Toward Proactive Policy Design: Identifying “To-Be” Energy-Poor Households Using SHAP for Early Intervention

Reproducibility Package

This repository contains the code and data for calculating the results presented in the paper. It provides a step-by-step guide to reproduce the results and generate the tables and figures presented in the paper.

Abstract

Identifying at-risk populations is essential for designing effective energy poverty interventions. Using data from the HILDA Survey, a longitudinal dataset representative of the Australian population, and a multidimensional index of energy poverty, we develop a machine learning model combined with SHAP (SHapley Additive exPlanations) values to document the short- and long-term effects of individual and contextual factors—such as income, energy prices, and regional conditions—on future energy poverty outcomes. The findings emphasize the importance of policies focused on income stability and may be used to shift the policy focus from reactive measures, which address existing poverty, to preventive strategies that target households showing early signs of vulnerability.

Highlights

- Introduces SHAP values in energy poverty analysis, improving interpretability and insight into predictive factors.
- Reveals the role of income and income volatility on future energy poverty using longitudinal data.
- Identifies critical periods for policy intervention, emphasizing proactive rather than reactive measures.
- Highlights the varying impacts of energy prices and household composition on energy poverty risk over time.

About the dataset

The HILDA (Household, Income and Labour Dynamics in Australia) dataset is a longitudinal study that tracks Australian households annually. It collects data on income, employment, health, education, and family relationships. The study began in 2001 with about 7,600 households and 13,000 individuals and has been updated to account for panel attrition. It maintains a high retention rate, which supports its use in long-term analyses, though some checks address potential biases from participants leaving the study.

The original dataset is available in the dataset folder.

Package requirements for the project

It is recommended to use a virtual environment to install the packages. To create a virtual environment using, e.g., conda, run the following command:

```
conda create -n myenv python=3.11
```

where `myenv` is the name of the virtual environment. To activate the virtual environment, run the following command:

```
conda activate myenv
```

This project requires Python 3.11, and some packages that can be installed using `pip`. To install the required packages, run the following command:

```
pip install -r requirements.txt
```

For reproducibility, the versions of the packages used in this project are locked. The versions can be found in the requirements.txt file.

One-line command to run the project

In order to run the project, you can use the following command:

```
bash run.sh
```

This command must be run in the root directory of the project, and it will run the scripts in the correct order and generate the results for the paper.

Each step of the command is printed in the terminal.

This script assumes that the dataset is available in the dataset (`Hildabalin.csv`), and that the required packages are installed.

Please note that we fixed the random seed in the scripts to ensure the reproducibility of the results (i.e., `random_state=42`).

Depending on the machine's specifications, the scripts may take a while (specially the `4_ObtainSHAPsExplain.py` script) to run.

Description of the scripts

The scripts are numbered in the order they should be run. Here is a brief of each script:

utils.py

- Contains utility functions used in the main scripts.

1_DataPreparation.py

- Processes the original dataset (available in the dataset directory) by:

  - Cleaning and engineering features, including age group categorization, year-on-year changes for economic indicators, and lagged variables.
  - Filtering users based on consecutive years of data.
  - Generating subsets for analysis.
- Saves results into CSV files (saved in the filtered_data directory).

2_ModelTraining.py

- Evaluates different machine learning models using various hyperparameter configurations.
- Performs cross-validation to assess model performance based on metrics like accuracy, precision, recall, and sensitivity.
- Calculates class-specific sensitivity metrics from confusion matrices.
- Scales features using a robust scaler to handle outliers.
- Saves:
  - Trained models with unique identifiers in the `models` directory.
  - Evaluation results in CSV files in the `results` directory.

3_ModelsResults.py

- Consolidates performance results from multiple classifiers by:
  - Loading their respective CSV files.
  - Tagging each dataset with the model name and merging them into a unified DataFrame.
  - Extracting key parameters (threshold and window) using regex.
  - Calculating metrics for grouping by model, threshold, and window.
  - Identifying the best-performing models.
- Saves:
  - Aggregated results to `all_results.csv`.
  - Best-model datasets to `best_models.csv`.

4_ObtainSHAPsExplain.py

- Computes SHAP (SHapley Additive exPlanations) values to explain the predictions of pre-trained machine learning models.
- Processes a list of models and their parameters.
- Validates model performance using sensitivity metrics.
- Computes feature importance using SHAP.

5_ResultsFiguresScatters.py & 5_ResultsFigures.py

- Generates figures based on results from previous scripts.
- Creates visualizations showing the characteristics of SHAP values.
- Corresponds to Figures 1-8 in the paper.

5_ResultsTables.py

- Evaluates trained machine learning models on test data.
- Calculates performance metrics, such as:
  - Sensitivity, specificity, precision, recall, F1 score, ROC AUC, and balanced accuracy.
- Organizes results into a unified table.
- Corresponds to Table C.3 (and Table 1) in the paper.

Please note that the scripts are designed to be run in sequence, as they depend on the output of the previous script. However, if you opt to run the `one-line command` mentioned above, the scripts will be run in the correct order.

Results

The results are saved in the results directory. The following table provides the links to the results and their corresponding table/figure of the paper:

Table 1

- results/tables/TableC.3.md

Table C.3

- results/tables/TableC.3.md

Figures

Figure 1

- results/figures/balancedBaggingClassifier/window_8_0.1_notime.png

Figure 2

- results/figures/balancedBaggingClassifier/window_8_0.1.png

Figure 3

- results/figures/balancedBaggingClassifier/window_8_0.1_temporal.png

Figure 4

- results/figures/balancedBaggingClassifier/window_4_0.1_hhsizeequiv.png
- results/figures/balancedBaggingClassifier/window_4_0.1_ln_yearseduc.png
- results/figures/balancedBaggingClassifier/window_4_0.1_lnndincome_heu.png

- results/figures/balancedBaggingClassifier/window__4__0.1__lnndincome__heu__change.png
- results/figures/balancedBaggingClassifier/window__4__0.1__parttimerate.png

Figure 5

- results/figures/balancedBaggingClassifier/window__2__0.1__notime.png
- results/figures/balancedBaggingClassifier/window__4__0.1__notime.png
- results/figures/balancedBaggingClassifier/window__12__0.1__notime.png
- results/figures/balancedBaggingClassifier/window__14__0.1__notime.png

Figure 6

- results/figures/balancedBaggingClassifier/window__2__0.1.png
- results/figures/balancedBaggingClassifier/window__4__0.1.png
- results/figures/balancedBaggingClassifier/window__12__0.1.png
- results/figures/balancedBaggingClassifier/window__14__0.1.png

Figure 7

- results/figures/balancedBaggingClassifier/window__8__0.2__notime.png
- results/figures/balancedBaggingClassifier/window__8__0.4__notime.png

Figure 8

- results/figures/balancedBaggingClassifier/window__8__0.2.png
- results/figures/balancedBaggingClassifier/window__8__0.4.png