//Part1: Pre-processing

//Part2: Create a spatial and temporal filter

//create spatial filter

var SpatFiltered = sentinel.filterBounds(studyarea);

//create a temporal filter

var SA2022 = SpatFiltered.filterDate('2022-01-01', '2022-05-31');

//filter to keep images with less tha 10%cloud cover

var qualityfilter = SA2022.filterMetadata('CLOUDY\_PIXEL\_PERCENTAGE', 'less\_than', 10);

// Step3: Create a cloud mask using Sentinal 2 QA60 band

//Function to mask clouds using the sentinel 2 QA Band

function maskS2clouds(image){

var qa =image.select('QA60');

//Bits 10 and 11 are clouds abd cirrus, respectively. Here we select those bits

var cloudBitMask = 1<<10;

var cirrusBitMask = 1<<11;

//Both flags should be set to zero, indicating clear conditions

//we keep the pixel if the bit is zero

var mask = qa.bitwiseAnd(cloudBitMask).eq(0).and(

qa.bitwiseAnd(cirrusBitMask).eq(0));

// return the masked bands and scale data using metadata scaling factor

return image.updateMask(mask).divide(10000)

.select("B.\*")

.copyProperties(image, ["system:time\_start"]);

}

//once the function is created we can apply function to the selected images

var CloudMasked = qualityfilter.map(maskS2clouds);

print(CloudMasked);

//Step4: Combine all masked images using median operator and visualize

var CloudMaskMedian = CloudMasked.median().clip(studyarea);

//we then visualise the false color composite

var falseColor1 = {

bands: ["B11", "B8", "B4"],

min: 0,

max: 0.5

};

Map.addLayer(CloudMaskMedian, falseColor1, "median cloud masked false color");

print('median of all filtered', CloudMaskMedian);

//Step7: After finishing collecting all training sites, Merge them into single feature collection

// merge training data

var MergedTrain = Forest.merge(Agriculture)

.merge(Water)

.merge(Wetland)

.merge(Urban);

//Part3: Creating signatures for classification

// Step8: specify the bands to use in the classification

var bands = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B11', 'B12'];

//Step9: Extract the training data

var points = CloudMaskMedian.select(bands).sampleRegions({

collection: MergedTrain,

properties: ['Label'],

scale: 30

}).randomColumn(); //to divide into training and validation

print(points.first(), points);

//Step10: visualize signature

//visualize signature Water

var waterSig = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.mean(),Water,30);

print (waterSig, 'waterSig')

//visualize signature Agriculture

var agricultureSig = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.mean(),Agriculture,30);

print (agricultureSig, 'agricultureSig')

//Forest Signature

var forestsig = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.mean(), Forest, 30);

print(forestsig, 'forestsig')

// exracting maximum and minimum value

var forestSigMin = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.min(), Forest, 30);

print (forestSigMin, 'forestSigMin');

var forestSigMax = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.max(), Forest, 30);

print (forestSigMax, 'forestSigMax');

//visualize signature Wetland

var wetlandSig = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.mean(),Wetland,30);

print (wetlandSig, 'wetlandSig');

var urbanSig = CloudMaskMedian.select(bands)

.reduceRegion(ee.Reducer.mean(),Urban,30);

print(urbanSig,'urbanSig')

//Randomly split the samples to set some aside for testing the models accuracy

//using the "random" column. Roughly 80% for training, 20% for training

var split = 0.8; //set split to 80%

var training = points.filter(ee.Filter.lt('random', split)); //less than 80%

var testing = points.filter(ee.Filter.gte('random', split)); //remaing 20% for testing

//print these variabkes to see how much training and testing data you are using

print('samples n =', points.aggregate\_count('.all'));

print('Training n =', training.aggregate\_count('.all'));

print('Testing n =', testing.aggregate\_count('.all'));

//Run Supervised Classification

//Run the minimum distance supervised classification

var classifierMD = ee.Classifier.minimumDistance('euclidean',1).train({

features: training,

classProperty: 'Label',

inputProperties: bands

});

//Apply training classifier to the image

var classifiedMD = CloudMaskMedian.select(bands).classify(classifierMD);

//CREATE PALETTE FOR FINAL LANDCCOVER MAP CLASSIFCATION

var Palette =

'<RasterSymbolizer>'+

'<ColorMap type = "intervals">' +

'<ColorMapEntry color = "#202bb6" quantity = "1" Label = "Water"/>'+

'<ColorMapEntry color = "#feb50d" quantity = "2" Label = "Agriculture"/>'+

'<ColorMapEntry color = "#54dd6d" quantity = "3" Label = "Forest"/>'+

'<ColorMapEntry color = "#1eddf4" quantity = "4" Label = "Wetland"/>'+

'<ColorMapEntry color ="#f41e3f" quantity="5" label="urban"/>' +

'</ColorMap>'+

'</RasterSymbolizer>';

//Add final map to the display with the specified palette

Map.addLayer(classifiedMD.sldStyle(Palette),{},"Land Classification: Minimum Distance");

//center the map for display

Map.setCenter(-90.1871,16.7167);

//Run the random forest supervised classification

var classifierRF = ee.Classifier.smileRandomForest(300,5).train({

features: training,

classProperty: 'Label',

inputProperties: bands

});

//Apply training classifier to the image

var classifiedRF = CloudMaskMedian.select(bands).classify(classifierRF);

//CREATE PALETTE FOR FINAL LANDCCOVER MAP CLASSIFCATION

var Palette =

'<RasterSymbolizer>'+

'<ColorMap type = "intervals">' +

'<ColorMapEntry color = "#202bb6" quantity = "1" Label = "Water"/>'+

'<ColorMapEntry color = "#feb50d" quantity = "2" Label = "Agriculture"/>'+

'<ColorMapEntry color = "#54dd6d" quantity = "3" Label = "Forest"/>'+

'<ColorMapEntry color = "#1eddf4" quantity = "4" Label = "Wetland"/>'+

'<ColorMapEntry color ="#f41e3f" quantity="5" label="urban"/>' +

'</ColorMap>'+

'</RasterSymbolizer>';

//Add final map to the display with the specified palette

Map.addLayer(classifiedRF.sldStyle(Palette),{},"Land Classification: Random Forest");

//center the map for display

Map.setCenter(-90.1871,16.7167);

//Part 5 : Accuracy Assessment

//use testing data to evaluate the accuracy of the classification

//Print Confusion matricx and overall accuracy of MD

var confusionMatrix = classifierMD.confusionMatrix(); // goodness of fit of model- accuracy of trainng

print('Confusion matrix:', confusionMatrix);

print('Training Overall Accuracy:', confusionMatrix.accuracy());

print('Training Users Accuracy:', confusionMatrix.consumersAccuracy());

print('Training Producers Accuracy:', confusionMatrix.producersAccuracy());

var validation = testing.classify(classifierMD); //ACCURACY ASESSMENT ON INDIPENDENT POINTS

var testAccuracy = validation.errorMatrix('Label', 'classification');

print ('Validation Error Matrix MD', testAccuracy);

print('Validation Overall Accuracy MD:', testAccuracy.accuracy());

print('Validation Users Accuracy MD:', testAccuracy.consumersAccuracy());

print('Validation Producers Accuracy MD:', testAccuracy.producersAccuracy());

//Print Confusion matricx and overall accuracy of RF

var confusionMatrix = classifierRF.confusionMatrix(); // goodness of fit of model- accuracy of trainng

print('Confusion matrix RF:', confusionMatrix);

print('Training Overall Accuracy RF:', confusionMatrix.accuracy());

print('Training Users Accuracy RF:', confusionMatrix.consumersAccuracy());

print('Training Producers Accuracy RF:', confusionMatrix.producersAccuracy());

var validation = testing.classify(classifierRF); //ACCURACY ASESSMENT ON INDIPENDENT POINTS

var testAccuracy = validation.errorMatrix('Label', 'classification');

print ('Validation Error Matrix RF', testAccuracy);

print('Validation Overall Accuracy RF:', testAccuracy.accuracy());

print('Validation Users Accuracy RF:', testAccuracy.consumersAccuracy());

print('Validation Producers Accuracy RF:', testAccuracy.producersAccuracy());

//Part 6: Compare the two results

// Compare algorithms

var Intersection = classifiedRF.addBands(classifiedMD).reduceRegion({

reducer: ee.Reducer.frequencyHistogram().unweighted().group({

groupField: 1,

groupName: 'classifiedMD'

}),

geometry: studyarea,

scale: 10,

maxPixels: 1e9,

});

print(Intersection);