Pengembangan Aplikasi Sampling Penutup Lahan

Khairil Fahmi Faisal
Gemasakti Adzan

August 29, 2021

1 Ringkasan

`Aplikasi Sampling Penutup Lahan` dapat digunakan untuk mendesain dan melabeli sampel untuk proses klasifikasi citra penginderaan jauh dengan teknik *supervised classification machine learning*. Aplikasi ini dibangun pada platform `Jupyter Notebook` dengan menggunakan beberapa Python package seperti `ee`, `geopandas`, `ipywidgets`, dan `ipyleaflet`. Secara garis besar aplikasi ini terdiri dari dua tahapan yaitu 1) membangun set data sampel yang distribusinya didasarkan atas hasil K-Means Clustering dengan proporsi jumlah sesuai luas setiap kluster; 2) memberikan label pada setiap titik sampel secara interaktif melalui *map widget* berbasis `ipyleaflet`. Aplikasi ini dapat digunakan untuk berbagai macam citra multispektral dan hiperspektral. Melalui aplikasi ini pengguna memiliki opsi untuk menampilkan koleksi data dari katalog maupun asset `Google Earth Engine`. Hal ini akan membuat proses mendesain sampel menjadi efektif dan efisien, karena pengguna tidak perlu mengunduh data terlebih dahulu untuk dibuka pada perangkat lunak RS/GIS. Penulisan dalam platform `Jupyter Notebook` dapat digunakan sebagai arsip dan pencatatan yang baik dalam sebuah rangkaian penelitian. Selain itu aplikasi ini berkesinambungan dengan proses terkait lainnya dalam skema besar Land Cover Monitoring System yang juga dibangun dalam platform `Jupyter Notebook`.

2 Latar belakang

Pendekatan *machine-learning* dengan teknik *supervised classification* merupakan metode yang banyak dipakai untuk memproduksi peta penutup lahan

dari citra multispektral dan hiperspektral. Beberapa metode yang banyak digunakan misalnya metode *spectral distance-based* (Minimum Distance, Maximum Likelihood, Support Vector Machine/SVM); *decision tree-based* (Decision Trees/DT, Classification and Regression Trees/CART, dan Random Forests/RF); dan *neural-based* (Artificial Neural Network, K-Neural Network). Teknik *supervised classification* memerlukan *labelled samples* dalam proses membangun *classifier model*. Setiap sampel akan "dilatih" dengan membangun hubungan antara *labelled samples* dengan statistik dari *predictors* (berupa *image bands*, *transformation index*, maupun hasil *feature engineering* lainnya) dengan metode *supervised classification* di atas. Kualitas *classifier model* dalam memetakan penutup lahan (digambarkan melalui akurasi *training samples* dan *test samples*) sangat bergantung dari kualitas sampel yang digunakan. Oleh karena itu diperlukan dataset sampel yang baik, yang benar-benar merepresentasikan statistik dari suatu tipe penutup lahan.

Labelled samples salah satunya dapat diperoleh dari dataset observasi lapangan, misalnya plot sampel dan hasil observasi lapangan. Namun terkadang untuk area pemetaan yang besar, sampel tersebut tidak cukup representatif dari segi kuantitas dan distribusi spasialnya untuk digunakan sebagai sampel dalam model *supervised classification*. Oleh karena itu sampel lebih sering dibangun dengan membuat dataset sampel dengan proses observasi dan digitisasi (berupa titik maupun area) pada citra penginderaan jauh yang relevan digunakan sebagai referensi. Permasalahannya, proses ini seringkali menghasilkan dataset sampel yang tidak cukup ideal untuk model *supervised classification*. Misalnya, banyak sampel yang *redundant* dan secara statistik tidak meliput distribusi nilai piksel keseluruhan secara utuh. Artinya kecenderungan tidak terambilnya "kluster" sampel dengan karakteristik tertentu bisa terjadi.

Solusi yang dapat digunakan untuk mengatasi permasalahan tersebut di atas beberapa di antaranya adalah pendekatan *semisupervised learning* dan *active learning*. Dalam proses *semisupervised learning*, pertama dilakukan proses *unsupervised clustering* untuk mengelompokkan piksel menjadi beberapa kluster. Setiap kluster kemudian diberikan label tipe penutup lahan, beberapa di antaranya mungkin masih berupa percampuran beberapa tipe penutup lahan. Melalui observasi, kluster-kluster yang menggambarkan kelas penutup lahan murni kemudian dijadikan acuan untuk membuat *labelled sampel*. Proses pada *active learning* pada dasarnya juga dilakukan berdasarkan proses *clustering*, namun yang membedakan setiap kluster tersebut kemudian dilatih dengan sejumlah kecil *labelled samples* menggunakan metode *supervised classification* tertentu, misalnya Support Vector Machine (SVM)[1]. *Feature space* kemudian digunakan untuk mengelompokkan objek

sampel yang *robust* (hasil *medoid*) dan membuang objek sampel yang ambigu (dalam *feature space* digambarkan sebagai objek yang lokasinya beririsan dengan kluster atau kelas lain).

3 Tujuan

Projek ini bertujuan untuk mendesain sebuah aplikasi pembuatan sampel sebagai *input* dalam *supervised classification* dengan mempertimbangkan berbagai permasalahan di atas. Dalam projek ini, pendekatan yang digunakan adalah *semisupervised learning* dengan menggunakan metode K-Means Clustering. Dari setiap kluster yang dihasilkan, sejumlah titik sampel kemudian diambil secara acak dengan jumlah proporsi berdasarkan luas setiap kluster. Proses berikutnya adalah *data labelling*, yaitu memberikan label pada setiap titik sampel tersebut. Aplikasi ini dibuat pada platform `Jupyter Notebook`, sehingga prosesnya dapat direpetisi dan juga mengakomodasi perubahan yang dapat dilakukan untuk pengembangan atau perbaikan. Aplikasi ini dapat digunakan untuk berbagai citra multispektral maupun hiperspektral, namun dalam projek ini dicontohkan dengan menggunakan liputan citra Landsat 8 OLI.

4 Aplikasi Sampling Penutup Lahan

Aplikasi ini terdiri dari dua bagian, yakni `Sample generator` untuk membuat set data sampel dan `Sample data labelling` untuk memberikan label pada setiap titik sampel yang telah dibuat secara interaktif. Pengguna dapat memasukkan *input parameter* utamanya berupa `GEE asset id`, jumlah kluster, dan juga jumlah titik sampel yang ingin dibuat.

4.1 `Sample generator`

Seperti telah dikemukakan sebelumnya, set data sampel dibangun dengan mempertimbangkan hasil *unsupervised learning* baik untuk distribusi sebaran maupun kuantitasnya. Hal tersebut untuk meminimalisir unsur subjektifitas yang kerap dijumpai ketika sebaran titik sampel dibuat secara manual. Selain itu hal tersebut dilakukan untuk memastikan bahwa sampel mewakili kondisi data secara menyeluruh, yang diwakili oleh representasi kluster hasil *unsupervised learning*.

Metode *unsupervised learning* yang digunakan adalah K-Means Clustering. Pada dasarnya, K-Means Clustering berupaya untuk mengelompokkan set data sampel ke dalam sejumlah kluster/*centroids* (ditentukan sebagai

variabel `k`) berdasarkan kesamaan karakteristik data prediktor (dalam hal ini nilai spektral/piksel citra penginderaan jauh). Setiap data sampel akan dialokasikan pada kluster yang memiliki kemiripan dengannya. Setiap kluster akan memiliki titik pusat kluster yang posisinya akan berubah-ubah hingga menemukan kondisi stabil dan optimal di mana jarak tiap titik dalam sebuah kluster memiliki variasi paling kecil.

Aplikasi ini menggunakan fungsi K-Means Clustering dari package `ee` (`ee.Clusterer.WekaKMeans()`) yang diaplikasikan pada set data 10 ribu titik sampel acak. K-Means Clustering dijalankan untuk sejumlah `k` kluster yang nilainya telah ditentukan pengguna. Dari sejumlah `k` kluster tersebut kemudian dihitung proporsi luasnya untuk dijadikan dasar dalam membuat *stratified random samples* (`ee.stratifiedSample()`) yang jumlah total sampelnya ditentukan oleh pengguna. Hasil keluaran berupa set data *unlabelled samples* ini dapat diekspor oleh pengguna ke `Google Drive` untuk proses berikunya yaitu *data labelling*.

4.2 Sample data labelling

Bagian berikutnya dari `Aplikasi Sampling Penutup Lahan` adalah proses memberikan label pada setiap objek *unlabelled data* hasil *proportionally stratified random sampling* sebelumnya. Pengguna dapat melakukan proses labelisasi secara interaktif dengan mengacu pada visualisasi citra multispektral dan juga *basemaps* citra satelit penginderaan jauh resolusi tinggi. *User interface* yang dibangun pada aplikasi ini terinspirasi dari proses validasi sampel penutup lahan pada LACO-Wiki[1]. *Widget* peta interaktif akan menampilkan sebuah titik sampel berdasarkan `unique id` yang bisa diubah oleh pengguna melalui tombol `Next feature`, `Prev. feature`, dan juga `Jump to feature` dengan memasukkan `unique id` yang diinginkan.

Pada setiap titik sampel tersebut kemudian pengguna dapat memberikan label penutup lahan melalui *dropdown list* yang pilihannya dapat ditentukan sendiri oleh pengguna (dalam prototipe ini digunakan kelas `Forest` dan `No forest`). Pengguna harus meng-klik tombol *trigger* `Save edit` untuk menyimpan hasil labelisasi. Hasil akhir proses labelisasi ini akan tersimpan dalam format `geopandas.GeoDataFrame` yang dapat diekspor ke dalam berbagai format (misalnya `ESRI Shapefiles`) untuk dapat diperbarui lagi. Pengguna dapat mengekspor hasil final ke `GEE asset` agar dapat diinteraksikan dengan `ee object` lainnya dalam proses *supervised classification*.

[1] https://www.laco-wiki.net/

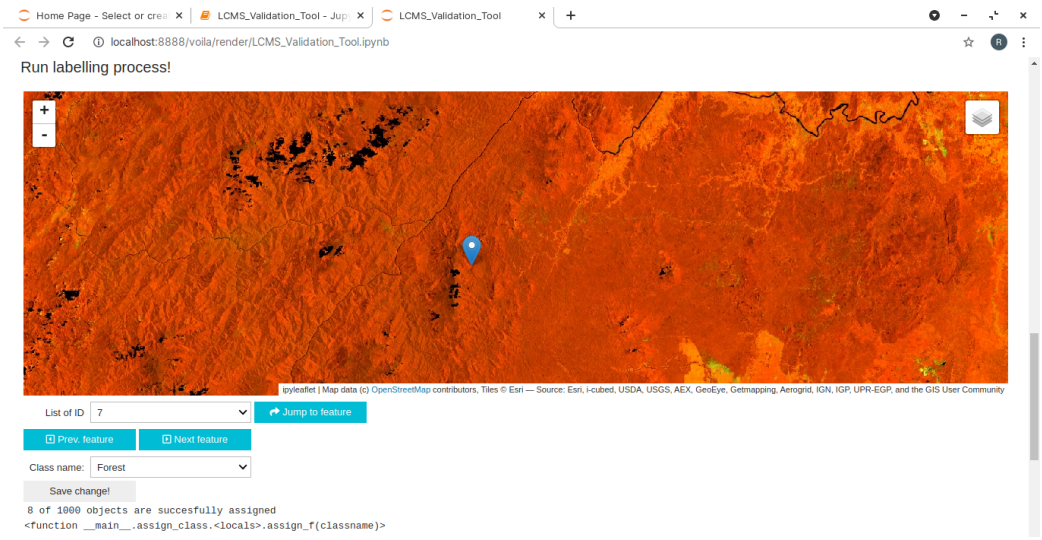

Figure 1: *User interface* berbasis `ipyleaflet` dan `ipywidgets` untuk labelisasi data secara interaktif

References

[1] S. Patra and L. Bruzzone, "A Batch-Mode Active Learning Technique Based on Multiple Uncertainty for SVM Classifier," *IEEE Geosci. Remote Sens. Lett.*, vol. 9, no. 3, pp. 497–501, 2012.

Land Cover Sampling Tool

Date: 2021-08-23 ## Bagian 1: Generate stratified random samples

User Editable Part

Anda dipersilakan mengisikan input/parameter pada bagian ini.

```
# Bagian 1A: nama file object Earth Engine berikut alamat asset-nya
# Contoh: "users/kfaisal/LCMS_Borneo_2016/L8_Borneo_2016_int"
img_asset = "users/kfaisal/LCMS_Borneo_2016/L8_Borneo_2016_int"

# Bagian 1B: kombinasi RGB
# Bands: {B2 = Blue, B3 = Green, B4 = Red, B5 = NIR, B6 = SWIR1, B7 = SWIR2}
# Pilihan kombinasi RGB yang tersedia: "432" (komposit warna asli), "543",
# "562", "563", "564", "567"
comp_rgb = "562"

# Bagian 2: Study area
# Pilihan: Sumatera, Jawa, Bali_NusaTenggara, Borneo, Sulawesi, Maluku, Papua
studyArea = "Borneo" # jangan lupa membubuhkan tanda petik

# Bagian 3: Jumlah K-Means clusters
num_cluster = 15

# Bagian 4: Jumlah titik sampel yang diinginkan
num_out_samples = 1000

# Bagian 5: Parameter ekspor samples ke Google Drive
folderName = "LCMS_samples"
exportName = "Borneo_2016"
```

Packages

```
import ee
```

Authenticate & Initialize GEE account

```
# Authenticate
# Anda dipersilakan melakukan otentifikasi cukup sekali
# ee.Authenticate()

# Initialization
ee.Initialize()
```

Define datasets

```
# Study area
Sumatera = ee.Geometry.Rectangle(94.972663, 6.07694, 109.167015, -6.168225)
Jawa = ee.Geometry.Rectangle(105.099847, -5.042965, 116.270189, -8.78036)
BaliNusaTenggara = ee.Geometry.Rectangle(114.431623, -6.634793, 127.303383, -11.007615)
Borneo = ee.Geometry.Rectangle(108.028633, 7.370019, 119.507898, -5.101769)
Sulawesi = ee.Geometry.Rectangle(117.481334, 5.565816, 127.163961, -7.516092)
Maluku = ee.Geometry.Rectangle(124.144139, 2.645058, 134.908244, -8.347357)
Papua = ee.Geometry.Rectangle(128.909404, 1.081204, 141.029936, -9.140145)

if studyArea == "Sumatera":
    ROI = Sumatera
elif studyArea == "Jawa":
    ROI = Jawa
elif studyArea == "Bali_NusaTenggara":
    ROI = BaliNusaTenggara
elif studyArea == "Borneo":
    ROI = Borneo
elif studyArea == "Sulawesi":
    ROI = "Sulawesi"
elif studyArea == "Maluku":
    ROI = Maluku
elif studyArea == "Papua":
    ROI = Papua

# Landsat image
landsat_image = ee.Image(img_asset)

# SRTM image
SRTM_image = ee.Image("USGS/SRTMGL1_003").clip(ROI)

# Stack image
stacked_image = ee.Image([landsat_image, SRTM_image])

# Landsat visualization (ipyleaflet tile layer format)
RGB432 = {"min": 400, "max": 4000, "gamma": 0.99, "bands": ['B4','B3','B2']}
RGB543 = {"min": 400, "max": 4000, "gamma": 0.99, "bands": ['B5','B4','B3']}
RGB562 = {"min": 400, "max": 4000, "gamma": 0.99, "bands": ['B5','B6','B2']}
RGB564 = {"min": 400, "max": 4000, "gamma": 0.99, "bands": ['B5','B6','B4']}
RGB567 = {"min": 400, "max": 4000, "gamma": 0.99, "bands": ['B5','B6','B7']}

if comp_rgb == "432":
    vizParam = RGB432
elif comp_rgb == "543":
```

```
    vizParam = RGB543
elif comp_rgb == "562":
    vizParam = RGB562
elif comp_rgb == "564":
    vizParam = RGB564
elif comp_rgb == "567":
    vizParam = RGB567
```

K-Means Clustering

```
# Generate 10000 random samples and extract image values

## Sampling
random_samples = ee.FeatureCollection.randomPoints(
    region = ROI,
    points = 10000
)
## Image sampling
samp_image = stacked_image.sampleRegions(
    collection = random_samples,
    scale = 30
)
## Run K-Means clustering
kmeans = ee.Clusterer.wekaKMeans(num_cluster).train(samp_image)
cluster_k = stacked_image.cluster(kmeans).rename("cluster")
```

Generate proportionally stratified random samples

```
# Proportional random sampling
areaImage = ee.Image.pixelArea().addBands(cluster_k)

# Calculate per-cluster area
areas = areaImage.reduceRegion(
    reducer = ee.Reducer.sum().group(
        groupField = 1,
        groupName = "cluster"),
    geometry = ROI,
    scale = 100,
    maxPixels = 1e13)
classAreas = ee.List(areas.get("groups"))

# List of cluster
def classnum(item):
    areaDict = ee.Dictionary(item)
```

```
    classNumber = ee.Number(areaDict.get("cluster"))
    return classNumber
classNumLists = classAreas.map(classnum)
numValues = ee.Array(classNumLists).round().int().toList()

# List of per-class area
def classarea(item):
    areaDict = ee.Dictionary(item)
    area = ee.Number(areaDict.get("sum")).divide(1e4).round()
    return area
classAreaLists = classAreas.map(classarea)
totalArea = ee.Number(classAreaLists.reduce(ee.Reducer.sum()))

# Generate number of points proportionally
numPoints = ee.Array(classAreaLists).divide(totalArea)
          .multiply(num_out_samples).round().int().toList()

# Generate proportional sample points
stratified = cluster_k.stratifiedSample(
  numPoints = num_out_samples,
  classBand = "cluster",
  classValues = numValues,
  classPoints = numPoints,
  scale = 100,
  region = ROI,
  geometries = True)
```

Export output samples to Google Drive

```
# Export samples to Drive
from datetime import date
today = date.today()
todaydate = today.strftime("%b-%d-%Y")

exportTask = ee.batch.Export.table.toDrive(
    collection = stratified,
    description = todaydate + '_' + exportName + '_stratifiedsamples_'
                + str(num_cluster) + '_' + str(num_out_samples),
    folder = folderName,
    fileFormat = 'SHP')
exportTask.start()
```

Akhir dari bagian pembuatan titik sampel

Cek hasil ekspor titik sampel pada `Google Drive` Anda, kemudian unduh file tersebut untuk Anda gunakan pada proses berikutnya: proses pemberian label pada titik sampel.## Bagian 2: Labelling

Note: Jika Anda memulai sesi baru, silakan menjalankan code cell **User Editable Part** hingga **Define datasets** pada Bagian 1 di atas.

User Editable Part

Anda dipersilakan mengisikan input/parameter pada bagian ini.

```
# Bagian 1: nama dan folder path lokasi penyimpanan file titik sampel yang dibuat
# berdasarkan hasil clustering dengan K-Means pada tahap sebelumnya
# (ataupun titik yang telah selesai diisi atributnya sebagian)
# Contoh: "./samples/Aug-06-2021_Borneo_2016_stratifiedsamples_15_1000_gcs.shp"
path_to_shp = "./samples/Aug-06-2021_Borneo_2016_stratifiedsamples_15_1000_gcs.shp"

# Bagian 2: daftar label tutupan lahan
class_opt = ["Forest", "No-forest"]
```

Packages

```
import geopandas
from ipywidgets import fixed, interact, interact_manual, widgets
from IPython.display import display, clear_output
import ipyleaflet
import geemap
import numpy
```

Define datasets

```
# import shapefiles as geodataframe
gdf = geopandas.read_file(path_to_shp)

# create some columns for first time only
if 'ID' in gdf:
    pass
else:
    gdf['ID'] = numpy.arange(len(gdf))
    gdf['lat'] = gdf['geometry'].y
    gdf['lon'] = gdf['geometry'].x
    gdf['Class'] = None
```

```
# filter out geodataframe with NA value on "Class" column
filt_gdf = gdf[gdf['Class'].isna()]

# create list of ID
filt_id_list = filt_gdf['ID'].tolist()

# function to read tile layer on ipyleaflet
def GetTileLayerUrl(ee_image_object):
  map_id = ee.Image(ee_image_object).getMapId()
  tile_fetcher = map_id['tile_fetcher']
  return tile_fetcher.url_format

# Check prior dataframe
filt_gdf.head()
```

List of widgets

```
nextButton = widgets.Button(
    description = 'Next feature',
    button_style = 'info',
    tooltip = 'Next feature',
    icon = 'toggle-right'
)

prevButton = widgets.Button(
    description = 'Prev. feature',
    button_style = 'info',
    tooltip = 'Prev. feature',
    icon = 'toggle-left'
)
jumpButton = widgets.Button(
    description = 'Jump to feature',
    button_style = 'info',
    tooltip = 'Jump to',
    icon = 'mail-forward'
)
class_assign = widgets.Dropdown(
    options = class_opt,
    description = 'Class name:',
    value = None
)
id_selector = widgets.Dropdown(
    options = filt_id_list,
    description = "List of ID",
```

```
    value = filt_id_list[0]
)
out = widgets.Output()
toolbar_wid = widgets.VBox()
toolbar_wid.children = (
    widgets.HBox([id_selector,jumpButton]),
    out,
    widgets.HBox([prevButton,nextButton])
)
```

Define ipyleaflet objects

```
# basemaps
basemap_Esri = ipyleaflet.basemap_to_tiles(
    basemap = ipyleaflet.basemaps.Esri.WorldImagery)

# map components
m = ipyleaflet.Map(scroll_wheel_zoom = True)
layer_control = ipyleaflet.LayersControl(position = 'topright')
m.add_control(layer_control)
m.add_layer(basemap_Esri)
m.add_layer(ipyleaflet.TileLayer(url = GetTileLayerUrl(
    landsat_image.visualize(min = vizParam['min'],
                            max = vizParam['max'],
                            gamma = vizParam['gamma'],
                            bands = vizParam['bands']))))

idx = 2 # index of list

# Ipyleaflet object
init_obj = filt_gdf[filt_gdf['ID'] == filt_id_list[idx]]
geodata = ipyleaflet.GeoData(geo_dataframe = init_obj,
    name = 'Layer ' + str(filt_id_list[idx]))

lat = init_obj.iloc[0]['lat']
lon = init_obj.iloc[0]['lon']

m.center = (lat, lon)
m.zoom = 10
m.add_layer(geodata)

# # functions

def assign_class():
```

```
def get_idx(id):
    return filt_id_list.index(id)

def jump_button_clicked(b):
    with out:
        clear_output()
        global idx
        global geodata
        idx = get_idx(id_selector.value)
        print('jump to ID:', id_selector.value)
        m.remove_layer(geodata)
        geo_dataframe = filt_gdf.loc[filt_gdf['ID'] == filt_id_list[idx]]
        geodata = ipyleaflet.GeoData(
    geo_dataframe = geo_dataframe,
    name = 'Layer ' + str(filt_id_list[idx]))
        m.add_layer(geodata)
        sel = filt_gdf.loc[filt_gdf['ID'] == filt_id_list[idx]]
        lat = sel.iloc[0]['lat']
        lon = sel.iloc[0]['lon']
        m.center = (lat, lon)

def next_button_clicked(b, incr = 1):
    with out:
        clear_output()
        global idx
        global geodata
        print('Prev. ID:', filt_id_list[idx])
        idx = idx + incr
        # print('Current ID selected:', filt_id_list[idx])
        m.remove_layer(geodata)
        geo_dataframe = filt_gdf.loc[filt_gdf['ID'] == filt_id_list[idx]]
        geodata = ipyleaflet.GeoData(
    geo_dataframe = geo_dataframe,
    name = 'Layer ' + str(filt_id_list[idx]))
        print('Current ID Class:', geo_dataframe['Class'])
        m.add_layer(geodata)
        sel = filt_gdf.loc[filt_gdf['ID'] == filt_id_list[idx]]
        lat = sel.iloc[0]['lat']
        lon = sel.iloc[0]['lon']
        m.center = (lat, lon)

def prev_button_clicked(b):
    return next_button_clicked(b, -1)

def assign_f(classname):
    global idx
```

```
        gdf.loc[gdf['ID'] == filt_id_list[idx], 'Class'] = classname
        # print(gdf.loc[gdf['ID'] == filt_id_list[idx]])
        print('{} of {} objects are succesfully assigned' \
    .format(1000 - len(gdf[gdf['Class'].isna()]), len(gdf)-1))

    display(m)

    display(toolbar_wid)

    jumpButton.on_click(jump_button_clicked)
    nextButton.on_click(next_button_clicked)
    prevButton.on_click(prev_button_clicked)

    im = interact_manual(assign_f, classname = class_assign,
             gdf = widgets.fixed(gdf))
    im.widget.children[0].description = 'Class name:'
    im.widget.children[1].description = 'Save change!'
    display(im)
```

Run labelling process!

```
assign_class()
```

Check dataframe

```
# Check after editing
gdf.head(10)
```

Menyimpan hasil sample labelling

```
# export current results

# Encoding to 0 (No-forest) and 1 (Forest)
gdf['Class_code'] = numpy.where(gdf['Class'].isnull(), 99, \
    numpy.where(gdf['Class'] == 'Forest', 1, 0))

gdf.to_file(path_to_shp)
```

Ekspor hasil final ke GEE Asset

```
# Export to Asset, later used as machine-learning classification input in GEE
from datetime import date
today = date.today()
```

```
todaydate = today.strftime("%b-%d-%Y")

# Convert geodataframe to ee object
ee_export = geemap.geopandas_to_ee(gdf)

exportTask = ee.batch.Export.table.toAsset(
    collection = ee_export,
    description = str(todaydate) + '_' + studyArea + '_forestCoverSamples',
    assetId = 'users/gemasaktiadzan/' + description)
exportTask.start()
```