Land Cover Mapping (Supervised and Unsupervised Classification) in GEE

Created by: N. Lakmal Deshapriya (lakmal@ait.ac.th, lakmalnd@yahoo.com)
Geoinformatics Centre (GIC), Asian Institute of Technology (AIT), Thailand

First, let's create a new notebook. Then, let's install the required libraries, import them into our current notebook, authenticate, and initialize.

```
!pip install geemap >> /dev/null
!pip install pycrs >> /dev/null

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import ee
import geemap

ee.Authenticate()
ee.Initialize(project='[Add Your Project ID]')
```

Content

1) Supervised Classification

In the context of "Supervised Classification" for landcover mapping, we provide examples of various landcover types within our study area to a machine learning algorithm. We then task a machine learning algorithm with determining optimal ways to categorize all pixels in the study area into different landcover classes.

In the following section, let's try to map landcover of the surrounding area of a city call "Krabi", located in the southern part of Thailand.

In this case study, we will use Sentinel-2 Surface Reflectance data. First, let's define image collection covering "Krabi" city in the dry season of 2020, where cloud cover is less compared to the wet season. Then we will use Medium reduction over the ImageCollection to combine all images in the ImageCollection to a single image. After that, we will clip by study area, and select Red (B4), Green (B3), Blue (B2) and NIR (B8) bands.

```
studyArea = ee.Geometry.Rectangle(98.82, 7.98, 99.02, 8.18)

KrabiCol = ee.ImageCollection('COPERNICUS/S2_SR').filterDate('2020-01-01','2020-04-30').filterBounds(studyArea)

KrabiImg = KrabiCol.median().clip(studyArea).select(['B4','B3','B2', 'B8'])

Map = geemap.Map(center = [8.08,98.92], zoom = 12)
vis_Para = {'min': 0.0, 'max': 1000, 'bands':['B4','B3','B2']}
Map.addLayer(KrabiImg, vis_Para, name="Krabi RGB Image")
Map
```

Note: Medium reduction operation over an ImageCollection, helps getting rid of cloud, as well as cloud shadow. Try the Minimum reduction operation as well, and see the difference.

For this case study, we will use pre-existing ground-truth data that consisted of approximately 300 example points across 4 landcover classes, as listed below.

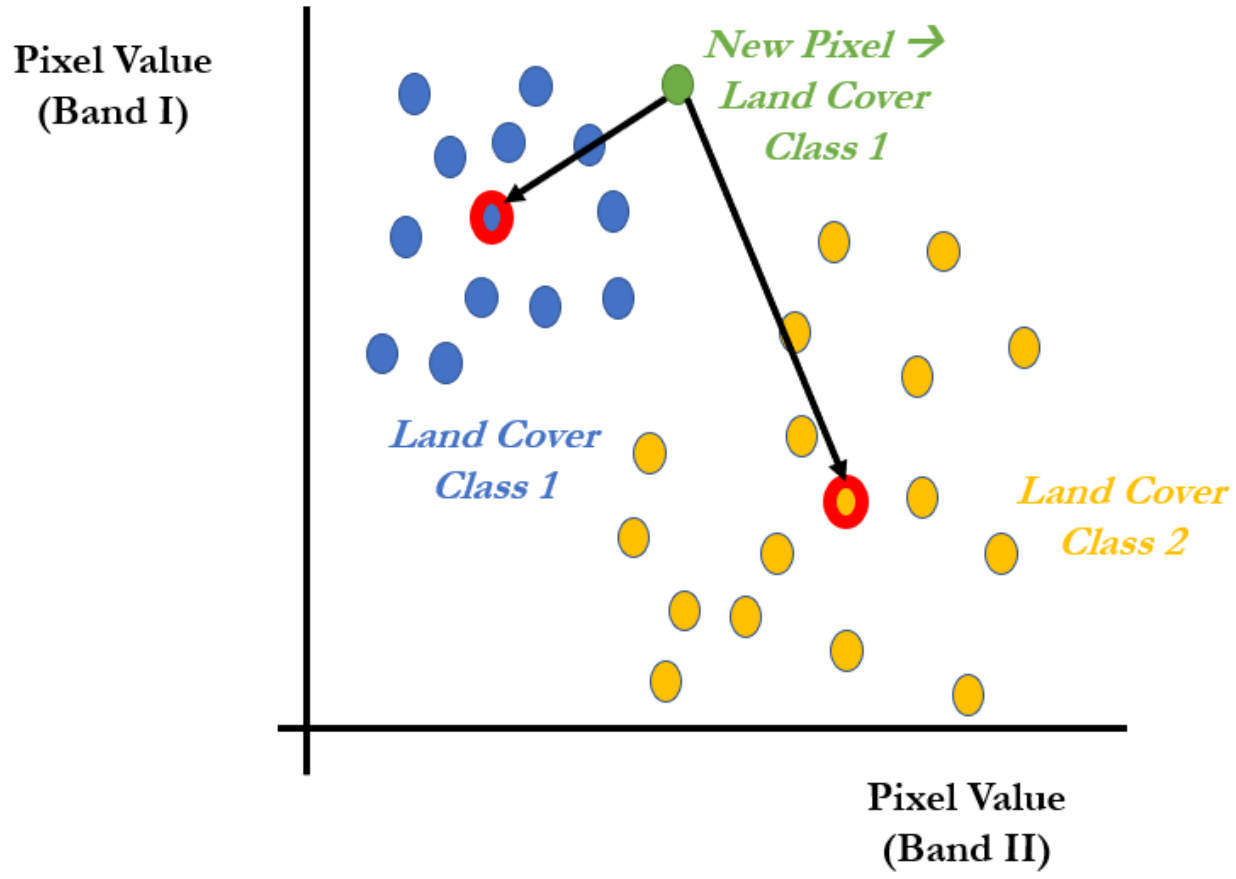
- 1) Forest class -> (id = 1)
- 2) Urban class -> (id = 2)
- 3) Water class -> (id = 3)
- 4) Agriculture class -> (id = 4)

Now, let's upload *"krabi_gt_train.shp"* (with *.prj*, *.dbf*, and *.shx*) Shapefile to GEE's Session Storage. And import the uploaded Shapefile to GEE using *geemap* library. Then let's extract pixel values of *"KrabiImg"* image at ground-truth points in *"krabi_gt_train.shp"* Shapefile, as below.

```
lc_gt_train = geemap.shp_to_ee('/content/krabi_gt_train.shp')
example_pnts = KrabiImg.sampleRegions(lc_gt_train, scale=30)
```

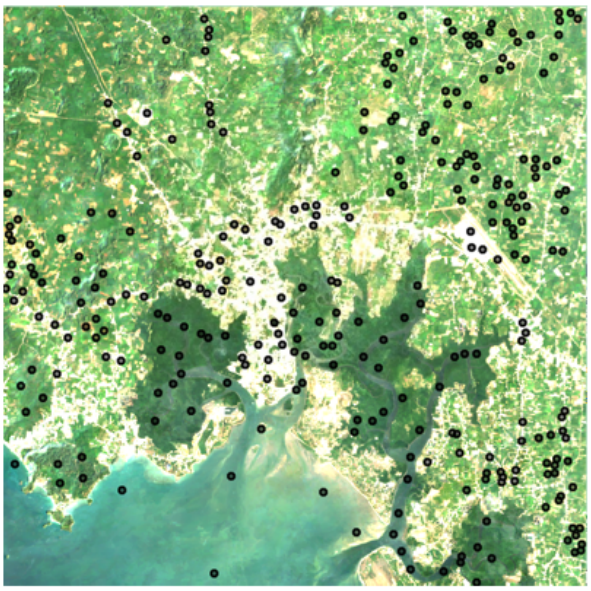
"First, we can utilize the Minimum Distance Classification algorithm provided by GEE to train a classifier for predicting landcover, as shown below,

Note: The "Minimum Distance" Classification algorithm is one of the simplest and well-known classification algorithms. In this case study, the algorithm first estimates the centers of landcover classes in the feature space created from our example points in *"krabi_gt_train.shp"*. The class of all other pixels in the image will be determined by the minimum distance to the centers of landcover classes. For example, if a new pixel is closer to the urban landcover class than to other landcover classes, that new pixel will be assigned to the urban landcover class, and vice versa. This concept is visually represented in the figure below, which illustrates the feature space.

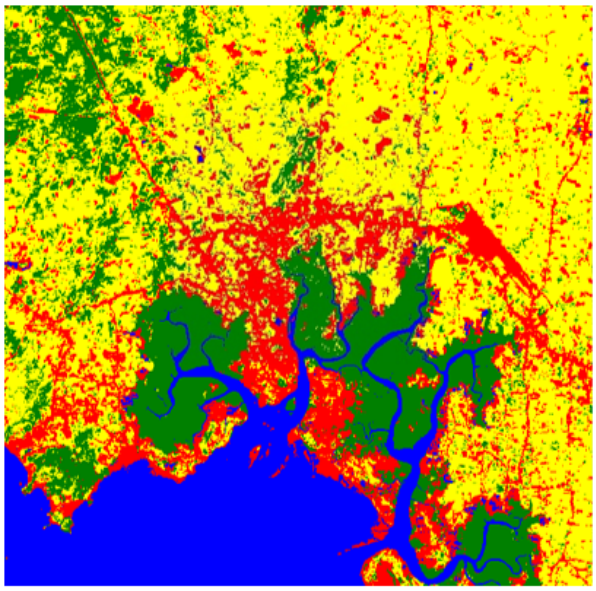



```
my_classifier = ee.Classifier.minimumDistance().train(example_pnts, 'id')
KrabiClassImg = KrabiImg.classify(my_classifier)
```

```
Map = geemap.Map(center = [8.08,98.92], zoom = 12)
vis_Para = {'min': 1, 'max': 4, 'palette':['green', 'red', 'blue', 'yellow']}
Map.addLayer(KrabiClassImg, vis_Para, name="Krabi - Landcover")
Map
```



Input RGB Image



Classified Landcover Map

Note: Here, we are utilizing the Minimum Distance Classifier. Google Earth Engine (GEE) offers several classification algorithms, among which the Random Forest Classifier stands out as a well-known option in the remote sensing community. It is considered as better and more sophisticated than the Minimum Distance Classifier. Therefore, the Random Forest Classifier is more suitable for advanced classifications, such as landcover mapping. The Random Forest classifier is based on decision trees and employs multiple decision trees to classify an image.

Exercise: Try the Random Forest Classifier (Sample Code: *".smileRandomForest(numberOfTrees=100)"*) as well and compare the results with Minimum Distance Classifier.

Exercise: Change "numberOfTrees" parameter in Random Forest Classifier and observe the effect on the results.

Exercise: Rerun the classification adding more bands and compare the results.

Exercise: Export final landcover map and open it in QGIS to make a nice map layout.

2) Accuracy Assessment

Even though the classified results appear good on the map, it is essential to numerically determine the accuracy of our classification results. Typically, this is done using a separate ground-truth dataset (*"krabi_gt_val.shp"*), referred to as the validation dataset. Since our training was based on the *"krabi_gt_train.shp"* ground-truth dataset, there's a possibility that our classifier might be biased towards this particular dataset. Therefore, it is always better to use a different ground-truth dataset for accuracy assessment / validation.

Now, let's upload *"krabi_gt_val.shp"* Shapefile (with *.prj*, *.dbf*, and *.shx*) to GEE's Session Storage. Then let's import the uploaded Shapefile to Google Colab and extract pixel values of *"KrabiImg"* image at ground-truth points in *"krabi_gt_val.shp"* Shapefile.

```
lc_gt_val = geemap.shp_to_ee('/content/krabi_gt_val.shp')
validation_pnts = KrabiImg.sampleRegions(lc_gt_val, scale=30)
```

Then let's classify these validation points using our trained Classifier (*"my_classifier"*) and print a sample feature from the classified validation dataset to inspect.

```
validation_class = validation_pnts.classify(my_classifier)

print(validation_class.first().getInfo())
```

Results ->
{'geometry': None, 'id': '0_0', 'properties': {'B2': 252.5, 'B3': 438, 'B4': 250, 'B8': 3224.5, 'classification': 1, 'id': 1}, 'type': 'Feature'}

Now, we can see that, original landcover class is in the *"id"* field/property and the classified / predicted landcover class is in *"classification"* field/property. Now we can generate an "Error Matrix" based on these 2 fields using GEE's *"errorMatrix"* method. And we can print Total Accuracy and Kappa Coefficient (*optional*) from the Error Matrix. In the Error Matrix rows are corresponding to the actual classes/values, and columns are corresponding to the predicted classes/values.

```
val_accuracy = validation_class.errorMatrix('id', 'classification')

print(val_accuracy.getInfo())
print(val_accuracy.accuracy().getInfo())
print(val_accuracy.kappa().getInfo()) # optional
```

Finally, we can see that the Total Accuracy is 94% with Random Forest Classifier which is pretty good. This means 94% of validation points have been correctly classified.

3) Unsupervised Classification (Clustering)

In the above section, we have used examples to guide the classification. That process is known as Supervised Classification. Even without examples, we can perform classification. This is known as Clustering in Machine Learning. And in the remote sensing field, this is known as Unsupervised Classification.

In Clustering/Unsupervised Classification, the classification is performed automatically based on the proximity of pixels (density of pixels) in a feature space. If a set of pixels is close together in a feature space, they will be classified as one cluster automatically.

In the "k means" clustering algorithm that we are going to use in this section, we can define number of expected clusters (using *"nClusters"* parameter). So the "k means" clustering algorithm will automatically cluster pixels into *"nClusters"* number of clusters.

This short video shows how the "k means" clustering algorithm works, https://www.youtube.com/watch?v=5I3Ei69I40s . The steps in the "k means" clustering algorithm are also summarized below.

First let's assume, the *"nClusters"* is K,

- Step-1) Select random K points or centroids for each expected cluster
- Step-2) Assign each data point to their closest centroid
- Step-3) Calculate centroid of newly created clusters, and place a new centroids of each K clusters
- Step-4) Repeat 2nd and 3rd steps until centroids are not changed

Let's revisit our previous land cover case study in "Krabi city." First, let's define the image.

```
studyArea = ee.Geometry.Rectangle(98.82, 7.98, 99.02, 8.18)

KrabiCol = ee.ImageCollection('COPERNICUS/S2_SR').filterDate('2020-01-01','2020-04-30').filterBounds(studyArea)

KrabiImg = KrabiCol.median().clip(studyArea).select(['B4','B3','B2', 'B8'])
```

```
Map = geemap.Map(center = [8.08,98.92], zoom = 12)
vis_Para = {'min': 0.0, 'max': 1000, 'bands':['B4','B3','B2']}
Map.addLayer(KrabiImg, vis_Para, name="Krabi RGB Image")
Map
```

The entire image is too large to run the clustering algorithm, so let's first select a random sample of 2500 pixels from the FCC image, as shown below:

```
KrabiImg_sample = KrabiImg.sample(numPixels=2500, scale=30)
```

Then we can run our clustering algorithm on the randomly selected 2500 pixels/samples for 4 classes (*"nClusters=4"*) as below,

```
my_clustering = ee.Clusterer.wekaKMeans(nClusters=5).train(KrabiImg_sample)
KrabiImg_class = KrabiImg.cluster(my_clustering)

Map = geemap.Map(center = [8.08,98.92], zoom = 12)
vis_Para = {'min': 0, 'max': 3, 'palette':['green', 'red', 'blue', 'yellow']}
Map.addLayer(KrabiImg_class, vis_Para, name="Clustered - 4 Classes")
Map
```

Exercise: Try visualize the classified image with *"KrabiImg"* in the background to see how good the classification result is.

Exercise: Use "Inspector" tool in the GEEMAP map's toolbox to find classified pixel values of "Agriculture" land cover class. Then produce binary map showing only "Agriculture" and "Non-Agriculture" land cover classes. ***Hint:*** *Use "equals" Boolean operation.*

Exercise: Change the *"nClusters"* parameter in the "k means" clustering algorithm and compare the results. In the case of Unsupervised Classification, we often classify images into more classes than needed. Later, we reclassify them back to the desired number of classes. This is a common practice.

4) Case Study in Afghanistan

In this case study, we will try to create a simple land cover map of central region of Afghanistan.

First, let's create a new notebook. Then, let's install the required libraries, import them into our current notebook, authenticate, and initialize.

```
!pip install geemap >> /dev/null
!pip install pycrs >> /dev/null

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import ee
import geemap

ee.Authenticate()
ee.Initialize(project='[Add Your Project ID]')
```

Then let's upload our study area shapefile *"central_r.shp"* (with *.prj*, *.dbf*, and *.shx*) to GEE's Session Storage, and import it in to our current Google Colab notebook. Then let's create a Landsat-8 image covering our study area by minimizing the presence of clouds and cloud shadows as much as possible.

```
afg_central = geemap.shp_to_ee('/content/central_r.shp')
```

```
imgCol = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterDate('2018-01-01', '2018-12-31').filterBounds(afg_central)
imgCol = imgCol.filter(ee.Filter.lt('CLOUD_COVER', 20))

img2018 = imgCol.reduce(ee.Reducer.percentile([20]))

img2018 = img2018.select(['SR_B2_p20', 'SR_B3_p20', 'SR_B4_p20', 'SR_B5_p20', 'SR_B6_p20']).clip(afg_central)

Map = geemap.Map(center = [33.9,67.7], zoom = 6)
vis_Para = {'min': 0, 'max': 30000, 'bands': ['SR_B4_p20', 'SR_B3_p20', 'SR_B2_p20']}
Map.addLayer(img2018, vis_Para, name="Exaple LANDSAT Image")
Map
```

Then let's upload our training data shapefile *"afg_gt_val.shp"* (with *.prj*, *.dbf*, and *.shx*) to GEE's Session Storage, and import it in to our current Google Colab notebook. And then extract pixel values of *"img2018"* image at training points in the *"afg_gt_val.shp"* Shapefile.

```
lc_gt_train = geemap.shp_to_ee('/content/afg_gt_train.shp')
example_pnts = img2018.sampleRegions(lc_gt_train, scale=30)
```

The *"afg_gt_train.shp"* is consisted of 6 classes as below. Land cover class ID is in *"lc_id"* field.

- 1) Other landcover class -> (id = 0)
- 2) Forest class -> (id = 1)
- 3) Urban class -> (id = 2)
- 4) Water class -> (id = 3)
- 5) Agriculture class -> (id = 4)
- 6) Rangeland class -> (id = 5)

Now, let's classify the image using the Random Forest Classifier and visualize the classification results.

```
my_classifier = ee.Classifier.smileRandomForest(numberOfTrees=200).train(example_pnts, 'lc_id')
afg_central_class = img2018.classify(my_classifier)

Map = geemap.Map(center = [33.9,67.7], zoom = 6)
vis_Para = {'min': 0, 'max': 5, 'palette':['grey', 'green', 'red', 'blue', 'yellow', 'purple']}
Map.addLayer(afg_central_class, vis_Para, name="Afg Central Region - LC")
Map
```

The classification result isn't particularly promising. Nevertheless, let's move forward and calculate the accuracy of the classification to get a clearer assessment.

```
lc_gt_val = geemap.shp_to_ee('/content/afg_gt_val.shp')
validation_pnts = img2018.sampleRegions(lc_gt_val, scale=30)

validation_class = validation_pnts.classify(my_classifier)
val_accuracy = validation_class.errorMatrix('lc_id', 'classification')
print(val_accuracy.accuracy().getInfo())
```

The total accuracy is around 62%, which is not particularly high. Next, let's explore ways to improve the accuracy. There are various approaches, such as collecting more ground-truth data or adding more meaningful bands to the satellite image. However, collecting additional ground-truth data is not currently feasible, so let's focus on adding more meaningful bands to the satellite image.

Given that our classification involves grassland, forest, and agriculture classes, we can hypothesize that incorporating time-series NDVI data may enhance accuracy. So let's try that first.

Now let's add quarterly NDVI images to the satellite image that is being classified.

```
def getNDVI(startY, endY, startM, endM):

  imgCol = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterBounds(afg_central)
  imgCol = imgCol.filter(ee.Filter.calendarRange(startY,endY,'year')).filter(ee.Filter.calendarRange(startM,endM,'month
  imgCol = imgCol.filter(ee.Filter.lt('CLOUD_COVER', 20))

  tempImg = imgCol.reduce(ee.Reducer.percentile([20])).clip(afg_central)

  ndvi = tempImg.expression('(NIR-RED)/(NIR+RED)', {'NIR': tempImg.select('SR_B5_p20'), 'RED': tempImg.select('SR_B4_p2

  return ndvi

ndvi_q1 = getNDVI(2016, 2020, 1, 3)
ndvi_q2 = getNDVI(2016, 2020, 4, 6)
ndvi_q3 = getNDVI(2016, 2020, 7, 9)
ndvi_q4 = getNDVI(2016, 2020, 10, 12)

img_stack = ee.Image([img2018, ndvi_q1, ndvi_q2, ndvi_q3, ndvi_q4])
```

Now let's extract pixel values of *"img_stack"* image at training points in the *"afg_gt_val.shp"* Shapefile.

```
example_pnts = img_stack.sampleRegions(lc_gt_train, scale=30)
```

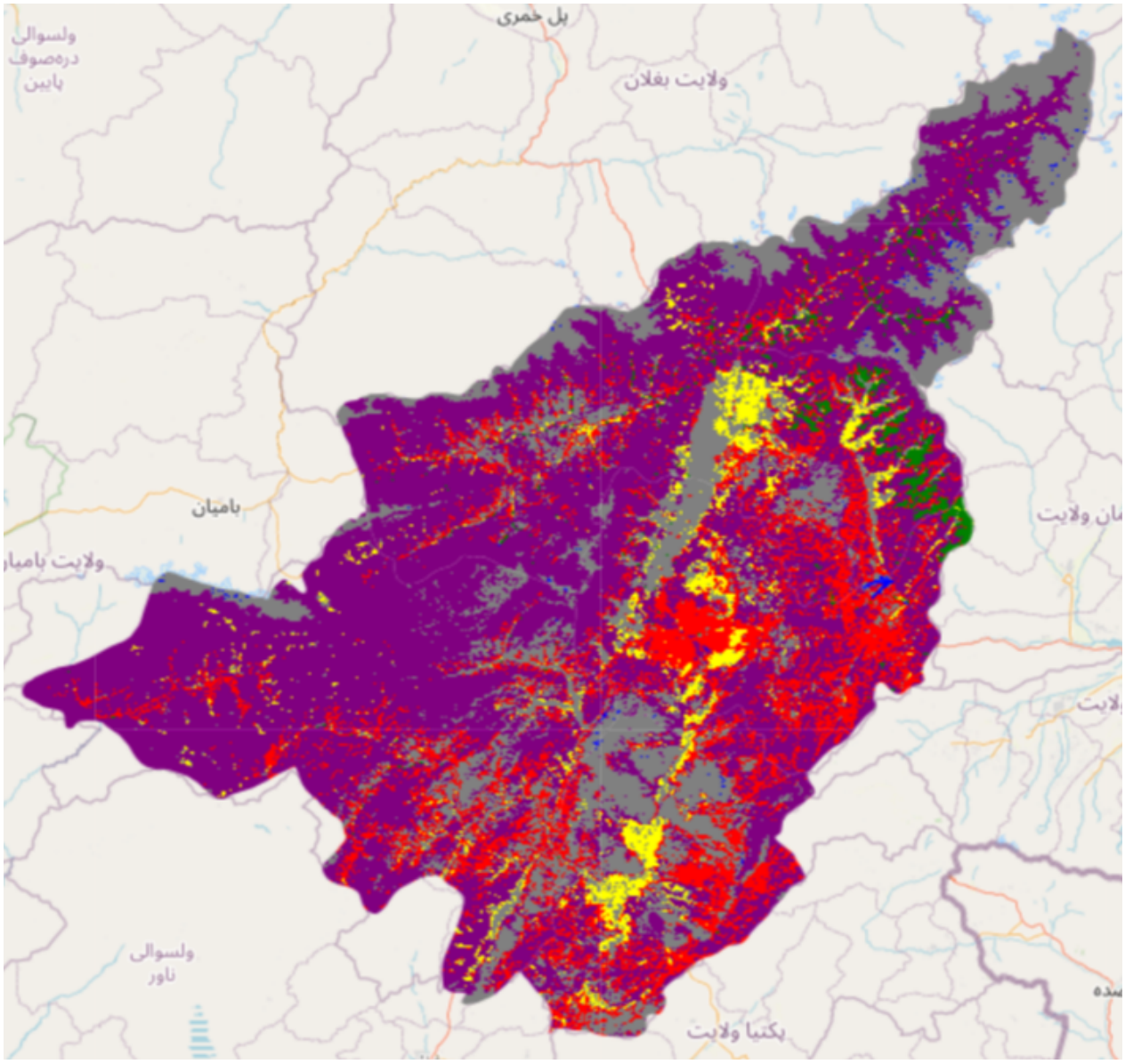
Then let's classify the new image, and visualize the results,

```
my_classifier = ee.Classifier.smileRandomForest(numberOfTrees=200).train(example_pnts, 'lc_id')
afg_central_class = img_stack.classify(my_classifier)

Map = geemap.Map(center = [33.9,67.7], zoom = 6)
vis_Para = {'min': 0, 'max': 5, 'palette':['grey', 'green', 'red', 'blue', 'yellow', 'purple']}
Map.addLayer(afg_central_class, vis_Para, name="Afg Central Region - LC")
Map
```

Finally let's try to calculate accuracy again.

```
lc_gt_val = geemap.shp_to_ee('/content/afg_gt_val.shp')
validation_pnts = img_stack.sampleRegions(lc_gt_val, scale=30)

validation_class = validation_pnts.classify(my_classifier)
val_accuracy = validation_class.errorMatrix('lc_id', 'classification')
print(val_accuracy.accuracy().getInfo())
```

Now we can see that the accuracy has increased to 77.5%, which is significantly higher than the previous classification.


پل خمری
ولسوالی
دره‌صوف
پایین
ولایت بغلان
بامیان
ولایت بامیان
ولسوالی
ناور
یکتیا ولایت