Poverty Mapping in Off-Census Years

Paul Corral, Heath Henderson, and Sandra Segovia

World Bank Summer University 2024

June 3, 2024

Introduction

- Recent developments in poverty mapping in off-census years have departed from standard practice in important ways.
- In brief, standard practice consists of two basic steps:
  1. Fit a parametric statistical model by regressing a survey-based welfare measure on some area-level covariates, typically census aggregates.
  2. Predict the welfare measure for all geographic entities using the census aggregates.
- In contrast to this "traditional" approach, the "modern" approach:
  1. Relies on non-parametric, machine-learning methods rather than parametric models
  2. Tends to rely more on non-traditional, remotely-sensed covariates in the estimation and prediction stage
- In what follows, we'll discuss this modern approach in more detail and work through a simple application.

An Example

RESEARCH ARTICLE | ECONOMIC SCIENCES

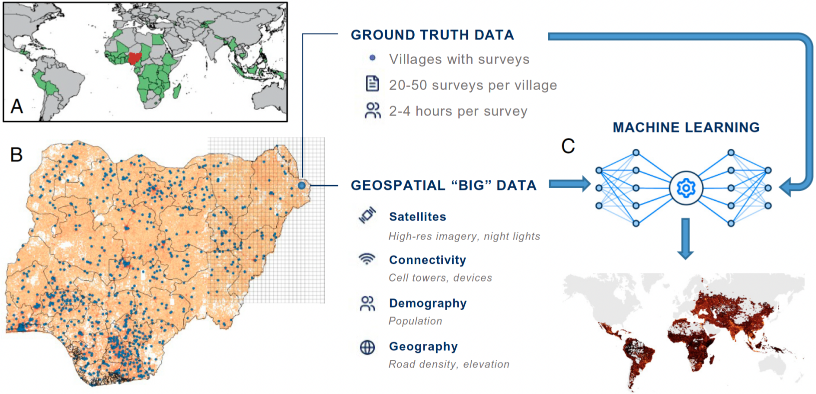
Microestimates of wealth for all low- and middle-income countries

Guanghua Chi, Han Fang, Sourav Chatterjee, and Joshua E. Blumenstock Authors Info & Affiliations

Edited by Jose Scheinkman, Department of Economics, Columbia University, New York, NY; received July 24, 2021; accepted November 14, 2021

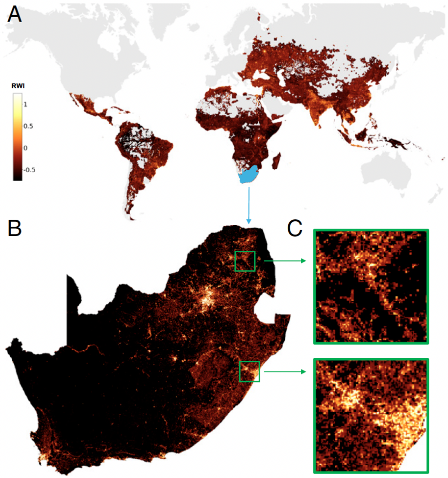
January 11, 2022 | 119 (3) e2113658119 | https://doi.org/10.1073/pnas.2113658119

An Example



A
B
GROUND TRUTH DATA
Villages with surveys
20-50 surveys per village
2-4 hours per survey
GEOSPATIAL “BIG” DATA
Satellites
High-res imagery, night lights
Connectivity
Cell towers, devices
Demography
Population
Geography
Road density, elevation
C
MACHINE LEARNING

An Example



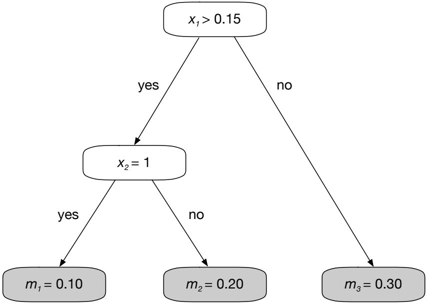
Machine Learning

- Machine learning plays a critical role in the modern approach to poverty mapping.
- The goal of machine learning is to develop high-performance algorithms for prediction, classification, and clustering/grouping tasks.
- Broadly speaking, machine-learning methods can be divided into two basic types:
  1. Unsupervised learning: Seeks to identify clusters of observations that are similar
  2. Supervised learning: Uses a set of features to predict some outcome of interest
- Supervised learning can be further divided into regression and classification tasks:
  1. Regression: Concerned with predicting continuous outcomes
  2. Classification: Focuses on predicting categorical outcomes

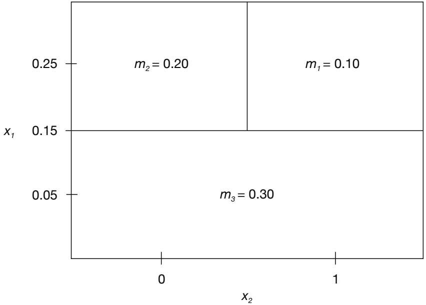
Machine Learning

- There are many machine-learning methods for regression and classification (e.g., random forests, support vector machines, or Bayesian additive regression trees).
- To fix ideas, we'll focus on one method that's been especially popular for poverty mapping: gradient boosting.
- Gradient boosting machines combine a large number of weak "learners" to form a stronger "ensemble" prediction:
  - Models are added to the ensemble sequentially by fitting new learners to the negative gradient of the loss function.
  - With a squared-error loss function, this amounts to sequentially fitting new models to the current residuals of the ensemble.
- Extreme gradient boosting (XGBoost) is a particularly popular implementation that uses classification and regression trees as the base models.

Machine Learning



(a) Regression tree



(b) Covariate space

Machine Learning

- For any given observation, the ensemble prediction used by XGBoost is the sum of that observation's predictions across all trees in the ensemble:

$$y_i^p = \sum_t f_t(x_i)$$

- To build the trees in the ensemble, XGBoost minimizes the following objective function:

$$\sum_i l(y_i^d, y_i^p) + \sum_t r(f_t)$$

- XGBoost uses the following specification for the regularization term:

$$r(f_t) = \gamma M_t + \frac{1}{2}\lambda \sum_j m_{tj}^2$$

Machine Learning

- To minimize the objective function, the model is trained in a sequential manner, building one tree at a time.
- Ideally, one would select trees by enumerating all possible structures and then using the one that minimizes the objective function, but this is often intractable.
- XGBoost instead seeks to minimize the objective function by greedily optimizing one level of the tree at a time:
  1. For a given node, calculate the reduction in the objective function for every possible split rule for every available covariate.
  2. Split the node using the covariate and split rule that maximally reduces the objective function.
- XGBoost continues splitting nodes until some user-specified stopping rule is met.

Machine Learning

- Like many machine-learning models, XGBoost relies on various hyperparameters that must be selected by the user:
    - The regularization term (i.e., γ and λ)
    - Maximum tree depth and number of trees
    - Feature subsampling
    - The learning rate
- There are several different ways one might approach hyperparameter selection:
    - Use the default hyperparameters
    - Grid search
    - Bayesian hyperparameter optimization

An Application

- Poverty maps in off-census years are generally produced with two types of data:
  - Direct poverty estimates for a subset of areas
  - A list of covariates or predictors for all areas
- For a simple application, we'll use the 2015 Mexican Intercensal Survey (MIS) for both data sources:
  - The sample consists of 5.9 million households
  - Representative at the national, state, and municipality level
  - Gathered information on household income, location, demographics, etc.
- We'll treat the MIS as if it were a real census and use it to simulate the data used for poverty maps in off-census years:
  - Sample households from the MIS to obtain direct poverty estimates for a subset of municipalities.
  - Collapse the micro-data to the municipality level to obtain area-level covariates.

An Application

municipality	direct	true	hhsize	age_hh	male_hh	piped_water	ıo_piped_wate	no_sewage	sewage_pub
1011001	0.0731939	0.0792447	-0.392702	-0.656265	-0.327982	0.713719	-0.703915	-0.776471	1.31692
1011002	0.311203	0.262542	1.16187	-0.751027	0.693776	0.687339	-0.683918	-0.444965	0.983853
1011003	0.231818	0.243805	0.249574	0.0228985	0.326906	0.709911	-0.698422	-0.764816	1.19186
1011004	0.182292	0.246614	1.39054	-0.284892	1.05251	0.760751	-0.762797	-0.701359	1.32361
1011005	0.0783132	0.0922531	0.223972	-1.61063	0.866597	0.695103	-0.689421	-0.769329	1.22863
1011006	0.145161	0.142484	0.767608	-0.98952	-0.0114516	0.678938	-0.670958	-0.72303	1.24327
1011007	0.365188	0.239514	1.35238	-0.888183	0.294417	0.675333	-0.678623	-0.667009	1.23939
1011008	0.212766	0.207656	0.977491	-1.11958	-0.69044	0.636178	-0.624729	-0.560338	1.19124
1011009	0.224299	0.228771	0.798937	-0.761821	1.27636	0.705323	-0.693837	-0.660832	1.18033
1011010	0.230769	0.236398	0.740872	-0.791986	1.41223	0.565258	-0.553847	-0.424855	0.884507
1011011	0.08	0.107069	-0.297909	-2.6176	0.715367	0.717107	-0.705614	-0.783883	1.35315
1021001	0.0316456	0.0720229	-1.41713	-1.37729	-1.40387	-0.0613733	0.0676727	-0.166492	0.0262108
1021002	0.127098	0.0514764	-1.53969	-1.3291	-0.777536	0.598097	-0.591137	-0.548799	0.956378
1021003	0.0377359	0.053013	-1.19462	-1.31876	-1.1787	0.135678	-0.137525	-0.644508	0.553619
1021004	0.0105125	0.0438967	-1.35347	-1.54744	-1.15381	0.54075	-0.534839	-0.663636	1.10457
1021005	nan	0.0627038	-1.09161	-1.20586	-1.1212	-0.0864749	0.06466	-0.66397	0.0944137
1031001	0.0664452	0.0982665	-1.55223	-0.407757	-0.48133	0.333902	-0.339917	-0.438449	0.33179

An Application

```python
# Import libraries
import xgboost as xgb
import pandas as pd

# Set directory
path = '/Users/hendersonhl/Desktop/Summer University/Application/'

# Import data
data = pd.read_csv(path + 'data.csv', header = 0)
sample = data.dropna()
y = sample['direct']
X = sample.drop(columns = ['municipality', 'direct', 'true'])

# Implement XGBoost
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100,
        max_depth=6, eta=0.3)
model.fit(X, y)

# Generate poverty estimates
X_all = data.drop(columns = ['municipality', 'direct', 'true'])
y_pred = model.predict(X_all)
```

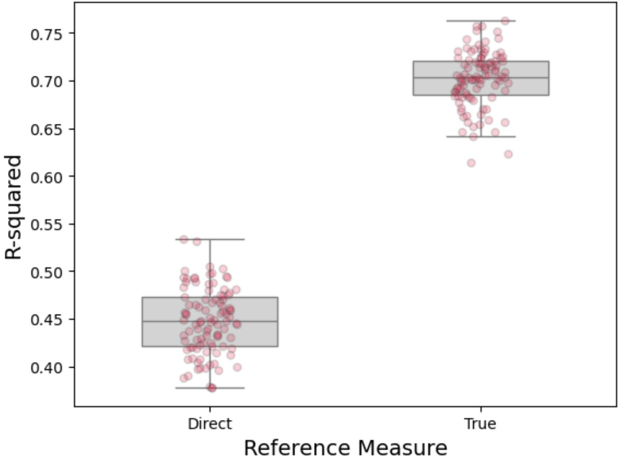
An Application

```
# Import additional functions
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Create empty lists
r2_direct = []
r2_true = []

# Run loop
for i in range(100):
    # Split data and fit model
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
    model.fit(X_train, y_train)
    # Get predicted and true values
    y_pred = model.predict(X_test)
    y_true = [sample['poor'][i] for i in y_test.index]
    # Save R-squared results
    r2_direct.append(r2_score(y_test, y_pred))
    r2_true.append(r2_score(y_true, y_pred))
```

An Application



Resources

- Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5): 1189–1232.
- Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.
- Natekin, A. and Kroll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7(21): 1–21.
- Corral, P., Henderson, H., and Segovia, S. (2023). Poverty mapping in the age of machine learning. World Bank Policy Research Working Paper No. 10429.