From Shadows to Light Illuminating Costa Rican Household Poverty Prediction

1. Introduction:

- This project focuses on predicting household poverty levels in Costa Rica. The goal is to develop an accurate predictive model that can assist in identifying households in need and provide targeted interventions. The dataset used for this project is the 'train.csv' file.

2. Data Preprocessing:

- The necessary libraries are imported, including pandas, numpy, matplotlib, seaborn, sklearn, bayes_opt, lightgbm, xgboost, and catboost.
- The dataset is loaded into a pandas DataFrame using the `pd.read_csv()` function.
- The dimensions of the DataFrame are printed to understand its size.
- The structure and information of the DataFrame are displayed using `df.info()`.
- Columns with missing values are identified and stored in the `missing_cols` variable.
- Missing values are imputed using the backward fill (`bfill`) and forward fill (`ffill`) methods to fill missing values.
- It is ensured that there are no more missing values in the DataFrame.
- The data types of the columns are checked, and numeric and categorical columns are identified and stored in separate variables.
- The 'edjefe' and 'edjefa' columns, representing education level, are converted to numeric format.
- The 'dependency' column is recalculated by taking the square root of the 'SQBdependency' column.
- Unnecessary columns ('Id' and 'idhogar') are dropped from the DataFrame.

3. Exploratory Data Analysis:

- Descriptive statistics for the numeric columns are calculated using df.describe() and displayed. This provides measures such as count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum for each numeric feature, giving an overview of their distribution and variability.

- Duplicate rows in the dataset are checked using df.duplicated().sum(). This helps identify if there are any duplicated instances in the data, which may need to be handled or removed to avoid bias in the analysis.
- The distribution of the target variable is visualized using a count plot from the seaborn library. This plot shows the number of occurrences of each target class, providing insights into the class distribution and potential class imbalances.
- The correlation between numeric features is analyzed using a heatmap visualization created with sns.heatmap(). This plot displays a matrix of correlation coefficients between pairs of numeric features, allowing us to identify strong positive or negative correlations. The heatmap helps identify potential multicollinearity issues and informs feature selection or engineering decisions.

4. Feature Engineering:

- Additional features are created to enhance the predictive power of the model.
- Ratios, per-person features, and comparisons of household attributes are calculated and added as new columns to the DataFrame.
- Each newly created feature is explained in the code comments for better understanding.

5. Model Optimization:

- The XGBoost classifier is optimized using Bayesian Optimization and cross-validation. The xgb_cv() function defines the objective for optimization, which evaluates the performance of the XGBoost model with specific parameter values. The optimize_xgb() function performs the optimization process by using Bayesian Optimization to search for the optimal combination of hyperparameters, such as the number of estimators, maximum depth, gamma, and subsample. The optimization process aims to maximize the cross-validation score.
- Similarly, the CatBoost classifier is optimized using Bayesian Optimization and cross-validation. The cb_cv() function defines the objective for optimization, which evaluates the performance of the CatBoost model with specific parameter values. The optimize_cb() function performs the optimization process by using Bayesian Optimization to search for the optimal combination of hyperparameters, such as the number of estimators and depth. The optimization process aims to maximize the cross-validation score.

- The LightGBM classifier is optimized using Bayesian Optimization and cross-validation. The lgb_cv() function defines the objective for optimization, which evaluates the performance of the LightGBM model with specific parameter values. The optimize_lgb() function performs the optimization process by using Bayesian Optimization to search for the optimal combination of hyperparameters, such as the number of estimators, num_leaves, min_child_samples, and subsample. The optimization process aims to maximize the cross-validation score.
- The optimized parameter values for each model are reported, indicating the hyperparameters that yielded the best performance during the optimization process. These optimized parameter values can be used to train the final models with improved configurations.

6. Model Training and Evaluation:

- The dataset is split into training and testing sets using train_test_split() from the sklearn library. This division allows us to train the models on a portion of the data and evaluate their performance on unseen data. The training set is typically larger to ensure sufficient data for model learning, while the testing set serves as an independent dataset for performance evaluation.
- The XGBoost, CatBoost, and LightGBM classifiers are instantiated with their optimized parameters obtained from the optimization process. These classifiers are powerful gradient boosting algorithms that have shown effectiveness in various machine learning tasks.
- A stacking classifier is built using the three models. The stacking classifier combines the predictions of the base models (XGBoost, CatBoost, and LightGBM) to make the final prediction. It leverages the strengths of each individual model and aims to improve overall performance.
- The stacking classifier is trained on the training data using the fit() method. This process involves fitting the base models on the training data and training the meta-model to learn the optimal combination of base model predictions.
- Predictions are made on the test data using the predict() method of the stacking classifier. The test data is passed to the stacking classifier, which utilizes the trained base models and meta-model to generate predictions for the target variable.
- The accuracy of the model is evaluated by comparing the predicted labels with the true labels using the accuracy_score() function from the sklearn library. This metric measures the proportion of correctly classified instances and provides an indication of the model's overall accuracy in predicting the target variable.

- The achieved accuracy is reported, indicating the performance of the stacking classifier on the test data. A higher accuracy score suggests better predictive performance.