Write-up: CSMC12300 Group Project

SateLIFE

Contributors: Cooper Nederhood, Beth Bailey, Laurence Warner, Jo Denby

Major coding contributions by: Beth Bailey, Jo Denby, Cooper Nederhood

Description of data

For our analysis we relied on publicly available satellite imagery obtained and pre-processed through the Google Earth Engine (EE) platform, a satellite imagery and geo-spatial analysis platform from Google.

Raw Data

Satellite imagery is a rich form of data with different satellites gathering different information, called bands. We utilized the following data types/bands:

Landsat 7

- 30m resolution
- Cleaned yearly composites from 2000-2012
- 3 band RGB (red-green-blue) which yields a traditional image of the area of interest, as it appears to an observer

LST

- 1km resolution
- Cleaned yearly composite from 2000-2012
- 1 band quantifying the land surface temperature of the area of interest

NDVI

- 30m resolution
- Cleaned yearly composites from 2000-2012
- 1 band, Normalized Difference Vegetation Index, essentially captures the 'greenness' of the land below. This is a measure of the vegetation

Night_lights

- 1km resolution
- Cleaned yearly composites from 2000-2012
- 1 band, a measure of the average nightime luminosity of the area of interest. This has been shown to be correlated with economic activity and has been widely used as a proxy for economic activity within the development literature

GeoJSON

- Not obtained through Google but through OpenStreetMaps, a crowd-source mapping platform
- .gpx files (geojson type) containing GPS coordinates describing polygons of the neighborhood boundaries within the city of Kinshasa
- Described in greater detail in "Getting boundaries" below

Cleaning & Exporting Satelite imagery

Raw satellite imagery can be extremely noisy with missingness due to cloud cover, distortions caused when the satellite passes over different areas of the Earth at different times, etc. We utilized the Google EE code editor platform to perform cleaning and rectifying (fitting satellite images together into a clean composite). The EE code editor is written in javascript syntax. The script "geeimage.js" is our script for downloading, processing, and exporting the massive files to Google Drive.

Google EE exports all images as .tiff files, which we then load into Python as numpy arrays. A particular challenge when using any geospatial data is to have the data aligned geographically, so that each pixel within the numpy arrays corresponds to the same geographic location in another .tiff file. Again, we utilized Google EE to ensure the geographic alignment of our raw data.

Getting Boundaries:

In addition to making statements about the Kinshasa-Brazzaville area, we wanted to make finer-grained statements comparing the two countries, the two-cities, the river between, the city neighborhoods, etc. This required combining the spatial satellite data with GPS indexed boundary data.

OpenStreetMaps, as part of project to map the Democratic Republic of the Congo, provides GPS coordinate data in the form of .gpx (geojson) files which define neighborhoods within Kinshasa. The script "clean_gps.py" parses the .gpx files. However, to be compatible the GPS data needs to be processed through Google EE. Thus, the "clean_gps.py" takes the parsed .gpx files and outputs a .txt file of EE compatible code which builds a tiff image containing the

gps boundaries. This txt file is then the foundation for the script "kinshasa_boundaries.js"

Google EE has two image types: raster and feature. The images describe above are rasters and the GPS boundaries are feature types. Again, using EE we convert the GPS boundaries from feature types into raster/image types for compatibility. We also manually draw boundaries where GPS data is not available from OpenStreetMaps. The boundary analsysis is layered, with different codes denoting country differences and different areas within country. Thus, the analysis in EE yields 3 boundary tiff files which are then composited into 1 comprehensive boundary file. This occurs in the "clean_boundaries.py" script

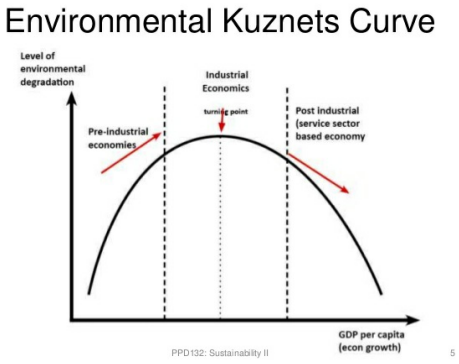
Class Design

As described in the comments and header of the "util.py" file, the SatData class facilitates the construction and analysis of the satelite tiff image files.

Hypotheses

The Kuznets Curve

See the diagram below for an example of the Kuznets curve hypothesis.



Developing nations like DRC & Republic of Congo are expected to be on the left hand side of the graph, and undergo environmental degradation as they develop over time.

How the data we collected correspond to measures of environmental degradation:

- Reduction of green spaces: NDVI - Normalized Difference Vegatation Index
- Light pollution: Night Lights Index
- Urban build-up: Land Surface Temperature

Our hypothesis is that these indices should increase over time. We also predict spatial variation between neighborhoods.

A Tale of Two Cities

Brazzaville & Kinshasa are an interesting case study because they are the world's two most proximate capital cities (ignoring the Vatican City).

However, in Kinshasa's country, DRC, there has been a brutal civil war raging throughout the period.

We hypothesise that Brazzaville has seen greater economic development. This can be measured by proxy using the Night Lights Index, as in Mellander et al. (2013). Hence we expect Brazzaville's Night Lights growth to outstrip Kinshasa's.

Algorithms

Autocorrelation

Autocorrelation is the correlation of a signal with a delayed copy of itself (wikipedia). In other words, it allows us to determine whether the value at a certain pixel within a band is correlated with the value in the next period (or periods). To calculate this, we normalize a each band such that it is mean zero. We then multiply each normalized pixel value in year=t with the corresponding normalized pixel avlue in year=t-k, where k is the level of autocorrelation. Our code currently tests autocorrelations of k=1,2,3 to determine how persistent any correlation is and whether the nature changes with respect to the degree. We then divide by each std deviation and take the expected value of the transformed variable, thus calculating the statistic across the entire band. See "util.py" and "auto_correlation.py" for code.

K-Nearest Neighbors

This recently added calculation is still a work in progress, but we present it here for its coding usefulness. Unlike our other statistics, this calculation is made not across an entire band but across a small k-near-neighborhood of a given pixel. Thus, for any given k x k neighborhood we can calculate a proxy for the spatial correlation within the neighborhood. We essentially slide this neighborhood "filter" across the band calculating with each change, resulting in a new 2D surface. This is very similar to the convolution calculations within a convolutional neural network. See "util.py" and "k_nearest.py" for code

Big data

Automation via Bash Scripts

In order to maximize our use of MPI via the Google Cloud VM platform while avoiding the tedium of manually initializing, preparing, and linking instances/nodes, we wrote a handful of

Bash scripts that would automate the process. For `gcloudsetup.sh`, the user simply runs the script specifying the nubmer of VM instances desired, and the shell, using the `gcloud` command-line interface, will create those instances, send over requisite files/data via `SCP`, and install necessary software and packages. From within each node, the script also calls `chain.sh`, which connects each node to each other nodes via `SSH` to facilitate MPI. Finally, `update.sh` serves to automate the process of sending files and directories to all nodes at once. Look within the scripts for explicit documentation.

Results

Kuznets curve

Our results do not support the Kuznets Curve hypothesis.

Challenges