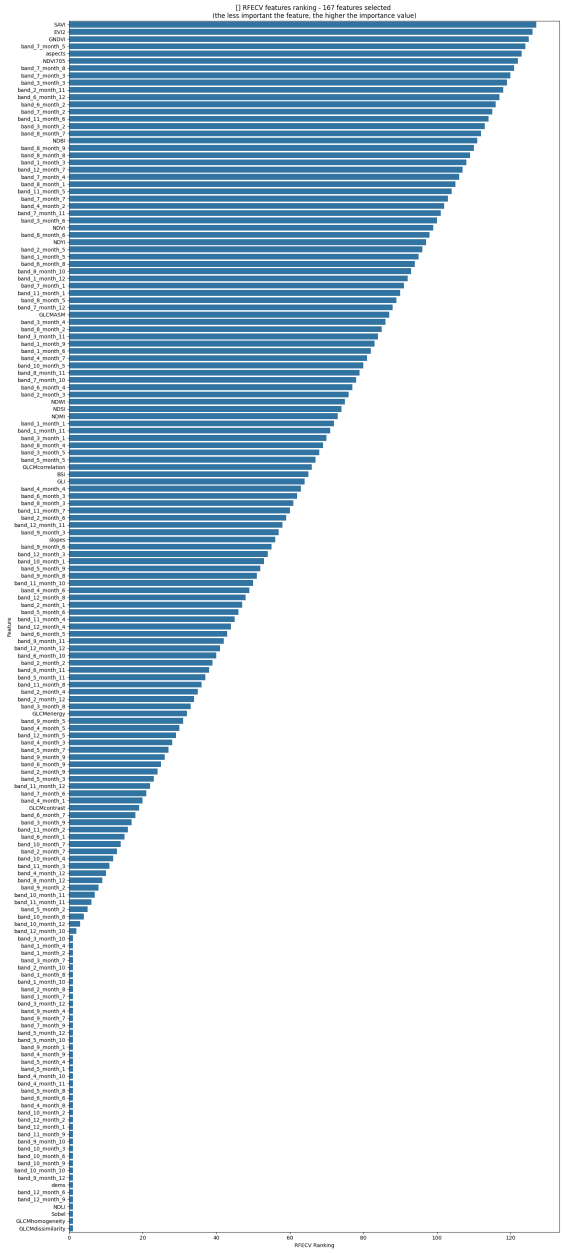


Fig. 41: RFECV-assigned feature rankings. Features with a ranking of 1 are retained, while higher values indicate less importance

- **Feature elimination:** Within each pair of highly correlated features, the one with the lower combined metric is removed. However, if one of the features belongs to a user-specified list of fixed features (parameter `fixed_features`), it is preserved regardless of the metric.

After selection, the correlation structure and information contribution of the remaining features can be re-evaluated. The final correlation matrix is reported in Figure 42 and the updated mutual information scores in Figure 43.

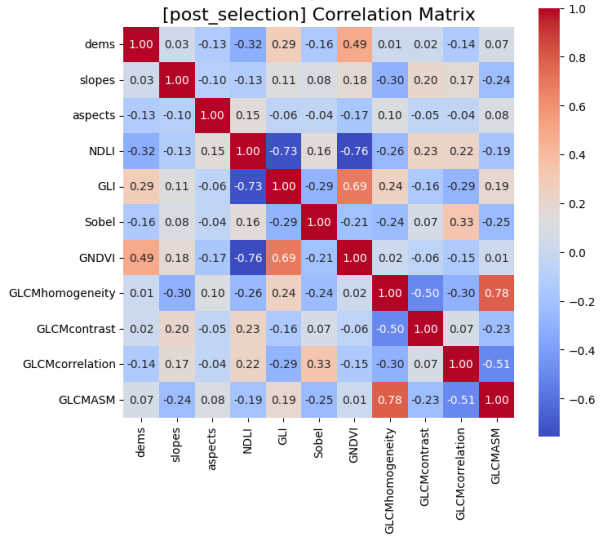


Fig. 42: Correlation matrix after multicollinearity analysis (UniTN standard dataset, tile 21KUQ). Redundancy is reduced among the retained features with respect to Figure 38.

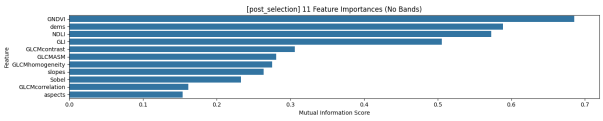


Fig. 43: Mutual information scores after multicollinearity analysis (UniTN standard dataset, tile 21KUQ).

In summary, the reimplemented pipeline supports two automated feature selection strategies: `RFECV` and multicollinearity analysis, each offering a different approach to features selection. However, this phase can also be skipped entirely by setting the `features_selection_strategy` parameter to `Null`, in which case all input features will be retained for the subsequent stages of the pipeline.

E. Feature Reduction with PCA

After feature selection, the reimplemented pipeline optionally supports a dimensionality reduction step based on PCA. PCA is a widely used linear transformation technique that projects the original features into a new set of orthogonal axes, called *principal components*, which are ranked according to the amount of variance they capture from the data. The transformation maximizes the variance along the first few components, enabling the representation of the dataset in a lower-dimensional space while retaining most of its information.

The PCA implementation relies on the `PCA` class from `scikit-learn` [27], and is applied only to the training set. The transformation is then applied to the test and calibration sets using the stored PCA metadata. The process involves the following steps:

- **Feature selection:** The columns used as input to PCA are those remaining after the previous feature selection step. Non-feature columns, as defined by `non_features_columns`, are excluded. Additionally, the parameter `pca_use_bands` can be used to explicitly include or exclude spectral band features from the reduction.
- **PCA fitting and component selection:** PCA is used to compute the principal components and their associated variance. The number of components to retain is determined using one of three selection strategies, specified through `pca_components_selection_strategy`:
  - `Cumulative Explained Variance (CEV)`: selects the smallest number of components such that the cumulative explained variance exceeds a user-defined threshold (typically 99%).
  - `Elbow Curve (EC)`: uses the ”elbow” point in the explained variance curve, identified via the `KneeLocator` function.
  - `Average`: selects the average between the CEV-based and EC-based component counts.

  The cumulative explained variance plot is shown in Figure 44, where the green, purple and red vertical lines respectively indicate the component counts from the CEV, average and selected strategies. It can be seen that with approximately 30 components, nearly all the original variance is retained.

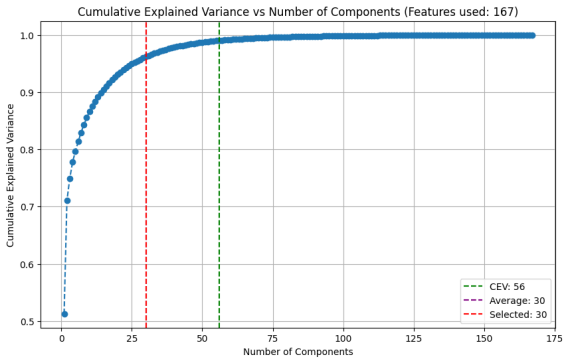


Fig. 44: Cumulative explained variance vs. number of components. The dashed lines indicate the number of components from different selection strategies: CEV (green), Average (purple), and Selected (red).

The explained variance ratio for each individual component is shown in the elbow curve (Figure 45). This

chart visualizes how variance is distributed among the components.

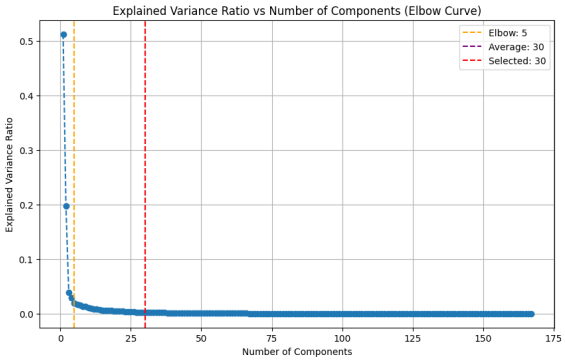


Fig. 45: Explained variance ratio per component. The elbow point (orange), average (purple), and selected (red) lines indicate the positions used to determine the number of retained components.

- **Transformation and reconstruction:** Once the number of components is selected, the training set is projected into the lower-dimensional space defined by the chosen principal components. This transformation is also applied to the test and calibration sets using the stored PCA model. If bands are excluded from PCA computation they are re-added after the transformation.
- **Metadata storage:** All relevant PCA statistics (e.g., selected components, explained variance, means, projection matrix) are stored to ensure that downstream datasets can be consistently transformed in the same way.

As a post-analysis diagnostic, the pipeline also computes the mutual information scores of the final set of principal components with respect to the target labels. Figure 46 shows these importance scores for the selected components. The first few components are notably more informative, confirming that PCA preserved the most relevant information in the first components generated.

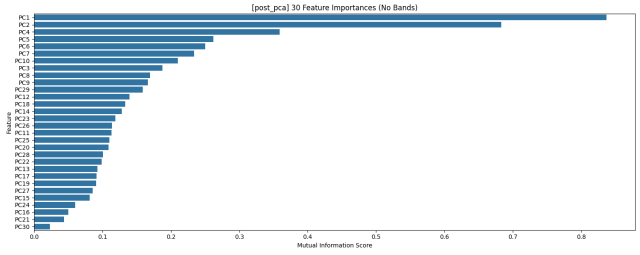


Fig. 46: Mutual information scores of selected principal components with respect to the target labels. The most informative directions are concentrated in the first few PCs.

F. SVM Training and Evaluation procedures overview

The core part of the pipeline involves training SVMs using two parallel strategies: a single multiclass SVM trained using an OvR configuration and a set of binary SVMs trained independently for each class, also using the OvR strategy but implemented manually. Both approaches include hyperparameter optimization via grid search and support probability calibration.

a) Multiclass SVM:: A single SVM model is trained using `scikit-learn`'s `SVC` class. In this setup, one binary classifier is trained per class, treating it as positive while all others are treated as negative. During prediction, the class with the highest decision function output is selected.

Hyperparameter tuning is performed using `GridSearchCV` [28], which systematically evaluates combinations of hyperparameters (`C`, `gamma`, `class_weight`) for the `rbf` kernel. A 3-fold cross-validation is used and the `f1_macro` score serves as the selection metric.

After identifying the best model, it is evaluated on the test set and then calibrated using isotonic regression, implemented via `CalibratedClassifierCV` [29]. This calibration step adjusts the raw decision scores to produce better probability estimates, using a separate calibration set to avoid information leakage. Both the uncalibrated and calibrated versions are evaluated.

b) Binary SVMs:: The pipeline trains a separate binary SVM for each class using the same OvR principle. Each classifier distinguishes one class from the rest by relabeling the dataset into a binary problem. This approach allows per-class tuning and evaluation. Each binary classifier undergoes independent hyperparameter optimization using `GridSearchCV` with the same parameter grid and cross-validation setup as the multiclass model. Once trained, each binary SVM is calibrated using `CalibratedClassifierCV` with isotonic regression, just like the multiclass model.

c) Evaluation and Reporting:: Model performance is assessed using `scikit-learn`'s `classification_report` [30] and confusion matrices. The evaluation is handled by the `test_svm` function, which returns structured dictionaries for further processing.

G. Model Packaging and Prediction with the `SVMS` Class

This section introduces the custom `SVMS` class, specifically designed to encapsulate all the components of a single experiment. The class aggregates the trained models (both multiclass and binary SVMs), as well as the associated preprocessing metadata and dimensionality reduction parameters. This structure allows the entire pipeline state to be serialized and stored to disk for reuse in the final prediction module responsible for generating land cover maps. Moreover, `SVMS` exposes a `scikit-learn`-like interface with `predict()` and `predict_proba()` methods, enabling inference using either the multiclass SVM or the binary classifiers. This section also details the internal structure used to store model metadata and the mechanisms through which predictions are generated.

a) Class Structure: The `SVMS` class is designed to aggregate all elements necessary for performing predictions. Internally, it stores:

- The multiclass SVM model and its calibrated counterpart.
- A dictionary of binary SVMs, one per class, along with their respective calibrated versions.
- The complete set of features used during training, to ensure compatibility with new input data.
- The full training report and evaluation metrics for both multiclass and binary SVMs.
- Metadata related to feature preprocessing and selection to ensure reproducibility.
- Optional PCA transformation metadata, if dimensionality reduction was applied.
- A copy of the experiment configuration dictionary.

The prediction behavior of the class is fully configurable through three internal boolean attributes:

- `use_binary_svms`: if set to `True`, the class uses the group of binary SVMs to compute predictions.
- `use_binary_svms_with_softmax`: if `True`, this modifies the binary SVMs behavior by applying a softmax normalization to the output probabilities. This option requires `use_binary_svms` to also be `True`.
- `use_multiclass_svm_calibrated`: when using the multiclass SVM, this flag determines whether the calibrated or uncalibrated model is used for predictions.

Exactly one prediction strategy is active at any given time, and it is determined by the combination of these flags. If `use_binary_svms` and `use_multiclass_svm_calibrated` are `False`, the multiclass SVM is used by default.

To interface with downstream components, the class provides two main methods: `predict()` for generating class labels and `predict_proba()` for estimating class probabilities. Internally, these methods delegate to the appropriate prediction strategy depending on the class configuration flags. When the multiclass SVM is used (either calibrated or not), predictions are obtained directly from the corresponding model. In contrast, when binary SVMs are enabled, the class aggregates their outputs using one of the dedicated methods described in the next section.

b) Probability Estimation with Binary SVMs: When the `SVMS` class is configured to use the binary classifiers, it supports two alternative strategies for estimating multiclass probabilities by aggregating the outputs of the individual class-specific SVMs. These aggregation procedures are implemented in two dedicated methods: `predict_proba_with_binary_svms()` and `predict_proba_with_binary_svms_softmax()`. Both methods aim to replicate the logic behind the internal implementation of `scikit-learn`'s `CalibratedClassifierCV`, particularly the behavior of the `_CalibratedClassifier` class [31].

- **Probabilities estimation**: the `predict_proba_with_binary_svms()` method consists of the following steps:
  1) For each class $k \in \{1, \ldots, K\}$:
     a) Apply the uncalibrated binary SVM $f_k(\mathbf{x})$ to compute the decision function value:
     $$s_k = f_k(\mathbf{x}) \in \mathbb{R}$$
     b) Use the corresponding fitted isotonic calibrator $h_k(\cdot)$ to obtain a calibrated probability estimate:
     $$p_k = h_k(s_k) \in [0, 1]$$
  2) Construct the raw probability vector $\mathbf{p} = [p_1, p_2, \ldots, p_K]$.
  3) Normalize $\mathbf{p}$ to ensure it forms a valid probability distribution:
     $$\hat{p}_k = \frac{p_k}{\sum_{j=1}^{K} p_j}, \quad \text{if } \sum_j p_j > 0$$
     If $\sum_j p_j = 0$ for a given sample (i.e., all p_k are zero), the fallback is to assign a uniform distribution:
     $$\hat{p}_k = \frac{1}{K}, \quad \forall k$$
  4) For binary classification ($K = 2$), normalization is simplified as:
     $$\hat{p}_0 = 1 - p_1, \quad \hat{p}_1 = p_1$$
  5) Finally, return the normalized probability vector $\hat{\mathbf{p}}$.

  This strategy preserves the calibrated nature of the class-specific scores while enforcing probabilistic consistency across all classes.
- **Probabilities estimation with Softmax**: in the variant `predict_proba_with_binary_svms_softmax()`, each binary classifier is still used to compute a calibrated probability estimate for its positive class, following the same steps as above. However, instead of directly normalizing with the sum of values, the vector of calibrated outputs is passed through the softmax function:
  1) As before, for each class k:
     $$s_k = f_k(\mathbf{x}), \quad p_k = h_k(s_k)$$
  2) Construct the probability vector $\mathbf{p} = [p_1, p_2, \ldots, p_K]$.
  3) Apply the softmax transformation:
     $$\hat{p}_k = \frac{\exp(p_k)}{\sum_{j=1}^{K} \exp(p_j)}$$
  4) Return the softmax-normalized probabilities $\hat{\mathbf{p}}$.

  The softmax function ensures that all class probabilities are positive and sum to 1, while also preserving the relative differences between the calibrated outputs. This approach is advantageous when the magnitude of scores across classes differs significantly, providing smoother and more balanced prediction probabilities.
- **Prediction**: the method `predict()` internally delegates to `predict_proba()` and performs a simple class selection:
  $$\hat{y} = \arg\max_k \hat{p}_k$$

where $\hat{p}_k$ is the predicted probability for class k obtained using the configured prediction strategy. Thus the returned prediction will correspond to the class with the highest probability.

H. Calibration Curves

The calibration curves presented in this section are generated using the Unitn standard dataset for tile 21KUQ. In this experiment, the SVM pipeline was configured to use all available features with standard preprocessing for both band and non-band features and without applying feature selection or PCA. Calibration curves visually assess how well the predicted probabilities from an SVM model align with the true likelihoods of the target outcomes. A well-calibrated model produces probability estimates such that, for example, among all samples predicted with 70% probability, approximately 70% actually belong to the positive class. These curves are generated using the `CalibrationDisplay` class from `scikit-learn` [32] and computed with the strategy set to `'quantile'`, which is particularly useful for addressing class imbalance in probability bins by avoiding bins with too few samples.

a) Binary SVM Calibration Curves:: By default, binary SVMs are trained with `probability=False`, which prevents the use of the `predict_proba` method; hence, calibration curves cannot be generated in that case. However, when training with `probability=True`, the pipeline can produce calibration curves for both uncalibrated and calibrated binary classifiers.

- **Uncalibrated binary SVMs:** When `probability=True` is set during training, the uncalibrated binary SVMs provide probability estimates that can be plotted. Figure 47 illustrates the calibration curves for each binary classifier, showing the alignment between predicted probabilities and the observed fraction of positives.

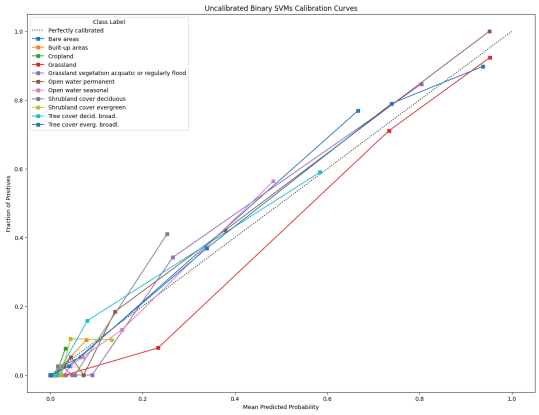


Fig. 47: Calibration curves for uncalibrated binary SVMs (using the quantile strategy) for each class with the Unitn standard dataset (tile 21KUQ).

- **Calibrated binary SVMs:** For the same binary classifiers, after calibration with isotonic regression, probability estimates become more reliable. Figure 48 demonstrates how calibration aligns the predicted probabilities with the empirical frequencies across classes.

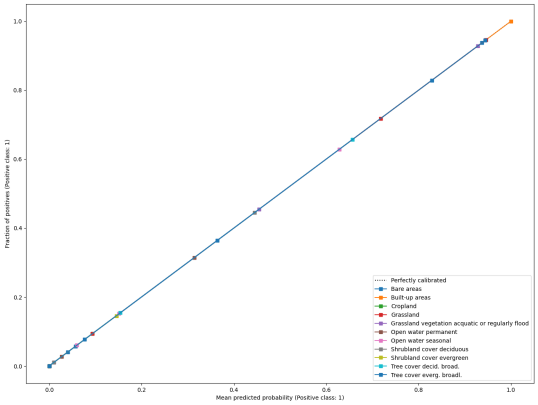


Fig. 48: Calibration curves for calibrated binary SVMs (using the quantile strategy) for each class on the Unitn standard dataset (tile 21KUQ).

b) Multiclass SVM Calibration Curves:: The multiclass SVM is trained with `probability=True`, which directly enables the use of the `predict_proba` method for both calibrated and uncalibrated versions of the classifier. For each class, the true labels are first binarized and then calibration curves are plotted. Figure 49 presents the calibration curves for the uncalibrated multiclass SVM, while Figure 50 shows the curves after isotonic regression calibration. These plots illustrate how calibration improves the reliability of probability estimates for each class in the multiclass setting.

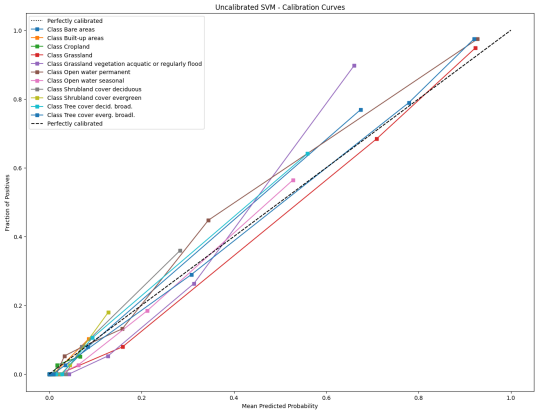


Fig. 49: Calibration curves for the uncalibrated multiclass SVM (using the quantile strategy) on the Unitn standard dataset (tile 21KUQ).

Calibration curves serve as a critical diagnostic tool to verify that the predicted probabilities are meaningful. They also help assess whether the calibration process (via isotonic regression) effectively adjusts the model outputs to better reflect the true outcome frequencies.

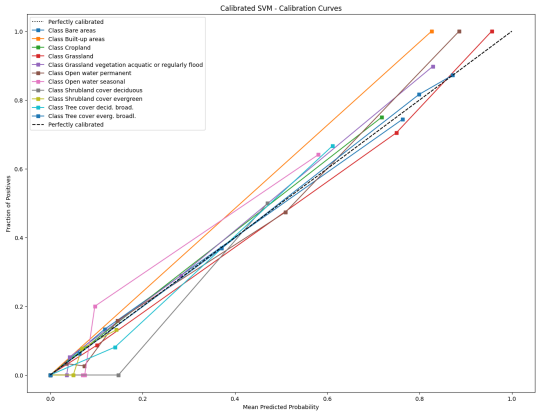


Fig. 50: Calibration curves for the calibrated multiclass SVM (using the quantile strategy) for each class on the Unitn standard dataset (tile 21KUQ).

I. Experimental Setup and Grid Search for SVM Pipeline Parameters

To assess the impact of various configuration parameters on SVM performance, the pipeline adopts a grid search strategy. This approach systematically explores combinations of key parameters, presented in Section VIII-A, that influence dataset selection, feature preprocessing, feature selection and dimensionality reduction. The aim is to identify the most promising configurations through exhaustive experimentation.

Parameters used for grid search are specified in a YAML file using the `gs_` prefix. Each parameter is associated with a list of candidate values, which are used to generate all possible combinations in the search space. For instance, the feature selection strategy can be defined as `gs_features_selection_strategy` with values such as `[rfecv, multicollinearity_analysis, Null]`.

The grid search function automatically detects all parameters prefixed with `gs_`, generates the full Cartesian product of their values and applies pruning rules to remove redundant or invalid configurations. For example, if `apply_pca` is set to `False`, PCA-related parameters such as `pca_use_bands` are ignored and excluded from the final configuration. This filtering avoids unnecessary duplication and ensures computational resources are focused on meaningful combinations.

This automated exploration phase allows the identification of promising configurations, which are then selected for more targeted manual evaluation. The following paragraphs present the results of the grid search and introduce the best-performing parameter values.

a) Grid Search: After filtering, a total of 1020 distinct parameter combinations were tested. These configurations were generated from the following set of parameter values:

- **Dataset path (`gs_dataset_path`):** Different CSV datasets, such as standard, enhanced and fullLC variants produced with the dataset generation component presented in VII.
- **Preprocessing strategies:**
  - `gs_band_features_preprocessing`: `[minmax, standard]`
  - `gs_other_features_preprocessing`: `[minmax, standard]`
- **Features to use:**
  - `gs_use_band_features`: `[True]`
  - `gs_use_glcm_features`: `[True, False]`
  - `gs_use_other_features`: `[True, False]`
- **Feature selection strategy (`gs_features_selection_strategy`):** `[Null, rfecv, multicollinearity_analysis]`
- **Dimensionality reduction via PCA:**
  - `gs_apply_pca`: `[True, False]`
  - `gs_pca_use_bands`: `[True, False]`

By systematically evaluating these combinations, the grid search provides a view of how different configurations affect model performance. Each experiment's result is recorded in a CSV file, for a subsequent analysis.

J. Grid Search Results Discussion

To better understand the influence of individual configuration parameters on model performance, the results of the grid search were analyzed by aggregating the average macro F1 scores. The analysis shown below focuses exclusively on the top 10 best-performing configurations; this filtering highlights the parameter settings that consistently appear in the most successful pipeline configurations.

For each parameter involved in the grid search, the plots display the mean SVM macro F1 score observed among the top-performing configurations that include each specific value. This helps identify which parameter values lead to better performances.

1) Multiclass SVM: The analysis of the top-10 multiclass SVM configurations, shown in Figure 51, confirms and further clarifies several previously observed trends. The `std` dataset variant, corresponding to the UniTN standard format, remains the most reliable option, consistently yielding the highest average F1 scores across top-performing configurations. In contrast, modified variants such as `enhanced`, `eroded` and especially `fullLC` perform considerably worse. This suggests that the alterations applied to these datasets may negatively impact data quality, introducing inconsistencies or distortions that have a significant impact to the SVM performance.

Configurations that include `GLCM` features are again associated with improved performance, achieving an average F1 score of 80.8% in top configurations. On the other hand, the inclusion of auxiliary features (described in Section VI-B) appears to have a slightly negative effect in this iteration, with configurations excluding them reaching a higher mean score (80.97% vs. 77.27%).

For features preprocessing, the standardization of band features clearly outperforms both `minmax` and `minmax_q` normalization methods, reaching an average of 80.71%. For non-band features, the best results are obtained using `minmax`

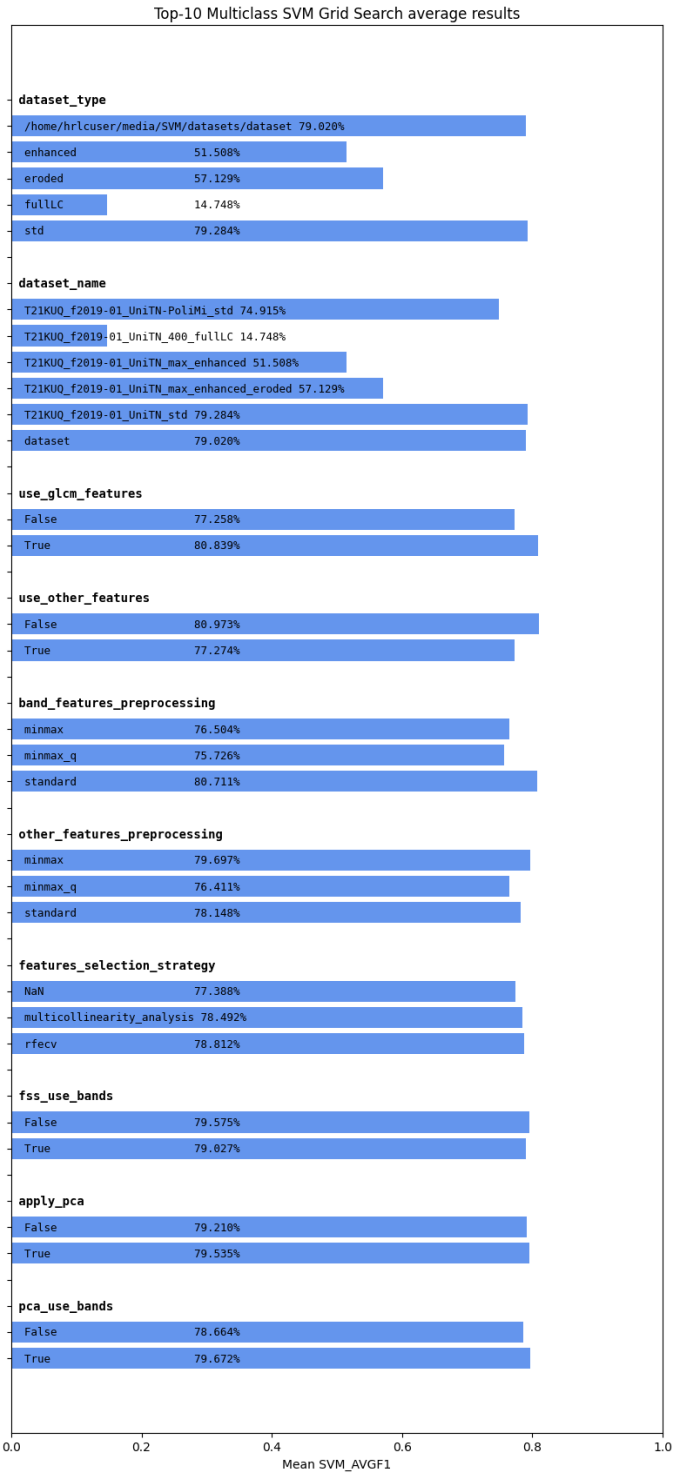


Fig. 51: Average macro F1 scores by parameter value for the top-10 multiclass SVM grid search configurations.

normalization (79.69%), followed by standardization and `minmax_q`.

Regarding feature selection, although all strategies yield fairly similar performance, RFECV slightly outperforms multicollinearity analysis and configurations without any feature selection, suggesting a modest but consistent benefit when used. The inclusion of bands in the selection process (`fss_use_bands=True`) appears to have a minor negative effect this time, slightly reducing the average F1 score compared to excluding bands from the selection.

Finally, PCA remains a non-decisive factor. Configurations with and without PCA show nearly identical performance (79.53% vs. 79.21%). Similarly, whether band features are included in the PCA transformation or not has only a marginal impact, with a slight edge (79.67% vs. 78.66%) in favor of their inclusion.

2) Binary SVMs: The results for the binary SVMs, shown in Figure 52, confirm several trends observed in the multiclass case, while also highlighting specific differences. As expected, the binary classifiers reach higher average macro F1 scores compared to their multiclass counterparts, benefiting from the simpler two-class setting where each classifier can focus on a single class versus the rest.

Configurations based on the UniTN standard dataset continue to dominate the top-performing group, reaffirming its suitability across both multiclass and binary classification tasks. Enhanced and eroded variants perform slightly better in the binary setting than in the multiclass one, but still lag behind the standard dataset. The fullLC dataset, by contrast, consistently ranks at the bottom, suggesting that the label integration strategy used during its construction introduces noise or inconsistencies.

In terms of feature selection, the inclusion of GLCM texture descriptors again proves beneficial, slightly improving performance across the top configurations. However, the use of other auxiliary features does not lead to consistent improvements, with top-performing runs found both with and without them. This indicates that while GLCM features contribute positively, the auxiliary features have a more variable or redundant effect in the binary setting.

Preprocessing strategies show consistent patterns: standardization remains the most effective approach for band features, followed by both min-max and quantile-based normalization. For non-band features, all preprocessing methods (standard, minmax, minmax_q) yield comparable performance with only slight differences. This suggests that preprocessing has a more limited influence on binary classification compared to multiclass SVMs.

As for feature selection strategies, RFECV again outperforms multicollinearity analysis and the absence of selection. Although the differences are small, RFECV consistently provides the highest F1 scores, confirming its utility in isolating the most informative features even in the binary setting. However, the inclusion or exclusion of band features in the selection process has only a marginal effect on final performance.

Interestingly, the use of PCA also appears neutral in this context. Whether or not PCA is applied, and regardless of whether band features are included in the dimensionality reduction process, the differences in macro F1 scores are minimal. This suggests that the feature space in binary classification is already sufficiently compact, and dimensionality reduction offers no substantial advantage.

As with the multiclass analysis, the grid search results can be interactively explored using the accompanying notebook. This interface enables users to filter results by K (top configurations) and inspect how each parameter setting impacts performance, this analysis is available in the SVMs related notebook of the project.

3) Top-Performing Model Configurations: This section presents the detailed configurations and performances of the best models identified during the grid search. These include both the multiclass SVM and the binary SVMs. The selected configurations correspond to those that achieved the highest average macro F1 score across the entire grid search. In the following paragraphs, the performance metrics and calibration behavior of these two models are presented.

a) Best Multiclass SVM Configuration: The optimal multiclass SVM configuration is based on the UniTN standard dataset, and includes a combination of band and GLCM features. It applies RFECV for feature selection (with bands included in the selection), followed by PCA for dimensionality reduction (excluding band features from the PCA transformation). The feature preprocessing strategy involves standardization for band features and min-max normalization for non-band features. To summarize the parameters used are the following:

- **Dataset:** Existing pipeline dataset
- **Used features:** Bands and GLCM; other features excluded
- **Feature preprocessing:** Bands: `standard`; Other: `minmax`
- **Feature selection:** `rfecv`; Bands included in selection
- **PCA:** Applied; Band features excluded
- **SVM parameters:**
  - $C = 100.0$
  - $\gamma = 0.001$
  - `probability = True`

The uncalibrated version of this model achieves an overall accuracy of 80% and a macro-averaged F1 score of 0.77. Precision and recall are high for dominant classes such as *Tree cover evergreen broadleaf* ($F1 = 0.93$), *Grassland* ($F1 = 0.89$), and *Open water seasonal* ($F1 = 0.77$). Conversely, minority classes such as *Cropland* and *Shrubland cover deciduous* exhibit lower performance, reflecting their underrepresentation in the dataset (2 and 8 samples, respectively).

As shown in Figure 53, the model's predictions are skewed toward majority classes, with *Tree cover evergreen broadleaf*, *Grassland*, and *Open water seasonal* being the most frequently predicted labels. This imbalance, combined with the observed performance, suggests a tendency to favor well-represented categories in the decision boundaries.

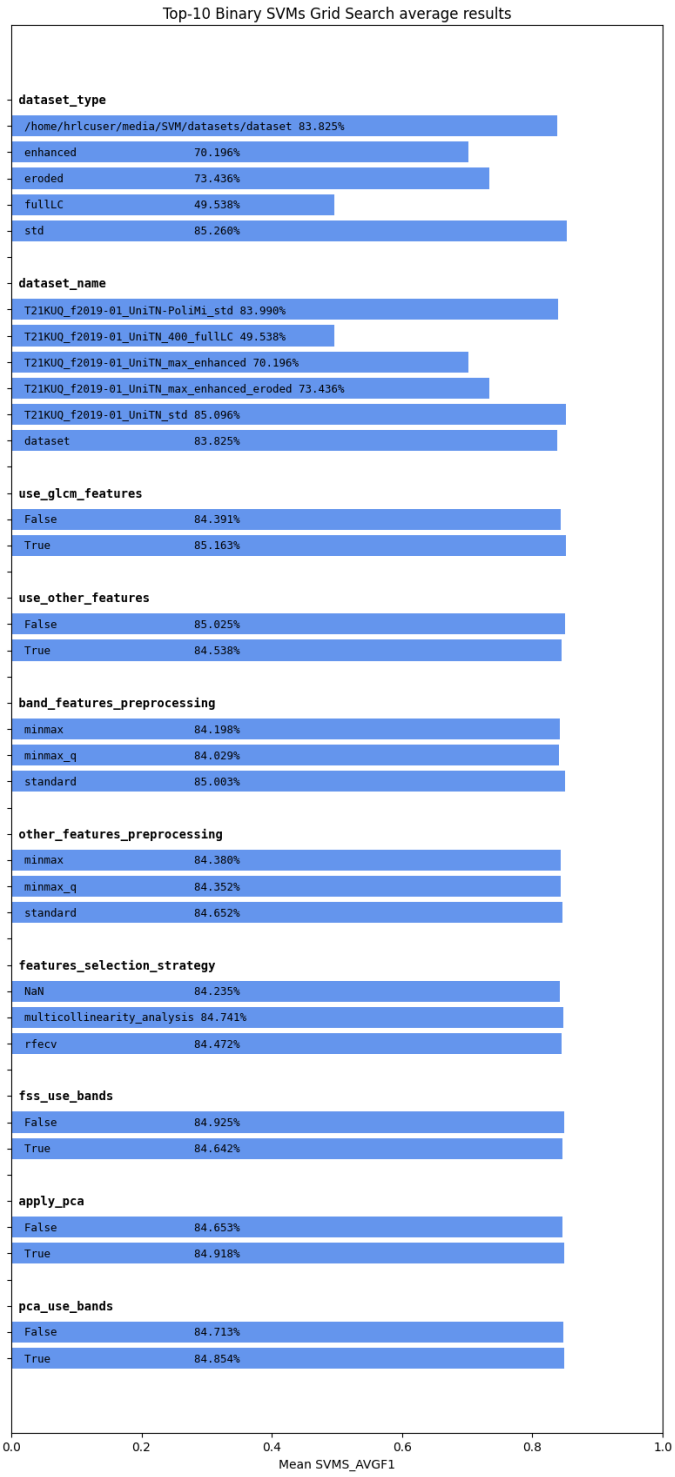


Fig. 52: Average macro F1 scores by parameter value for the top-10 binary SVMs configurations.

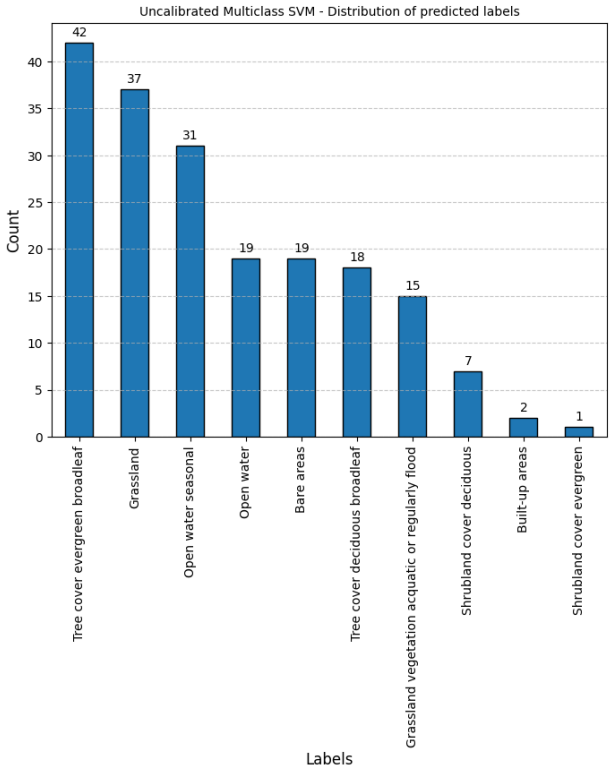


Fig. 53: Predicted label distribution for the uncalibrated multiclass SVM.

Although the uncalibrated model delivers quite good classification results, the reliability of its probability estimates is less optimal. This is evident in the calibration curves presented in Figure 54. Several classes exhibit probability estimates that diverge from the ideal diagonal, especially in the low and mid confidence regions. This misalignment suggests that, while the model may classify correctly, the predicted probabilities may not accurately reflect the likelihood of those predictions.

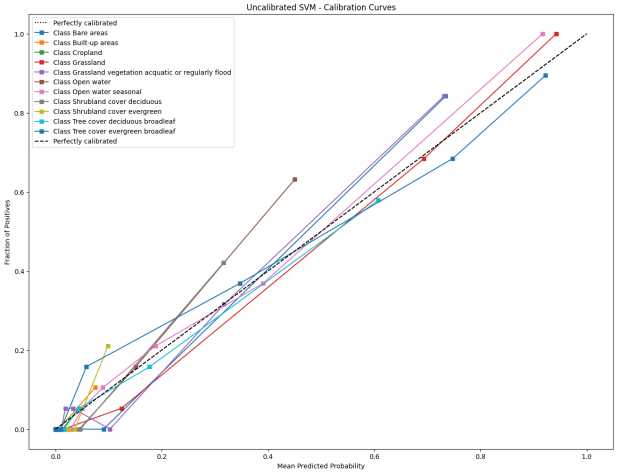


Fig. 54: Calibration curves for the uncalibrated multiclass SVM.

To improve probabilistic interpretability, the model is recalibrated using isotonic regression. After calibration, accuracy remains stable at 0.80, while the macro-averaged F1 score decreases slightly to 0.71. This drop is primarily due to fluctuations in classes with few instances, such as *Cropland*, which are no longer correctly classified.

Despite the minor decline in classification metrics, the predicted label distribution (Figure 55) remains largely consistent with the uncalibrated case, reinforcing that calibration does not distort prediction patterns but improves probability estimates.

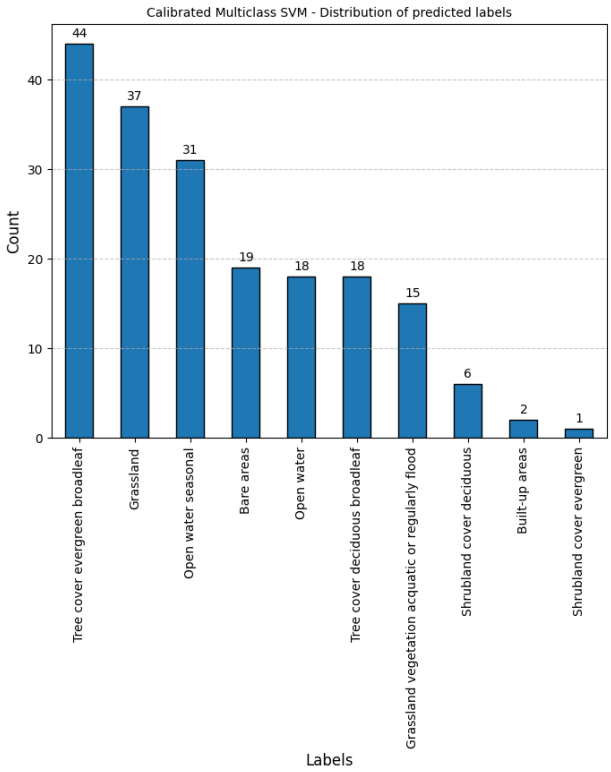


Fig. 55: Predicted label distribution for the calibrated multiclass SVM.

Figure 56 illustrates the positive impact of calibration: the curves are more tightly aligned with the diagonal, particularly in mid-to-high probability regions. This improved calibration is important for downstream tasks that depend on well-calibrated confidence values.

b) Best Binary SVMs Configuration: The best-performing binary SVMs share several similarities with the multiclass setup, including the use of the UniTN standard dataset and the same feature preprocessing strategies. However, it differs in the inclusion of all three feature groups—bands, GLCM, and other auxiliary features, while omitting both feature selection and PCA. Each binary classifier is trained with `probability=False`, and is subsequently calibrated using isotonic regression to obtain probability estimates. The configuration details are summarized below:

- **Dataset:** UniTN standard (`T21KUQ_f2019-01_UniTN_std.csv`)

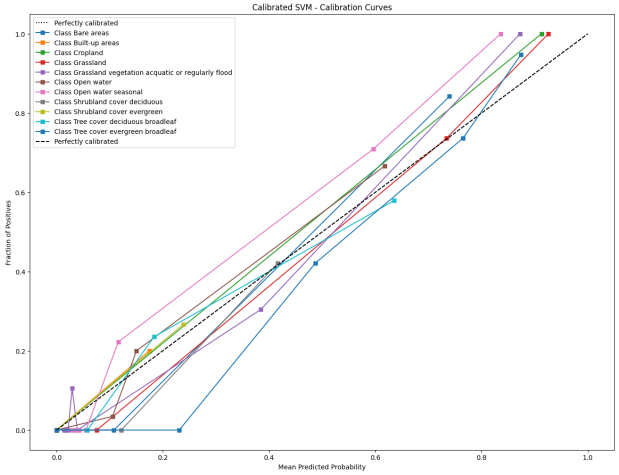


Fig. 56: Calibration curves for the calibrated multiclass SVM.

- **Used features:** Bands and GLCM
- **Feature preprocessing:** Bands: `standard`; Other: `standard`
- **Feature selection:** Multicollinearity analysis
- **PCA:** Applied excluding bands
- **SVM parameters:**
  - $C = 550.0$
  - $\gamma = 0.001$
  - `probability = False`

The binary SVMs achieves an overall accuracy of 0.81 and a macro-averaged F1 score of 0.69. Strong classification performance is observed across dominant classes such as *Tree cover evergreen broadleaf*, *Grassland*, and *Open water seasonal*. However, the SVMs struggles with rare classes such as *Cropland* and *Shrubland cover evergreen*, which suffer from very low support and are frequently misclassified or not detected at all.

Figure 57 highlights the dominance of classes such as *Tree cover evergreen broadleaf*, *Grassland* and *Open water seasonal*, which are also among the most frequent in the training dataset. Conversely, rare classes like *Cropland*, *Shrubland cover evergreen* and *Built-up areas* are significantly underrepresented in the predictions. This fact is consistent with the results seen in the multiclass configuration.

However, the key strength of the binary SVM setup lies in the quality of its probability estimates. Thanks to the post-training isotonic calibration, the model outputs well-aligned probabilities across all classifiers. As illustrated in Figure 58, the calibration curves closely follow the ideal diagonal, indicating high reliability of the predicted confidence scores.

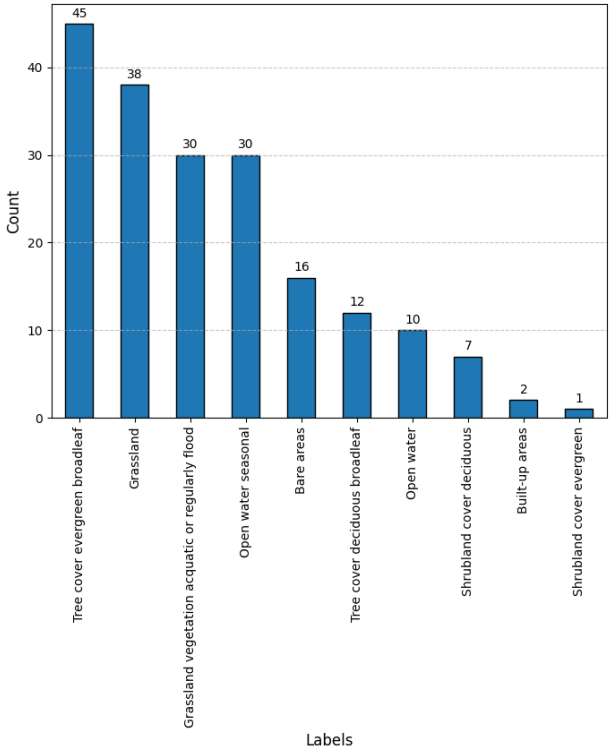


Fig. 57: Distribution of predicted labels for the calibrated binary SVMs.

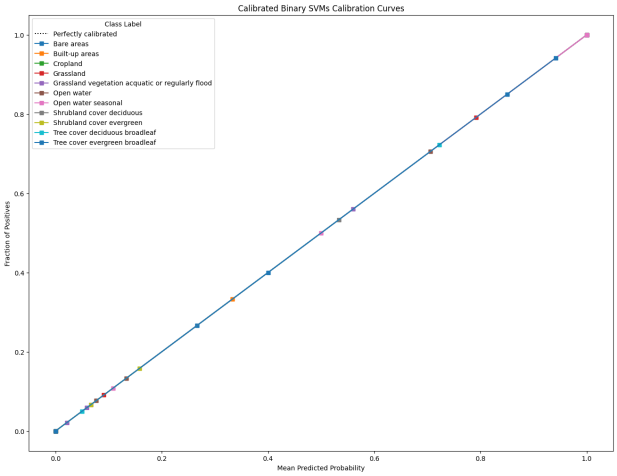


Fig. 58: Calibration curves for the calibrated binary SVMs.

In summary, although the binary SVM ensemble yields a slightly lower macro F1 score compared to the best multiclass configuration, it demonstrates significantly better calibration performance.

IX. Reimplemented Pipeline – Land Cover Maps Generation

This final component of the pipeline is responsible for generating the output LC maps using a custom `SVMS` model produced by the classification pipeline and described in Section VIII-G0a. This module corresponds functionally to the last step of the original pipeline but introduces a redesigned classification workflow built upon `xarray` and `dask`.

A. Parameters

The parameters for this component are organized into three categories: data selection, resource paths and classification control.

These parameters filter which input products are used for classification.

- **Composites year** (`composites_year`): selects input composites matching the specified year.
- **DEM year** (`dems_year`): selects DEM products for the specified year.
- **Tile identifier** (`tile_id`): filters input files to include only those corresponding to the specified tile.

This group of parameters defines the directories and files required for processing.

- **Composites directory** (`composites_path`): path to the directory containing input composite images.
- **DEMs directory** (`dems_path`): path to the directory containing DEM products.
- **Features directory** (`features_path`): path to the directory (formatted as `YYYY-MM`) where features, as described in Section VI, are stored.
- **Labels file path** (`labels_path`): path to the file defining the class labels.
- **Output directory** (`output_path`): destination directory for the final LC map.

The last group of parameters controls how the classification process is executed.

- **Path to custom model** (`model_path`): path to a serialized `SVMS` model, trained and stored by the previous pipeline component.
- **SVM configurations to use** (`svms_to_use`): this parameter is a list of identifiers (`svmMc`, `svmMcCal`, `svmsBin`, `svmsBinSoftmax`) specifying which SVM configurations should be used to generate the LC maps. Multiple configurations can be selected in a single execution, allowing the generation of several LC maps simultaneously.
- **Chunk size** (`chunk_size`): specifies the spatial size of square chunks to be processed in parallel by `xarray`.
- **Debug chunk limit** (`chunks_limit`): optional parameter for limiting the number of chunks processed during debugging.

B. Workflow for Land Cover Map Generation

The final component of the reimplemented pipeline is responsible for applying a trained `SVMS` model to multispectral and auxiliary features in order to generate LC classification maps. This process is designed to be tile-independent: LC maps can be generated for any tile, if all required inputs such as composite images, DEM and derived features are available for the selected location and time period. This section describes the complete workflow followed by the component, from data loading to the export of the prediction outputs.

a) Workflow Overview: The classification component applies the trained `SVMS` model over the input dataset using a patch-based strategy leveraging `xarray` and `dask`. The workflow proceeds as follows:

1) **Model and Metadata Loading:** The stored `SVMS` model is loaded from disk. This object contains all necessary elements for inference, as presented in Section VIII-G0a.
2) **Data Acquisition and Fusion:** The component also retrieves all the required composite data, DEM and GLCM features. These data are merged into a unified `xarray.Dataset` containing all features needed for classification.
3) **Preprocessing and Feature Flattening:** The input dataset is transformed by expanding the temporal and spectral dimensions into separate feature variables, to facilitate subsequent steps. Then, the data are filtered to retain only the features used during training.
4) **Patch-Based Parallel Inference:** The feature dataset is divided into square patches of configurable size and empty GeoTIFF files (initialized with zeros) are generated and stored on disk, one for each configuration specified in `svms_to_use`. For each patch:
   - Features are preprocessed and, if enabled, projected onto the PCA-reduced space.
   - Predictions are computed using the svm configurations specified though the parameter `svms_to_use`.
   - Alongside the predicted labels, class-wise probability maps are also generated.
   - The predictions and corresponding probabilities are directly written to the correspondant GeoTIFF files.

   Once all patches are processed, the GeoTIFF images contain complete prediction and probability maps for the entire scene.
5) **Export to PNG:** For quick visual inspection, the band representing predicted labels is also exported as a `.png` image, providing a simplified preview of the final land cover classification map.

C. Performance Comparison

This section presents a comparative analysis of the computational resources used by the existing pipeline and the reimplemented version for generating LC maps. In order to ensure a fair comparison, the reimplemented version was configured to emulate the behavior of the existing system by limiting the execution to a single SVM configuration via the `svms_to_use` parameter. This setup ensures that both pipelines process one LC map at a time.

Figure 59 reports memory usage and execution time for two runs of each version. As shown, the reimplemented pipeline exhibits a considerably higher memory footprint, with average memory usage around 10.1 GB and peak values reaching nearly 19.3 GB. In contrast, the existing version maintains a lower and more gradual memory profile, averaging about 3.8 GB and peaking at approximately 12.6 GB.

Despite the increased memory consumption, the new implementation demonstrates improved efficiency in terms of runtime. On average, it completes the generation of an LC map in approximately 6038 seconds (1.68 hours), compared to 12510 seconds (3.47 hours) required by the older system. This suggests that the reimplementation trades higher memory usage for significantly faster processing times.

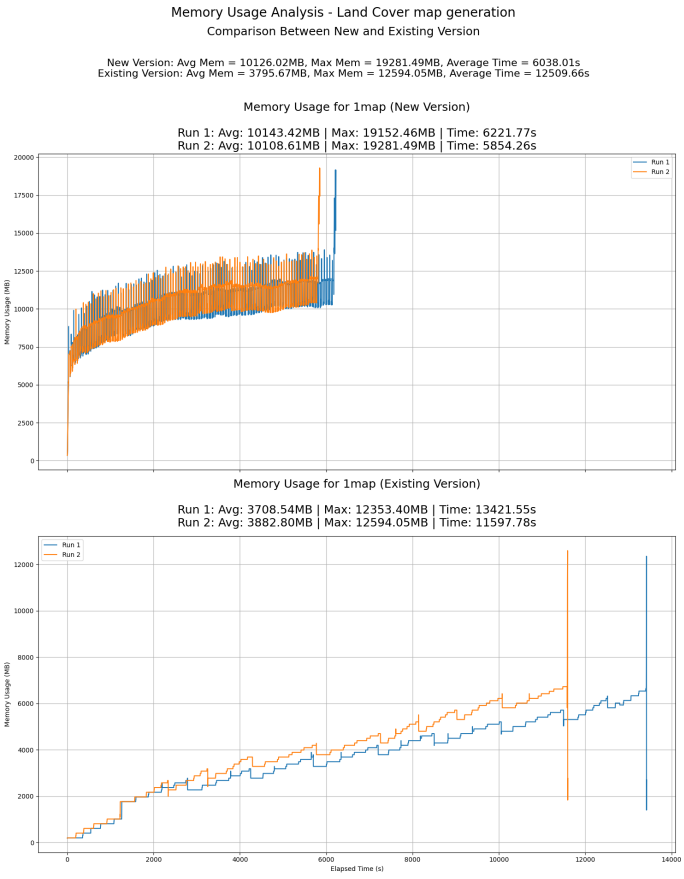


Fig. 59: Comparison of memory usage and execution time between the reimplemented and existing pipeline for generating one LC map.

In conclusion, the reimplemented pipeline achieves a substantial reduction in runtime at the cost of increased memory usage. This trade-off may be beneficial in modern computational environments where memory is less of a bottleneck than time, especially when scaling to multi-map inference.

For completeness, Figure 60 presents a report similar to the previous comparison, with the key difference that the reimplemented pipeline was configured to generate **four** LC maps in a single run. This was achieved by specifying multiple SVM configurations simultaneously via the `svms_to_use` parameter.

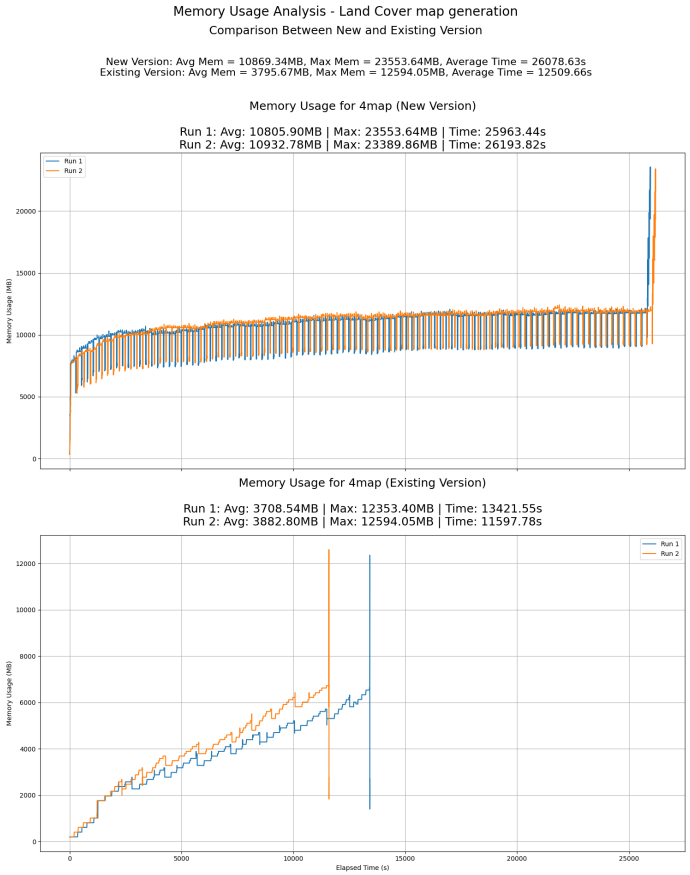


Fig. 60: Comparison of memory usage and execution time between the reimplemented pipeline (producing 4 LC maps in a single execution) and the existing pipeline (generating one map per run).

The total runtime of the reimplemented version is nearly doubled compared to the single-map case, reaching an average of about 7.2 hours. However, it remains more efficient than running the existing pipeline four times, which takes approximately 13.9 hours in total.

In terms of memory, the reimplemented pipeline maintains a similar footprint to the single-map scenario, averaging around 10.9 GB and peaking near 23.5 GB. This confirms that the memory usage does not scale linearly with the number of maps, thanks to the underlying reuse of shared data structures and models during execution.

In conclusion, this test highlights the scalability of the reimplemented component and its ability to handle multiple SVM configurations efficiently within a single execution cycle.

D. Final Land Cover Maps

To illustrate the results of the classification component, this section presents the final LC maps generated for tile `21KUQ` (January 2019). Each map includes a visualization of the classified scene and the corresponding distribution of predicted

classes. These maps provide a qualitative comparison of the classification performance across the different configurations. Four different maps are shown in the next page, corresponding to the following configurations:

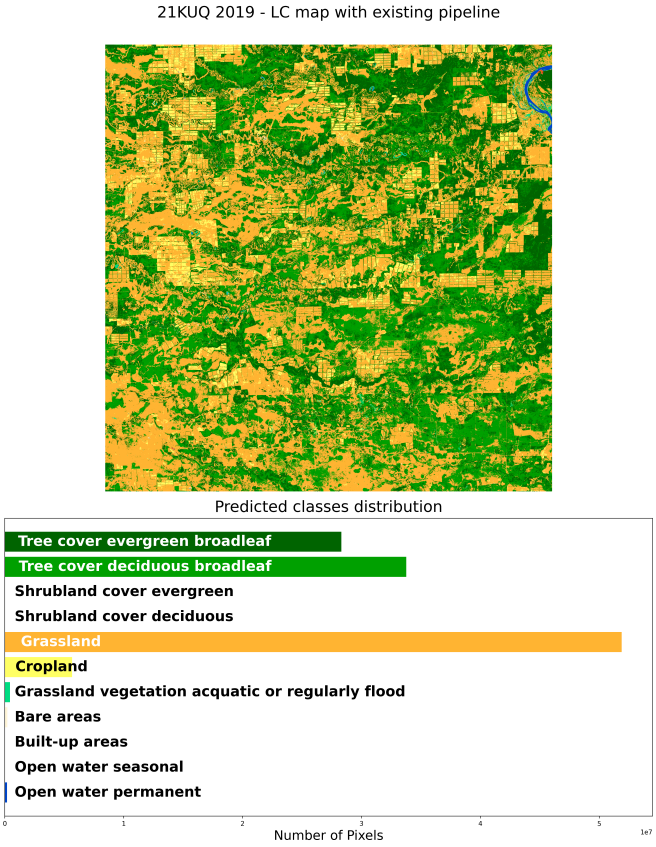
- LC map produced by the **existing pipeline** (Figure 61a);
- LC map produced by the reimplemented pipeline using the **best multiclass SVM** (Figure 61b);
- LC map produced by the reimplemented pipeline using the **best multiclass SVM with calibration** (Figure 61c);
- LC map produced by the reimplemented pipeline using the **best binary SVMs** (Figure 61d)

These LC maps have been generated by applying, for each configuration, the corresponding SVM model trained with the best-performing parameter settings identified during the training and evaluation phase. In particular, for the **existing pipeline**, the SVM model corresponds to the configuration that achieved the highest macro F1-score on the available test set. Similarly, for each SVM configuration in the **reimplemented pipeline** (multiclass, calibrated multiclass, and binary), the model and parameters that achieved the best performance during grid search and validation have been selected and used to generate these final LC maps.

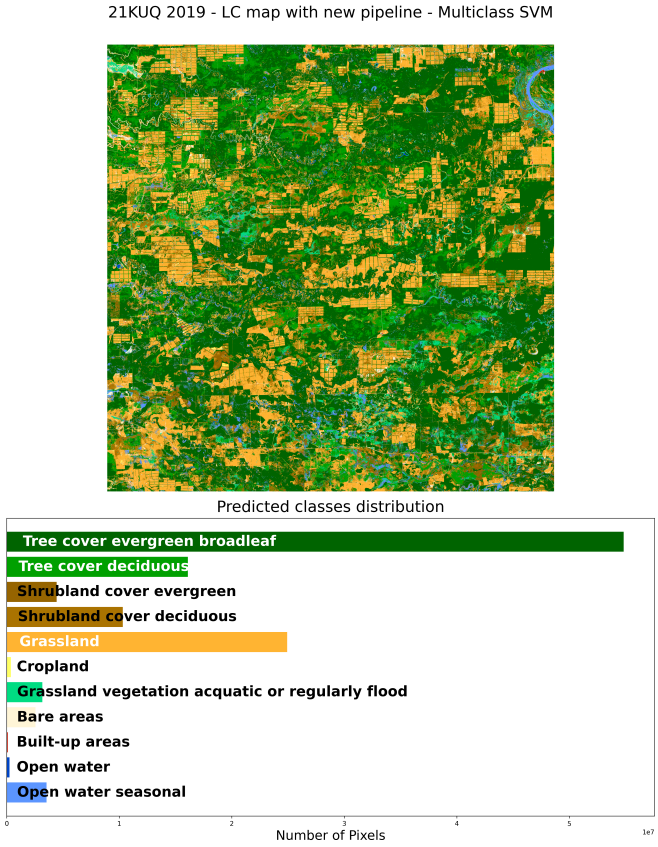
1) Qualitative discussion: The four LC maps reveal differences in the distribution of predicted land cover classes over tile `21KUQ`, located in the south-eastern Amazon region. This area should in theory presents a heterogeneous landscape composed of `Tree cover evergreen broadleaf` (primary forest), `Grassland` and `Cropland` areas resulting from land use change, smaller patches of `Shrubland`, and water bodies.

The map generated by the **existing pipeline** shows a predominance of `Grassland`, while the maps produced with the **reimplemented pipeline** exhibit a broader representation of `Tree cover`, `Shrubland`, and `Open water` classes. Differences are also visible in the spatial patterns of minority classes, such as `Built-up areas` and `Open water seasonal`.

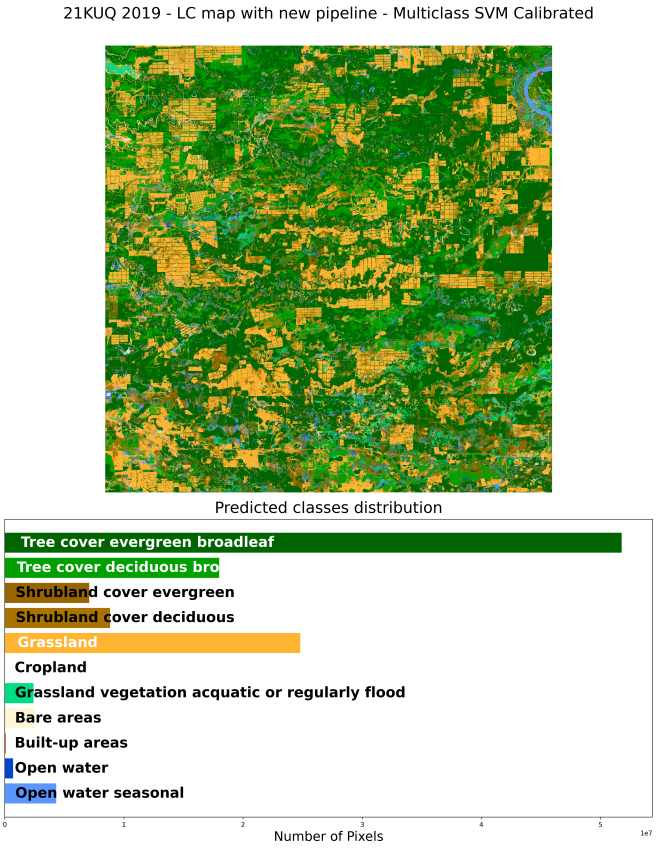
Variations among the maps obtained with different SVM configurations (multiclass, calibrated multiclass, and binary SVMs) mainly affect the representation of minority classes. It should be noted that a complete reference annotation for the entire tile is not available; therefore, these comparisons are intended to provide a qualitative overview of the differences between the generated LC maps.



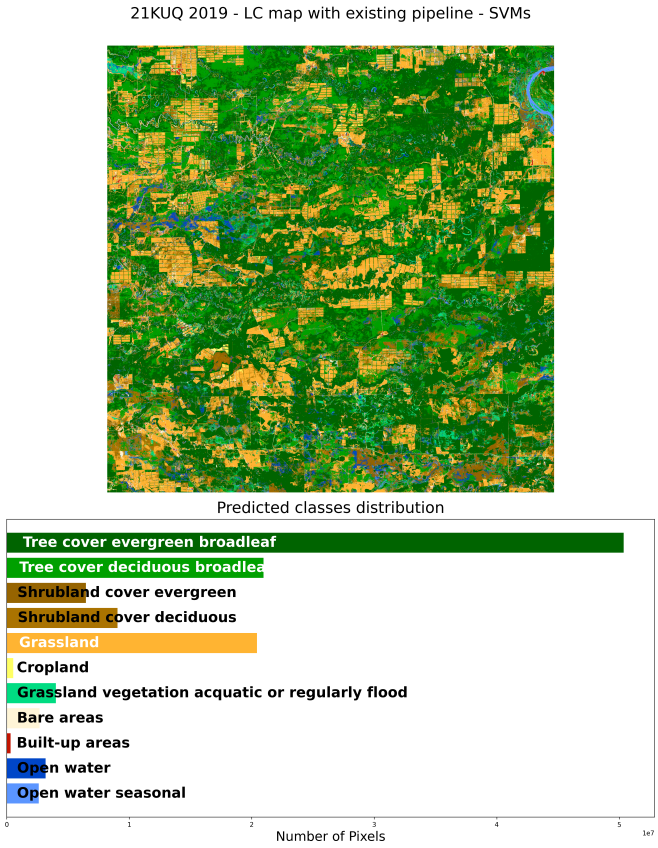
(a) Land Cover Map generated by the **existing pipeline**.



(b) Land Cover Map generated with **Multiclass SVM**.



(c) Land Cover Map generated with **Multiclass SVM Calibrated**.



(d) Land Cover Map generated with **Binary SVMs**.

Fig. 61: Comparison of Land Cover Maps generated with the existing and reimplemented pipelines.

X. Conclusion

This project is my first experience working with radar-related technologies and the processing of multispectral satellite data. The primary objective was to reimplement a pipeline originally developed for land cover classification, redesigning it using more modern, modular and scalable tools. The reimplemented system was built around `xarray` and `dask`, two libraries that support parallel and memory-efficient computation over large datasets. The new pipeline replicates and extends all core functionalities of the original one, including sen2cor masks refinement, preprocessing of Sentinel-2 composites, feature extraction and selection, SVM model training and evaluation, and the generation of final land cover maps.

The development process involved several phases. The first was dedicated to understanding and restoring the original pipeline, which required analyzing a series of existing scripts, many of which were a bit disorganized. To make them executable and inspect their behavior, I had to clean and refactor them, both to understand their logic and to establish a benchmark for comparison with the new implementation. This step proved essential, especially for replicating components such as the Sen2Cor mask refinement and the land cover classification workflow, whose performance was then compared to the performance of the same reimplemented components. In both cases, the performance of the reimplemented operations was faster, in term of time taken.

Working with large and heterogeneous data pushed me to deeply explore parallel and memory intensive computation strategies. Libraries like `xarray` and `dask` were challenging to use at first, particularly due to the need to balance computational efficiency with memory constraints. Many of the methods I initially implemented were rewritten and refined multiple times as I gained experience with these tools and understood how to structure computations more effectively. Managing parallel execution and memory-aware processing became a central aspect of the pipeline design, especially in the masks refinement, composites preprocessing and land cover map generation phases.

Despite the technical challenges, I believe the final outcome represents a robust and extensible base. While the SVM-based classification results of the new pipeline are slightly lower in performance compared to those produced by the original system, it is important to consider that the two approaches are based on different processing chains and modeling strategies, making a direct comparison difficult. Nevertheless, the new pipeline offers improved modularity and clarity, enabling faster experimentation and easier integration of new techniques.

From a broader perspective, the work carried out here goes beyond a simple software reimplementation. It was an opportunity to deeply engage with geospatial data processing, machine learning applied to remote sensing and the design of scalable data pipelines. The experience taught me how to approach complex codebases, adapt to unfamiliar tools, and reason about performance in real-world, data-intensive environments.

Finally, I would like to thank engineer Gianmarco Perantoni, who guided me throughout the entire project. His support was fundamental, both in clarifying technical aspects and in encouraging a methodical approach to software development. Thanks to his mentorship, I also acquired development practices and analytical skills that will undoubtedly be useful in future research and professional contexts.

References

[1] ESA CCI HRLC Team, "Algorithm Theoretical Basis Document (ATBD)," European Space Agency (ESA), Tech. Rep. CCI_HRLC_Ph1-D2.2, 2022, accessed: 2025-03-11. [Online]. Available: https://climate.esa.int/media/documents/CCI_HRLC_Ph1-D2.2_ATBD_v4.0.pdf

[2] Sentinel Hub, "Sentinel-2 level-2a data," https://docs.sentinel-hub.com/api/latest/data/sentinel-2-l2a/, 2025, accessed: 2025-03-11.

[3] Xarray Development Team, "Xarray documentation," https://docs.xarray.dev/, 2025, accessed: 2025-03-11.

[4] Dask Development Team, "Dask documentation," https://docs.dask.org/, 2025, accessed: 2025-03-11.

[5] H. Zhai, H. Zhang, L. Zhang, and P. Li, "Cloud/shadow detection based on spectral indices for multi/hyperspectral optical remote sensing imagery," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 144, pp. 235–253, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271618301989

[6] A. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics Tools*, vol. 9, 01 2004.

[7] Visioterra, "Digital elevation model for sentinel-2," https://visioterra.fr/web/Digital-Elevation-Model-for-Sentinel-2-1935?lang=en, 2025, accessed: 2025-03-12.

[8] C. Aybar, L. Bautista, D. Montero, J. Contreras, D. Ayala, F. Prudencio, J. Loja, L. Ysuhuaylas, F. Herrera, K. Gonzales, J. Valladares, L. A. Flores, E. Mamani, M. Quiñonez, R. Fajardo, W. Espinoza, A. Limas, R. Yali, A. Alcántara, M. Leyva, R. Loayza-Muro, B. Willems, G. Mateo-García, and L. Gómez-Chova, "Cloudsen12+: The largest dataset of expert-labeled pixels for cloud and cloud shadow detection in sentinel-2," *Data in Brief*, vol. 56, p. 110852, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340924008163

[9] Sentinel Hub, "Normalized difference vegetation index (ndvi)," https://custom-scripts.sentinel-hub.com/sentinel-2/ndvi/, accessed: 2025-04-01.

[10] ——, "Green normalized difference vegetation index (gndvi)," https://custom-scripts.sentinel-hub.com/sentinel-2/gndvi/, accessed: 2025-04-01.

[11] A. A. Gitelson and M. N. Merzlyak, "Remote sensing of chlorophyll concentration in higher plant leaves," *Advances in Space Research*, vol. 22, no. 5, pp. 689–692, 1998, synergistic Use of Multisensor Data for Land Processes. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117797011332

[12] Sentinel Hub, "Normalized difference yellow index (ndyi)," https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/ndyi/, accessed: 2025-04-01.

[13] ——, "Enhanced vegetation index 2 (evi2)," https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/evi2/, accessed: 2025-04-01.

[14] E. R. Hunt Jr., C. S. T. Daughtry, J. U. H. Eitel, and D. S. Long, "Remote sensing leaf chlorophyll content using a visible band index," *Agronomy Journal*, vol. 103, no. 4, pp. 1090–1099, 2011. [Online]. Available: https://acsess.onlinelibrary.wiley.com/doi/abs/10.2134/agronj2010.0395

[15] Sentinel Hub, "Soil adjusted vegetation index (savi)," https://custom-scripts.sentinel-hub.com/sentinel-2/savi/, accessed: 2025-04-01.

[16] ——, "Normalized difference moisture index (ndmi)," https://custom-scripts.sentinel-hub.com/sentinel-2/ndmi/, accessed: 2025-04-01.

[17] L. Serrano, J. Peñuelas, and S. L. Ustin, "Remote sensing of nitrogen and lignin in mediterranean vegetation from aviris data: Decomposing biochemical from structural signals," *Remote Sensing of Environment*, vol. 81, no. 2, pp. 355–364, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425702000111

[18] Sentinel Hub, "Normalized difference water index (ndwi)," https://custom-scripts.sentinel-hub.com/sentinel-2/ndwi/, accessed: 2025-04-01.

[19] Y. Zha, J. Gao, and S. Ni, "Use of normalized difference built-up index in automatically mapping urban areas from tm imagery," *International Journal of Remote Sensing - INT J REMOTE SENS*, vol. 24, pp. 583–594, 02 2003.
[20] Sentinel Hub, "Normalized difference snow index (ndsi)," https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/ndsi/, accessed: 2025-04-01.
[21] ——, "Barren soil index (bsi)," https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/barren_soil/, accessed: 2025-04-01.
[22] SciPy, "Sobel filter," https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.sobel.html, accessed: 2025-04-01.
[23] ESA, "Esa tiling system for sentinel-2," https://hls.gsfc.nasa.gov/products-description/tiling-system/, accessed: 2025-04-01.
[24] Wikipedia contributors, "Leakage (machine learning) — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Leakage_(machine_learning), 2024, accessed: 2025-03-31.
[25] scikit-learn developers, "sklearn.feature_selection.rfecv," https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html, 2024, accessed: 2025-04-01.
[26] ——, "sklearn.feature_selection.mutual_info_classif," https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html, 2024, accessed: 2025-04-01.
[27] ——, "sklearn.decomposition.pca — scikit-learn documentation," https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, 2024, accessed: 2025-04-02.
[28] ——, "Gridsearchcv — scikit-learn documentation," https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html, accessed: 2025-04-02.
[29] ——, "Calibratedclassifiercv — scikit-learn documentation," https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html, accessed: 2025-04-02.
[30] ——, "classification_report — scikit-learn documentation," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html, accessed: 2025-04-02.
[31] ——, "scikit-learn: Source code for _calibratedclassifier," https://github.com/scikit-learn/scikit-learn/blob/6e9039160f0dfc3153643143af4cfdca941d2045/sklearn/calibration.py#L739C1-L765C21, 2024, accessed: 2025-04-02.
[32] ——, "sklearn.calibration.calibrationdisplay," https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibrationDisplay.html, 2024, accessed: 2025-04-02.