MVP 2

Advanced Geocomputation

Matt Braaksma

Table of contents

Predicting Land Supply Elasticities

Description

Deforestation and cropland expansion are significant global challenges, contributing to issues such as greenhouse gas emissions and biodiversity loss. These land-use changes are largely influenced by land supply elasticities, which measure how responsive land expansion is to price changes. However, in developing countries where price data is often unavailable, estimating these elasticities becomes difficult. Villoria and Liu (2018) address this gap by estimating gridded land supply elasticities based on factors like market access and land suitability. This Minimum Viable Product (MVP) focuses on developing an alternative framework to predict land supply elasticities with the aim of comparing the econometric model predictions to the machine learning predictions.

The key tasks for this MVP include:

- Gather the raster data from Villoria and Liu (2018)
- Reproject and resample the raster data
- Generate the outcome data from Graesser et al. (2015)
- Rasterize the outcome data
- Find a suitable sample for the CNN model
  - Look for a small square with no missing values
- Combine the predictive rasters into a raster stack
- Build a CNN to predict land elasticities

Code

1. Import modules

Import the necessary data, geospatial, visualization modules.

```
# Standard Library Imports
import os
import numpy as np

# Data Visualization Imports
import matplotlib.pyplot as plt

# Geospatial Data Imports
from osgeo import gdal
import geopandas as gpd
import rasterio
from rasterio.features import rasterize

# Machine Learning Imports
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Set dir
# Define the path to your working directory
data_dir = '../../../base_data/advgeocomp2024/mvp02'
data_dir_land_supply = '../../../base_data/land_supply'
os.makedirs(data_dir, exist_ok=True)
```

2a. Raster Data

```
# Raster paths
base_raster_paths = [
    os.path.join(data_dir_land_supply, 'anntotprecip/anntotprecip'),
    os.path.join(data_dir_land_supply, 'builtupland/builtupland'),
    os.path.join(data_dir_land_supply, 'elevation/elevation'),
    os.path.join(data_dir_land_supply, 'gl-croplands-geotif/cropland.tif'),
    os.path.join(data_dir_land_supply, 'HWSD2_RASTER/HWSD2.bil'),
    os.path.join(data_dir_land_supply, 'irragland/irragland'),
    os.path.join(data_dir_land_supply, 'minutes_to_market/minutes_to_market_10s.tif'), # rep
    os.path.join(data_dir_land_supply, 'potentialveg/potveg'),
    os.path.join(data_dir_land_supply, 'soilcarbon/soilcarbon'),
    os.path.join(data_dir_land_supply, 'SOILPH/soilph')
]

# Corresponding target paths
target_raster_paths = [
    os.path.join(data_dir_land_supply, 'aligned/anntotprecip.tif'),
    os.path.join(data_dir_land_supply, 'aligned/builtupland.tif'),
    os.path.join(data_dir_land_supply, 'aligned/elevation.tif'),
    os.path.join(data_dir_land_supply, 'aligned/cropland.tif'),
```

```
    os.path.join(data_dir_land_supply, 'aligned/HWSD2.tif'),
    os.path.join(data_dir_land_supply, 'aligned/irragland.tif'),
    os.path.join(data_dir_land_supply, 'aligned/minutes_to_market_10s.tif'),
    os.path.join(data_dir_land_supply, 'aligned/potentialveg.tif'),
    os.path.join(data_dir_land_supply, 'aligned/soilcarbon.tif'),
    os.path.join(data_dir_land_supply, 'aligned/SOILPH.tif')
]

# Create the target directory if it doesn't exist
if not os.path.exists(os.path.join(data_dir_land_supply, 'aligned')):
    os.mkdir(os.path.join(data_dir_land_supply, 'aligned'))

# Loop through the base rasters and apply the gdal.Warp function
for base_raster_path, target_raster_path in zip(base_raster_paths, target_raster_paths):
    # Using gdal.Warp to reproject and resize
    gdal.Warp(
        target_raster_path,
        base_raster_path,
        xRes=0.7,
        yRes=0.7,
        resampleAlg='bilinear',
        dstSRS='EPSG:4326'
    )
    print(f"Raster succesfully processed and saved to {target_raster_path}.")
```

```
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/anntotpreci
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/builtupland
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/elevation.t
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/cropland.ti
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/HWSD2.tif.
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/irragland.t
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/minutes_to_
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/potentialve
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/soilcarbon.
Raster succesfully processed and saved to ../../../base_data/land_supply/aligned/SOILPH.tif.
```

2b. Vector Data

```
# Load shapefile
ecos_path = os.path.join(data_dir, 'TerrestrialEcos')
gdf = gpd.read_file(ecos_path)

# Define the eco_names and their observed values from the figure in Graesser et al. (2015)
eco_name_observed_values = {
    'Araucaria moist forests': 0.12,
```

```
    'Alto Paraná Atlantic forests': 0.15,
    'Uruguayan savanna': 0.18,
    'Llanos': 0.10,
    'Espinal': 0.08,
    'Humid Pampas': 0.07,
    'Humid Chaco': 0.09,
    'Peten-Veracruz moist forests': 0.05,
    'Chiquitano dry forests': 0.03,
    'Dry Chaco': 0.20,
    'Cerrado': 0.25,
    'Mato Grosso seasonal forests': 0.60
}

# Filter the GeoDataFrame to include only the specified eco_names
subset_gdf = gdf[gdf['ECO_NAME'].isin(eco_name_observed_values.keys())].copy()

# Add a new column 'observed_value' with the corresponding observed values
subset_gdf.loc[:, 'observed_value'] = subset_gdf['ECO_NAME'].map(eco_name_observed_values)

# Load the template raster file
with rasterio.open(target_raster_paths[0]) as src:
    template_meta = src.meta.copy()
    template_shape = src.shape
    template_transform = src.transform

# Prepare geometries and values for rasterization
# Each geometry should have its 'observed_value' mapped to it
shapes = ((geom, value) for geom, value in zip(subset_gdf.geometry, subset_gdf['observed_val

# Rasterize using the template raster's shape and transform
rasterized_data = rasterize(
    shapes=shapes,
    out_shape=template_shape,
    transform=template_transform,
    fill=0,  # background value, e.g., 0 for no data
    dtype='float32'  # use a suitable data type
)

# Update metadata to match the output raster
output_meta = template_meta.copy()
output_meta.update({"dtype": 'float32', "count": 1})

# Save the rasterized output to a new file
output_path = os.path.join(data_dir, 'rasterized_ecos.tif')
with rasterio.open(output_path, 'w', **output_meta) as dst:
```

```
    dst.write(rasterized_data, 1)
```

3. Build CNN model

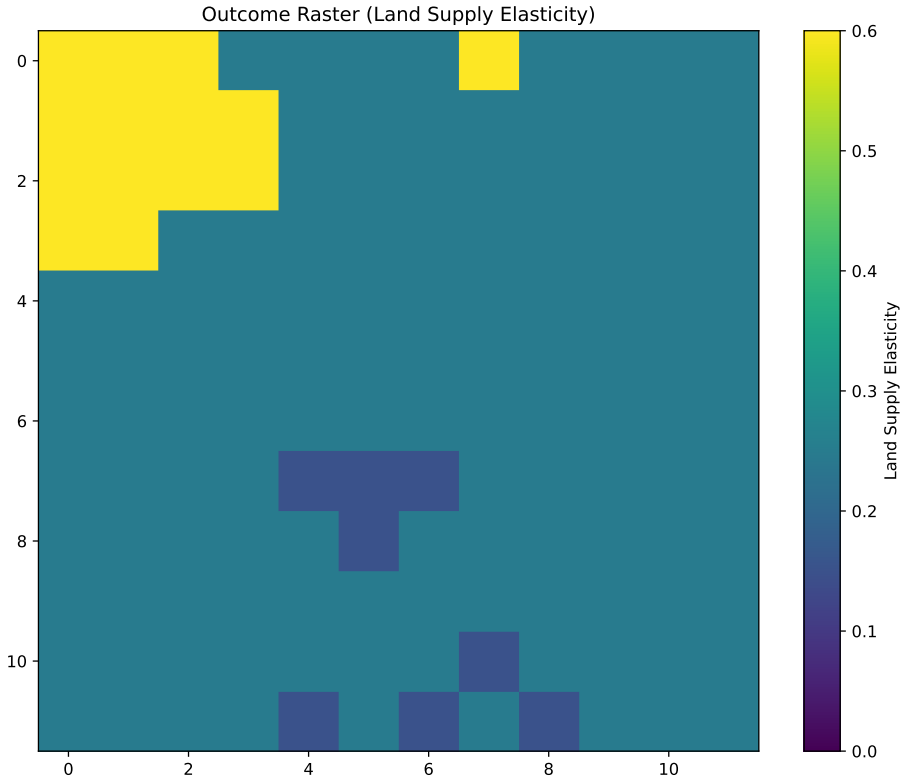
data_dir_land_supply

```
# Function to load a raster file and return its array representation
def load_raster(raster_path):
    """Loads a raster file from the given path and returns it as a NumPy array."""
    dataset = gdal.Open(raster_path)  # Open the raster file
    array = dataset.ReadAsArray()     # Read the raster data into a NumPy array
    dataset = None
    return array

# Function to load and stack multiple raster files
def load_and_stack_rasters(raster_paths):
    """
    Loads multiple raster files and stacks them along a new axis for CNN input
    (height, width, channels).
    """
    raster_stack = []  # List to store individual raster arrays

    # Loop through each raster path, load the raster, and append it to the list
    for path in raster_paths:
        array = load_raster(path)
        raster_stack.append(array)
        print(f"Loaded {path} with shape {array.shape}")

    # Stack the loaded rasters along a new axis (creating the 'channels' dimension)
    return np.stack(raster_stack, axis=-1)

# Load and stack the rasters
raster_paths = [
    os.path.join(data_dir_land_supply, 'aligned/anntotprecip.tif'),
    # os.path.join(data_dir_land_supply, 'aligned/builtupland.tif'), # Not the same dimensio
    os.path.join(data_dir_land_supply, 'aligned/elevation.tif'),
    os.path.join(data_dir_land_supply, 'aligned/cropland.tif'),
    os.path.join(data_dir_land_supply, 'aligned/HWSD2.tif'),
    os.path.join(data_dir_land_supply, 'aligned/irragland.tif'),
    os.path.join(data_dir_land_supply, 'aligned/minutes_to_market_10s.tif'),
    os.path.join(data_dir_land_supply, 'aligned/potentialveg.tif'),
    os.path.join(data_dir_land_supply, 'aligned/soilcarbon.tif'),
    os.path.join(data_dir_land_supply, 'aligned/SOILPH.tif')
]
raster_stack = load_and_stack_rasters(raster_paths)
```

```
# Function to find a square area without missing data (zeros) in the outcome raster
def find_square_with_complete_outcome(outcome_raster, raster_stack, square_size):
    """
    Finds a square region in the outcome raster with no zero values,
    then extracts the corresponding square region from the raster stack.
    """
    height, width = outcome_raster.shape  # Get dimensions of the outcome raster
    x_max_start = width - square_size     # Maximum starting x-coordinate for the square
    y_max_start = height - square_size    # Maximum starting y-coordinate for the square

    # Loop through the raster, stepping by the square size
    for y_start in range(0, y_max_start + 1, square_size):
        for x_start in range(0, x_max_start + 1, square_size):
            # Extract the square region from the outcome raster
            outcome_square = outcome_raster[y_start:y_start + square_size, x_start:x_start +

            # Check if there are any missing data (zeros) in the square region
            has_missing_data = (outcome_square == 0).any()

            # If no missing data, extract the corresponding region from the raster stack
            if not has_missing_data:
                predictor_square = raster_stack[y_start:y_start + square_size, x_start:x_sta
                return predictor_square, outcome_square

    # Raise an error if no valid square was found
    raise ValueError("No square region without zero values in the outcome raster was found."

# Path to the outcome raster (dependent variable)
outcome_raster_path = os.path.join(data_dir, 'rasterized_ecos.tif')

# Load the outcome raster
outcome_raster = load_raster(outcome_raster_path)

# Example usage: Find a 12x12 area without zero values in the outcome
# 12 is the largest available square
square_size = 12

try:
    # Find a valid square region
    predictor_square, outcome_square = find_square_with_complete_outcome(outcome_raster, ras
    print("Found a valid square area without zero values in the outcome raster.")

    # Plot the outcome square
    plt.figure(figsize=(10, 8))
    plt.imshow(outcome_square, cmap='viridis', vmin=np.min(outcome_raster), vmax=np.max(outc
```

```
    plt.colorbar(label='Land Supply Elasticity')
    plt.title('Outcome Raster (Land Supply Elasticity)')
    plt.show()
except ValueError as e:
    print(e)
```

```
Loaded ../../../base_data/land_supply/aligned/anntotprecip.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/elevation.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/cropland.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/HWSD2.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/irragland.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/minutes_to_market_10s.tif with shape (257, 514
Loaded ../../../base_data/land_supply/aligned/potentialveg.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/soilcarbon.tif with shape (257, 514)
Loaded ../../../base_data/land_supply/aligned/SOILPH.tif with shape (257, 514)
Found a valid square area without zero values in the outcome raster.
```




```
# Assuming `predictor_square` and `outcome_square` are lists of extracted squares
predictor_squares = np.array(predictor_square)  # Shape: (num_samples, square_size, square_s
outcome_squares = np.array(outcome_square)      # Shape: (num_samples, 1)
```