Geospatial Big Data Analysis - Lab 1

Konstantinos Papadakis - MSc Data Science and Machine Learning - 03400149

1. GIS comparison

Some popular cloud based Geographic Information Systems (GIS) platforms are those based on the Copernicus program, collectively called Data and Information Access Services (DIAS). There are five DIAS platforms, namely **CREODIAS , Sobloo, Mundi, Onda** and **WEkEO**. All DIAS platforms provide access to Copernicus Sentinel data, as well as to the information products from Copernicus' operational services, together with cloud-based tools (open source and/or on a pay-per-use basis). Each of the five competitive platforms also provides access to additional commercial satellite or non-space data sets as well as premium offers in terms of support or priority. Thanks to a single access point for the entire Copernicus data and information, DIAS allows the users to develop and host new applications in the cloud, while removing the need to download bulky files from several access points and process them locally.

More specifically, considering CREODIAS, WEkEO and Mundi:

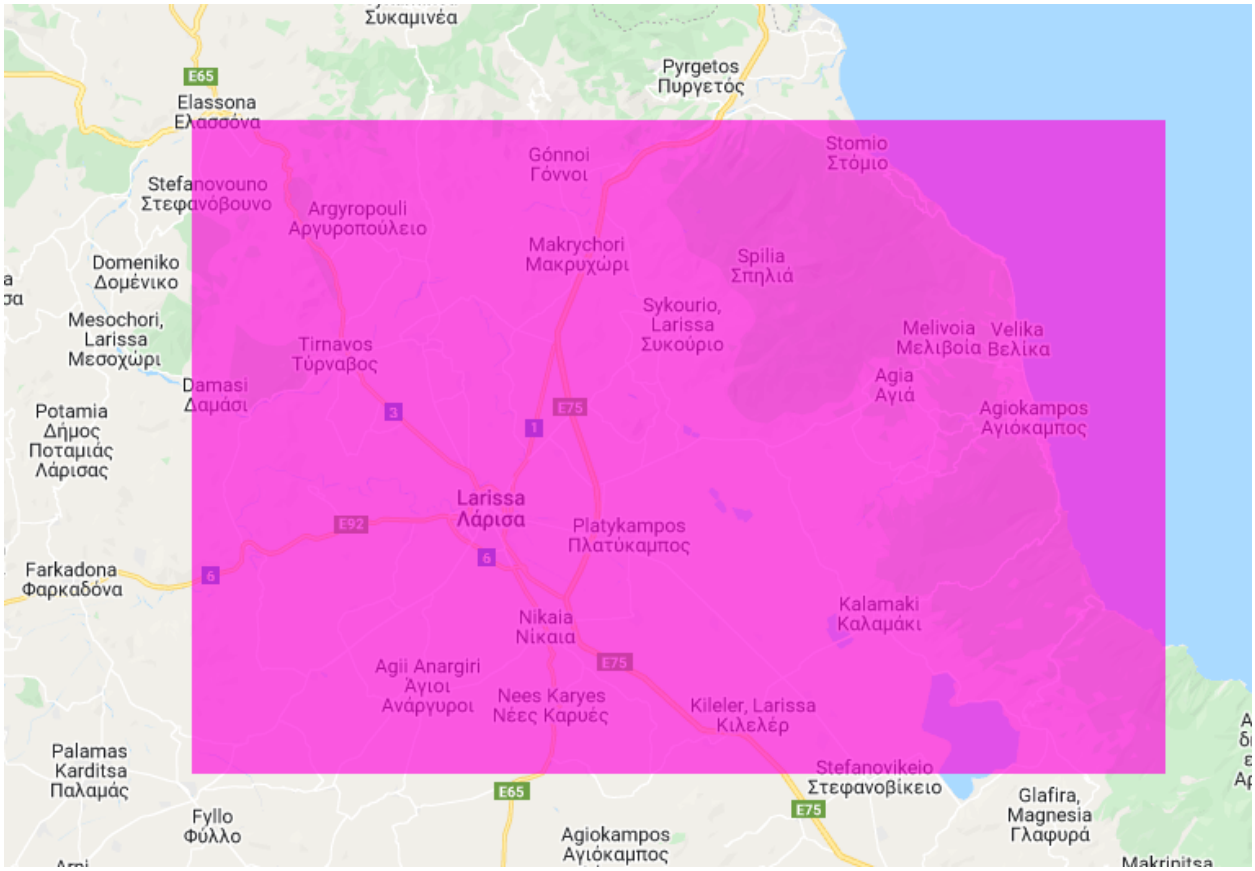
- **CREODIAS** provides access to almost all Copernicus services; CAMS, CEMS, CLMS, CMEMS, and satellite data from Sentinel-1, Sentinel-2, Sentinel-3, and Sentinel-5P. An extensive REST API is available with access to OpenStack (cloud system), Finder (data viewer) and billing data. Integrated JupyterHub and Notebooks support is available free of charge for every registered CREODIAS user.
- **WEkEO**, implemented by EUMETSAT, ECMWF and Mercator Ocean International, provides access to a partial set of the data produced by Copernicus services; CMEMS, CAMS and C3S, together with Sentinel-1, Sentinel-2, and Sentinel-3 data. WEkEO provides a basic API called Harmonised Data Access (HDA) API which allows access to the whole data catalog, with filtering, searching and downloading functionalities. This API allows datasets usage with Jupyter Notebooks and WEkEO virtual machines.
- **Mundi** provides access to the following Copernicus services: CEMS and CLMS, with Sentinel1, Sentinel-2, Sentinel-3, and Sentinel-5P data, as well as Landsat data. Jupyter Notebooks integration is available, with pre-installed tools to enhance the user experience.

Overall, the facilities, data, and costs associated with the DIAS are very similar across the board, with minor variance in costs.

Another very popular cloud based Platform is **Google Earth Engine,** which provides access to Landsat and MODIS data (NASA/USGS) as well as Sentinel data (ESA Copernicus), and others. Google Earth Engine provides full-featured JavaScript, Python and REST APIs, as well as access to a web-based code editor specialized for JavaScript API usage. Earth Engine is free for research, education, and nonprofit use. For commercial or operational applications, Google Earth Engine is currently in Private Preview through Google Cloud.

2. Least Cloudy Image of 2019

Using Google Earth Engine we select a part of Thessaly, Greece as our area of interest, as shown in the following image

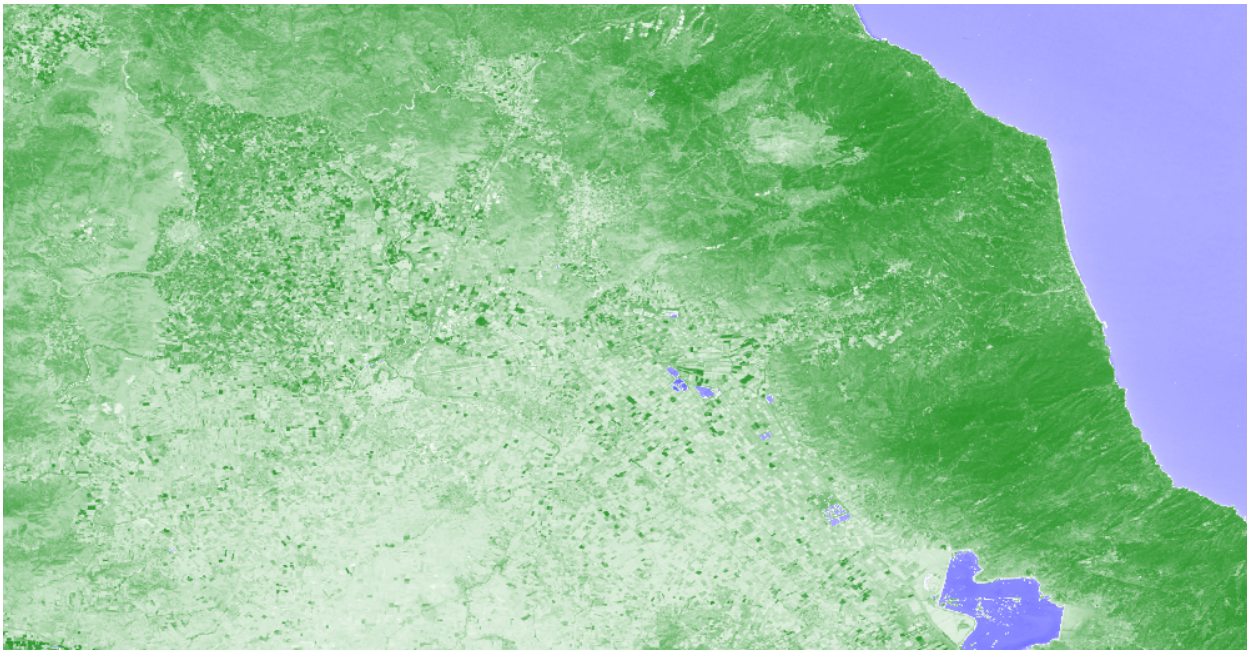


We find the date(s) that were the least cloudy in that region and display the corresponding:

- True Color Image (Red, Green, Blue)

- False Color Image (NIR, Red, Green)

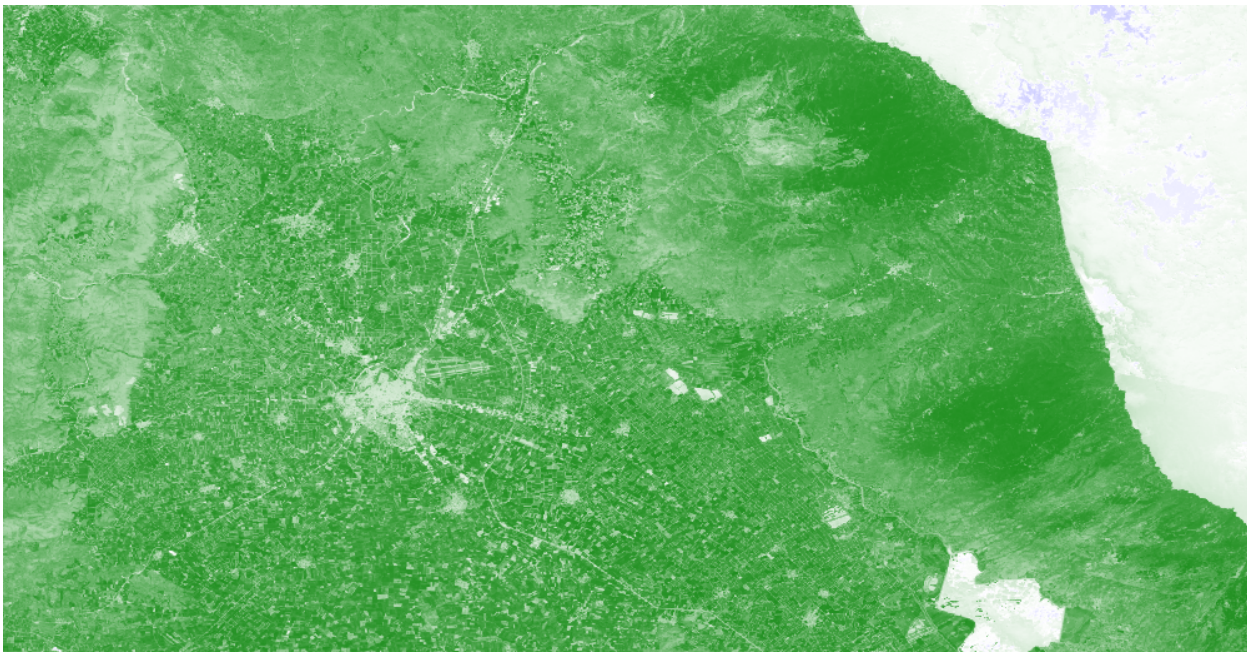
- NDVI Image



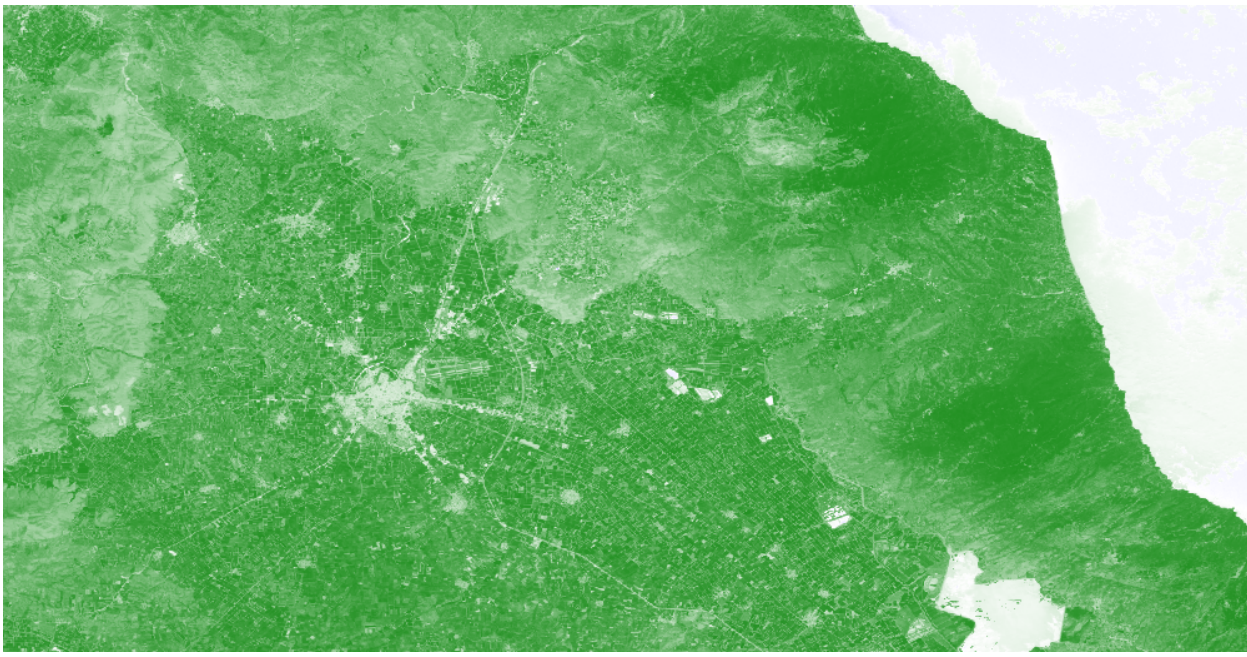
3. Max NDVI 2018 and 2019

We then proceed to create mosaics for the greenest images of the years for 2018 and 2019. We notice that in both years the images look roughly the same, with the few differences being mainly due cloud noise, and possibly crop rotation.

- NDVI values of the mosaic
  2018

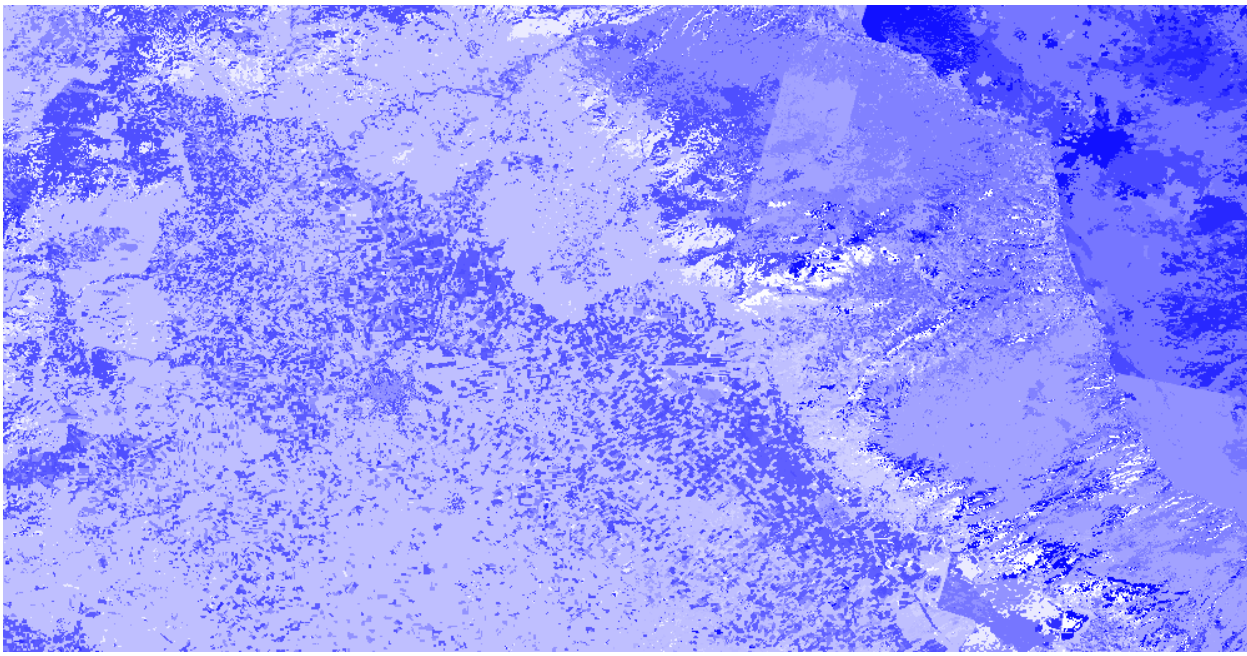


2019

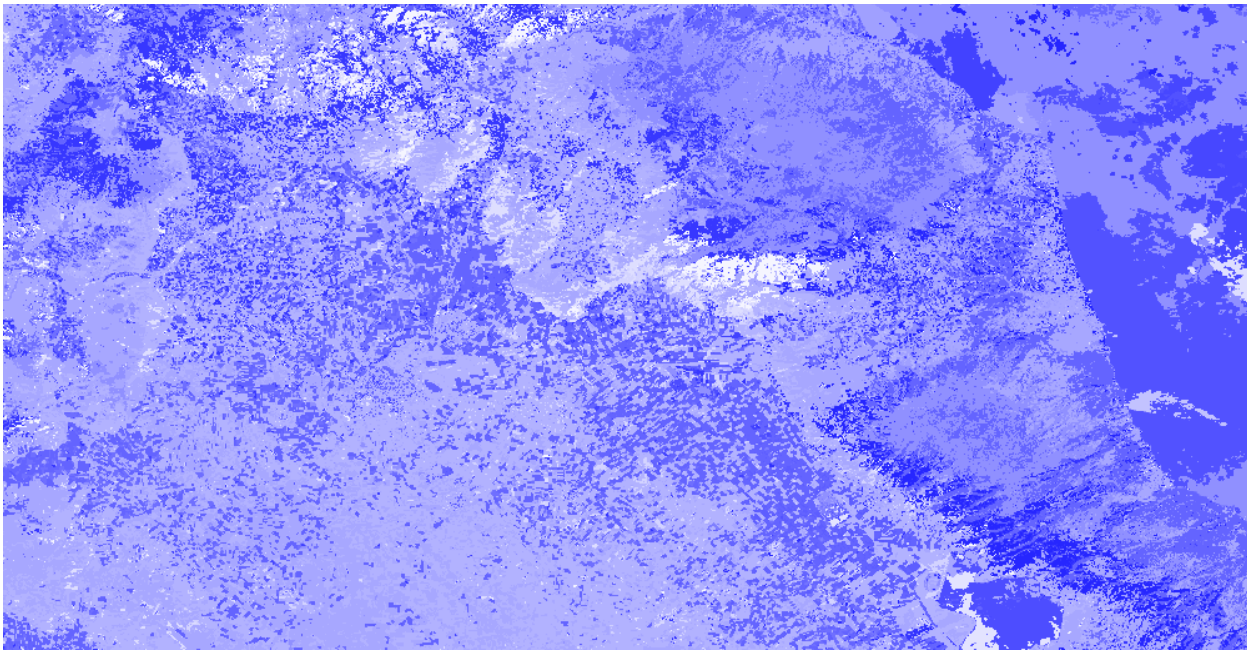


- Day of the year of the mosaic (white is the first day, blue is the last day)

2018

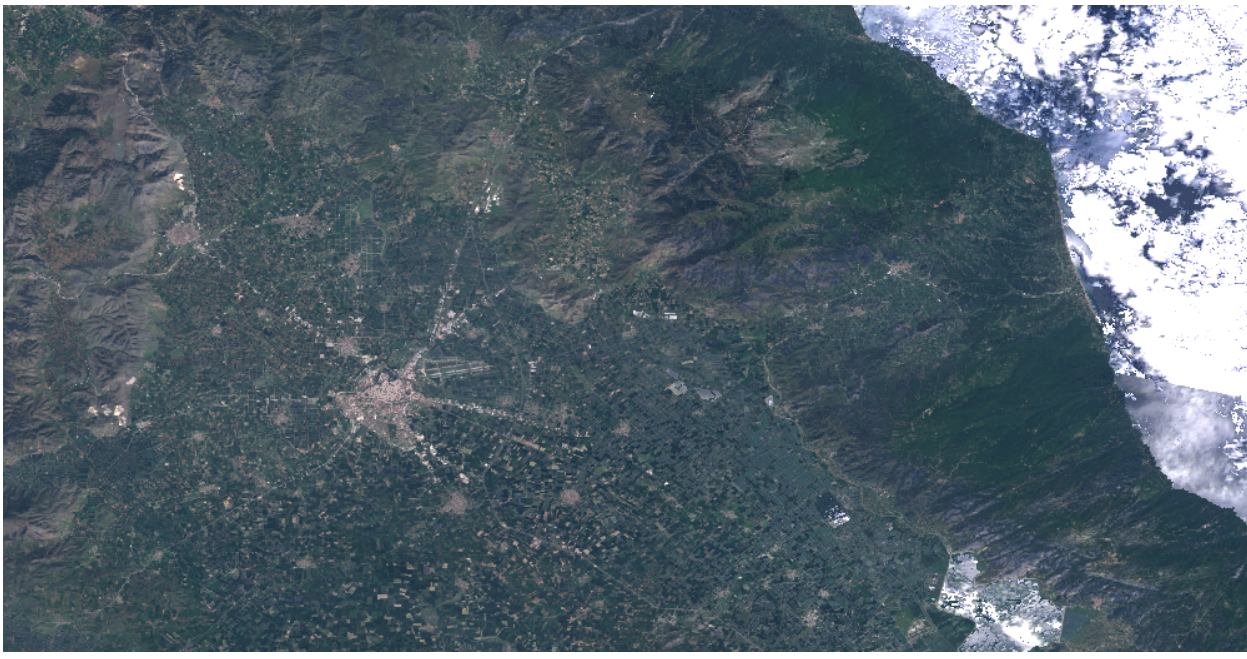


2019

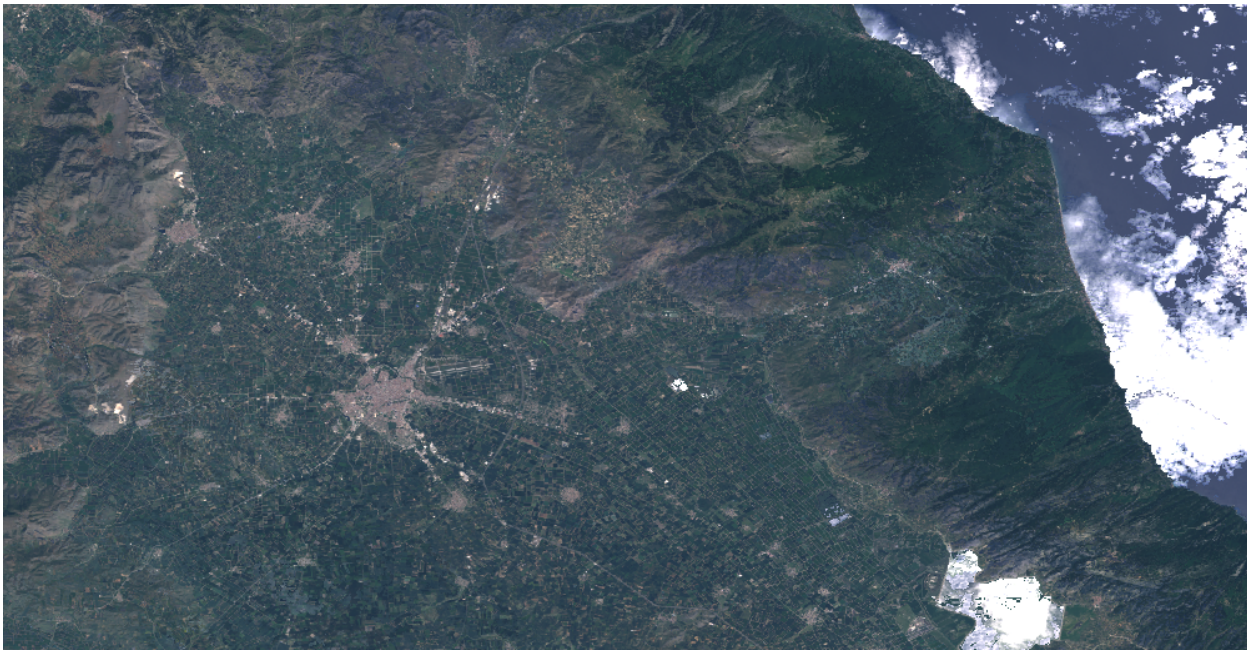


- True color of the mosaic

2018

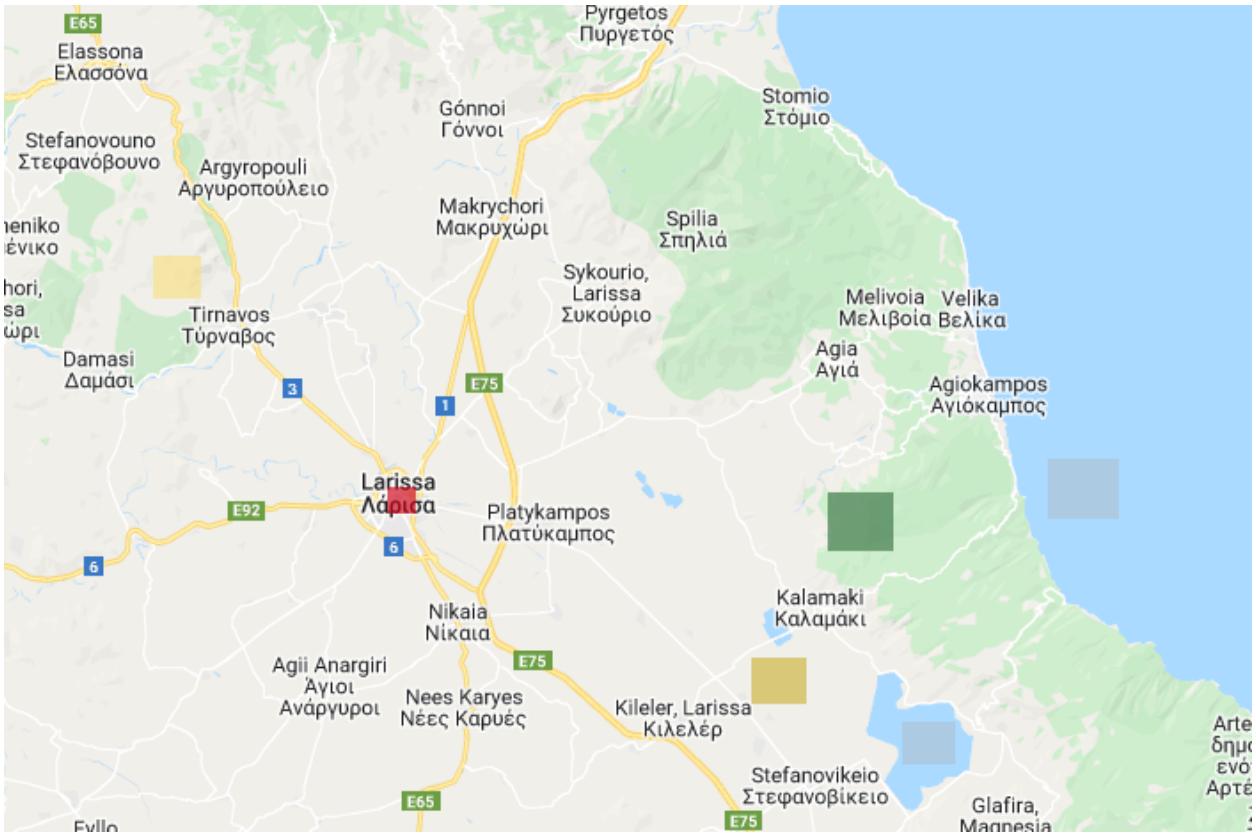


2019



4. NDVI Time Series

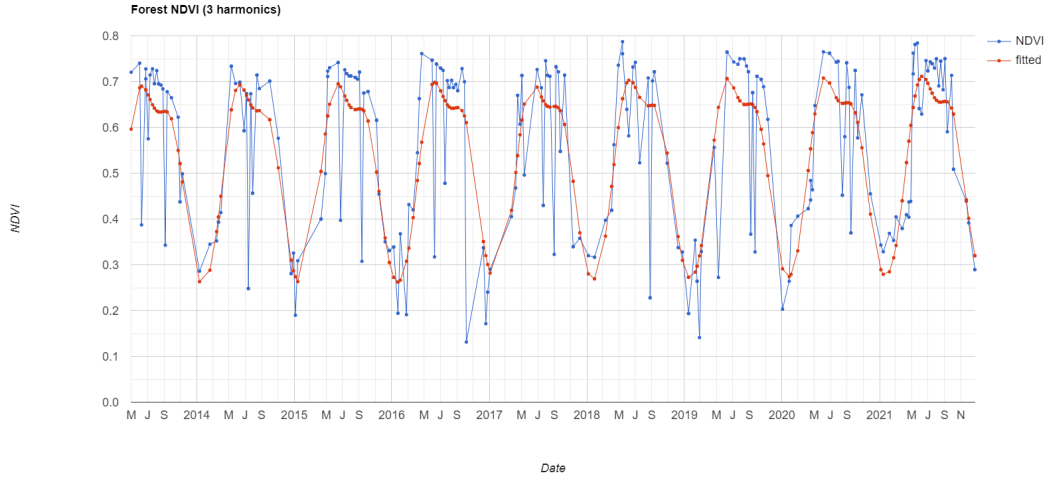
We choose 4 regions of different types, specifically **forest** (green), **cropland** (dark yellow), **barren** (light yellow), **urban** (red). In light blue we can also see a 5th type: **water,** which will be used in step 5 (classification).

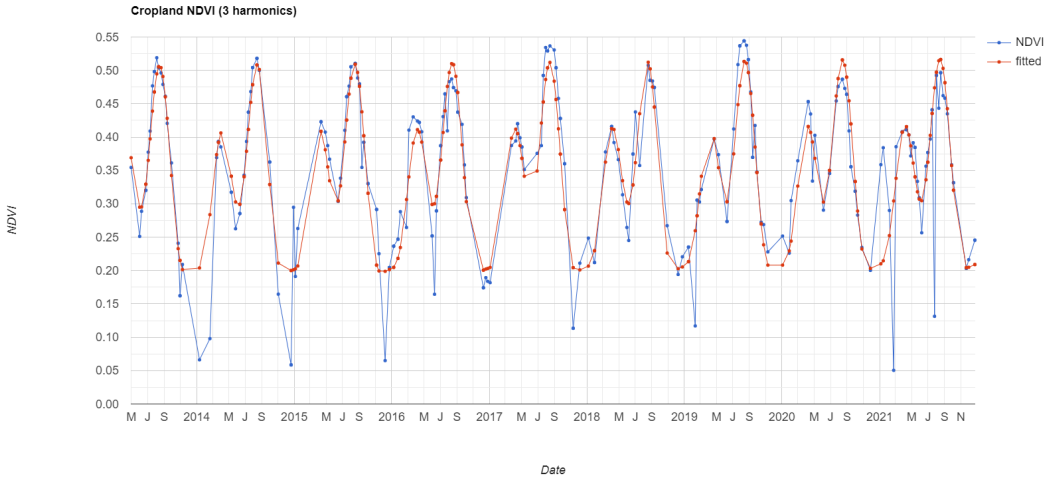


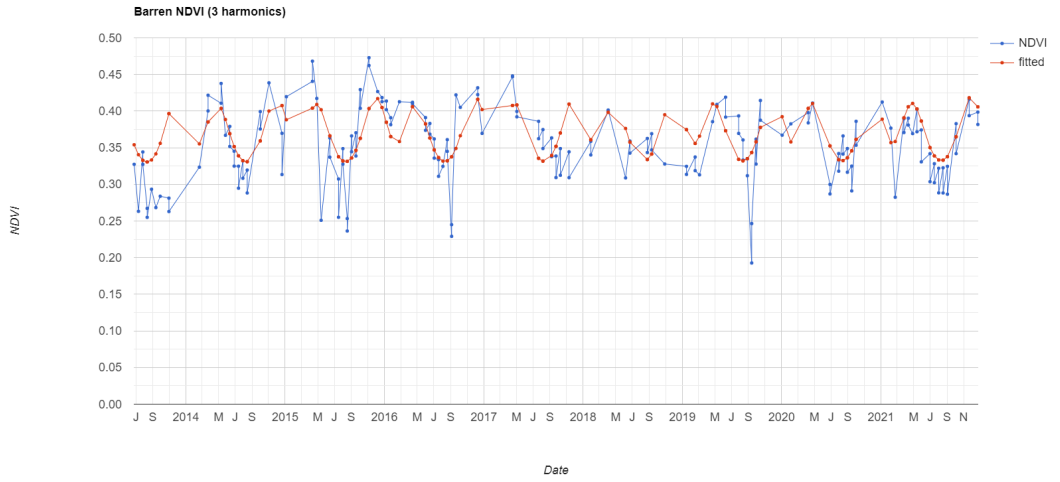
For each of these regions we plot their NDVIs over time and fit a wave composed of 3 harmonics using least squares. (*Note that in the app in step 6 one can alter the number of harmonics as well as plot other quality bands.*)

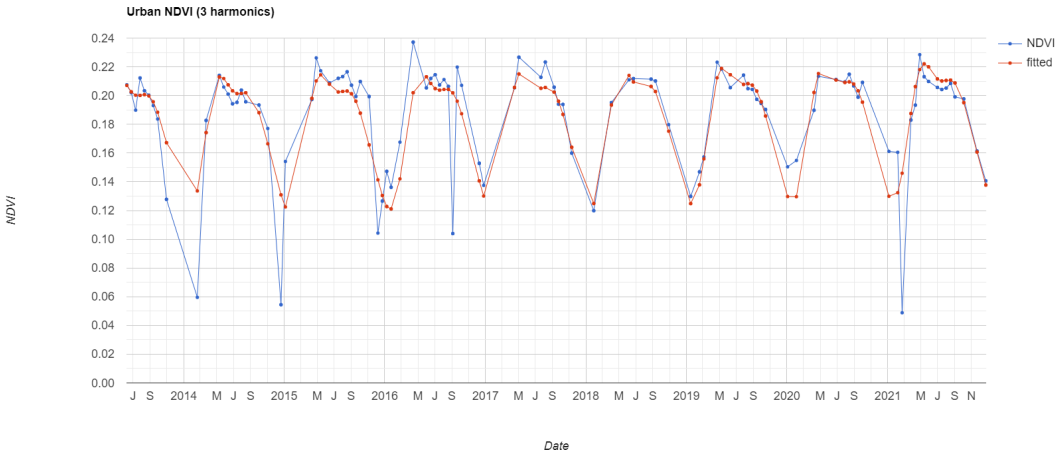
We observe that

- The *forest* NDVI goes through 1 cycle each year.
- The *cropland* goes through 1 cycle a year, with half-year “subcycles”, possibly due to planting and harvesting twice a year on the same land.
- The *barren land* has a relatively irregular and low NDVI.
- The *urban land* has a regular 1-year cycle, but with very low NDVI values, even at the peaks. Also the wave looks more of a cycloid than a sinusoid.


Forest NDVI (3 harmonics)
NDVI
fitted
NDVI
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
M J S 2014 M J S 2015 M J S 2016 M J S 2017 M J S 2018 M J S 2019 M J S 2020 M J S 2021 M J S N
Date


Cropland NDVI (3 harmonics)
NDVI
fitted
NDVI
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
M J S 2014 M J S 2015 M J S 2016 M J S 2017 M J S 2018 M J S 2019 M J S 2020 M J S 2021 M J S N
Date





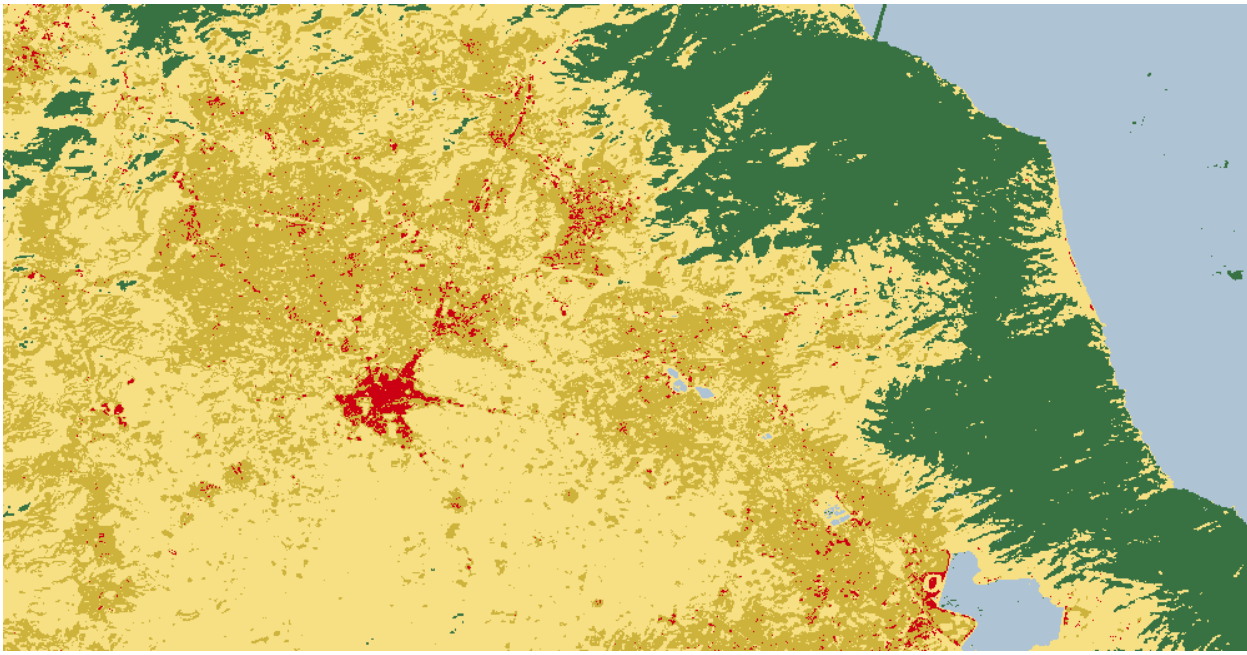
5. Classification of Land Usage Type

We first create an image using the median value of each pixel during 2019, so that most clouds (“extreme values”) get cleared. From each of the 5 region types of the previous step, we uniformly sample 1000 points, thus creating a fully balanced dataset of 5000 points in total. With this dataset as our training set we proceed to train a Support Vector Machine (radial basis kernel, γ=0.5, C=10), and a Random Forest (number of trees = 10), using bands 2-10 as the independent variables. The results are shown below.
We observe that the Random Forest did a very good job in classifying the regions, with its main mistakes being due to not providing it with enough classes: for example snow and coastlines are mistaken for urban areas. On the other hand, the SVM did poorly on differentiating between barren land and cropland, as well as detecting urban areas. The

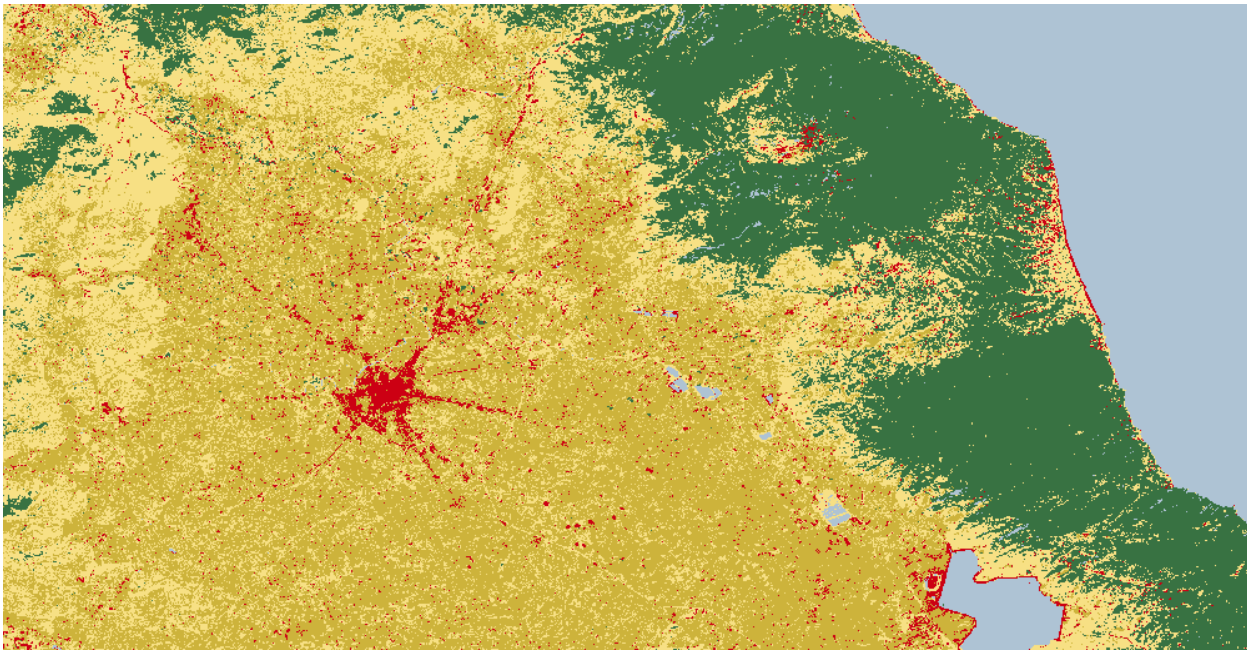
SVM seems to try to be less sensitive to small changes than the Random Forest is, and the svm image is less “fine grained” than the Random Forest one.
These results are to be expected since ensemble models like Random Forests tend to outperform non-ensemble ones.

- Median Image

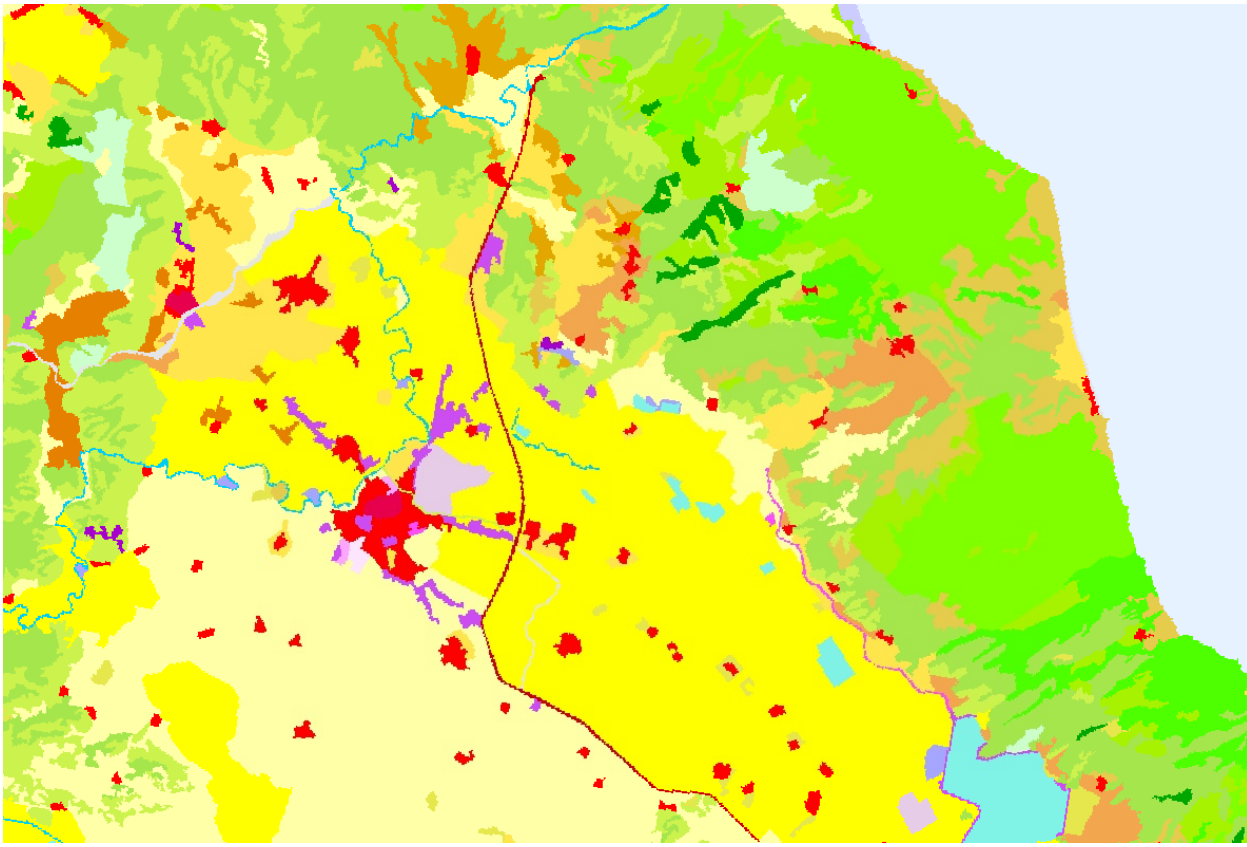
- Support Vector Machine



- Random Forest



- CORINE Land Cover 2018



6. Creating a Custom App

In the app https://konstantinospapadakis.users.earthengine.app/view/lab1 , one can pick a region and a quality band to plot a time series for that region and band. Additionally, a harmonic model is fitted and plotted, for a number of harmonics of the user's choice.

The code used for this document can be found in https://code.earthengine.google.com/?accept_repo=users/KonstantinosPapadakis/GeospatialLabs

We also list it here:

```
// IMPORTS
var l8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA"),
    aoi =
        /* color: #ff06db */
        /* shown: false */
        /* displayProperties: [
          {
            "type": "rectangle"
          }
        ] */
        ee.Geometry.Polygon(
            [[[22.167981530434947, 39.88956546831169],
            [22.167981530434947, 39.46679033875962],
            [22.986462975747447, 39.46679033875962],
            [22.986462975747447, 39.88956546831169]]], null, false),
    water =
        /* color: #aec3d4 */
        /* shown: false */
        /* displayProperties: [
          {
            "type": "rectangle"
          },
          {
            "type": "rectangle"
          }
        ] */
        ee.Geometry.MultiPolygon(
            [[[[22.808742141083965, 39.502764254564426],
```

```
          [22.808742141083965, 39.47732865085638],
          [22.850627517060527, 39.47732865085638],
          [22.850627517060527, 39.502764254564426]]],
          [[[22.921193509183556, 39.659993707477874],
          [22.921193509183556, 39.624038821007815],
          [22.976125149808556, 39.624038821007815],
          [22.976125149808556, 39.659993707477874]]]], null, false),
cropland =
    /* color: #cdb33b */
    /* shown: false */
    /* displayProperties: [
      {
        "type": "rectangle"
      }
    ] */
    ee.Geometry.Polygon(
        [[[22.6917256766282, 39.54117596679023],
        [22.6917256766282, 39.513370676356814],
        [22.734984343620386, 39.513370676356814],
        [22.734984343620386, 39.54117596679023]]], null, false),
forest =
    /* color: #387242 */
    /* shown: false */
    /* displayProperties: [
      {
        "type": "rectangle"
      }
    ] */
    ee.Geometry.Polygon(
        [[[22.75111645215788, 39.639330568005924],
        [22.75111645215788, 39.60468753476372],
        [22.8015848969821, 39.60468753476372],
        [22.8015848969821, 39.639330568005924]]], null, false),
urban =
    /* color: #cc0013 */
    /* shown: false */
    /* displayProperties: [
      {
        "type": "rectangle"
      }
```

```
        ] */
        ee.Geometry.Polygon(
            [[[22.41138267974303, 39.642188317437],
            [22.41138267974303, 39.62686948051071],
            [22.432046417993764, 39.62686948051071],
            [22.432046417993764, 39.642188317437]]], null, false),
    barren =
        /* color: #f7e084 */
        /* shown: false */
        /* displayProperties: [
          {
            "type": "rectangle"
          }
        ] */
        ee.Geometry.Polygon(
            [[[22.229529548781247, 39.78055743048465],
            [22.229529548781247, 39.75495982564443],
            [22.26592176069531, 39.75495982564443],
            [22.26592176069531, 39.78055743048465]]], null, false);


// 2. SHOW TRUE AND FALSE COLOR IMAGES OF 2019

var outputURLs = false;
var urlParams = { region: aoi, dimensions: 1000, format: 'png' };

var mergeDicts = function (a, b) {
    var res = {};
    for (var x in a) {
        res[x] = a[x];
    }
    for (var y in b) {
        res[y] = b[y];
    }
    return res;
};

var addQualityBands = function (image) {
    var ndvi = image.normalizedDifference(['B5', 'B4']).rename('NDVI');
    var evi = image.expression(
```

```
        '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))',
        {
            'NIR': image.select('B5'),
            'RED': image.select('B4'),
            'BLUE': image.select('B2')
        }
    ).rename('EVI');
    var ndwi = image.normalizedDifference(['B3', 'B5']).rename('NDWI');
    return image.addBands([ndvi, evi, ndwi]);
};

var leastCloudy2019 = l8
    .map(addQualityBands)
    .filterBounds(aoi)
    .filterDate(ee.Date('2019').getRange('year'))
    .sort('CLOUD_COVER', false)
    .mosaic();

var rgbVisParams = { bands: ['B4', 'B3', 'B2'], max: 0.3 };
var pseudVisParams = { bands: ['B5', 'B4', 'B3'], max: 0.3 };
var ndviVisParams = { bands: ['NDVI'], palette: ['blue', 'white',
'green'], min: -1, max: 1 };

Map.centerObject(aoi);
Map.addLayer(leastCloudy2019.clip(aoi), rgbVisParams, 'Clearest True RGB
2019');
Map.addLayer(leastCloudy2019.clip(aoi), pseudVisParams, 'Clearest Pseudo
RGB 2019');
Map.addLayer(leastCloudy2019.clip(aoi), ndviVisParams, 'Clearest NDVI
2019');

if (outputURLs) {
    print('Clearest True RGB 2019',
leastCloudy2019.getThumbURL(mergeDicts(rgbVisParams, urlParams)));
    print('Clearest Pseudo RGB 2019',
leastCloudy2019.getThumbURL(mergeDicts(pseudVisParams, urlParams)));
    print('Clearest NDVI 2019',
leastCloudy2019.getThumbURL(mergeDicts(ndviVisParams, urlParams)));
}
```

```
// 3. SHOW MAX NDVI for 2018 and 2019. ALSO SHOW THE DAY OF THE YEAR.

var addDOY = function (image) {
    var doy = image.date().getRelative('day', 'year');
    var doyBand = ee.Image.constant(doy).uint16().rename('DOY');
    return image.addBands(doyBand);
};

var doyVisParams = { bands: ['DOY'], palette: ['white', 'blue'], min: 0,
max: 366 };

['2018', '2019'].forEach(function (year) {
    var greenest = l8
        .map(addQualityBands)
        .filterBounds(aoi)
        .filter(ee.Filter.lt('CLOUD_COVER', 20))
        .map(addDOY)
        .filterDate(ee.Date(year).getRange('year'))
        .qualityMosaic('NDVI');
    Map.addLayer(greenest.clip(aoi), rgbVisParams, 'Greenest True RGB ' +
year);
    Map.addLayer(greenest.clip(aoi), ndviVisParams, 'Max NDVI ' + year);
    Map.addLayer(greenest.clip(aoi), doyVisParams, 'DOY ' + year);

    if (outputURLs) {
        print('Greenest True RGB ' + year,
greenest.getThumbURL(mergeDicts(rgbVisParams, urlParams)));
        print('Max NDVI ' + year,
greenest.getThumbURL(mergeDicts(ndviVisParams, urlParams)));
        print('DOY ' + year, greenest.getThumbURL(mergeDicts(doyVisParams,
urlParams)));
    }
});


// 4. PLOT NDVI TIME SERIES ALONGSIDE WITH THE FITTED HARMONICS

var addConstant = function (image) {
    return image.addBands(ee.Image(1).rename('constant'));
```

```
};

var addTime = function (image) {
    var date = image.date();
    var years = date.difference(ee.Date('1970-01-01'), 'year');
    var timeRadians = ee.Image(years.multiply(2 * Math.PI));
    return image.addBands(timeRadians.rename('t').float());
};

var getNames = function (base, nHarmonics) {
    // returns base_1, base_2, ..., base_nHarmonics
    return ee.List.sequence(1, nHarmonics).map(function (i) {
        return ee.String(base).cat(ee.Number(i).int());
    });
};

var addHarmonics = function (nHarmonics) {
    var freqs = ee.List.sequence(1, nHarmonics);
    var cosNames = getNames('cos_', nHarmonics);
    var sinNames = getNames('sin_', nHarmonics);
    return function (image) {
        var freqBands = ee.Image.constant(freqs);  // one band for each
freq
        var time = ee.Image(image).select('t');
        var cosines = time.multiply(freqBands).cos().rename(cosNames);
        var sines = time.multiply(freqBands).sin().rename(sinNames);
        return image.addBands(cosines).addBands(sines);
    };
};

var fitHarmonics = function (data, nHarmonics, dependent) {
    var cosNames = getNames('cos_', nHarmonics);
    var sinNames = getNames('sin_', nHarmonics);
    var independents = ee.List(['constant',
't']).cat(cosNames).cat(sinNames);
    var coeffs = data
        .select(independents.add(dependent))
        .reduce(ee.Reducer.linearRegression(independents.length(), 1))
        .select('coefficients')
        .arrayProject([0])
```

```
        .arrayFlatten([independents]);
    return data.map(function (image) {
        return image.addBands(
            image.select(independents)
                .multiply(coeffs)
                .reduce('sum')
                .rename('fitted'));
    });
};

var nHarmonics = 3;
var dependent = 'NDVI';

var fitted = fitHarmonics(
    l8
        .filter(ee.Filter.lt('CLOUD_COVER', 20))
        .map(addQualityBands)
        .map(addConstant).map(addTime).map(addHarmonics(nHarmonics)),
    nHarmonics,
    dependent
);

var makeChart = function (region, name) {
    return ui.Chart.image
        .series(
            fitted.select(['fitted', dependent]),
            region, ee.Reducer.mean(), 30)
        .setOptions({
            title: name + ' ' + dependent + ' (' + nHarmonics + '
harmonics)',
            hAxis: { title: 'Date' },
            vAxis: { title: dependent },
            lineWidth: 1,
            pointSize: 3
        });
};

var d = { 'Forest': forest, 'Cropland': cropland, 'Barren': barren,
'Urban': urban };
for (var key in d) {
```

```
    var chart = makeChart(d[key], key);
    print(chart);
}


// 5. CLASSIFY REGIONS

var image = l8
    .map(addQualityBands)
    .filterDate(ee.Date('2019').getRange('year'))
    .filterBounds(aoi)
    .filter(ee.Filter.lt('CLOUD_COVER', 20))
    .median();

var trainPolygons = [water, cropland, forest, urban, barren];  // classes
0, 1, ... , K
var colors = ['aec3d4', 'cdb33b', '387242', 'cc0013', 'f7e084'];
var addClass = function (i) { return function (feature) { return
feature.set({ 'class': i }) } };
var fcs = [];
trainPolygons.forEach(function (polygon, i) {
    var fc = ee.FeatureCollection.randomPoints(polygon, 1000,
42).map(addClass(i));
    fcs.push(fc);
});
var points = ee.FeatureCollection(fcs).flatten();
var bands = ['B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10'];
var training = image.select(bands).sampleRegions({
    collection: points,
    properties: ['class'],
    scale: 30
});

var svm = ee.Classifier.libsvm({
    kernelType: 'RBF',
    gamma: 0.5,
    cost: 10
});
svm = svm.train(training, 'class', bands);
print('SVM confusion matrix:', svm.confusionMatrix());
```

```
var svmPred = image.classify(svm);

var rf = ee.Classifier.smileRandomForest(10);
rf = rf.train(training, 'class', bands);
print('Random Forest confusion matrix:', rf.confusionMatrix());
print('Random Forest explanation:', rf.explain());
var rfPred = image.classify(rf);

var classVisParams = { min: 0, max: (colors.length - 1), palette: colors
};
Map.addLayer(image.clip(aoi),
    rgbVisParams,
    'Median True RGB 2019');
Map.addLayer(svmPred.clip(aoi),
    classVisParams,
    'SVM Predictions 2019');
Map.addLayer(rfPred.clip(aoi),
    classVisParams,
    'Random Forest Predictions 2019');

var corineLC =
ee.Image('COPERNICUS/CORINE/V20/100m/2018').select('landcover');
Map.addLayer(corineLC.clip(aoi), {}, 'CORINE Landcover 2018');

if (outputURLs) {
    print('Median True RGB 2019',
image.getThumbURL(mergeDicts(rgbVisParams, urlParams)));
    print('SVM Predictions 2019',
svmPred.getThumbURL(mergeDicts(classVisParams, urlParams)));
    print('Random Forest Predictions 2019',
rfPred.getThumbURL(mergeDicts(classVisParams, urlParams)));
}
```