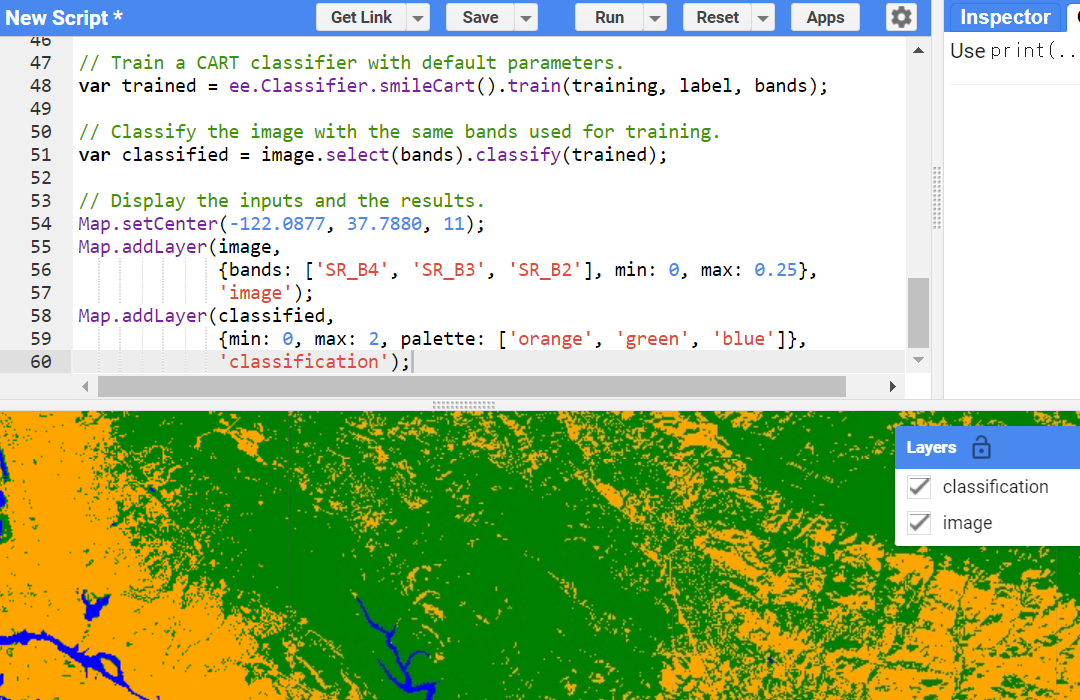
Supervised Classification

Classification and Regression Trees (CART)

https://developers.google.com/earth-engine/guides/classification



// Define a function that scales and masks Landsat 8 surface reflectance images.

function prepSrL8(image) {

// Develop masks for unwanted pixels (fill, cloud, cloud shadow).

var qaMask = image.select('QA\_PIXEL').bitwiseAnd(parseInt('11111', 2)).eq(0);

var saturationMask = image.select('QA\_RADSAT').eq(0);

// Apply the scaling factors to the appropriate bands.

var getFactorImg = function(factorNames) {

var factorList = image.toDictionary().select(factorNames).values();

return ee.Image.constant(factorList);

};

var scaleImg = getFactorImg([

'REFLECTANCE\_MULT\_BAND\_.|TEMPERATURE\_MULT\_BAND\_ST\_B10']);

var offsetImg = getFactorImg([

'REFLECTANCE\_ADD\_BAND\_.|TEMPERATURE\_ADD\_BAND\_ST\_B10']);

var scaled = image.select('SR\_B.|ST\_B10').multiply(scaleImg).add(offsetImg);

// Replace original bands with scaled bands and apply masks.

return image.addBands(scaled, null, true)

.updateMask(qaMask).updateMask(saturationMask);

}

// Make a cloud-free Landsat 8 surface reflectance composite.

var image = ee.ImageCollection('LANDSAT/LC08/C02/T1\_L2')

.filterDate('2021-03-01', '2021-07-01')

.map(prepSrL8)

.median();

// Use these bands for prediction.

var bands = ['SR\_B2', 'SR\_B3', 'SR\_B4', 'SR\_B5',

'SR\_B6', 'SR\_B7', 'ST\_B10'];

// Load training points. The numeric property 'class' stores known labels.

var points = ee.FeatureCollection('GOOGLE/EE/DEMOS/demo\_landcover\_labels');

// This property stores the land cover labels as consecutive

// integers starting from zero.

var label = 'landcover';

// Overlay the points on the imagery to get training.

var training = image.select(bands).sampleRegions({

collection: points,

properties: [label],

scale: 30

});

// Train a CART classifier with default parameters.

var trained = ee.Classifier.smileCart().train(training, label, bands);

// Classify the image with the same bands used for training.

var classified = image.select(bands).classify(trained);

// Display the inputs and the results.

Map.setCenter(-122.0877, 37.7880, 11);

Map.addLayer(image,

{bands: ['SR\_B4', 'SR\_B3', 'SR\_B2'], min: 0, max: 0.25},

'image');

Map.addLayer(classified,

{min: 0, max: 2, palette: ['orange', 'green', 'blue']},

'classification');

--

Support Vector Machine (SVM)

// Define a function that scales and masks Landsat 8 surface reflectance images.

function prepSrL8(image) {

// Develop masks for unwanted pixels (fill, cloud, cloud shadow).

var qaMask = image.select('QA\_PIXEL').bitwiseAnd(parseInt('11111', 2)).eq(0);

var saturationMask = image.select('QA\_RADSAT').eq(0);

// Apply the scaling factors to the appropriate bands.

var getFactorImg = function(factorNames) {

var factorList = image.toDictionary().select(factorNames).values();

return ee.Image.constant(factorList);

};

var scaleImg = getFactorImg([

'REFLECTANCE\_MULT\_BAND\_.|TEMPERATURE\_MULT\_BAND\_ST\_B10']);

var offsetImg = getFactorImg([

'REFLECTANCE\_ADD\_BAND\_.|TEMPERATURE\_ADD\_BAND\_ST\_B10']);

var scaled = image.select('SR\_B.|ST\_B10').multiply(scaleImg).add(offsetImg);

// Replace original bands with scaled bands and apply masks.

return image.addBands(scaled, null, true)

.updateMask(qaMask).updateMask(saturationMask);

}

// Make a cloud-free Landsat 8 surface reflectance composite.

var image = ee.ImageCollection('LANDSAT/LC08/C02/T1\_L2')

.filterDate('2018-01-01', '2019-01-01')

.map(prepSrL8)

.median();

// Use these bands for prediction.

var bands = ['SR\_B2', 'SR\_B3', 'SR\_B4', 'SR\_B5',

'SR\_B6', 'SR\_B7'];

// Manually created polygons.

var forest1 = ee.Geometry.Rectangle(-63.0187, -9.3958, -62.9793, -9.3443);

var forest2 = ee.Geometry.Rectangle(-62.8145, -9.206, -62.7688, -9.1735);

var nonForest1 = ee.Geometry.Rectangle(-62.8161, -9.5001, -62.7921, -9.4486);

var nonForest2 = ee.Geometry.Rectangle(-62.6788, -9.044, -62.6459, -8.9986);

// Make a FeatureCollection from the hand-made geometries.

var polygons = ee.FeatureCollection([

ee.Feature(nonForest1, {'class': 0}),

ee.Feature(nonForest2, {'class': 0}),

ee.Feature(forest1, {'class': 1}),

ee.Feature(forest2, {'class': 1}),

]);

// Get the values for all pixels in each polygon in the training.

var training = image.sampleRegions({

// Get the sample from the polygons FeatureCollection.

collection: polygons,

// Keep this list of properties from the polygons.

properties: ['class'],

// Set the scale to get Landsat pixels in the polygons.

scale: 30

});

// Create an SVM classifier with custom parameters.

var classifier = ee.Classifier.libsvm({

kernelType: 'RBF',

gamma: 0.5,

cost: 10

});

// Train the classifier.

var trained = classifier.train(training, 'class', bands);

// Classify the image.

var classified = image.classify(trained);

// Display the classification result and the input image.

Map.setCenter(-62.836, -9.2399, 9);

Map.addLayer(image,

{bands: ['SR\_B4', 'SR\_B3', 'SR\_B2'], min: 0, max: 0.25},

'image');

Map.addLayer(polygons, {color: 'yellow'}, 'training polygons');

Map.addLayer(classified,

{min: 0, max: 1, palette: ['orange', 'green']},

'deforestation');

--

Accuracy Assessment

To assess the accuracy of a classifier, use a ConfusionMatrix.

// Define a region of interest.

var roi = ee.Geometry.BBox(-122.93, 36.99, -121.20, 38.16);

// Define a function that scales and masks Landsat 8 surface reflectance images.

function prepSrL8(image) {

// Develop masks for unwanted pixels (fill, cloud, cloud shadow).

var qaMask = image.select('QA\_PIXEL').bitwiseAnd(parseInt('11111', 2)).eq(0);

var saturationMask = image.select('QA\_RADSAT').eq(0);

// Apply the scaling factors to the appropriate bands.

var getFactorImg = function(factorNames) {

var factorList = image.toDictionary().select(factorNames).values();

return ee.Image.constant(factorList);

};

var scaleImg = getFactorImg([

'REFLECTANCE\_MULT\_BAND\_.|TEMPERATURE\_MULT\_BAND\_ST\_B10']);

var offsetImg = getFactorImg([

'REFLECTANCE\_ADD\_BAND\_.|TEMPERATURE\_ADD\_BAND\_ST\_B10']);

var scaled = image.select('SR\_B.|ST\_B10').multiply(scaleImg).add(offsetImg);

// Replace original bands with scaled bands and apply masks.

return image.addBands(scaled, null, true)

.updateMask(qaMask).updateMask(saturationMask);

}

// Make a cloud-free Landsat 8 surface reflectance composite.

var input = ee.ImageCollection('LANDSAT/LC08/C02/T1\_L2')

.filterBounds(roi)

.filterDate('2020-03-01', '2020-07-01')

.map(prepSrL8)

.median()

.setDefaultProjection('EPSG:4326', null, 30)

.select(['SR\_B2', 'SR\_B3', 'SR\_B4', 'SR\_B5', 'SR\_B6', 'SR\_B7']);

// Use MODIS land cover, IGBP classification, for training.

var modis = ee.Image('MODIS/006/MCD12Q1/2020\_01\_01')

.select('LC\_Type1');

// Sample the input imagery to get a FeatureCollection of training data.

var training = input.addBands(modis).sample({

region: roi,

numPixels: 5000,

seed: 0

});

// Make a Random Forest classifier and train it.

var classifier = ee.Classifier.smileRandomForest(10)

.train({

features: training,

classProperty: 'LC\_Type1',

inputProperties: ['SR\_B2', 'SR\_B3', 'SR\_B4', 'SR\_B5', 'SR\_B6', 'SR\_B7']

});

// Classify the input imagery.

var classified = input.classify(classifier);

// Get a confusion matrix representing resubstitution accuracy.

var trainAccuracy = classifier.confusionMatrix();

print('Resubstitution error matrix: ', trainAccuracy);

print('Training overall accuracy: ', trainAccuracy.accuracy());

// Sample the input with a different random seed to get validation data.

var validation = input.addBands(modis).sample({

region: roi,

numPixels: 5000,

seed: 1

// Filter the result to get rid of any null pixels.

}).filter(ee.Filter.notNull(input.bandNames()));

// Classify the validation data.

var validated = validation.classify(classifier);

// Get a confusion matrix representing expected accuracy.

var testAccuracy = validated.errorMatrix('LC\_Type1', 'classification');

print('Validation error matrix: ', testAccuracy);

print('Validation overall accuracy: ', testAccuracy.accuracy());

// Define a palette for the IGBP classification.

var igbpPalette = [

'aec3d4', // water

'152106', '225129', '369b47', '30eb5b', '387242', // forest

'6a2325', 'c3aa69', 'b76031', 'd9903d', '91af40', // shrub, grass

'111149', // wetlands

'cdb33b', // croplands

'cc0013', // urban

'33280d', // crop mosaic

'd7cdcc', // snow and ice

'f7e084', // barren

'6f6f6f' // tundra

];

// Display the input and the classification.

Map.centerObject(roi, 10);

Map.addLayer(input.clip(roi),

{bands: ['SR\_B4', 'SR\_B3', 'SR\_B2'], min: 0, max: 0.25},

'landsat');

Map.addLayer(classified.clip(roi),

{palette: igbpPalette, min: 0, max: 17},

'classification');

--

var sample = input.addBands(modis).sample({

numPixels: 5000,

seed: 0

});

// The randomColumn() method will add a column of uniform random

// numbers in a column named 'random' by default.

sample = sample.randomColumn();

var split = 0.7; // Roughly 70% training, 30% testing.

var training = sample.filter(ee.Filter.lt('random', split));

var validation = sample.filter(ee.Filter.gte('random', split));

--

// Sample the input imagery to get a FeatureCollection of training data.

var sample = input.addBands(modis).sample({

region: roi,

numPixels: 5000,

seed: 0,

geometries: true,

tileScale: 16

});

// The randomColumn() method will add a column of uniform random

// numbers in a column named 'random' by default.

sample = sample.randomColumn();

var split = 0.7; // Roughly 70% training, 30% testing.

var training = sample.filter(ee.Filter.lt('random', split));

print(training.size());

var validation = sample.filter(ee.Filter.gte('random', split));

// Spatial join.

var distFilter = ee.Filter.withinDistance({

distance: 1000,

leftField: '.geo',

rightField: '.geo',

maxError: 10

});

var join = ee.Join.inverted();

// Apply the join.

training = join.apply(training, validation, distFilter);

print(training.size());

--

