Machine Learning with Google Earth Engine

Pasquale Luca Tommasino
Geomatics and Geoinformation, MSc. in Data Science, Sapienza University of Rome

This report describes a set of exercises carried out on Google Earth Engine with the aim of applying classification techniques to multispectral satellite imagery. The main objective was to identify different land cover classes, such as snow-covered areas or vegetation types, using both unsupervised algorithms (K-means) and supervised ones (Support Vector Machine and Random Forest). Each method was adapted by modifying key parameters such as the geographical area, observation period, number of clusters or decision trees, spectral bands, and dataset size. The results were evaluated through accuracy metrics and visual inspection of the resulting maps, with the goal of verifying model consistency and understanding the impact of technical choices on classification quality.

Keywords: Google Earth Engine, Machine Learning, Supervised Classification, K-means Clustering, Support Vector Machine, Random Forest

I. INTRODUCTION

The use of satellite-based multispectral remote sensing techniques, combined with machine learning algorithms, allows for the automatic extraction of complex information from the Earths surface. In particular, imagery acquired by sensors such as Landsat and Sentinel-2 provides a valuable data source for analyzing land use and monitoring environmental phenomena. The aim of this report is to describe a series of exercises focused on land cover classification using machine learning algorithms implemented in Google Earth Engine (GEE). The activities are divided into two main approaches: unsupervised classification using K-means clustering, and supervised classification using Support Vector Machine (SVM) and Random Forest. For each method, students were asked to select a suitable geographic region, define a time interval of interest, and adjust key algorithm parameters such as the number of clusters or trees, the spectral bands used, and the size of the dataset. In the supervised classification tasks, the ESA WorldCover land cover map was used as a reference layer to train and validate the models. The results were evaluated through classification accuracy, confusion matrices, and visual analysis of class coherence, with the goal of understanding the strengths and limitations of each method in realistic remote sensing scenarios.

II. THEORY

A. Unsupervised Classification: K-means Clustering

K-means clustering is an unsupervised learning technique used to divide a satellite image into groups of pixels with similar spectral characteristics. In the absence of labeled data, the algorithm identifies patterns within the dataset by assigning each observation to one of K clusters based on spectral similarity. Each group is represented by a centroid, and pixels are grouped around it according to the minimum distance in feature space. In the context of remote sensing, this method is particularly useful for segmenting an image into homogeneous regions in terms of land cover. Since the algorithm does not assign semantic meaning to the clusters, it is necessary to interpret the resulting classes visually or with the aid of external reference data. The process involves selecting relevant spectral bands, aggregating the image temporally (e.g., using a median composite), and applying the clustering algorithm to generate a thematic map. (Gandhi 2023, Ravanelli 2024)

B. Supervised Classification: Support Vector Machine

Support Vector Machine (SVM) is a supervised classification algorithm that assigns classes to data based on labeled training examples. Its core principle lies in finding an optimal hyperplane that separates the classes in the feature space, maximizing the margin between the closest observations from each class. Although originally designed for linear separation, SVMs can handle non-linear boundaries through the use of kernel functions that project the data into higher-dimensional spaces. In multispectral image classification, the user provides representative training areas for each target class (e.g., snow and non-snow), and the model learns to generalize this separation across the entire image. The classifier is trained on the selected spectral bands and then applied to assign a class to each pixel. The performance of the model can be assessed qualitatively by visually examining the classification map and quantitatively when validation data are available. (Gandhi

2023, Ravanelli 2024)

C. Supervised Classification: Random Forest

Random Forest is a supervised learning algorithm based on an ensemble of decision trees. Each tree is trained on a random subset of the data, and the final class assigned to each observation is determined by majority voting among all trees. This ensemble approach improves classifier robustness and reduces sensitivity to noise and local variability. When applied to satellite image classification, Random Forest enables the discrimination of multiple land cover types by leveraging a large number of spectral bands and labeled training samples. Labels can be obtained from thematic datasets such as the ESA WorldCover map, which provides a global land cover classification at high spatial resolution. Once the training dataset is prepared, the model is calibrated and applied to the aggregated image to generate a complete classification. Model performance is typically evaluated using confusion matrices and accuracy metrics derived from a reserved validation subset. (Gandhi 2023, Ravanelli 2024, Yiu 2020)

III. METHODS

A. Unsupervised Classification: K-means Clustering

For the unsupervised classification, the K-means algorithm was applied to Landsat-7 Surface Reflectance images. The selected study area is located in Scotland, and the analysis covered the entire year 2023. The data were pre-processed by filtering for cloud cover and subsequently aggregated using the median composite of the relevant spectral bands. The K-means algorithm was trained on a random sample of 5000 pixels extracted within the region of interest. Three different values of the number of clusters were tested $K = 3, 4, 5$ (See Figure 1). The classification result with $K = 5$ was visually the most informative, clearly distinguishing major land cover types such as water bodies, vegetated areas, urban zones, and mountainous regions. The classification output was visualized using a custom color palette to aid interpretation.

B. Supervised Classification: Support Vector Machine (SVM)

For the binary supervised classification, a Support Vector Machine (SVM) was applied to Landsat-8 imagery. The objective was to distinguish between snow-covered and non-snow-covered areas in a mountainous region of Iceland, selected due to the evident presence of seasonal snow cover. The temporal range used was the full year 2023, with the image filtered for cloud cover and aggregated using a median composite. Training areas were manually defined using six rectangular polygons (three for each class). The spectral bands used included visible, SWIR, and thermal bands, which were considered effective for snow discrimination. After training, the model was applied to the aggregated image to generate a binary classification map. The result aligned well with the distribution of snow visible in the RGB composite, with a clearly defined separation between the two classes See Figure 2.

C. Supervised Classification: Random Forest

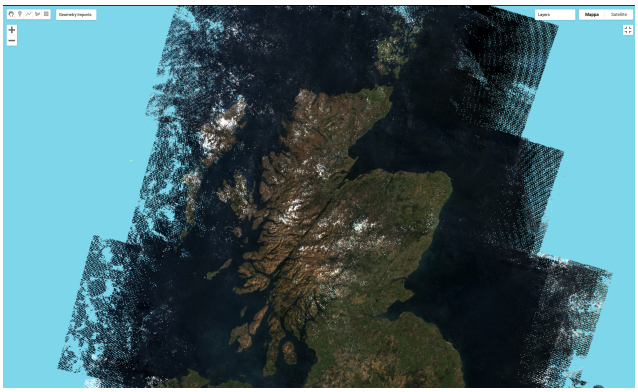
The multiclass classification was performed using the Random Forest algorithm applied to Sentinel-2 Harmonized images. The area analyzed is located in the state of Washington (USA), and the temporal range included the entire year 2024. After applying cloud masking and radiometric scaling, the images were aggregated using a median composite. The ESA WorldCover 2021 map was used as a reference for land cover labels. A stratified sampling procedure was performed, collecting 300 points per land cover class. The sampled data were split into a training set (80%) and a validation set (20%). The Random Forest classifier was configured with 50 decision trees. After classification, a confusion matrix was computed, and the validation accuracy reached 0.747, indicating good model performance. The resulting classified map showed a coherent spatial distribution of land cover classes consistent with the RGB composite image See Figure 3.

IV. CONCLUSIONS

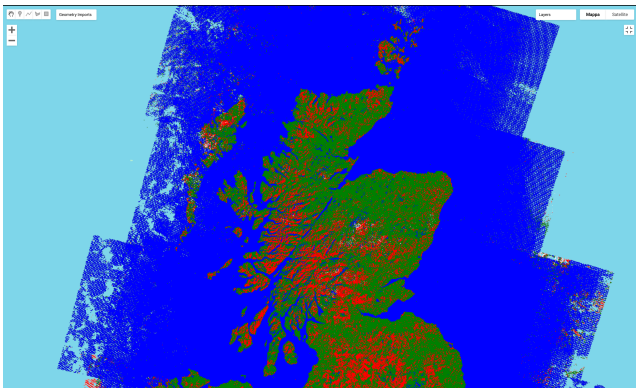
The exercises carried out allowed for the exploration of key techniques for satellite image classification, both supervised and unsupervised, using the Google Earth Engine platform. By combining multispectral data from Landsat and Sentinel-2, it was possible to practically analyze the performance of K-means, SVM, and Random Forest algorithms across different geographical and

temporal contexts. In the case of unsupervised classification, the use of K-means demonstrated the algorithms ability to effectively segment land cover based on spectral characteristics, even in the absence of labeled data. The choice of the number of clusters K proved to be a crucial factor in determining the visual quality of the result. The supervised classification with SVM showed good performance in the binary separation of snow-covered and non-snow-covered areas, thanks to a careful selection of spectral bands and a coherent training set. This approach proved particularly well suited to scenarios with distinct class boundaries. Finally, the Random Forest algorithm applied to multiclass classification, supported by ESA WorldCover data, produced accurate and consistent results, confirmed by a validation accuracy of 74.7%. The variation of algorithm parameters and training dataset size significantly influenced the models robustness. Overall, the activities performed provided insight into the operational potential of remote sensing and machine learning in land cover classification, while highlighting the importance of methodological choices and the quality of input data.

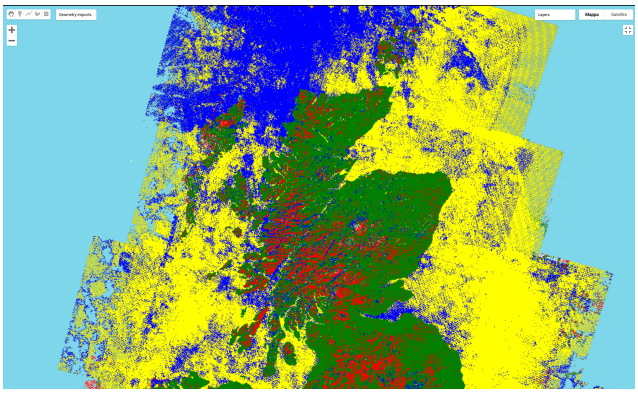
Appendix A: Images Visualization



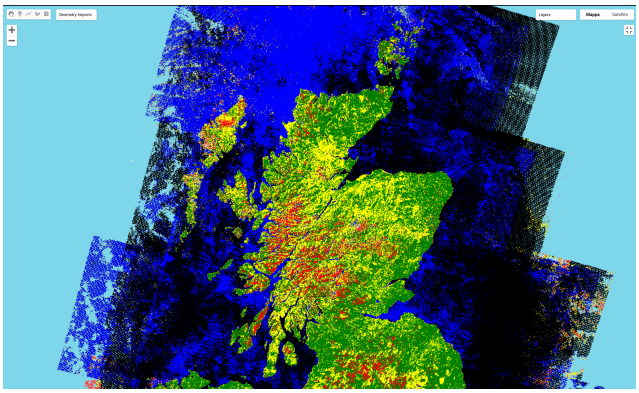
(a) RGB composite image (2023) of the study area.



(b) Unsupervised classification with K = 3 clusters.

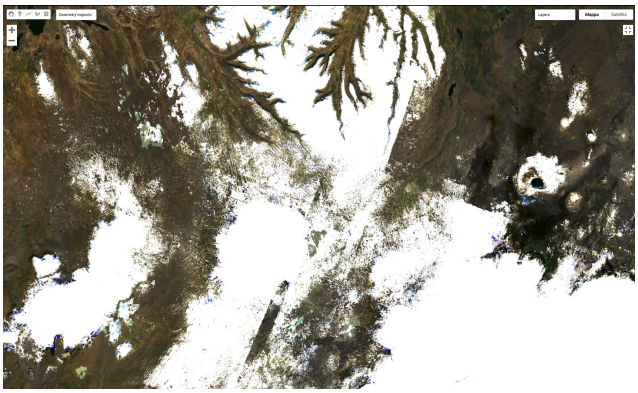


(c) Unsupervised classification with K = 4 clusters.

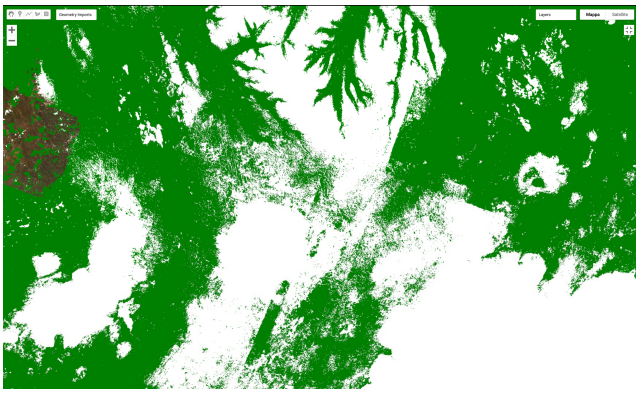


(d) Unsupervised classification with K = 5 clusters.

Figure 1: Comparison of K-means clustering results with different numbers of clusters ($K = 3, 4, 5$) applied to a median composite Landsat-7 image of Scotland (2023). As the number of clusters increases, the segmentation becomes more detailed, allowing a better distinction of land cover types such as water, vegetation, urban areas, and mountainous regions.



(a) RGB composite image (2023) of the selected area



(b) Binary classification result using SVM.

Figure 2: Support Vector Machine (SVM) classification applied to Landsat-8 imagery for detecting snow-covered areas in Iceland. The classification was trained using manually labeled polygons and spectral bands including visible, SWIR, and thermal information.

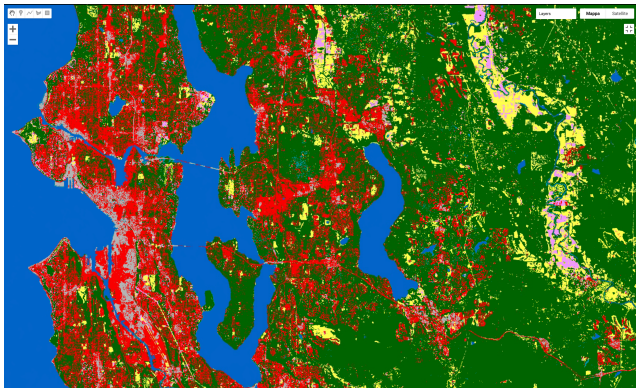


Figure 3: Land cover classification obtained using Random Forest on Sentinel-2 imagery (2024) in the Seattle region. The model was trained on ESA WorldCover 2021 reference labels and distinguishes multiple land cover types with high spatial detail.

Appendix B: Script on Google Earth Engine

1. K-means clustering (Scotland)

a. Selected ROI

```
var ROI = ee.Geometry.Polygon([
  [
    [-7.764189652125388,55.25526823853546],
    [-1.4360646521253884,55.25526823853546],
    [-1.4360646521253884,59.45730549423003],
    [-7.764189652125388,59.45730549423003],
    [-7.764189652125388,55.25526823853546]
  ]
]);
```

b. Script in common for all value of K

```
// Landsat-7 Surface Reflectance Tier 1 Collection 2
// https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LE07_C02_T1_L2#description
var L7_SR_coll = ee.ImageCollection("LANDSAT/LE07/C02/T1_L2");
// Filter the collection by period
// by location
// and by cloud cover property
var L7_SR_filtered_coll = L7_SR_coll.filterDate('2023-01-01', '2023-12-31')
                  .filterBounds(ROI)
                  .filter(ee.Filter.lt('CLOUD_COVER',50));
print('number of images in L7 filtered collection');
print(L7_SR_filtered_coll.size());
// Function that applies radiometric scaling factors
```

```
function applyRadiometricScaleFactors(image) {
  var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2);
  var thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0);

  return image.addBands(opticalBands, null, true)
              .addBands(thermalBands, null, true);
}
// Function that applies cloud masking
function cloud_maskL7sr(image) {
  // Bits 3 and 4 are cloud shadow and cloud, respectively
  var cloudShadowBitMask = 1 << 3;
  var cloudsBitMask = 1 << 4;
  // Get the pixel QA band.
  var qa = image.select('QA_PIXEL');
  // Both flags should be set to zero, indicating clear conditions
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
      .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  // Return the masked image without the QA bands
  return image.updateMask(mask);
}
// Applying radiometric scaling factors and cloud masking
L7_SR_filtered_coll = L7_SR_filtered_coll.map(applyRadiometricScaleFactors).map(cloud_maskL7sr);
// Check the effectiveness of the cloud masking
// on the first image of the filtered collection
Map.addLayer(L7_SR_filtered_coll.first(), {bands: ['SR_B4', 'SR_B3', 'SR_B2'], min:0,max: 0.3},
    'L7 SR first image 2023', false);
// Computing the aggregated image and selecting the bands of interest
var L7_SR_median_image = L7_SR_filtered_coll.median().select(['SR_B1', 'SR_B2','SR_B3','SR_B4',
    'SR_B5', 'SR_B7', 'ST_B6']);
print('L7_SR_median_image');
print(L7_SR_median_image);
// Display the aggregated image (input for the clusterer)
Map.addLayer(L7_SR_median_image, {'bands': ['SR_B3', 'SR_B2', 'SR_B1'], 'min':0, 'max':0.3},
    'Landsat-7 SR 2023 median image (RGB)');
Map.centerObject(ROI);
// Make the training dataset within the ROI
// We generate a sample of the input landsat 7 median image
var training_data = L7_SR_median_image.sample({
  region: ROI,
  scale: 30,
  numPixels: 5000,
  geometries: true
});
print('training_data');
print(training_data);
Map.addLayer(training_data, {color: 'black'}, 'training_data', false);
```

c. Number of clusters/classes (K) = 3

```
// Instantiate the clusterer
var number_of_clusters = 3;
var clusterer = ee.Clusterer.wekaKMeans(number_of_clusters);
```

```
// Train the clusterer
var trained_clusterer = clusterer.train(training_data);
// Cluster the input using the trained clusterer (inference)
var clustered_L7_SR_median_image = L7_SR_median_image
                                   .cluster(trained_clusterer);
print('clustered_L7_SR_median_image');
print(clustered_L7_SR_median_image);
// Display the clusters with a custom palette
// class 0: green
// class 1: red
// class 2: blue
Map.addLayer(clustered_L7_SR_median_image,
             {palette:['green', 'blue','red' ], min:0, max:2}, 'clustered_L7_SR_median_image');
Map.centerObject(ROI, 7.5);
```

d. Number of clusters/classes (K) = 4

```
// Instantiate the clusterer
var number_of_clusters = 4;
var clusterer = ee.Clusterer.wekaKMeans(number_of_clusters);
// Train the clusterer
var trained_clusterer = clusterer.train(training_data);

// Cluster the input using the trained clusterer (inference)
var clustered_L7_SR_median_image = L7_SR_median_image
                                   .cluster(trained_clusterer);
print('clustered_L7_SR_median_image');
print(clustered_L7_SR_median_image);
// Display the clusters with a custom palette
// class 0: green
// class 1: red
// class 2: blue
// class 3: yellow
Map.addLayer(clustered_L7_SR_median_image,
             {palette:['green', 'blue','red', 'yellow'], min:0, max:3}, 'clustered_L7_SR_median_image')
Map.centerObject(ROI, 7.5);
```

e. Number of clusters/classes (K) = 5

```
// Instantiate the clusterer
var number_of_clusters = 5;
var clusterer = ee.Clusterer.wekaKMeans(number_of_clusters);
// Train the clusterer
var trained_clusterer = clusterer.train(training_data);
// Cluster the input using the trained clusterer (inference)
var clustered_L7_SR_median_image = L7_SR_median_image
                                   .cluster(trained_clusterer);
print('clustered_L7_SR_median_image');
print(clustered_L7_SR_median_image);
```

```
// Display the clusters with a custom palette
// class 0: green
// class 1: red
// class 2: blue
// class 3: yellow
// class 4: black
Map.addLayer(clustered_L7_SR_median_image,
             {palette:['green', 'blue','red', 'yellow', 'black'], min:0, max:4},
             'clustered_L7_SR_median_image');
Map.centerObject(ROI, 7.5);
```

2. SVM classifier (Iceland)

a. Selected ROI

```
var ROI = ee.Geometry.Polygon([
  [-20.27263740270541, 64.50617269822484],
  [-16.48784736364291, 64.50617269822484],
  [-16.48784736364291, 65.41740879283175],
  [-20.27263740270541, 65.41740879283175],
  [-20.27263740270541, 64.50617269822484]
]);
```

b. Selected Area to analyze

```
//Area with snow
var snow1 = ee.Geometry.Rectangle([-17.139326798159843, 64.6057917545956,
    -16.941572891909843, 64.62698487810756]);
var snow2 = ee.Geometry.Rectangle([-20.125611412467123, 64.6530500364423,
    -20.056946861685873, 64.69182396791633]);
var snow3 = ee.Geometry.Rectangle([-16.68145038881973, 65.03754434348241,
    -16.626518747572284, 65.0977507433485]);
//Area without snow
var nonSnow1 = ee.Geometry.Rectangle([-19.577668297232748, 64.65246212617765,
    -19.513123619498373, 64.7012153618939]);
var nonSnow2 = ee.Geometry.Rectangle([-17.58526293103236, 65.10823536546012,
    -17.39849535290736, 65.13134633998428]);
var nonSnow3 = ee.Geometry.Rectangle([-18.802498361830096, 64.55303459590422,
    -18.753059885267596, 64.5922459860432]);
```

c. Remaining script

```
// Landsat-8 Surface Reflectance Tier 1 Collection 2
// https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C02_T1_L2
var L8_SR_coll = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2");
// Filter the collection by period
```

```
//                       by location
//                       by cloud cover property
var L8_SR_filtered_coll = L8_SR_coll.filterDate('2023-01-01', '2023-12-31')
                        .filterBounds(ROI)
                        .filter(ee.Filter.lt('CLOUD_COVER',50));
print('number of images in L8 filtered collection');
print(L8_SR_filtered_coll.size());
// Function that applies radiometric scaling factors
function applyRadiometricScaleFactors(image) {
  var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2);
  var thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0);
  return image.addBands(opticalBands, null, true)
              .addBands(thermalBands, null, true);
}
// Function that applies cloud masking
function cloud_maskL8sr(image) {
  // Bits 3 and 4 are cloud shadow and cloud, respectively
  var cloudShadowBitMask = 1 << 3;
  var cloudsBitMask = 1 << 4;
  // Get the pixel QA band.
  var qa = image.select('QA_PIXEL');
  // Both flags should be set to zero, indicating clear conditions
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
      .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  // Return the masked image without the QA bands
  return image.updateMask(mask);
}
// Applying radiometric scaling factors and cloud masking
L8_SR_filtered_coll = L8_SR_filtered_coll.map(applyRadiometricScaleFactors).map(cloud_maskL8sr);
// Check the effectiveness of the cloud masking
// on the first image of the filtered collection
Map.addLayer(L8_SR_filtered_coll.first(), {bands: ['SR_B4', 'SR_B3', 'SR_B2'], min:0, max: 0.3},
    'L8 first image 2018 (RGB)', false);
// Computing the aggregated image
var L8_SR_median_image = L8_SR_filtered_coll.median();
print('L8_SR_median_image');
print(L8_SR_median_image);
// Display the aggregated image (input for the classifier)
Map.addLayer(L8_SR_median_image, {bands: ['SR_B4', 'SR_B3', 'SR_B2'], min:0,max: 0.3},
    'L8 median image 2018 (RGB)');
// Make a FeatureCollection from the hand-made geometries
var polygons = ee.FeatureCollection([
  ee.Feature(nonSnow1, {'class': 0}),
  ee.Feature(nonSnow2, {'class': 0}),
  ee.Feature(nonSnow3, {'class': 0}),
  ee.Feature(snow1,    {'class': 1}),
  ee.Feature(snow2,    {'class': 1}),
  ee.Feature(snow3,    {'class': 1})
]);
print('polygons', polygons);
// Display the training polygons
Map.addLayer(polygons, {}, 'training polygons');
// Create the trining dataset
// Getting the values for all pixels in each polygon
var training_data = L8_SR_median_image.sampleRegions({
  // Get the sample from the polygons FeatureCollection
```

```
  collection: polygons,
  // Keep this list of properties from the polygons
  properties: ['class'],
  // Set the scale (spatial resolution) to get Landsat pixels in the polygons (m)
  scale: 30
});
print('training_data size');
print(training_data.size());
print('training_data first 50');
print(training_data.limit(50));
// Create an SVM classifier with custom parameters
var classifier = ee.Classifier.libsvm({
  kernelType: 'RBF',
  gamma: 0.5,
  cost: 10
});
// Bands to be used for the prediction
var bands = ['SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7', 'ST_B10'];
// Train the classifier
var trained_classifier = classifier.train(training_data, 'class', bands);
// Get information about the trained classifier
print('Trained classifier', trained_classifier.explain());
// Classify the L8 SR median image (inference)
var classified_L8_SR_median_image = L8_SR_median_image.classify(trained_classifier);
print('classified_L8_SR_median_image');
print(classified_L8_SR_median_image);
// Display the classification results
Map.addLayer(classified_L8_SR_median_image, {min: 0, max: 1, palette: ['green', 'white']},
    'classified_L8_SR_median_image (deforestation)');
Map.centerObject(ROI,8.2)
```

3. Random Forest classification

a. Selected ROI

```
var ROI = ee.Geometry.Rectangle([
  -122.44013986960687, 47.533604748040965,
  -121.85374460593499, 47.70023305247016
]);
```

b. Remaining script

```
// Sentinel-2 SR (HARMONIZED)
// https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR_HARMONIZED
var S2_SR_coll = ee.ImageCollection("COPERNICUS/S2_SR_HARMONIZED");
// Filter the collection by period
//                         by location
//                         by cloud cover property
var S2_SR_coll_filtered = S2_SR_coll.filterDate('2024-01-01', '2024-12-31')
```

```
                                  .filterBounds(ROI)
                                  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE',50));
print('number of images in S2 filtered collection');
print(S2_SR_coll_filtered.size());
// Function to mask clouds using the Sentinel-2 QA band
// and to apply the radiometric scaling factor
// @param  {ee.Image} image Sentinel-2 image
// @return {ee.Image} cloud masked and radiometrically scaled
//          Sentinel-2 image
function maskS2clouds_and_radiometricScaling(image) {
  var qa = image.select('QA60');
  // Bits 10 and 11 are clouds and cirrus, respectively.
  var cloudBitMask = 1 << 10;
  var cirrusBitMask = 1 << 11;
  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));
 // Masking cloud pixels        Applying radiometric scaling factor
 //            |                          |
 //            V                          |
 // making the pixels covered             |
 // by clouds transparent                 |
 // (ignored by reducers)                 |
 //            |                          |
 //            V                          V
  return image.updateMask(mask).multiply(0.0001);
}
// Applying radiometric scaling factors and standard cloud masking (QA60 band)
var S2_SR_coll_filtered_cloud_masked = S2_SR_coll_filtered.map(maskS2clouds_and_radiometricScaling);

// Check the effectiveness of the standard cloud masking
// on the first image of the filtered collection
//                                                        R->R  G->G   B->B
Map.addLayer(S2_SR_coll_filtered_cloud_masked.first(), {bands: ['B4', 'B3',  'B2'], min:0,max: 0.3},
    'S2 first image 2021 R-G-B (true color composite, standard masking)', false);
// Cloud Score+ image collection. Note Cloud Score+ is produced from Sentinel-2
// Level 1C data and can be applied to either L1C or L2A collections
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
// Use 'cs' or 'cs_cdf', depending on your use case; see docs for guidance
var QA_BAND = 'cs_cdf';
// The threshold for masking; values between 0.50 and 0.65 generally work well.
// Higher values will remove thin clouds, haze & cirrus shadows
var CLEAR_THRESHOLD = 0.60;
// Function to mask clouds using the Cloud Score+ cloud masking
// and to apply the radiometric scaling factor
function  cloud_score_plus_maskS2clouds_and_radiometricScaling(img) {
  //           Masking cloud pixels                    Applying radiometric scaling factor
  //                 |                                                  |
  //                 V                                                  |
  //            making the pixels covered                               |
  //            by clouds transparent                                   |
  //           (ignored by reducers)                                    V
      return img.updateMask(img.select(QA_BAND).gte(CLEAR_THRESHOLD)).multiply(0.0001);
    }
// Applying radiometric scaling factors and Cloud Score+ cloud masking
var S2_SR_coll_filtered_cloud_masked = S2_SR_coll_filtered.linkCollection(csPlus, [QA_BAND])
```

```
                                                .map(cloud_score_plus_maskS2clouds_and_radiometricScaling);
// Check the effectiveness of the Cloud Score+ cloud masking
// on the first image of the filtered collection
//                                                              R->R   G->G   B->B
Map.addLayer(S2_SR_coll_filtered_cloud_masked.first(), {bands: ['B4', 'B3',  'B2'], min:0,max: 0.3},
    'S2 first image 2021 R-G-B (true color composite, Cloud Score+ masking)', false);
// Computing the temporally aggregated image (median)
var S2_SR_aggregated_image = S2_SR_coll_filtered_cloud_masked.median();
//                                          R->R   G->G  B->B
Map.addLayer(S2_SR_aggregated_image, {bands:['B4', 'B3',  'B2'], min:0, max:0.3},
    'aggregated_image R-G-B (true color composite)');
//                                         R->SWIR1  G->NIR  B->G
Map.addLayer(S2_SR_aggregated_image, {bands: ['B11',  'B8',    'B3'], min: 0, max: 0.3},
    'aggregated_image SWIR1-NIR-G (false color composite)');
// The European Space Agency (ESA) WorldCover 10 m 2021 product provides a global land cover map
// for 2021 at 10 m resolution based on Sentinel-1 and Sentinel-2 data
// We will use it as label source in classifier training
var ESA_world_LCs = ee.ImageCollection("ESA/WorldCover/v200");
var ESA_2021_LC = ESA_world_LCs.first();
print('ESA 2021 WorldCover land cover map');
print(ESA_2021_LC);
// Remap the 11 land cover class values to a 0-based sequential series
var classValues = [10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 100];
var remapValues = ee.List.sequence(0, 10);
var label = 'lc';
ESA_2021_LC = ESA_2021_LC.remap(classValues, remapValues).rename(label).toByte();
// Defining the color palette for the LC classes
var LC_palette = ['006400',  // 0   Tree cover
                  'ffbb22',  // 1   Shrubland
                  'ffff4c',  // 2   Grassland
                  'f096ff',  // 3   Cropland
                  'fa0000',  // 4   Built-up
                  'b4b4b4',  // 5   Bare / sparse vegetation
                  'f0f0f0',  // 6   Snow and ice
                  '0064c8',  // 7   Permanent water bodies
                  '0096a0',  // 8   Herbaceous wetland
                  '00cf75',  // 9   Mangroves
                  'fae6a0']; // 10  Moss and lichen
Map.addLayer(ESA_2021_LC, {min:0,max:10, palette:LC_palette},
    'ESA_2021_LC (10 m spatial resolution)');
// Add land cover as a band of the aggregated image and sample 100 pixels at
// 10 m scale from each land cover class within the region of interest
// Stratified samplingwhere one samples specific proportions of individuals
// from various subpopulations (strata) in the larger populationis meant
// to ensure that the subjects selected will be representative of the population of interest
var datasetSample = S2_SR_aggregated_image.addBands(ESA_2021_LC).stratifiedSample({
  numPoints: 300,
  classBand: label,
  region: ROI,
  scale: 10,
  geometries: true,
  tileScale:4
});
print('datasetSample size',datasetSample.size());
print('datasetSample frequency');
print(datasetSample.reduceColumns(ee.Reducer.frequencyHistogram(),['lc']));
```

```
print('datasetSample first 100 values',datasetSample.limit(100));
// Add a random value field to the sample and use it to approximately split 80%
// of the features into a training set and 20% into a validation set
datasetSample = datasetSample.randomColumn();
print('datasetSample first 100 values',datasetSample.limit(100));
var trainingSample = datasetSample.filter('random <= 0.8');
var validationSample = datasetSample.filter('random > 0.8');
print('trainingSample size',trainingSample.size());
print('trainingSample frequency');
print(trainingSample.reduceColumns(ee.Reducer.frequencyHistogram(),['lc']));
print('trainingSample first 100 values',trainingSample.limit(100));
print('validationSample size',validationSample.size());
print('validationSample frequency');
print(validationSample.reduceColumns(ee.Reducer.frequencyHistogram(),['lc']));
print('validationSample first 100 values',validationSample.limit(100));
Map.addLayer(trainingSample, {color: 'black'}, 'Training sample', false);
Map.addLayer(validationSample, {color: 'white'}, 'Validation sample', false);
var RF_classifier =   ee.Classifier.smileRandomForest(50);
// Train a 50-tree random forest classifier from the training sample
var trainedRFClassifier =RF_classifier.train({
  features: trainingSample,
  classProperty: label,
  inputProperties: S2_SR_aggregated_image.bandNames()
});
// Get information about the trained classifier.
print('Results of trained classifier', trainedRFClassifier.explain());
// Get a confusion matrix and overall accuracy for the training sample
var trainAccuracy = trainedRFClassifier.confusionMatrix();
print('Training error matrix', trainAccuracy);
print('Training overall accuracy', trainAccuracy.accuracy());
// Get a confusion matrix and overall accuracy for the validation sample
validationSample = validationSample.classify(trainedRFClassifier);
var validationAccuracy = validationSample.errorMatrix(label, 'classification');
print('Validation error matrix', validationAccuracy);
print('Validation accuracy', validationAccuracy.accuracy());
// Classify the aggregated image with the trained classifier
var classified_S2_SR_aggregated_image = S2_SR_aggregated_image.classify(trainedRFClassifier);
Map.centerObject(ROI,12);
Map.addLayer(classified_S2_SR_aggregated_image, {min:0,max:10, palette:LC_palette},
    '2023 classified_S2_SR_aggregated_image');
```

U. Gandhi. End-to-end google earth engine, 2023. URL https://courses.spatialthoughts.com/end-to-end-gee.html. Accessed: 2025-06-19.

R. Ravanelli. Classification of satellite multispectral imagery with machine learning algorithms, 2024. Lecture slides, Remote Sensing and Geo Big Data, Sapienza University of Rome, Geodesy and Geomatics Division (DICEA).

T. Yiu. Understanding random forest, 2020. URL https://towardsdatascience.com/understanding-random-forest-58381e0602d2.