Exercise 6
Machine Learning with Google Earth Engine

Geomatics and Geoinformation
M.Sc. in Data Science

Cristiana Di Tullio

A.Y. 2023-2024

Introduction

What is Machine Learning?

"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."
- Arthur Samuel, 1959

Machine Learning (ML) is a branch of Artificial Intelligence that focuses on developing statistical algorithms capable to automatically learn from data, detecting hidden patterns and relations between features, to the ultimate goal of making their own predictions and gradually improve their performance. Machine Learning implements an intrinsically data-driven approach, training models on historical datasets and predicting outcomes for new, unseen data points. The prediction performance is then evaluated by means of several metrics, ranging from scalar scores like accuracy, precision, and recall to more complex indicators like the Area Under the ROC Curve (AUC). The ability to learn autonomously from data, with no need for explicit instructions, is what most differentiates Machine Learning from traditional computer programming, that typically adopts a rule-based approach where humans manually define the logic for the programs to make decisions.

Artificial Intelligence, Machine Learning and Deep Learning

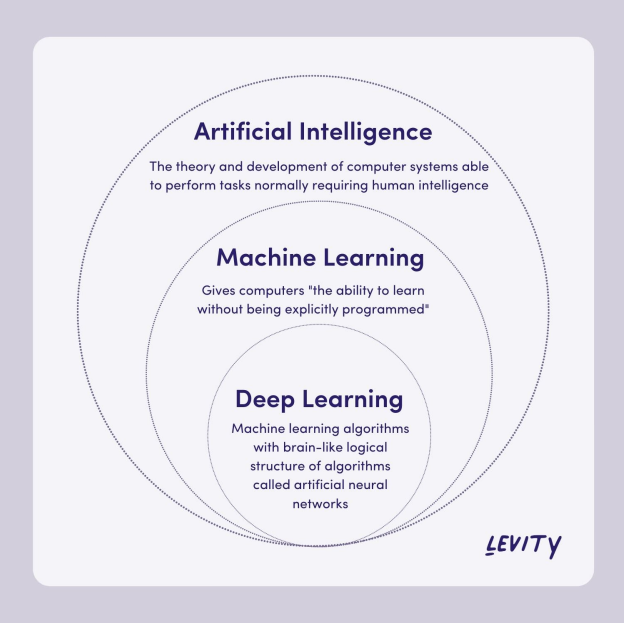

Figure 1: Artificial Intelligence, Machine Learning and Deep Learning relationship (source: Levity).

Machine Learning, Deep Learning and Artificial Intelligence are terms often used interchangeably, and although they aren't independent of each other they refer to different concepts. A simple depiction of their interaction is shown in Figure 1. Deep Learning has the important advantage of integrating feature engineering directly inside its learning process, with no need to carry it out manually as still happens in Machine Learning. While it is worth mentioning that Deep Learning architectures have recently obtained impressive results in Remote Sensing Change Detection (RSCD) research, their employment in Google Earth Engine is still limited and beyond our exercise's purposes.

Supervised vs Unsupervised Learning

Machine Learning relies on powerful models to carry out different kinds of tasks. In general, ML algorithms can be distinguished into three categories according to the learning method they implement:

- ***Supervised Learning***: the algorithms are trained on labeled data, i.e. they are provided with a dataset for the specific purpose of training and they learn by mapping input features to their corresponding target output. After the training phase, the models are able to generalize on previously unseen data. The performance evaluation is typically carried out on a validation or test dataset.
  Examples of Supervised Learning tasks are regression and classification.

- ***Unsupervised Learning***: the algorithms learn to recognize underlying patterns and distributions directly in the input data. There is no training phase. There are different heuristics to evaluate the performance of unsupervised learning, depending on the specific algorithms and task.
  Examples of Unsupervised Learning tasks are clustering, dimensionality reduction and association rules.

- ***Semi-supervised Learning***: here, techniques from both Supervised and Unsupervised Learning are combined for use cases in which a sufficient amount of labeled data are too difficult to acquire for some reason. Often, these algorithms are trained on datasets that also incorporate unlabeled data.

In general, choosing one approach or the other depends on the specific use case, the amount and the structure of available data. Integrating Machine Learning models in data analysis contexts offers several benefits, such as the ability to uncover complex patterns in high dimensional data and scale to large number of parameters, therefore this technology finds applications in many diverse fields, including remote sensing imaging.

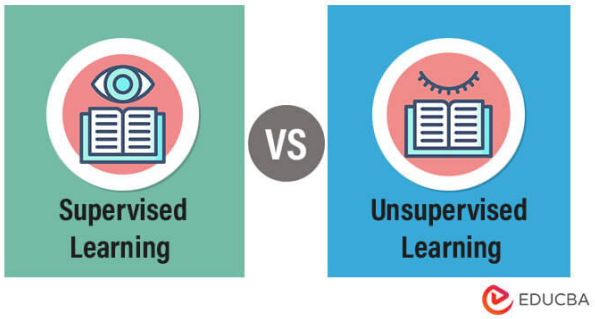

Figure 2: Supervised vs Unsupervised Learning approaches (source: EDUCBA).

Classification in Remote Sensing applications

In our particular field of study, "classification" is an improper term that refers rather to the notion of semantic segmentation in Data Science. This computer vision task takes an input an image and maps each pixel to a distinct category. In remote sensing image analysis, it means translating the colour observed in each pixel (represented as a reflectance value) into a thematic class that describes the dominant type of its land cover (like water, vegetation, desert, urban areas, etc.). Associating pixels to classes and reflectance values to land cover classes, the image becomes a *thematic map*. This turns out useful in each application that includes studying a geographical area in a certain period or monitoring the evolution of land use patterns across time. Within Google Earth Engine (GEE), standard Machine Learning algorithms are implemented by a specific API in the packages `ee.Classifier` for Supervised Classification, `ee.Clusterer` for Unsupervised Classification and `ee.Reducer` for Regression.

Decision Trees and Random Forests

Before introducing Random Forests, it is essential to recall what is a ***Decision Tree*** and how it works: it is a non-parametric Supervised Learning algorithm that employs a (generally binary) tree-like structure to build a flowchart of decisions. Each internal node of the tree tests a variable and the possible outcomes initiate a branching process that finally leads to a class label in the tree leaves. The branching process corresponds to a partitioning of the data space, as depicted in Figure 3. Every path from the root to a leaf represents a classification rule. Decision Trees and the models built on them can be used for both classification and regression tasks.

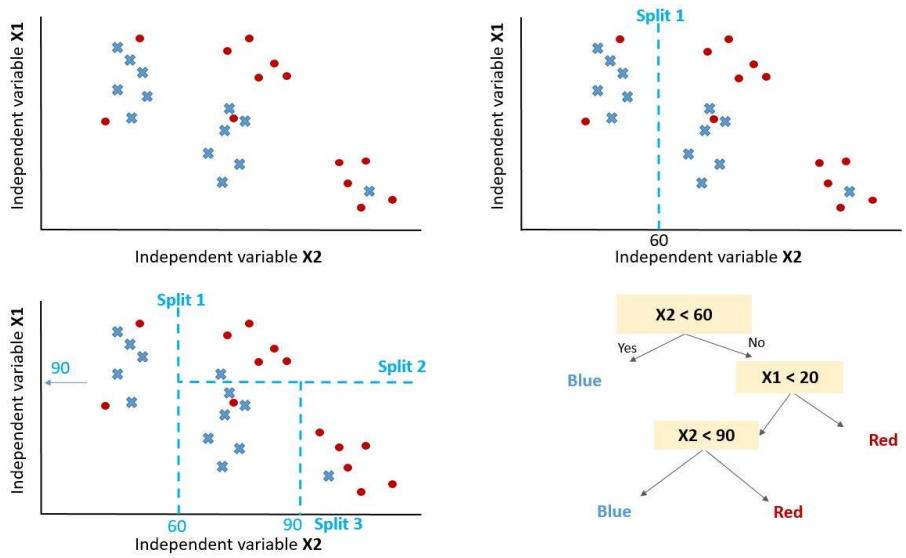

Figure 3: Branching process of a Decision Tree (source: Towards Data Science.)

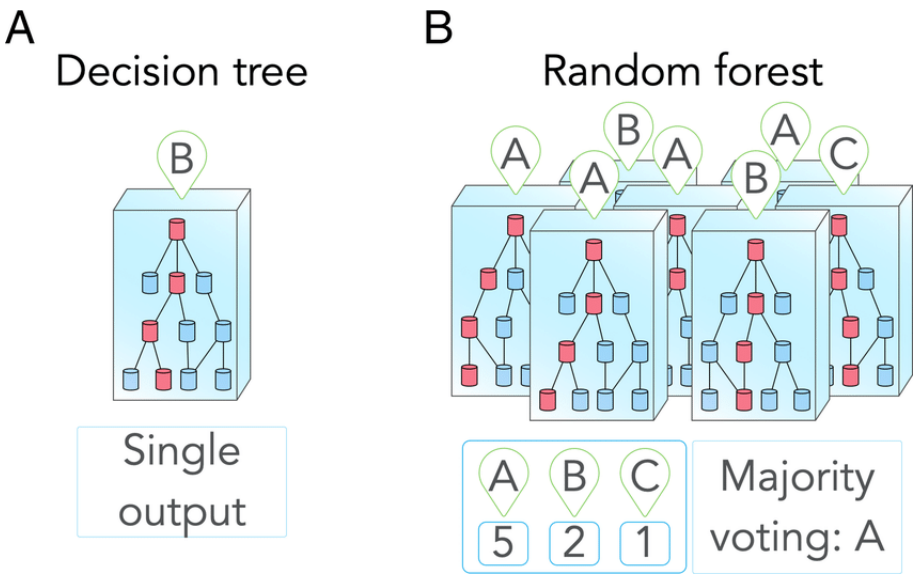

Figure 4: Decision Trees compared to Random Forests (source: ResearchGate).

Random Forest is a powerful ensemble algorithm that puts together the effort of a pre-defined number of individual Decision Trees, as shown in Figure 4. For each data point, the class prediction of every decision tree is collected and the final

model category is chosen by means of majority voting of all the predictions. Decision Trees, and therefore Random Forests, are ideal for *multi-class classification* as they allow to branch decisions in multiple outputs. Easy to interpret and explain, this model is highly popular in Data Science applications because of its simplicity and effectiveness. The number of trees to use is one of the parameters of the model.

Performance evaluation metrics

The ***Confusion Matrix*** is a standard quantitative method of evaluating classification performances. It allows to measure and compare the results of classification prediction with respect to the ground truth, in our case represented by the ESA World Cover dataset as we will see in the exercise implementation. There are four possible types of classification prediction, as shown in Figure 5:

- **True Positive (TP)**: correctly predicted positive class;
- **True Negative (TN)**: correctly predicted negative class;
- **False Positive (FP)**: incorrectly predicted positive class;
- **False Negative (FN)**: incorrectly predicted negative class.

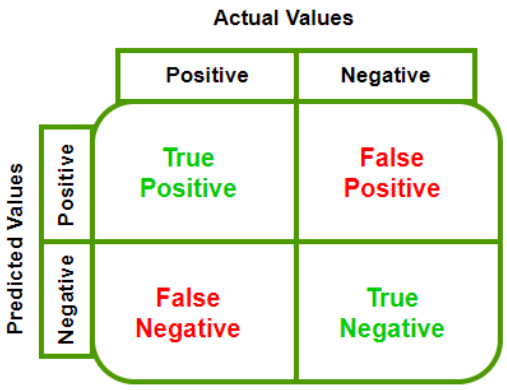

Figure 5: Confusion Matrix in the binary case (source: GeeksforGeeks).

While this simple example shows a 2×2 matrix, it's possible to use the confusion matrix in the multi-class case as well, just adding a row and a column for each additional class. In the general case, the number of all correct predictions will be contained on the diagonal elements. Ideally, a diagonal matrix would mean perfect classification. Based on the confusion matrix, several important metrics are defined to measure a different aspect of classification performance.

The most important are:

- ***Accuracy***: measures the amount of overall correct predictions.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- ***Precision***: measures the amount of the truly positive values, out of all the positive predictions.

$$precision = \frac{TP}{TP + FP}$$

- ***Recall***: measures the amount of the predicted positive, out of all truly positive values.

$$recall = \frac{TP}{TP + FN}$$

- ***F1 score***: measures the performance of the model class by class. It's defined as the harmonic mean of precision and recall, and it's often used in the case of imbalanced datasets.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

Exercise implementation

The aim of this exercise is to classify (segment) the pixels of an aggregated (median) image produced from Sentinel 2 Surface Reflectance images collected in a given region of interest (ROI) for a temporal period of interest.

The algorithm chosen to carry out this exercise is Random Forest.
The solution of this exercise follows the main steps of the workflow shown in Figure 6, excluding the more advanced phases of post processing and parameters tuning:

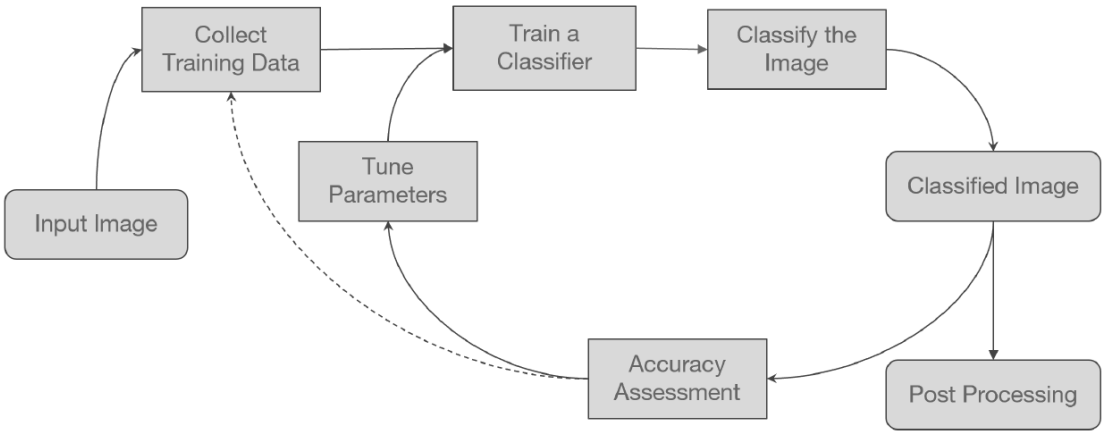

Image source: Copyright ©2021 Ujaval Gandhi: `www.spatialthoughts.com`

Figure 6: Typical workflow of a Supervised Learning process.

The implementative process is reported below:

1. Choosing a Region Of Interest (ROI): the region I have chosen is that of the Bosphorus Strait, which includes an urbanized part of Istanbul as well as vegetation areas on the land regions at its sides. In between Thrace and Anatolia, it divides the European and Asian continents. The ROI is shown in Figure 7;

2. Importing the Sentinel-2 MSI: MultiSpectral Instrument, Level-2A raw image collection containing Surface Reflectance values. After some filtering and preparation, this will be the base for the aggregate image that we want to give the model as input for classification;

3. Filtering the collection by chosen ROI and temporal period of interest: we extract all images collected between 1^{st} January 2021 and 1^{st} January 2024,

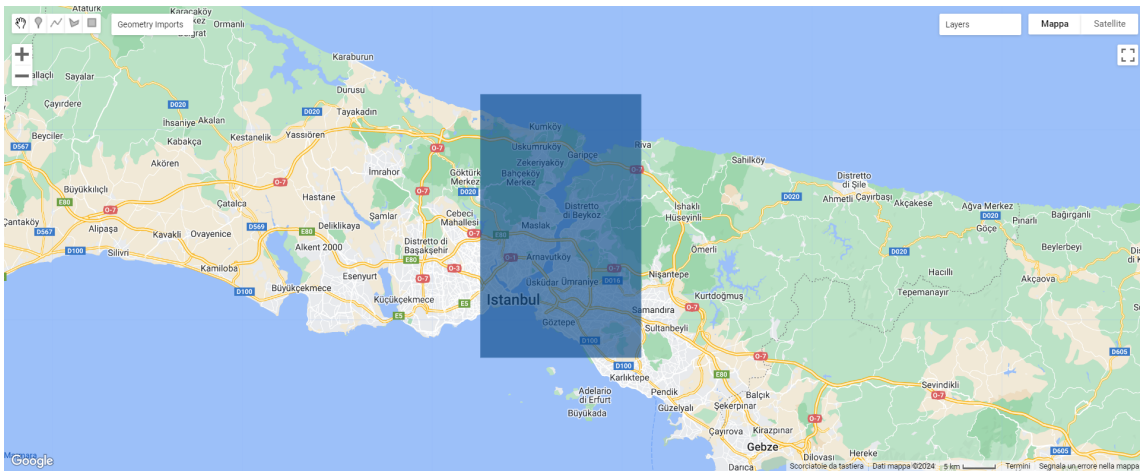

Figure 7: Selected Region Of Interest (ROI).

for a total of 3 years. Furthermore, we apply a filtering by property to exclude the images covered by more than 50% clouds through the attribute `CLOUDY_PIXEL_PERCENTAGE`. After the filtering, the retrieved images are 221;

4. The next step is cloud masking. Usually, we would do it using the pre-defined function found in the dataset web page, that uses the Sentinel-2 QA band. However, during the implementation turned out that this function leads to quite poor results, leaving behind a lot of unmasked clouds. So, in order to do better we use the Cloud Score+ S2 HARMONIZED V image collection, a dataset provided by Google with Deep Learning techniques. This dataset includes specific bands that allow for precise masking. We can use the pre-defined functions found at this web page to mask the clouds and also apply the radiometric scaling factor of 0.0001 get the reflectance data back to its original floating point value;

5. Aggregating the filtered collection into a single image by means of the `ee.median` method: it reduces an image collection by calculating the median of all values at each pixel across the stack of all matching bands. After the median, to reduce the computational load we only select the bands that contain specifically surface reflectance values. We can now visualize the true color composite (R, G, B) and a false color composite (SWIR, NIR, G) as an alternative representation of our area of interest, shown in Figure 8 and 9 respectively;

6. Importing the ESA WorldCover 10m v200 2021 product: with 11 land cover classes at 10 m resolution, shown in Figure 10 together with the color palette, this global map will help us building the training and validation datasets by providing a source of ground truth labels;

7. Remapping the 11 land cover classes with numbers in the range $0 - 10$ for internal computer reasons and checking visually that everything is working properly;

8. Preparing the data points that will later constitute the training and validation datasets: we need a new dataset, a new image containing all the bands of the aggregated median image plus one that is the Land Cover (LC) label. It is provided by the just imported ESA dataset. The new image is created with a stratified sample that collects a predefined number of points for each class

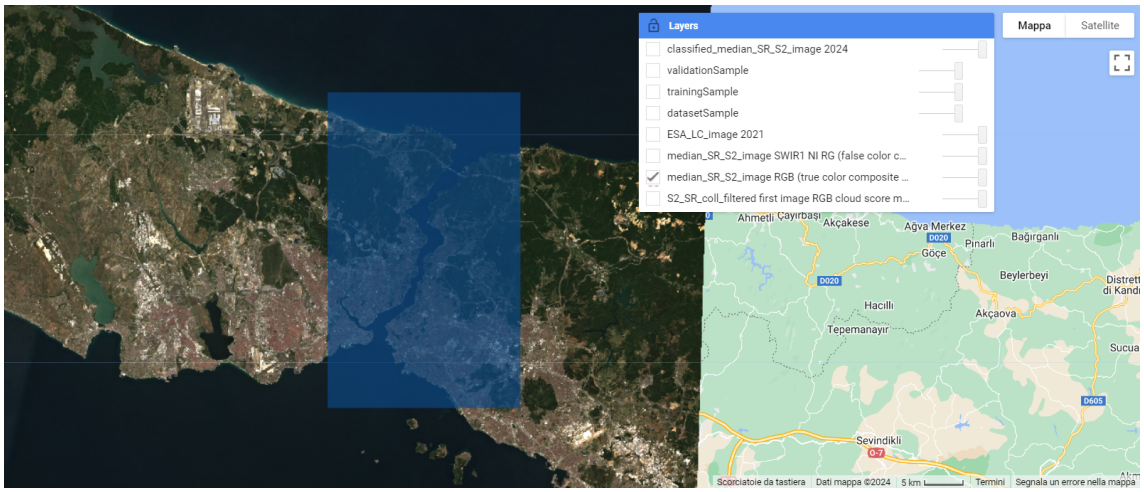

Figure 8: Median image visualized with true color composite (R-G-B).

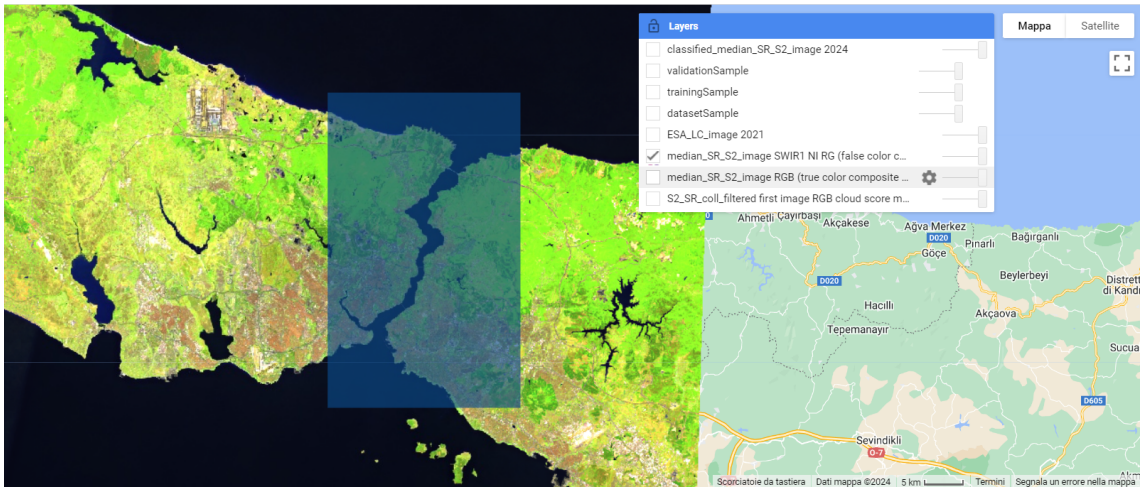

Figure 9: Median image visualized with false color composite (SWIR1-NIR-G).

Map Class Table

Value	Color	Description
10	#006400	Tree cover
20	#ffbb22	Shrubland
30	#ffff4c	Grassland
40	#f096ff	Cropland
50	#fa0000	Built-up
60	#b4b4b4	Bare / sparse vegetation
70	#f0f0f0	Snow and ice
80	#0064c8	Permanent water bodies
90	#0096a0	Herbaceous wetland
95	#00cf75	Mangroves
100	#fae6a0	Moss and lichen

Figure 10: Land cover classes contained in the ESA dataset.

value contained in the Land Cover feature. After a few trials, I found that 240 was the maximum number of points that could be collected in my case without running into a computational error or timeout. It is interesting to notice that, according to the sample definition, the dataset should contain $240 \times 11 = 2640$ points (the number of samples per class times the number of classes). Although, just 1920 points are collected. This can be due to various reasons, maybe an insufficient number of pixels for some classes in the region of interest (like the category snow and ice) or some pixels have been masked out when dealing with the cloud coverage.

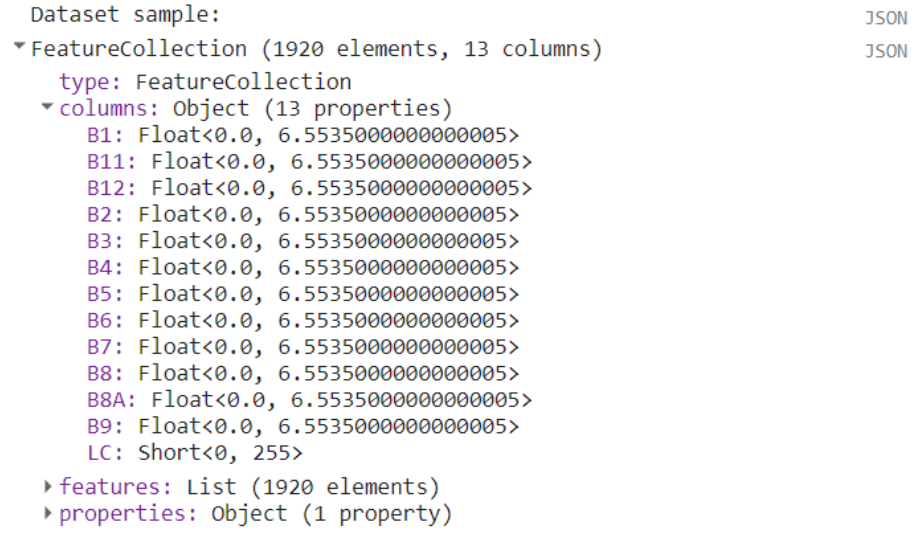

Figure 11: Dataset sample.

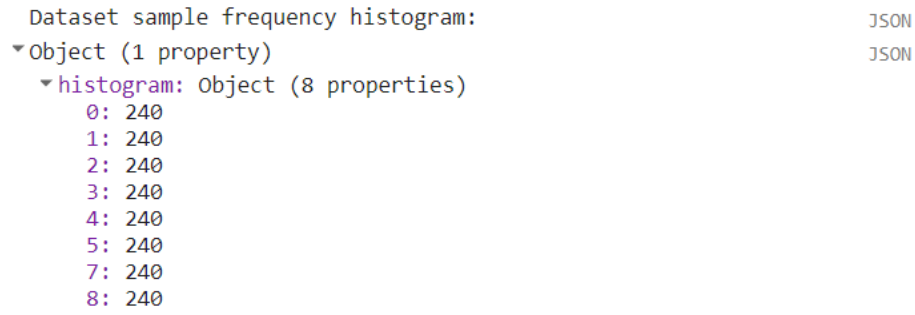

Figure 12: Histogram of classes contained in the dataset sample.

Figures 11 and 12 show the dataset features, specifying the number of elements, properties and histogram of classes frequency. As it appears, there are just 8 land cover classes in the area of our interest out of all the possible 11: class 6 (snow and ice), 9 (mangroves) and 10 (moss and lichen) are missing;

9. Splitting the dataset into training and validation sets. Precisely 80% of data will be dedicated to training and 20% to validation. The split is performed applying the function `randomColumn` that adds to each observation a float number, representing a uniform (by default) probability. The new feature is visible in Figure 13. At this point, the points will be assigned to training or validation dataset simply depending on the probability value being under or above the threshold 0.8 (i.e. 80%);

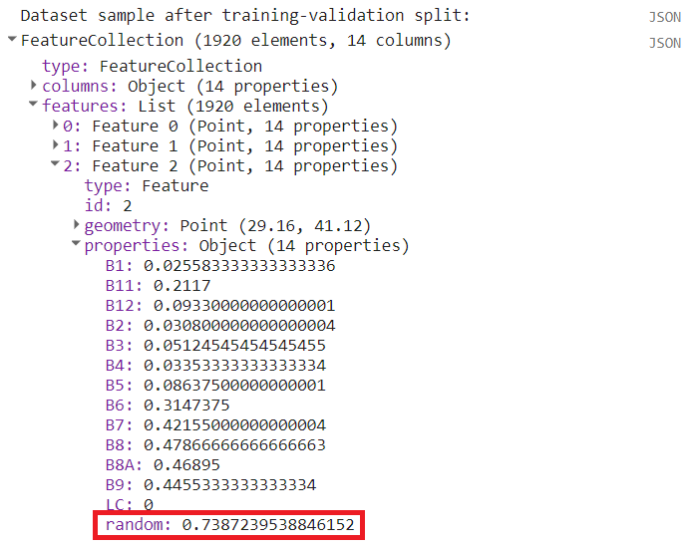

Figure 13: Data point with new feature. Here, the probability is lower than 0.8 so this emelent will be assigned to the training dataset.

10. Start of the Machine Learning part: first, I define the Random Forest classifier with the only specified parameter `number_of_trees = 100`;

11. Training the model to classify the images contained in the training dataset, with the images band names as input features and the Land Cover label as target variable to predict;

12. Computing the confusion matrix and accuracy on the *training* dataset. We expect to obtain a very high score, since we are evaluating the model on the data points it has been trained on. In fact, we retrieve an almost diagonal matrix with an accuracy of 0.9961265332472563, see Figure 15. The validation accuracy will surely be lower, but also more representative of the true capabilities of the model on unseen data. Steps from 10 to 12 are shown in Figure 14;

```
// STEP 10: set up Random Forest model
// Initialize parameter
var number_of_trees = 100;
// Define the idle classifier
var RF_classifier = ee.Classifier.smileRandomForest(number_of_trees);


// STEP 11: train the model
var trained_classifier = RF_classifier.train({
    features: trainingSample,                          // training dataset
    classProperty: 'LC',                               // our classification target (LC column)
    inputProperties: median_SR_S2_image.bandNames()    // input features for classification
});

print('Input features for classification:', median_SR_S2_image.bandNames());
print('Trained classifier, explained:', trained_classifier.explain());


// STEP 12: inspect training confusion matrix
var training_confusion_matrix = trained_classifier.confusionMatrix();

print('Training confusion matrix:', training_confusion_matrix);
print('Training accuracy:', training_confusion_matrix.accuracy());
```

Figure 14: Code to define and train the Random Forest classifier.

```
Training confusion matrix:                                    JSON
▼ List (9 elements)                                           JSON
  ▶ 0: [202,0,0,0,1,0,0,0,0]
  ▶ 1: [1,187,0,0,0,0,0,0,0]
  ▶ 2: [0,1,191,2,0,0,0,0,0]
  ▶ 3: [0,0,0,199,0,0,0,0,0]
  ▶ 4: [0,0,0,0,191,0,0,0,0]
  ▶ 5: [0,0,0,0,1,188,0,0,0]
  ▶ 6: [0,0,0,0,0,0,0,0,0]
  ▶ 7: [0,0,0,0,0,0,0,197,0]
  ▶ 8: [0,0,0,0,0,0,0,0,188]
```

Figure 15: Confusion matrix relative to training dataset.

13. Validating the model: now, the classifier is applied to the data points contained in the validation training set, that the model hasn't seen before. This way we can evaluate its generalization performance. The code is contained in Figure 17;

```
// STEP 13: validate the model
var classified_validation = validationSample.classify(trained_classifier);

print('Validation classified:');
print(classified_validation);


// STEP 14: inspect validation confusion matrix
var validation_confusion_matrix = classified_validation.errorMatrix('LC', 'classification');

print('Validation confusion matrix:', validation_confusion_matrix);
print('Validation accuracy:', validation_confusion_matrix.accuracy());
```

Figure 16: Model validation code.

14. Computing the confusion matrix and accuracy on the *validation* dataset. Now, the accuracy is 0.6684636118598383. A much worse result, as we expected. The matrix is shown in Figure 17 and we can see that it contains several mis-classified values out of the diagonal;

```
Validation confusion matrix:                                  JSON
▼ List (9 elements)                                           JSON
  ▶ 0: [29,3,3,1,1,0,0,0,0]
  ▶ 1: [1,41,6,4,0,0,0,0,0]
  ▶ 2: [0,7,10,13,7,6,0,0,3]
  ▶ 3: [1,2,16,19,2,1,0,0,0]
  ▶ 4: [1,0,6,3,30,8,0,0,1]
  ▶ 5: [0,0,2,1,10,38,0,0,0]
  ▶ 6: [0,0,0,0,0,0,0,0,0]
  ▶ 7: [0,0,0,0,0,0,0,43,0]
  ▶ 8: [2,3,2,0,2,2,0,3,38]
```

Figure 17: Confusion matrix relative to validation dataset.

15. Finally, it's time to classify our aggregated median image. The code is contained in Figure 18. The classification result is visualized immediately below, in Figure 19 and 20, together with the ground truth labels provided by the ESA dataset. Overall, the classification appears correct, even though the classes are a bit more "messy" than the original ones;

```
// STEP 15: apply the classifier to our input image
var classified_median_SR_S2_image = median_SR_S2_image.classify(trained_classifier);
// Visualize classification predictions
Map.addLayer(classified_median_SR_S2_image, {min:0, max:10, palette: LC_palette}, 'classified_median_SR_S2_image 2024');

// Checking performances

// Sample ESA WorldCover dataset within the borders of our ROI
var referenceSample = ESA_LC_image.sample({
    scale: 10,
    region: geometry,
    geometries: true,
    tileScale: 8,      // increased tileScale to mitigate computation problems
    numPixels: 1000
});

// Sample the classified image at the location of reference samples
var classified_sampled = classified_median_SR_S2_image.sampleRegions({
    scale: 10,
    collection: referenceSample,
    geometries: true,
    tileScale: 8,
    properties: ['LC']
});

// Compute the confusion matrix
var errorMatrix = classified_sampled.errorMatrix('LC', 'classification');

print('Confusion Matrix:', errorMatrix);
print('Accuracy:', errorMatrix.accuracy());
```

Figure 18: Median image classification and performance evaluation code.

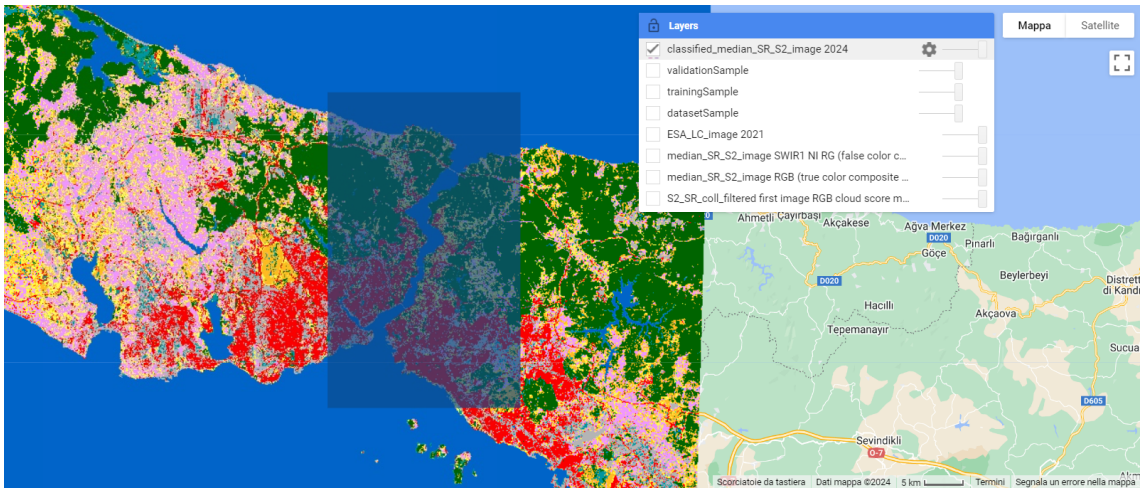

Figure 19: Median image classification result.

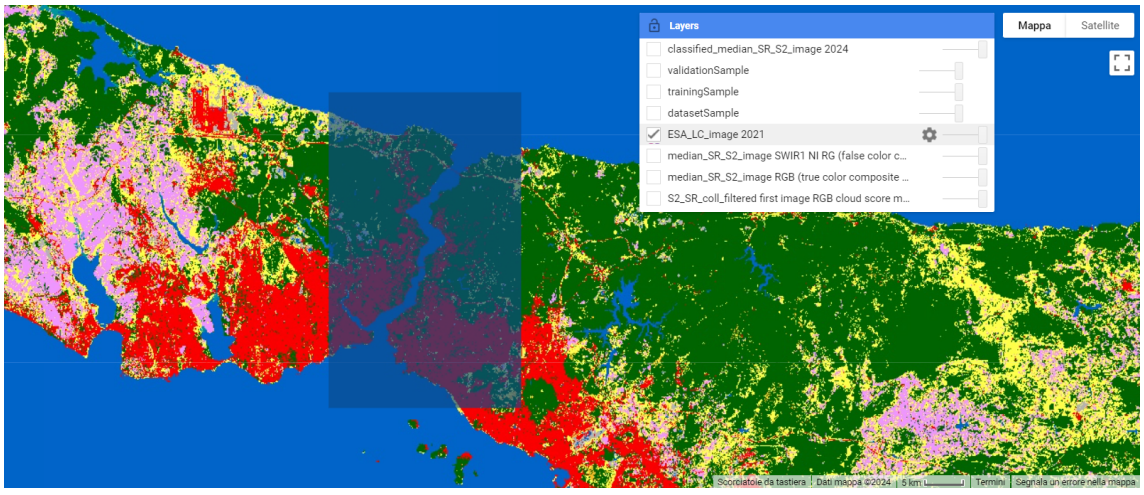

Figure 20: Ground truth ESA classes.

Also the confusion matrix and accuracy have been computed, but with a slight modification with respect to what has been done in class. Since the `errorMatrix` method is only available for FeatureCollection and ImageCollection objects, it cannot be directly used on the result of classification on our aggregated image. Therefore, it was necessary to sample some points from the classified image and compare them with the known land cover labels of the ESA dataset. At this point, having obtained another Feature Collection, `errorMatrix` could be applied again. A specific number of pixels to sample had to be specified in order to realize this idea, and I set it to 1000 as for higher values I always encountered time-out problems. The accuracy obtained this way is not too much significant, since it relates only to a subset of elements, but in my opinion it is still interesting.

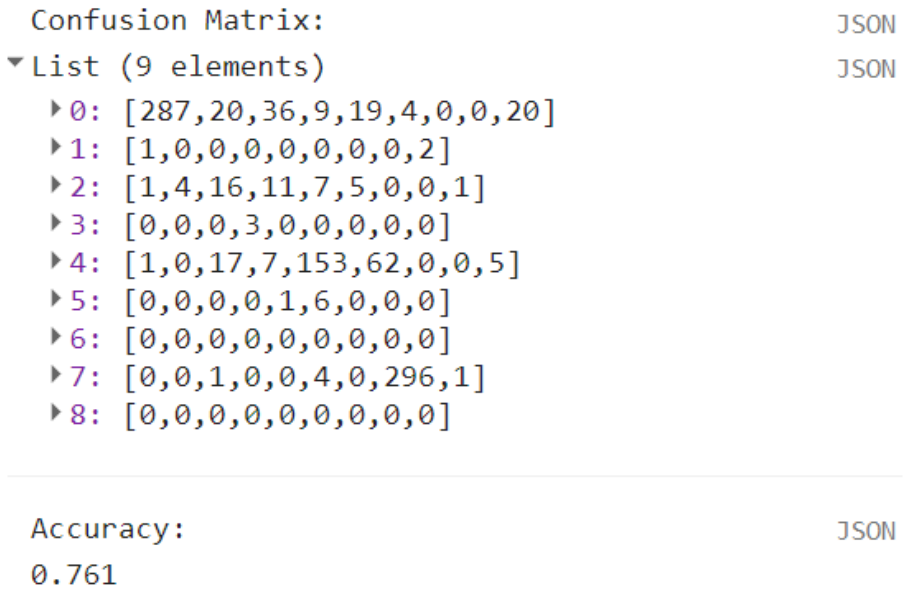

Figure 21: Classified median image confusion matrix and accuracy.

Link to the script

The script is available at GEE Code Editor.