Clarity AI
Data Science Take Home Exercise

POVER-T TESTS: PREDICTING POVERTY

Hosted by the World Bank

DRIVEN DATA

Authored by
Isabel Puche Marín

December 2020

This report intends to summarise step-by-step this exercise/project development. The jupyter notebook *Solutions_Pover-T_Tests.ipynb* (and to a less extent *Prepa_Pover-T_Tests.ipynb*) provides greater detail of analysis, procedures, model's specifications and supporting documentation, among others. Please, refer to it and the corresponding GitHub repository for more information.

Table of Contents

List of Abbreviations

NANs	Not a Number
ML	Machine Learning
DL	Deep Learning

1. Business Understanding

This project aims at **predicting whether or not a given household is poor or not**. This is a binary classification problem, which specifically posits the following question: what is the **probability of a given household of being poor**. This probability is actually the prediction value and the loss function score measures the prediction performance. For a binary classification, the typical loss function is the binary cross-entropy or **log loss**. In this case, we shall output a file with the probability for each household (in test dataset) of being poor.

Two .csv files are provided, a **training dataset and a test dataset**, with the variables:

– *non labelled column*: supposedly individuals' id, which is unambiguously related to a household id, that is, there is only one individual's iid for a household id, meaning that no information about household size can be extracted.

– *id*: households' identification

– *poor*: binary target variable corresponding to the assessment of whether a household is above or below the poverty line.

The remaining eight columns correspond each with a survey question. Each question is either multiple choice, in which case each choice has been encoded as random string, or it is a numeric value. Many of the multiple choice questions are about consumable goods--for example, *does your household have items such as Bar soap, Cooking oil, Matches, and Salt.* Numeric questions often ask things like, *how many working cell phones in total does your household own? or how many separate rooms do the members of your household occupy?* Among these variables, we have:

– *four categorical variables*

– *four numerical variables*

An **exploratory data analysis** uncovers the following:

– there are missing values for features both in train and in test datasets

– numerical variables follow skewed distributions and have outliers (fat tails)

– we assume three categorical variables of low dimension (less than 10 classes with uneven frequencies) and one variable of high dimension (31 classes with similar frequencies)

– the proportion of poor households is equitably distributed across train and test dataset

2. Data Preparation (Preprocessing)

Considering the above analysis, data needs preprocessing in order to (i) impute NANs (numerical and categorical), (ii) scale numerical data and (iii) encode categorical variables (with differentiated treatment). Failing to do so may affect the performance of the model or even hinder the model fit.

There are different ways to address this question:

– either building a preprocessor object (*Solution_Pover-T_Tests.ipynb*) or,

– preprocess the data *out-of-pipe* and store it in a new train/test dataset (*Prepa_Pover-T_Tests.ipynb)*

The first option has several advantages: it allows to (i) embed the preprocessor in a pipeline object with its classifier, (ii) perform some hyperparameter tuning and (iii) fit the model successively. The resulting estimator can be saved as a pickle to export to production. The second option has its drawbacks at production stage, as we shall see. However, for the sake of completeness, *Prepa_Pover-T_Tests.ipynb* notebook has been kept.

More concretely, the following steps should be accomplished in order to obtain a preprocessor object:

i. As for **numerical variables,** we shall in a **pipeline** impute NANs with a median strategy and scale the data with PowerTransformer function well suited for skewed distributions

ii. Regarding **categorical variables**, we will perform an **ordinal encoding** for low-dimensional variables and a **target encoding** for the high-dimensional variables. These have been preferred to one-hot encoding to avoid learning potential loss. However, we are aware that target encoding is more prone to overfitting (read discussion). As such, some noise will be introduced in the model fits.

iii. Finally, a **column transformer** has been used to process data by columns (features) and saved in an object for later use.

3. Model building

In this phase, a common procedure for ML and DL model building has been the following:

i. define a **model instance**

ii. define a **model pipeline** embedding the preprocessor and the model classifier previously defined

iii. perform some **hyperparameter tuning** with RandomizedSearchCV, which is apparently more efficient than GridSearchCV. This allows to plug our pipeline, identify best parameters, perform some cross validation and determine our scoring metric, among others

iv. **fit our model** on our train dataset

v. **predict the probability** of occurrence with our test dataset and,

vi. retrieve the **best scoring metric** for each model

Having said that, we have built seven models (as specified by the note of competition: gradient boosting methods and neural networks) and only trained six, since we encountered a problem at importing tensorflow and keras libraries that unfortunately could not be solved. Models' specifications and results can be checked in the notebook.

– Logistic regression as our baseline model

– Random Forest

– Extreme Gradient Boosting

– Light Gradient Boosting

– CatBoost model

– Stacking model of the previous models

– Neural network

The most performing one has been CatBoost although scorings have been very similar across models. A .csv file with results (probability of poor households) has been saved.

4. Productization Plan

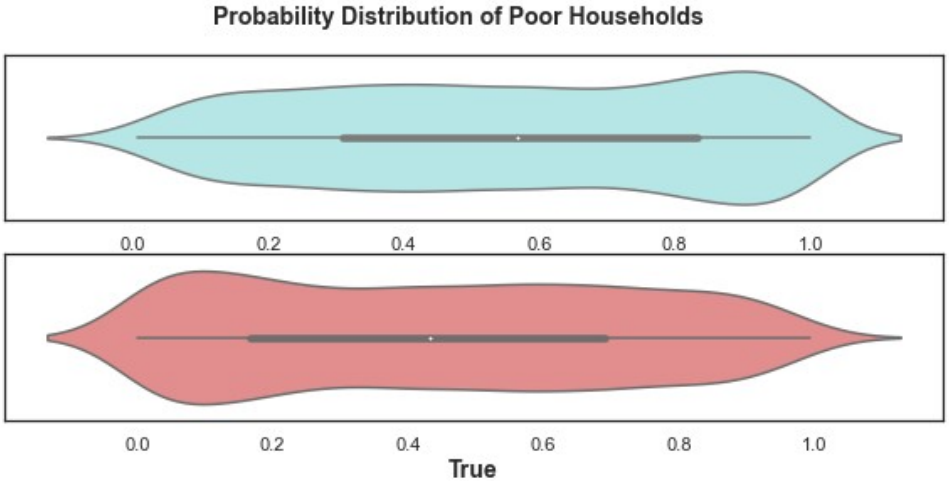
After building a model in lab conditions, we shall export a copy of it to the production server, where new data will be labelled. This process is called model deployment or pushing to production. To enable this, we shall serialize our model, that is, convert the fitted object of the best model (CatBoost) to a binary file with pickle.

As proceeded, we encapsulate the whole process into a single object to interact with production otherwise we will be forced to repeat one or various processes in production (which could happen if we preprocessed the data *out-of-pipe*). Our aim is to avoid bugs which can eventually cause financial problems.

5. Main Results and Future steps

5.1. Main Results

With a log-loss score of 0.4888, the best model (CatBoost) predicts a smaller amount of poor households with high probability. The contrary holds true for non-poor households.



In a binary classification problem ($N=2$) for balanced classes (every class has the same prevalence), the equation is the following: $Logloss=-\log(\frac{1}{2})$, equalling 0.693, in which case, 0.4888 might seem a good scoring.

However, classes in this problem are unbalanced (with a possible prevalence of 85% for poor households over non-poor ones). In such case, a good scoring would have to lie between 0.50 and 0.33, all the more so since the winners of this competition got 0.1447.

5.2. Future Steps

For that reason, further research can be conducted on the following:

– feature importance and selection

– further hyperparameter tuning

– greater stacking model complexity

– solve for tensorflow and keras libraries and fit neural network(s)