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

```python
# 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():
```

```python
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()
```

```python
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()
```