Predicting Poverty In Brazil From Satellite Imagery and Machine Learning

2019 Fall Geospatial Software Design Term Project Report

•••

Zixi Liu

Dec 17, 2019

Background: Kiva Predicting Borrowers' Poverty Rate in Brazil

Urbanization is a rapid transformation of human social roots on a global scale. This unprecedented movement of people is forecast to continue and intensify in the next few decades, but methods to trace urbanization on the globe remain limited. Therefore, evaluating urbanization process using remote sensing images is a cost-efficient approach.

We use machine learning for pixel-based image classification of built-up areas in Brazil at a large geographic scale using Landsat 8 and DMSP-OLS data. We combine nighttime-lights data and Landsat 8 to overcome the lack of extensive ground reference data. Finally, we will demonstrate the effectiveness of our methodology, which is implemented with Google Earth Engine API in Python, through the development of 30 m resolution maps that characterize built-up land cover in Brazil.

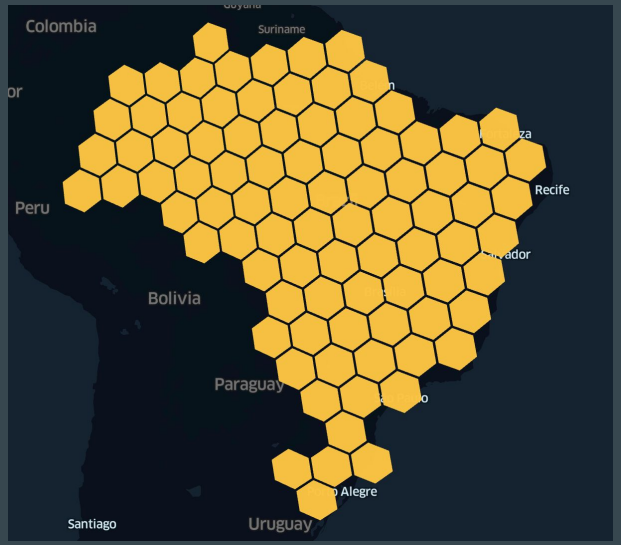
UNDP Global Multidimensional Poverty Index In Brazil Provided by Kiva Crowdfunding.

The higher the MPI, the higher percentage of the population that is multidimensionally poor.

Background: Breaking Down Study Areas with H3 Hexagons

Project Proposal:

- ❏ Visualize nighttime lights images (DMSP-OLS data) and Landsat 8 data using Python Earth Engine API.
- ❏ Use H3, a hexagonal tessellation mapping approach to handle large variation across regions.
- ❏ Compute NDWI, NIR, NDVI, and EVI and add bands.
- ❏ Partition Brazil with H3 equal-area hexagonal grids and calculate DMSP-OLS thresholds in each hexagon.
- ❏ Adopt Supervised Pixel-based Image Classification.



H3: Uber's Hexagonal Hierarchical Spatial Index is a geospatial indexing system using a hexagonal grid that can be (approximately) subdivided into finer and finer hexagonal grids.

Framework: Determine Thresholds, Train Classifier, and Predict

Optimizing Parameters

- ➔ Divide Brazil into 88 hexagons.
- ➔ Load DMSP-OLS Data and compute thresholds for each cell.
- ➔ Identify “high-lit” pixels and “low-lit” pixels in each cell.
- ➔ Load Landsat 8 images and mask out clouds.
- ➔ Calculate NDVI and NDWI thresholds for each cell.
- ➔ Mask out vegetation and bodies of water in each cell.

Training Data

- ➔ Select 8 hexagons as training dataset.
- ➔ Randomly sample 200 points from training Polygons.
- ➔ Hand-label built-up areas and non built up areas for training.
- ➔ Use Random Forest Classifier to train data.
- ➔ Parameter tuning to optimize the model.
- ➔ Identify optimal inputs to classifier.

Classifying Poverty

- ➔ Select 15 hexagons as evaluation dataset.
- ➔ Adopt trained classifier to pixels for evaluation.
- ➔ Identify classified “high-lit” pixels and “low-lit” pixels.
- ➔ Mask out vegetation and bodies of water.
- ➔ Gain built-up areas and non built-up areas in each cell.
- ➔ Post-process map for display.

```
!pip install --user git+https://github.com/matplotlib/basemap.git
!pip install geopandas
!apt-get install libgeos-3.5.0
!apt-get install libgeos-dev
!pip install https://github.com/matplotlib/basemap/archive/master.zip
import pandas as pd
from lxml import etree
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import mpl_toolkits
from matplotlib.colors import Normalize
from matplotlib.collections import PatchCollection
from mpl_toolkits.basemap import Basemap
from shapely.geometry import Point, Polygon, MultiPoint, MultiPolygon
from shapely.prepared import prep
from descartes import PolygonPatch
import fiona
from itertools import chain
from folium import Map, Marker, GeoJson
from folium.plugins import MarkerCluster
import geopandas as gpd

!pip install geojson
!pip install area

!pip install keplergl

!pip install earthengine-api
import ee
ee.Authenticate()

!pip install h3
from h3 import h3
```

Import Libraries, Folium Map, Google Earth Engine API, H3 Hexagonal Spatial Index System

Optimizing Parameters: Initialize Lib, Brazil Boundary and OLS

```
# Initialize the library.
ee.Initialize()
Brazil = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017').filter(ee.Filter.eq('country_na', 'Brazil'))
```

We first take a look at nighttime lights images (DMSP-OLS data). In general, the more lit up an area is at night, the richer and more developed it is. But night light image estimates are pretty rough and later we will use them for classification of built-up areas with daytime Landsat 30 m spatial resolution imagery.

```
import folium

nightlights = ee.ImageCollection("NOAA/DMSP-OLS/NIGHTTIME_LIGHTS").select('stable_lights').filterDate('2010-01-01', '2013-
annual_nl = nightlights.median()
nl  = annual_nl.clipToCollection(Brazil)

# Define a method for displaying Earth Engine image tiles to folium map.
def add_ee_layer(self, ee_image_object, vis_params, name):
  map_id_dict = ee.Image(ee_image_object).getMapId(vis_params)
  folium.raster_layers.TileLayer(
    tiles = map_id_dict['tile_fetcher'].url_format,
    attr = "Map Data © Google Earth Engine",
    name = name,
    overlay = True,
    control = True
  ).add_to(self)

# Add EE drawing method to folium.
folium.Map.add_ee_layer = add_ee_layer

# Set visualization parameters.
vis_params = {
  'min': 0,
  'max': 12,
  'palette': ['black','white']}

# Create a folium map object.
my_map = folium.Map(location=[-17, -50],zoom_start=4, height=500)

# Add the elevation model to the map object.
my_map.add_ee_layer(nl, vis_params, 'Night Light')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

- ➔ Initialize Brazil Feature Collection for image clipping.
- ➔ Load DMSP-OLS Data from Google Earth Engine for visualization.
- ➔ Clip nighttime lights images by Brazil boundary and use Folium Map to display image (DMSP-OLS data).

Optimizing Parameters: Load Landsat 8 Data, Mask Out Clouds

We apply a standard Top-of-Atmosphere (TOA) calibration on USGS Landsat 8 data and mask clouds in daytime and select different bands.

```
daytime = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT_TOA").filterDate('2013-01-01', '2014-01-01').filterBounds(Brazil)
day_list = daytime.toList(daytime.size())
col = ee.ImageCollection(day_list.splice(0,1))
```

```
col_band = col.map(lambda img: img.updateMask(ee.Algorithms.Landsat.simpleCloudScore(img).select(['cloud']).lte(10)))
```

```
dt = ee.Image(col_band.median())
trueColor = dt.select(['B6','B5', 'B4', 'B3', 'B2'])
```

Then we compute the NDWI, NIR, NDVI, and EVI.

```
ndwi = dt.normalizedDifference(['B3', 'B5']);
nir = dt.select('B5');
red = dt.select('B4');
ndvi = nir.subtract(red).divide(nir.add(red)).rename('NDVI');
```

```
dataset = ee.ImageCollection('LANDSAT/LC08/C01/T1_8DAY_EVI').filterDate('2013-01-01', '2014-01-01')
colorized = dataset.select('EVI').median()
evi = colorized.clipToCollection(Brazil)
colorizedVis = {
  min: 0.0,
  max: 1.0,
  'palette': [
    'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901',
    '66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01',
    '012E01', '011D01', '011301'
  ],}
```

```
# Create a folium map object.
my_map = folium.Map(location=[-18, -54],zoom_start=4, height=500)

# Add the elevation model to the map object.
my_map.add_ee_layer(evi, colorizedVis, 'EVI')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

- ➔ Load Landsat 8 Data from GEE API and clip to Brazil.
- ➔ Mask out clouds by cloud score.
- ➔ Compute NDWI, NIR, and NDVI.
- ➔ Load EVI data and display.

Landsat 8 Dataset

EVI Dataset

We divide our study area into a uniform hexagonal grid.

```
import json
import branca.colormap as cm
from branca.colormap import linear
from geojson.feature import *
from folium import Map, Marker, GeoJson
from folium.plugins import MarkerCluster
```

```
map_subzone = Map(location= [-17, -50], zoom_start=4, tiles="cartodbpositron",
                  attr= '© <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors © <a href="http:,
```

```
def reverse_lat_lon(hex_coords):
    geom_hex = []
    for lat_lon in hex_coords:
        geom_hex.append([lat_lon[1],lat_lon[0]])
    return geom_hex
```

```
from urllib.request import urlopen
from pandas.io.json import json_normalize
import json
req = urlopen("https://raw.githubusercontent.com/johan/world.geo.json/master/countries.geo.json").read()
json1 = json.loads(req)
df_BRZ = pd.DataFrame(json_normalize(json1['features']))
df_BRZ=df_BRZ[df_BRZ['properties.name']=="Brazil"]
```

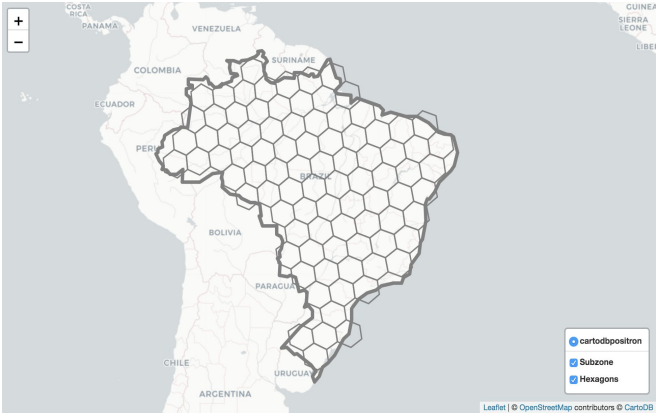
```
df_BRZ["geometry"] = df_BRZ.apply(lambda row:{"type" : row["geometry.type"], "coordinates": row["geometry.coordinates"]
df_BRZ
```

	type	id	properties.name	geometry.type	geometry.coordinates	geometry
23	Feature	BRA	Brazil	Polygon	[[[-57.625133, -30.216295], [-56.2909, -28.852...	{'type': 'Polygon', 'coordinates': [[[-57.6251...

```
def hexagons_dataframe_to_geojson(df_hex, file_output = None):

    '''Produce the GeoJSON for a dataframe that has a geometry column in geojson format already, along with the columns h

    list_features = []

    for i,row in df_hex.iterrows():
        feature = Feature(geometry = row["geometry"] , id=row["hex_id"], properties = {"value" : row["value"]})
        list_features.append(feature)

    feat_collection = FeatureCollection(list_features)

    geojson_result = json.dumps(feat_collection)

    #optionally write to file
    if file_output is not None:
        with open(file_output,"w") as f:
            json.dump(feat_collection,f)

    return geojson_result
```

Optimizing Parameters: Dividing Hexagonal Grids

- ➔ Load the boundary of Brazil and define functions to convert hexagons into data frames.
- ➔ Visualize hexagonal grids with folium map.
- ➔ Upload the grids as feature collection to GEE.
- ➔ Use hexagonal grids to clip the image data.



H3 Hexagonal Grid System, Brazil

```
#take the first and only row of the dataframe
row_sel = df_BRZ.iloc[0]

feature_sel = Feature(geometry = row_sel["geometry"], id=row_sel["id"])
feat_collection_sel = FeatureCollection([feature_sel])
geojson_subzone = json.dumps(feat_collection_sel)

GeoJson(
        geojson_subzone,
        style_function=lambda feature: {
            'fillColor': None,
            'color': 'grey',
            'weight': 5,
            'fillOpacity': 0
        },
        name = "Subzone"
    ).add_to(map_subzone)

map_subzone
```

```
set_hexagons = h3.polyfill(geo_json = row_sel["geometry"], res =2, geo_json_conformant = True)
list_hexagons = []
for ele in set_hexagons:
  list_hexagons.append(ele)
print("The subzone was filled with",len(list_hexagons), "hexagons at resolution 3.")
```

```
The subzone was filled with 88 hexagons at resolution 3.
```

```
df_fill_hex = pd.DataFrame({"hex_id": list_hexagons})
df_fill_hex["value"] = 0
df_fill_hex['geometry'] = df_fill_hex.hex_id.apply(lambda x: {"type" : "Polygon","coordinates": [reverse_lat_lon(h3.h3_to_

geojson_hx = hexagons_dataframe_to_geojson(df_fill_hex)

GeoJson(
        geojson_hx,
        style_function=lambda feature: {
            'fillColor': None,
            'color': 'grey',
            'weight': 2,
            'fillOpacity': 0
        },
        name = "Hexagons"
    ).add_to(map_subzone)

folium.map.LayerControl('bottomright', collapsed=False).add_to(map_subzone)
map_subzone
```

Optimizing Parameters: Dividing Hexagonal Grids

- ➔ Determine to use hexagonal grids at resolution 3.
- ➔ Fill Brazil boundary with hexagonal cells.
- ➔ Display hexagonal grids with folium map.
- ➔ Upload the grids as feature collection to GEE.

Optimizing Parameters: Identify High-lit and low-lit Pixels

```
polys = []
for i in range(0,88):
  {
       polys.append(ee.Feature(ee.Geometry.Polygon(df_fill_hex['geometry'][i]['coordinates']), { 'name': df_fill_hex['hex_
  }
```

```
grid = ee.FeatureCollection(polys)
```

Here we partition Brazil into an equal-area hexagonal grid (or a hexagonal tessellation).

```
nl_clip = nl.clip(grid)
```

```
def addPct2(feature):
      return feature.set('name',nl_clip.reduceRegion(ee.Reducer.percentile([50, 75, 90, 95, 99]), feature.geometry(), sc
```

```
gridAdded = grid.map(addPct2)
```

Identify "high lit" pixels. A pixel is characterized as highly-lit and as lowly-lit if its value exceeds and is below a given threshold, respectively.

```
def set_high(feature):
      threshold1 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p90'))
      threshold2 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p95'))
      threshold3 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p99'))
      if threshold1 != 0:
        threshold = threshold1
      elif (threshold1 == 0)&(threshold2 != 0):
        threshold = threshold2
      elif (threshold2 == 0)&(threshold3 != 0):
        threshold = threshold3
      else:
        threshold = threshold1
      hotspots = nl_clip.clip(feature.geometry()).gt(threshold)
      patchid = hotspots.connectedComponents(ee.Kernel.plus(1), 256)
      hotspots = hotspots.updateMask(hotspots)
      patches = nl_clip.clip(feature.geometry()).addBands(patchid)
      return patches
```

```
mapPatches = ee.ImageCollection(gridAdded.map(set_high))
```

```
mapPatches =  mapPatches.select( ['stable_lights', 'labels', 'stable_lights_1'], ['stable_lights', 'labels','high_lit'])
mapPatches.first().bandNames().getInfo()
```

```
['stable_lights', 'labels', 'high_lit']
```

- ➔ Upload Hexagonal Grids as Feature Collection to GEE.
- ➔ Clip nightimes light image with our grid.
- ➔ Determine separate thresholds for “high-lit” pixels and “low-lit” pixels in each cell.
- ➔ Label each high-lit pixel in each hexagon.

Optimizing Parameter: Identify High-lit and low-lit Pixels

```
mapPatches1 = mapPatches.median()
```

```
# Set visualization parameters.
vis_params = {
  'min': 0,
  'max': 1,
  'palette': ['black','white']}

# Create a folium map object.
my_map = folium.Map(location=[-17, -56],zoom_start=4, height=500)

# Add the elevation model to the map object.
my_map.add_ee_layer(mapPatches1.select('high_lit'), vis_params, 'Night Light')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

```
def addPct3(feature):
      return feature.set('name',nl_clip.reduceRegion(ee.Reducer.percentile([10, 25, 50,75]), feature.geometry(), scale=3(
```

```
gridAdded3 = grid.map(addPct3)
```

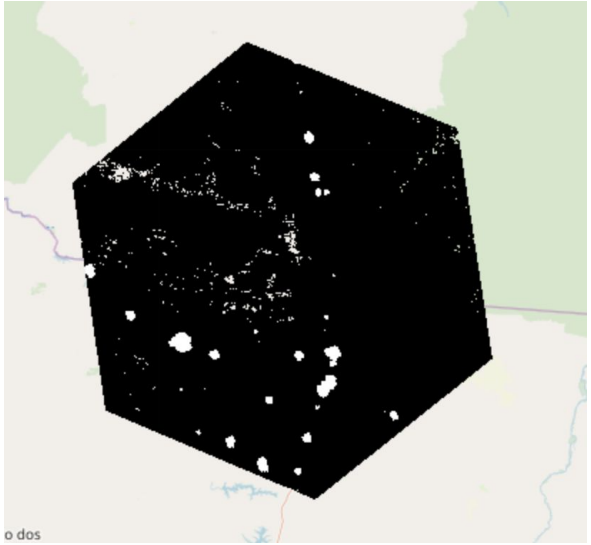
```
def set_low(feature):
     threshold1 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p25'))
     threshold2 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p50'))
     threshold3 = ee.Number(ee.Dictionary(feature.get('name')).get('stable_lights_p75'))
     if threshold1 != 0:
       threshold = threshold1
     elif (threshold1 == 0)&(threshold2 != 0):
       threshold = threshold2
     elif (threshold2 == 0)&(threshold3 != 0):
       threshold = threshold3
     else:
       threshold = threshold1
     coldspots = nl_clip.clip(feature.geometry()).gt(threshold)
     patchid2 = coldspots.connectedComponents(ee.Kernel.plus(1), 256)
     patches2 = nl_clip.clip(feature.geometry()).addBands(patchid2)
     return patches2
```

```
mapPatches2 = ee.ImageCollection(gridAdded3.map(set_low))
```

```
mapPatches2 =  mapPatches2.select( ['stable_lights', 'labels', 'stable_lights_1'], ['stable_lights', 'labels','low_lit'])
mapPatches2.first().bandNames().getInfo()
```

```
['stable_lights', 'labels', 'low_lit']
```

- ➔ Clip OLS image again with our grids.
- ➔ Set thresholds for each cell and label low-lit pixels.
- ➔ A closer look at a specific cell clipped and labeled.



Optimizing Parameter: Masking Out Vegetation and Water

```
ndvi_clip = ndvi.clip(grid)
def addPct_ndvi(feature):
      return feature.set('name',ndvi_clip.reduceRegion(ee.Reducer.percentile([50,75,90,95,99]), feature.geometry(), scal
```

```
gridAdded_ndvi = grid.map(addPct_ndvi)
```

Due to the spatial resolution of DMSP-OLS and the blooming effect, areas identified as highly lit may potentially include non-built-up land cover. We want to remove these pixels based on Landsat's per-pixel NDVI and NDWI values. Hence we exclude pixels when its NDVI value exceeds its threshold and remove bodies of water if its NDWI value is positive.

```
def set_ndvi(feature):
      threshold1 = ee.Number(ee.Dictionary(feature.get('name')).get('NDVI_p75'))
      threshold2 = ee.Number(ee.Dictionary(feature.get('name')).get('NDVI_p90'))
      threshold3 = ee.Number(ee.Dictionary(feature.get('name')).get('NDVI_p95'))
      if threshold1.gt(0.7):
        threshold = threshold1
      elif (threshold1.lt(0.7))&(threshold2.gt(0.7)):
        threshold = threshold2
      elif (threshold2.lt(0.7))&(threshold3.gt(0.7)):
        threshold = threshold3
      else:
        threshold = 0.7
      vegspots = ndvi_clip.clip(feature.geometry()).lt(threshold)
      filtered = mapPatches.filterBounds(feature.geometry()).max()
      patches3 = filtered.updateMask(vegspots)
      return patches3.set('geometry',feature.geometry())
```

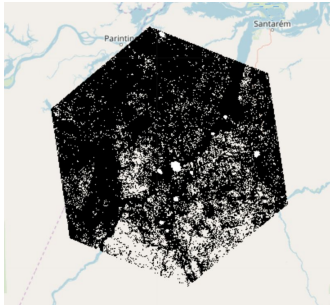
```
mask_ndvi = ee.ImageCollection(gridAdded_ndvi.map(set_ndvi))
```

```
ndwi_clip = ndwi.clip(grid)
```

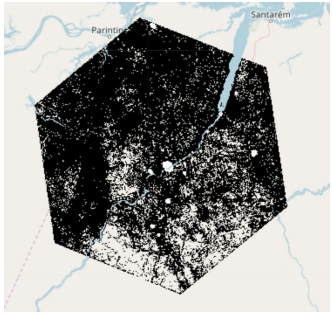
```
def set_ndwi(feature):
      waterspots = ndwi_clip.clip(feature.get('geometry')).lt(0)
      filtered = mask_ndvi.filterBounds(feature.get('geometry')).max()
      patches4 = filtered.updateMask(waterspots)
      return patches4.set('geometry',feature.get('geometry'))
```

```
mask_ndwi = ee.ImageCollection(mask_ndvi.map(set_ndwi))
```

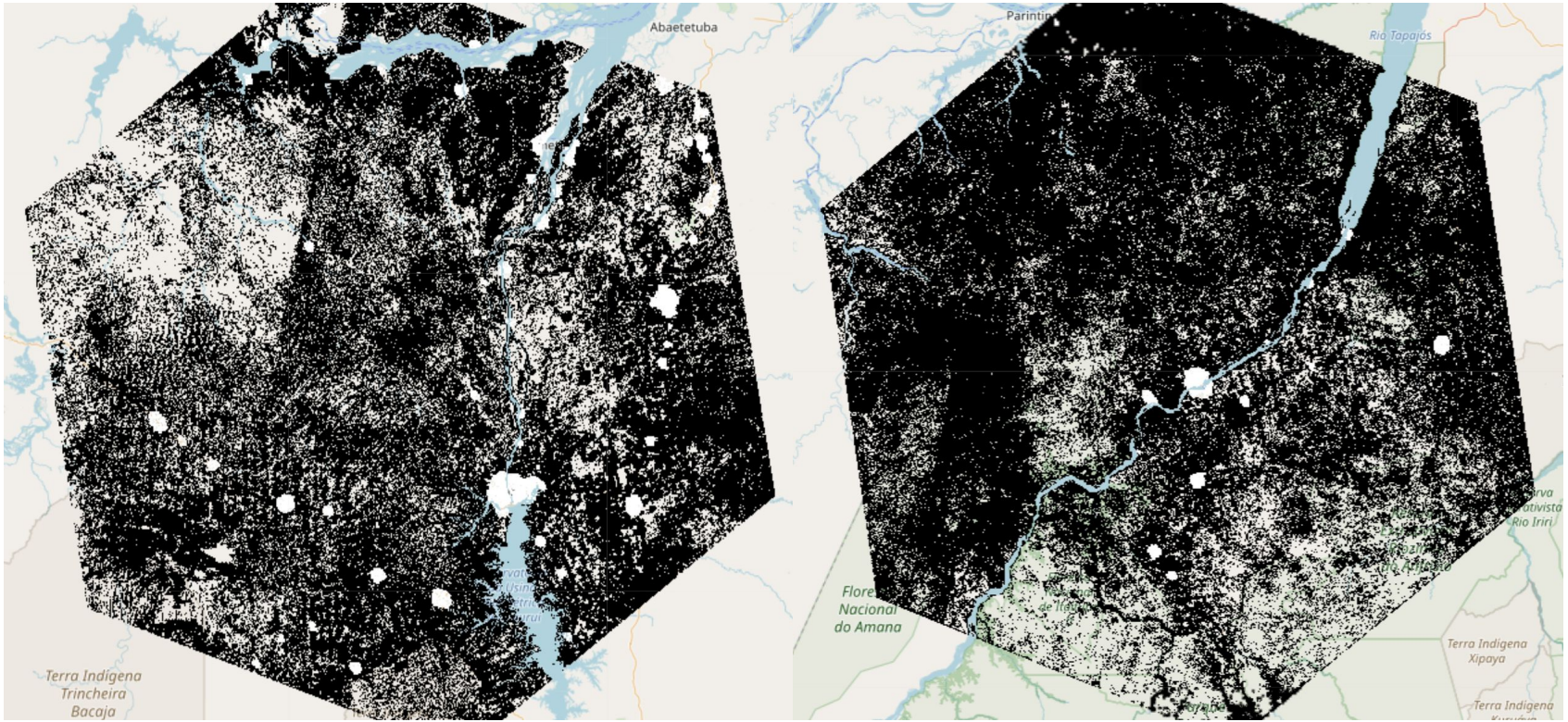
➔ A cell masked out vegetation using NDVI.



➔ A cell further masked out water using NDWI.



Optimizing Parameter: Masking Out Vegetation and Water



Training Data: Data Split for Training Hexagonal Cells

Use some pre-made geometries to sample the stack in strategic locations. Specifically, these are hand-made polygons in which to take the 256x256 samples. Display the sampling polygons on a map, red for training polygons, blue for evaluation.

```
trainingPolys = grid.filter(ee.Filter.gt("system:index", '80'))
evalPolys = grid.filter(ee.Filter.lt("system:index", '15'))

polyImage = ee.Image(0).byte().paint(trainingPolys, 1).paint(evalPolys, 2)
polyImage = polyImage.updateMask(polyImage)

vis_params = {
  'min': 1,
  'max': 2,
  'palette': ['red','blue']}

my_map = folium.Map(location=[-20,-50], zoom_start=4)

# Add the elevation model to the map object.
my_map.add_ee_layer(polyImage, vis_params, 'Training_Eval')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

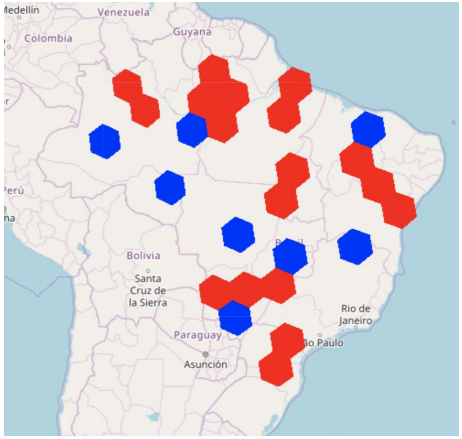
```
WARNING:googleapiclient.http:Sleeping 1.80 seconds before retry 1 of 5 for request: POST https://earthengine.googl
```

Sampling

```
# Convert the feature collections to lists for iteration.
trainingPolysList = trainingPolys.toList(trainingPolys.size())
evalPolysList = trainingPolys.toList(trainingPolys.size())

# These numbers determined experimentally.
n = 200 # Number of shards in each polygon.
N = 2000 # Total sample size in each polygon.
```

We randomly sample 200 points from training Polygons and hand-label pixels as built-up areas and non-built-up areas.

- ➔ Split the data in training and evaluation sets.
- ➔ Color training cells by blue and evaluation cells by red. (reversed color set in code)
- ➔ Randomly sample 200 points from training cells.



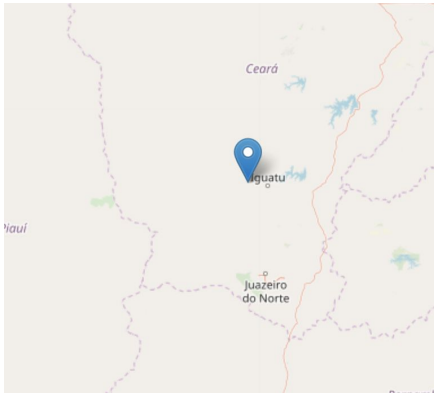
Training Data: Hand-label Build-up and Non Built-up Areas

```
pts = ee.FeatureCollection.randomPoints(trainingPolys,200,0,1)
```

```
mp = mapPatches.median()
mp_train = mp.clip(trainingPolys)
```

```
NBU = ee.Geometry.MultiPoint([[-53.8666937326277,-30.90309259768932], [-60.35114790261921, -6.156233883089041],
                              [-59.75058578049829, -10.724091981668563],[-50.23516311711308, -29.200893429586614],
                              [-48.86043606247113, -22.935326532345695],[-61.58222100350182, 1.5836508730276935],
                              [-60.90715625375103, 3.327988175379846],[-52.827982141665956, -12.215724823065658],
                              [-48.816947610296715, -27.343398004144095],[-50.05305224504645, -22.51087457134498],
                              [-55.64924045555567, -9.246243494099462],[-52.03707318653847, -12.35202080122596],
                              [-60.376114265116804, -7.912657679492538],[-53.86444878377972, -10.476157645884259],
                              [-52.91847472295991, -13.011255370745706],[-53.009432592296235, -13.321501361049638],
                              [-60.273357103796556, 3.29801633532492],[-52.827982141665956, -12.215724823065658],
                              [-48.816947610296715, -27.343398004144095],[-50.05305224504645, -22.51087457134498],
                              [-55.64924045555567, -9.246243494099462],[-52.729876191177986,-16.152932779252524],
                              [-53.589876191177986,-17.152932779252524],[-52.589876191177986,-17.052932779252524],
                              [-51.589876191177986,-17.052932779252524],[-52.789876191177986,-17.929327792525241],
                              [-52.789876191177986,-15.92932779252524],[-54.29876191177986,-14.92932779252524],
                              [-54.29876191177986, -15.82932779252524],[-53.74776191177986,-19.45932779252524],
                              [-60.49776191177986, 2.35932779252524],[-61.79776191177986, 2.75932779252524],
                              [-42.79776191177986, -4.05932779252524],[-41.79776191177986,-4.05932779252524],
                              [-40.79776191177986, -6.05932779252524],[-39.79776191177986,-6.05932779252524],
                              [-39.79776191177986, -7.05932779252524],[-39.79776191177986, -8.05932779252524],
                              [-37.79776191177986, -9.05932779252524],[-37.79776191177986, -8.05932779252524],
                              [-37.79776191177986, -7.05932779252524],[-39.79776191177986,-7.05932779252524],
                              [-39.99776191177986,-7.05932779252524],[-40.69776191177986,-7.13932779252524],
                              [-38.69776191177986,-7.13932779252524],[-39.49776191177986,-6.33932779252524]])
BU = ee.Geometry.MultiPoint([[-50.05305224504645,-23.51087457134498],[-51.789876191177986,-17.92932779252524],
                             [-52.29876191177986,-15.92932779252524],[-54.39876191177986,-15.82932779252524],
                             [-54.29876191177986,-15.51932779252524],[-54.29776191177986,-15.51932779252524],
                             [-55.19776191177986, -15.55932779252524],[-55.19776191177986, -15.54932779252524],
                             [-55.74776191177986,-15.45932779252524],[-54.74776191177986,-20.45932779252524],
                             [-54.54776191177986, -20.45932779252524],[-55.74776191177986,-20.45932779252524],
                             [-53.74776191177986,-20.45932779252524],[-54.04776191177986, -19.55932779252524],
                             [-54.58776191177986,-19.35932779252524],[-58.49776191177986,-10.35932779252524],
                             [-60.79776191177986,2.85932779252524],[-60.79776191177986,2.75932779252524],
                             [-40.79776191177986,-4.75932779252524],[-42.79776191177986,-5.35932779252524],
                             [-42.79776191177986, -5.05932779252524],[-39.69776191177986,-7.13932779252524],
                             [-39.39776191177986,-7.13932779252524],[-39.39776191177986,-7.33932779252524]])
```

- ➔ Clip images with training hexagons.
- ➔ Use a marker to decide whether pixel is high-lit.
- ➔ Hand-label built-up and non built-up areas for classification.



Training Data: Apply Random Forest Classifier to Train Data

```
polygons = ee.FeatureCollection([
  ee.Feature(BU, {'class': 1}),
  ee.Feature(NBU, {'class': 0})
])
```

```
training = featureStack.clip(trainingPolys)
```

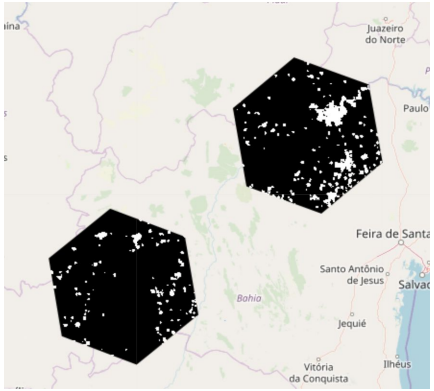
```
# Select the bands for training
bands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10', 'B11', 'NDVI', 'NDWI', 'EVI']

#Sample the input imagery to get a FeatureCollection of training data.
trdata = featureStack.select(bands).sampleRegions(polygons, ['class'],scale= 30)
```

```
classifier = ee.Classifier.randomForest(10).train(trdata, 'class')
```

```
classified = training.classify(classifier)
```

```
my_map = folium.Map(location=[-20,-50], zoom_start=4)

# Set visualization parameters.
vis_params = {
  'min': 0,
  'max': 1,
  'palette': ['black','white']}

# Add the elevation model to the map object.
my_map.add_ee_layer(classified, vis_params, 'Training_Eval')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

- ➔ Make a feature collection of built-up and non built-up pixels.
- ➔ Train our data with random forest classifier from GEE.
- ➔ Use classifier to classify training data.



Classifying Poverty: Accuracy and Classify Evaluation Data

```
trainAccuracy = classifier.confusionMatrix();
```

```
trainAccuracy.accuracy().getInfo()
```

```
0.8615384615384616
```

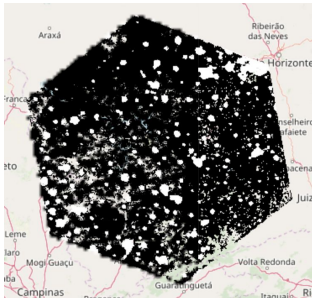
```
#Get a confusion matrix representing resubstitution accuracy.
print('RF error matrix: ', classifier.confusionMatrix().getInfo())
#print('RF accuracy: ', classifier.confusionMatrix().accuracy())
```

```
RF error matrix:  [[39, 2], [7, 17]]
```

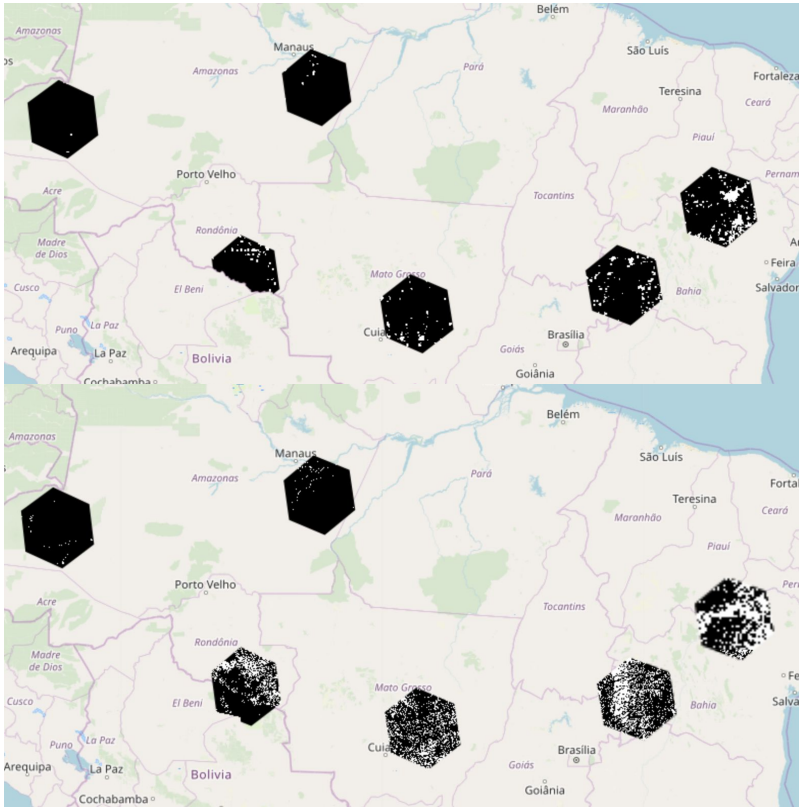
```
validation = featureStack.clip(evalPolys)
validated = validation.classify(classifier)
```

```
my_map = folium.Map(location=[-20,-50], zoom_start=4)

# Set visualization parameters.
vis_params = {
  'min': 0,
  'max': 1,
  'palette': ['black','white']}

# Add the elevation model to the map object.
my_map.add_ee_layer(validation.select('high_lit'), vis_params, 'Training_Eval')

# Add a layer control panel to the map.
my_map.add_child(folium.LayerControl())

# Display the map.
display(my_map)
```

Since we have labeled limited number of points, our classification result is rough. But still, we can get a sense of this methodology of supervised pixel-based classification. Below is an example of classified cell.



Classifying Poverty: Visualizing Classified Evaluation Data



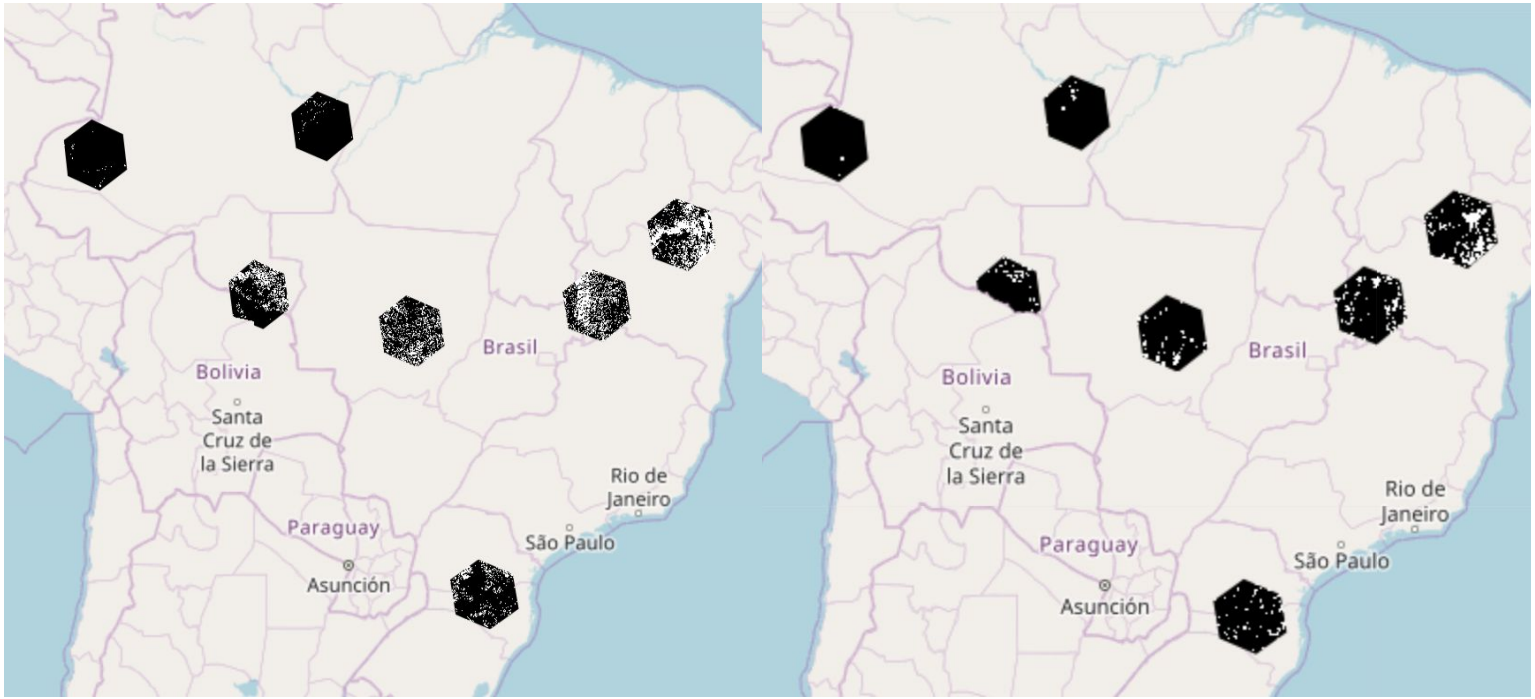
➔ Evaluation Hexagons where high-lit pixels are labeled.

Since masking out vegetation and bodies of water will take up much computation time and easily reach memory limit even with a GPU, I used reclassified night-time lights image here to get a sense of the classification methodology.

➔ Classified Hexagons using random forest classifier.

We may also do parameter tuning for the random forest classifier and hand-label more pixels to improve prediction results.

Classifying Poverty: Visualizing Classified Evaluation Data



Whole picture of our classification results. On the left is the evaluation data; on the right is our classified data.

Future Improvement: Thoughts on Deep Learning U-net

```
# Tensorflow setup.
import tensorflow as tf

tf.enable_eager_execution()
print(tf.__version__)
```

```
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tensorflow_version 1.x magic: more info.

1.15.0
```

```
# Specify names locations for outputs in Cloud Storage.
FOLDER = 'fcnn-demo'
TRAINING_BASE = 'training_patches'
EVAL_BASE = 'eval_patches'

# Specify inputs (Landsat bands) to the model and the response variable.
opticalBands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7']
thermalBands = ['B10', 'B11']
BANDS = opticalBands + thermalBands
RESPONSE = 'high_lit'
FEATURES = BANDS + [RESPONSE]

# Specify the size and shape of patches expected by the model.
KERNEL_SIZE = 256
KERNEL_SHAPE = [KERNEL_SIZE, KERNEL_SIZE]
COLUMNS = [
  tf.io.FixedLenFeature(shape=KERNEL_SHAPE, dtype=tf.float32) for k in FEATURES
]
FEATURES_DICT = dict(zip(FEATURES, COLUMNS))

# Sizes of the training and evaluation datasets.
TRAIN_SIZE = 4000
EVAL_SIZE = 250

# Specify model training parameters.
BATCH_SIZE = 16
EPOCHS = 3
BUFFER_SIZE = 2000
OPTIMIZER = 'SGD'
LOSS = 'MeanSquaredError'
METRICS = ['RootMeanSquaredError']
```

Since we used Google Earth Engine API in Google Collab, we can further perform classification with deep learning methods such as U-net.

Here, we have limited time and RAM for training, but with a Google Cloud Bucket we can save training and evaluation data in tf formats and adopt the U-net with training. In order to reach optimal classification results, we need parameter tuning, changing the epochs, buffers etc.

Link to code:
https://nbviewer.jupyter.org/github/zixi-liu/GoogleEarthEngine/blob/master/BrazilProject.ipynb

Web Resource: Two Choropleth Maps to Visualize GDP& Pop