Optimal Policy Learning using Stata

Giovanni Cerulli
IRCrES-CNR
Rome, Italy
giovanni.cerulli@ircres.cnr.it

Abstract. This article introduces the Stata package `OPL` for optimal policy learning, facilitating ex-ante policy impact evaluation within the Stata environment. Despite theoretical progress, practical implementations of policy learning algorithms are still poor within popular statistical software. To address this limitation, `OPL` implements three popular policy learning algorithms in Stata – *threshold-based*, *linear-combination*, and *fixed-depth decision tree* – and provides practical demonstrations of them using a real dataset. Also, the paper presents policy scenario development proposing a *menu* strategy, particularly useful when selection variables are affected by welfare *monotonicity*. Overall, the paper contributes to bridging the gap between theoretical advancements and practical applications in the field of policy learning.

Keywords: Ex-ante policy evaluation, Optimal policy learning, optimal treatment assignment, Machine learning

Acknowledgment: This work was supported by FOSSR (Fostering Open Science in Social Science Research), funded by the European Union - NextGenerationEU under the NPRR Grant agreement n. MUR IR0000008.

1 Introduction

This article introduces the Stata package `OPL`, designed for conducting *optimal policy learning* within the Stata environment. It offers a software platform comprising various Stata commands to delve into improving the ex-ante evaluation of policy impacts by leveraging statistical outcomes derived from prior ex-post evaluations (Manski, 2004; Dehejia, 2005). Termed as *policy learning* (Athey and Wager, 2021), *optimal policy assignment* (Bhattacharya and Dupas, 2012), or *empirical welfare maximization* (Kitagawa and Tetenov, 2018; Manski, 2004), ex-ante evaluation of policy impacts based on statistical learning from past policies is a focal point for informing policymakers and decision-makers.

Despite recent theoretical advancements in this field, the practical implementation of policy learning algorithms is limited. Consequently, there is a significant dearth of applied analysis with well-defined protocols and a thorough exploration of challenges encountered in real policy scenarios. This area remains considerably neglected, if not entirely unexplored.

This paper aims to address this gap by implementing three popular policy learning algorithms in Stata, corresponding to three common policy classes: *threshold-based*

(TB), *linear-combination* (LC), and *decision tree* (DT).

Beginning with a brief overview of the state-of-the-art in the OPL literature, the paper proceeds to establish the primary statistical foundations of the OPL framework. Subsequently, it provides the algorithm protocol application common to the three policy classes.

After presenting the syntax of the proposed OPL commands, a practical demonstration is provided using the popular LaLonde (1986) training program database.

The final section is dedicated to scenario development, considering issues related to *angle solutions* resulting from the so-called *monotonicity*, which may manifest in practice for certain selection variables chosen by policymakers.

The structure of the paper unfolds as follows: Section 2 provides a concise overview of the current OPL literature. Section 3 outlines the OPL statistical framework, specifically focusing on optimal treatment assignment, and provides the OPL implementation protocol. Section 3 presents the OPL package and the syntax of its commands (see the related subsections). Section 5 delineates and scrutinizes the empirical application by doing this command-by-command. Section 6 discusses scenario building, by stressing the policymaker perspective in terms of choices among a *menu* of alternative choices entailing a trade-off between the size of the policy (i.e., the percentage of treated units) and the achieved level of welfare. Section 7, finally, concludes the paper.

2 A brief account of the OPL literature

The OLP literature, though relatively recent, has witnessed notable contributions, briefly surveyed in what follows, that refers only to the case of a binary treatment.

Kitagawa and Tetenov (2018) employ the semi-parametric inverse-probability weighing estimator, assuming selection-on-observables, revealing that with a known policy propensity score, OPL rules achieve a convergence rate of at least $N^{-1/2}$ to the maximum attainable welfare uniformly. The uniform convergence rate hinges on the richness of decision rule candidates, the distribution of conditional treatment effects, and the absence of knowledge regarding the propensity score. Regrettably, in cases where the propensity score is unknown, this optimal convergence rate is elusive, and only slower rates are attainable.

Athey and Wager (2021) devise an alternate OPL learning algorithm, establishing an optimal regret bound for binary-action policy learning, even in the absence of knowledge regarding the propensity score. They extend their analysis to scenarios where the selection-on-observables assumption falters but instrumental variables are available. Proposing a cross-validated estimation algorithm for empirical welfare, they apply it using a fixed-depth policy tree.

Zhou, Athey, and Wager (2023) extend the findings of Kitagawa and Tetenov (2018) and Athey and Wager (2021) from the binary setting to a multi-action policy setting. Building on the theory of efficient semi-parametric inference, they introduce and imple-

ment a policy learning algorithm that achieves asymptotically minimax-optimal regret. Their work addresses computational challenges arising when implementing the EWM method, particularly when the policy is confined to a decision tree structure.

Additional noteworthy contributions to the literature include Mbakop and Tabord-Meehan (2018), showcasing how to select from a growingly complex collection of policy classes in a data-dependent manner, and Nie et al. (2019), extending the OPL framework to cases where policymakers must decide not only whom to treat but also when to initiate treatment.

By emphasizing the empirical perspective, Cerulli (2023) contributed to the discussion of challenges that arise when implementing optimal policy learning in practical settings. The author concentrates on threshold-based policies, outlining the theoretical foundations of the policy-maker selection problem. Also, a solution is proposed to address common challenges, as *angle* solution problems, that typically emerge in real-world applications of optimal policy assignment.

In the wake of this literature, this paper underscores the policymaker perspective through a practical implementation in Stata of the main OPL algorithms.

3 The OPL statistical framework

Consider an individual's vector of characteristics denoted by X, with Y representing an outcome of interest, and T being a binary treatment $(T = 0, 1)$.[1] A policy assignment rule, denoted as $\mathcal{G}$, is a function that maps X onto T, essentially determining which individuals should be treated or untreated:

$$\mathcal{G} : X \to T$$

The population policy conditional average treatment effect (CATE), denoted here as $\tau(X)$, is defined as the difference between the expected outcomes in the two states of the world (treated vs. untreated) given the control variables X, that is:

$$\tau(X) = E(Y_1|X) - E(Y_0|X)$$

where Y_1 and Y_0 are the potential outcomes, and $E_X[\tau(X)] = \tau$ represents the average treatment effect.

The estimated policy actual total effect, referred to as *welfare* $(\widehat{W})$, is expressed as[2]:

$$\widehat{W} = \sum_{i=1}^{N} T_i \cdot \widehat{\tau}(X_i)$$

1. This section draws on Kitagawa and Tetenov(2018), and Cerulli (2023).
2. For brevity, we directly consider the sample counterparts of the population parameters.

where $\widehat{\tau}(X_i)$ is a consistent estimation of $\tau(X_i)$. Additionally, the estimated policy unconstrained optimal total effect, denoted as $\widehat{W}^*$, is given by:

$$\widehat{W}^* = \sum_{i=1}^{N} \widehat{T}_i^* \cdot \widehat{\tau}(X_i)$$

Here, $\widehat{T}_i^* = \mathbf{1}[\widehat{\tau}(Xi) > 0]$ represents the estimated optimal unconstrained policy assignment. Kitagawa and Tetenov(2018) refers to this as the *first-best* policy rule.

The disparity between the estimated unconstrained maximum achievable welfare and the estimated welfare associated with the implemented policy is termed *regret*, and it is defined as:

$$\widehat{regret} = \widehat{W}^* - \widehat{W}$$

The regret tends to be positive, indicating the inability of decision-makers to select the optimal treatment assignment, especially true in randomized control trials (RCT) where treatment lacks deliberate selection objectives.

Due to various constraints, policymakers often resort to a constrained assignment (T'), as they are unable to implement the optimal unconstrained policy assignment. The resulting welfare, W', may or may not be lower than the unconstrained maximum welfare (W^*). The question arises whether policymakers can achieve the largest viable constrained welfare. Answering this question involves focusing on specific classes of policies and identifying the optimal assignment within them, known as optimal constrained policy assignment. Commonly adopted policy classes include: *threshold-based*, *linear-combination*, and *fixed-depth decision tree* (Kitagawa and Tetenov, 2018). Within these classes of policies, policymakers select crucial variables (e.g., firm size, bank assets, individual age) and specific threshold values to distinguish between treated and untreated units. Below, I consider the three classes separately.

Threshold-based (or quadrant) policy class. Within this policy class, given a single selection variable x with threshold c_x, the estimated assignment to treatment is a function of x and c_x:

$$\widehat{T}_i(x, c_x) = \widehat{T}_i^* \cdot \mathbf{1}[x >= c_x]$$

where $\mathbf{1}[A]$ is an index function taking 1 when A is true and zero otherwise. The corresponding welfare is given by:

$$\widehat{W}(x, c_x) = \sum_{i=1}^{N} \widehat{T}_i(x, c_x) \cdot \widehat{\tau}(X_i)$$

The optimal threshold c_x^* maximizes $\widehat{W}(x, c_x)$ over c_x:

$$c_x^* = \texttt{argmax}_{c_x}[\widehat{W}(x, c_x)]$$

If c_x^* exists, the estimated optimal constrained welfare will thus be equal to $\widehat{W}(c_x^*)$.

Expanding the approach to include more than one selection variable is a straightforward progression. Let's consider a scenario with two selection variables, denoted as x_1 and x_2, each associated with corresponding thresholds, c_1 and c_2. In this instance, the estimated assignment to treatment becomes a function of both thresholds, taking the following form:

$$\widehat{T}_i(c_1, c_2) = \widehat{T}_i^* \cdot \mathbf{1}[x_1 >= c_1] \cdot \mathbf{1}[x_2 >= c_2]$$

This assignment rule, in two dimensions, is termed *quadrant assignment*, as it selects the upper-right quadrant among the four generated by setting the two thresholds.

Linear-combination policy class. Within this policy class, given two selection variables x_1 and x_2, and three weights c_1, c_2, and c_3, the estimated assignment to treatment is:

$$\widehat{T}_i(c_1, c_2, c_3) = \widehat{T}_i^* \cdot \mathbf{1}[c_1 \times x_1 + c_2 \times x_2 >= c_3]$$

The corresponding welfare is given by:

$$\widehat{W}(x, c_1) = \sum_{i=1}^{N} \widehat{T}_i(c_1, c_2, c_3) \cdot \widehat{\tau}(X_i)$$

By denoting $c_x = [c_1, c_2, c_3]$, the optimal threshold c_x^* maximizes $\widehat{W}(x, c_x)$ over c_x:

$$c_x^* = \texttt{argmax}_{c_x}[\widehat{W}(x, c_x)]$$

If c_x^* exists, the estimated optimal constrained welfare will thus be equal to $\widehat{W}(c_x^*)$.

Fixed-depth decision tree. Within this policy class, given two selection variables x_1 and x_2, and three thresholds c_1, c_2, and c_3, the estimated assignment to treatment is:

$$\begin{aligned} \widehat{T}_i(z(1), z(2), z(3), c_1, c_2, c_3) = \widehat{T}_i^* \cdot \{&\mathbf{1}[z(1) >= c_1] \cdot \mathbf{1}[z(2) >= c_2] \\ &+ (1 - \mathbf{1}[z(1) >= c_1]) \cdot \mathbf{1}[z(3) >= c_3]\} \end{aligned} \tag{1}$$

where each $z(j)$ – with $j = 1, 2, 3$ – can be either x_1 and x_2.

The corresponding welfare is given by:

$$\widehat{W}(x, c_1) = \sum_{i=1}^{N} \widehat{T}_i(z(1), z(2), z(3), c_1, c_2, c_3) \cdot \widehat{\tau}(X_i)$$

By denoting $z = [z(1), z(2), z(3)]$ and $c_x = [x_1, x_2, x_3]$, the optimal triplet of variables z^* and thresholds c_x^* maximize $\widehat{W}(z, c_x)$ over z and c_x:

$$[z^*, c_x^*] = \texttt{argmax}_{z, c_x}[\widehat{W}(z, c_x)]$$

If z^* and c_x^* exist, the estimated optimal constrained welfare will thus be equal to $\widehat{W}(z^*, c_x^*)$.

Figure 1 sets out the decision boundary and selection area for the three policy classes (again, *threshold-based*, *linear-combination*, and *fixed-depth decision tree*). We clearly see that the selection area is a quadrant for the the threshold-based, a triangle for the linear-combination, and four adjacent regions for a 2-layer fixed-depth decision tree.

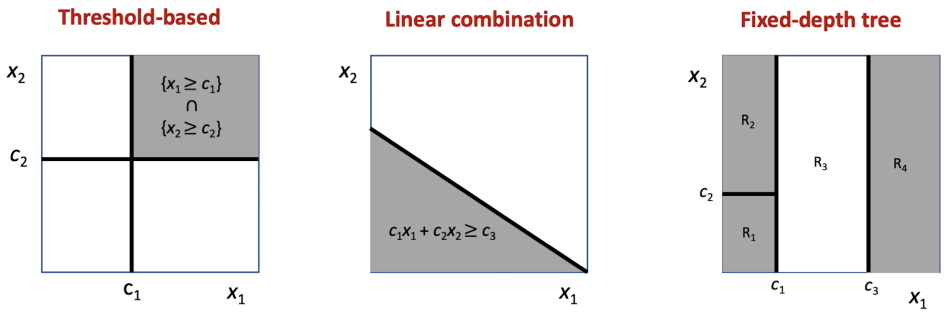

Figure 1: Decision boundary and selection areas for the policy classes: *threshold-based*, *linear-combination*, and *fixed-depth decision tree*.

Seeking optimal thresholds (and variables to split on, for the case of a decision tree) can pose challenges. There is a possibility that, at the optimal values, the share of units to treat might be excessively small or, conversely, too large. In such cases, implementing a policy based on an impractically small or large number of treated units becomes meaningless. To address this issue, policymakers can consider budget constraints, such as the maximum number of units treatable given a specific budget, or predefine a target number of units to treat (e.g., a treatment share between 30% and 40% of the entire reference population). While this approach may lead to a reduction in welfare, it ensures the policy's feasibility.

A related issue arises when there is a monotonic effect of the selection variable(s) on welfare, termed the *angle solution*. For instance, consider the educational attainment of individuals as a selection variable. In many policy contexts, welfare increases monotonically with higher education levels, potentially leading to the selection of individuals with the highest educational attainment. However, this might be impractical for two reasons: the policy aims to target poorly educated individuals, and the number of treated units could become too small as individuals with high educational attainment are generally few. The solution involves the policymaker introducing threshold limits

or predefined ranges of treatment shares to address these constraints. While this may result in a reduction in welfare, it maintains the possibility of eventually implementing the policy.

For optimal assignment implementation, independently of the policy class considered, Table 1 lists a series of steps for finding empirically by *grid search* the optimal selection parameters, i.e. those maximizing the constrained welfare.

Step	**Action**
Step 1 →	Assume having data from a Randomized Control Trial (RCT) or an observational study, consisting of the information triple (Y, X, T) available for every unit involved in the program.
Step 2 →	Run a quasi-experimental method with observable heterogeneity, estimate $\tau(X)$, and compute the (estimated) actual total welfare of the policy $\widehat{W}$.
Step 3 →	Identify the estimated optimal unconstrained policy $\widehat{T}^*$, and compute $\widehat{W}^*$, i.e., the estimated maximum total welfare achievable by the policy, and estimate the regret as $\widehat{W}^* - \widehat{W}$.
Step 4 →	Consider an estimated constrained selection rule $\widehat{T}(x, c_x)$ based on a given set of selection variables, x (two, in our case), and related thresholds/parameters, c_x, and define the estimated maximum constrained welfare as $\widehat{W}(x, c_x)$.
Step 5 →	Build a grid of K possible values for $c_x \in \{c_{x,1}, \ldots, c_{x,K}\}$, compute the optimal vector of thresholds/parameters c_{x,k^*} and the corresponding maximum estimated welfare $\widehat{W}(x, c_{x,k^*})$ thus achieved.

Table 1: Procedure for finding empirically by grid search the optimal selection parameters, i.e. those maximizing the constrained welfare.

In the forthcoming sections, after introducing the syntax of the proposed OPL commands, we will demonstrate the practical application of this procedure on a real dataset pertaining to a specific policy example.

4 The OPL package

OPL is a package for learning optimal policies from data for empirical welfare maximization. Specifically, OPL allows to find *treatment assignment rules* that maximize the overall welfare, defined as the sum of the policy effects estimated over all the policy beneficiaries.

OPL learns the optimal policy empirically, i.e. based on data and observations

obtained from previous (same or similar) implemented policies. OPL carries out empirical welfare maximization within three policy classes: (i) *threshold-based*; (ii) *linear-combination*; and (iii) *2-layer fixed-depth decision-tree*. OPL considers only *two* selection variables chosen by the the policymaker among a set of variables she can control for selection purpose.

Empirical welfare maximization requires the estimation of the Conditional Average Treatment Effect (CATE) of the past policy. Currently, OPL estimates CATE via linear and non-linear Regression Adjustment (RA), allowing for the target outcome to be continuous, binary, count, or fractional. For doing that, OPL employs the built-in Stata command `teffects ra`. The treatment variable of reference must be binary 0/1.

The OPL package relies on the following commands:

- `make_cate`: predicting conditional average treatment effect (CATE) on a *new* policy based on the training over an *old* policy;
- `opl_tb`: implementing threshold-based optimal policy learning;
- `opl_tb_c`: implementing threshold-based policy learning at specific threshold values;
- `opl_lc`: implementing linear-combination optimal policy learning;
- `opl_lc_c`: implementing linear-combination policy learning at specific parameters' values;
- `opl_dt`: implementing 2-layer fixed-depth decision-tree optimal policy learning;
- `opl_dt_c` : implementing 2-layer fixed-depth decision-tree optimal policy learning at specific splitting variables and threshold values.

4.1 Syntax for `make_cate`

`make_cate` is a command generating conditional average treatment effect (CATE) for both a training dataset and a testing (or new) dataset related to a binary (treated vs. untreated) policy program. It provides the main input for running `opl_tb` (optimal policy learning of a threshold-based policy), `opl_tb_c` (optimal policy learning of a threshold-based policy at specific thresholds), `opl_lc` (optimal policy learning of a linear-combination policy), `opl_lc_c` (optimal policy learning of a linear-combination policy at specific parameters), `opl_dt` (optimal policy learning of a decision-tree policy), `opl_dt_c` (optimal policy learning of a decision-tree policy at specific thresholds and selection variables). See Kitagawa and Tetenov (2018) for an outline of the main econometrics supported by these commands.

`make_cate` *outcome features* `, treatment(`*varname* `) model(`*model_type*`)`
`new_cate(`*name*`) train_cate(`*name*`) new_data(`*name*`)`

where *outcome* is a numerical variable and *features* a list of numerical variables representing the features.

Options

`treatment(`*varname*`)` defines the treatment variable adopted in the old (ex-post) policy run. It must be a 0/1 dummy (1=treated; 0=untreated).

`model(`*model_type*`)` indicates the treatment model used for estimating and predicting the conditional average treatment effect (CATE). The implemented estimation methods are linear and non-linear regression adjustment. As *model_type* use the following options: `linear`, if the outcome variable is gaussian (numerical and continuous); `logit`, if the outcome variable is binary (0/1); `poisson`, if the outcome variable is countable; `flogit`, if the outcome variable is fractional.

`new_cate(`*name*`)` indicates by *name* the variable that will be generated containing the prediction over `new_data` of the conditional average treatment effect (CATE).

`train_cate(`*name*`)` indicates by *name* the variable that will be generated containing the prediction over the training dataset of the conditional average treatment effect (CATE).

`new_data(`*name*`)` indicates by *name* the dataset stored in the home directory containing the data of the new policy run (i.e., the features of the would-be beneficiaries).

As returns, `make_cate` provides the following macros:

`e(cate_new)` is the name of the CATE in the new policy data.

`e(cate_train)` is the name of the CATE in the old (training) policy data.

As returns, `make_cate` provides the following variables:

`_train_new_index` is a flag variable indicating the training (i.e., old-policy) and the new-policy observations.

`cate_train` is the variable containing training (i.e., old-policy) predictions for CATE.

`cate_new` is the variable containing new (i.e., new-policy) predictions for CATE.

4.2 Syntax for `opl_tb`

`opl_tb` is a command implementing optimal ex-ante treatment assignment using as policy class a threshold-based (or quadrant) approach.

`opl_tb , xlist(`*var1 var2* `) cate(`*varname*`)`

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides

to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generated using the command `make_cate`.

As returns, `opl_tb` provides the following scalars:

`e(best_c1)` is the threshold which maximizes the welfare over *var1*.

`e(best_c2)` is the threshold which maximizes the welfare over *var2*.

4.3 Syntax for `opl_tb_c`

`opl_tb_c` is a command implementing ex-ante treatment assignment using as policy class a threshold-based (or quadrant) approach at specific threshold values `c1` and `c2` for respectively the selection variables *var1* and *var2*.

`opl_tb_c , xlist(`*var1 var2*`) cate(`*varname*`) c1(`*number*`) c2(`*number*`)` [`,`
`depvar(`*name*`) graph`]

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generated using the command `make_cate`.

`c1(`*number*`)` puts into *number* the value of the threshold value `c1` for the first selection variable. This number must be chosen between 0 and 1.

`c2(`*number*`)` puts into *number* the value of the threshold value `c2` for the second selection variable. This number must be chosen between 0 and 1.

`depvar(`*name*`)` assigns the specified *name* to the dependent variable for display in the results table. While this option does not impact computations, it ensures a meaningful label for the dependent variable in the results table.

`graph` visualizes selected treated and untreated within the *var1* and *var2* quadrant.

As returns, `opl_tb_c` provides the following scalars:

`e(c1)` is the chosen threshold for *var1*.

`e(c2)` is the chosen threshold for *var2*.

`e(W_opt_unconstr)` is the value of the unconstrained welfare at threshold values `c1` and `c2`.

`e(W_opt_constr)` is the value of the constrained welfare at threshold values `c1` and `c2`.

`e(perc_treat)` is the percentage over the entire sample of the beneficiaries to treat at threshold values `c1` and `c2`.

As return, `opl_tb_c` provides the following variable:

`_units_to_be_treated` is a flag variable indicating the policy beneficiaries at threshold values `c1` and `c2`.

4.4 Syntax for `opl_lc`

`opl_lc` is a command implementing optimal ex-ante treatment assignment using as policy class a linear-combination of the selection variables *var1* and *var2*. This class is based on the following linear combination: $\texttt{c1} * \textit{var1} + \texttt{c2} * \textit{var2} = \texttt{c3}$.

`opl_lc , xlist(`*var1 var2*`) cate(`*varname*`)`

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generate using the command `make_cate`.

As returns, `opl_lc` provides the following scalars:

`e(best_c1)` is the linear-combination parameter for *var1* which maximizes the welfare.

`e(best_c2)` is the linear-combination parameter for *var2* which maximizes the welfare.

`e(best_c3)` is the linear-combination parameter which maximizes the welfare.

4.5 Syntax for `opl_lc_c`

`opl_lc_c` is a command implementing ex-ante treatment assignment using as policy class a linear-combination approach at specific parameters' values `c1`, `c2`, and `c3` for the linear-combination of variables *var1* and *var2*. This class is based on the following linear combination: $\texttt{c1} * \textit{var1} + \texttt{c2} * \textit{var2} = \texttt{c3}$.

`opl_lc_c , xlist(`*var1 var2*`) cate(`*varname*`) c1(`*number*`) c2(`*number*`)`
`c3(`*number*`)` [`, depvar(`*name*`) graph`]

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides

to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generate using the command `make_cate`.

`c1(`*number*`)` puts into *number* the value of the linear-combination parameter `c1` for the first selection variable. This number must be chosen between 0 and 1.

`c2(`*number*`)` puts into *number* the value of the linear-combination parameter `c2` for the second selection variable. This number must be chosen between 0 and 1.

`c3(`*number*`)` puts into *number* the value of the parameter `c3` of the linear-combination. This number must be chosen between 0 and 1.

`depvar(`*name*`)` assigns the specified *name* to the dependent variable for display in the results table. While this option does not impact computations, it ensures a meaningful label for the dependent variable in the results table.

`graph` visualizes selected treated and untreated within the *var1* and *var2* quadrant.

As returns, `opl_lc_c` provides the following scalars:

`e(c1)` is the parameter of *var1* in the linear-combination.

`e(c2)` is the parameter of *var2* in the linear-combination.

`e(c3)` is the third parameter of the linear-combination.

`e(W_opt_unconstr)` is the value of the unconstrained welfare at parameters' values `c1`, `c2`, and `c3`.

`e(W_opt_constr)` is the value of the constrained welfare at parameters' values `c1`, `c2`, and `c3`.

`e(perc_treat)` is the percentage over the entire sample of the beneficiaries to treat at linear combination parameters' values `c1`, `c2`, and `c3`.

As return, `opl_tb_c` provides the following variable:

`_units_to_be_treated` is a flag variable indicating the policy beneficiaries at linear combination parameters' values `c1`, `c2`, and `c3`.

4.6 Syntax for `opl_dt`

`opl_dt` is a command implementing optimal ex-ante treatment assignment using as policy class a 2-layer fixed-depth decision-tree based on selection variables *var1* and *var2*.

`opl_dt , xlist(`*var1 var2*`) cate(`*varname*`)`

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generated using the command `make_cate`.

As returns, `opl_dt` provides the following scalars:

`e(best_c1)` is the optimal threshold level of the first variable used for splitting which maximizes the welfare.

`e(best_c2)` is the optimal threshold level of the second variable used for splitting which maximizes the welfare.

`e(best_c3)` is the optimal threshold level of the third variable used for splitting which maximizes the welfare.

As returns, `opl_dt` provides the following local macros:

`e(best_x1)` is the name of the first variable to split on.

`e(best_x2)` is the name of the second variable to split on.

`e(best_x3)` is the name of the third variable to split on.

4.7 Syntax for `opl_dt_c`

`opl_dt_c` is a command implementing ex-ante treatment assignment using as policy class a 2-layer fixed-depth decision-tree at specific splitting variables and threshold values.

`opl_dt_c , xlist(`*var1 var2* `) cate(`*varname*`) c1(`*number*`) c2(`*number*`)`
`c3(`*number*`) x1(`*varname*`) x2(`*varname*`) x3(`*varname*`)` [`, depvar(`*name*`)`
`graph`]

Options

`xlist(`*var1 var2*`)` defines the two variables – *var1* and *var2* – the policymaker decides to use for selecting policy beneficiaries.

`cate(`*varname*`)` puts into *varname* a variable already present in the dataset containing the conditional average treatment effect (CATE). This variable can be generate using the command `make_cate`.

`c1(`*number*`)` puts into *number* the value of the threshold value `c1` for the first splitting variable. This number must be chosen between 0 and 1.

`c2(`*number*`)` puts into *number* the value of the threshold value `c2` for the second splitting

variable. This number must be chosen between 0 and 1.

`c3(`*number*`)` puts into *number* the value of the threshold value `c3` for the third splitting variable. This number must be chosen between 0 and 1.

`x1(`*varname*`)` puts into *varname* the first variable to split on in the decision-tree.

`x2(`*varname*`)` puts into *varname* the second variable to split on in the decision-tree.

`x3(`*varname*`)` puts into *varname* the third variable to split on in the decision-tree.

`depvar(`*name*`)` assigns the specified *name* to the dependent variable for display in the results table. While this option does not impact computations, it ensures a meaningful label for the dependent variable in the results table.

`graph` visualizes selected treated and untreated within the *var1* and *var2* quadrant.

As returns, `opl_dt_c` provides the following scalars:

`e(c1)` is the threshold level of the first variable used for splitting.

`e(c2)` is the threshold level of the second variable used for splitting.

`e(c3)` is the threshold level of the third variable used for splitting.

`e(W_opt_unconstr)` is the value of the unconstrained welfare at threshold values `c1` for the first splitting variable, `c2` for the second splitting variable, and `c3` for the third splitting variable.

`e(W_opt_constr)` is the value of the constrained welfare at threshold values `c1` for the first splitting variable, `c2` for the second splitting variable, and `c3` for the third splitting variable.

`e(perc_treat)` is the percentage over the entire sample of the beneficiaries to treat at decision-tree parameters' values `c1`, `c2`, and `c3`.

As return, `opl_tb_c` provides the following macros:

`e(x1)` contains the name of the first splitting variable.

`e(x2)` contains the name of the second splitting variable.

`e(x3)` contains the name of the third splitting variable.

As return, `opl_tb_c` provides the following variable:

`_units_to_be_treated` is a flag variable indicating the policy beneficiaries at decision tree parameters' values `c1`, `c2`, and `c3` and related splitting variables.

5 Applications

As an illustrative example of the use of the previous commands, I utilize the well-known LaLonde (1986) dataset `jtrain2.dta`, which was employed by Dehejia and Wahba (1999) to assess various propensity-score matching methods in an ex-post policy evaluation. In their investigation, the authors aimed to estimate the impact of participating in a job training program administered in 1976 (indicated by the binary variable `train`, taking the value 1 for treated individuals and 0 for untreated) on real earnings in 1978 (variable `re78`) for a group of individuals in the United States. The dataset comprises a total of 445 observations, with 185 individuals treated and 260 untreated. For the sake of simplicity, I consider a simplified specification of the model.

5.1 Example using `make_cate`

In this first example, we use `make_cate` to predict the conditional average treatment effect (CATE) for a binary *new* policy using training data from an *old* policy. The Stata code is:

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training) and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)

* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"

* Set the selection variables
. global z "age mostrn"

* Run "make_cate" and generate training (old policy) and
* testing (new policy) CATE predictions
. make_cate $y $x , treatment($w) model("linear") new_cate("my_cate_new") ///
train_cate("my_cate_train") new_data("jtrain_test")

Iteration 0:  EE criterion =  2.104e-28
```

```
Iteration 1:  EE criterion =  1.665e-30

Treatment-effects estimation                    Number of obs     =        267
Estimator      : regression adjustment
Outcome model  : linear
Treatment model: none
------------------------------------------------------------------------------
             |               Robust
        re78 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
ATE          |
       train |
   (1 vs 0)  |   .4035484   .7557555     0.53   0.593    -1.077705    1.884802
-------------+----------------------------------------------------------------
POmean       |
       train |
          0  |   4.598974   .4505875    10.21   0.000     3.715839     5.48211
------------------------------------------------------------------------------
```

The command `make_cate` generates the following variables: `svar`, `_train_new_index`, `my_cate_train`, and `my_cate_new`. The variable `svar` takes value 1 for training data and 2 for new data. Similarly, the variable `_train_new_index` is a string variable taking two values, that either "train" or "new", for training and new data respectively. The variable `my_cate_train` contains the values of the CATE estimated in-sample, that is in the training dataset, whereas the variable `my_cate_new` contains the CATE values predicted over the new policy dataset. As main returns, `make_cate` provides the name of the training CATE into the macro `e(cate_train)`, and the name of the CATE in the new policy dataset into the macro `e(cate_new)`. The latter is the basis for implementing the subsequent OPL commands.

5.2 Example using `opl_tb`

Using the same dataset of the previous section, we apply `opl_tb` for optimal ex-ante treatment assignment using as policy class a threshold-based (or quadrant) approach. The main output of this command is to return the threshold which maximizes the welfare over *var1* (i.e., `c1`), and the threshold which maximizes the welfare over *var2* (i.e., `c2`), where *var1* and *var2* are the two selection variables adopted by the policymaker to select the best set of policy beneficiaries. In this case, we use as *var1* the variable `age` (i.e., age of the person), and as *var2* the variable `mostrn` (i.e., months of training). Below we set out the Stata code.

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training)
```

```
* and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)

* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"

* Set the selection variables
. global z "age mostrn"

* Run "make_cate" and generate training (old policy) and
* testing (new policy) CATE predictions

. make_cate $y $x , treatment($w) model("linear") new_cate("my_cate_new") ///
train_cate("my_cate_train") new_data("jtrain_test")

<output omitted>

* Generate a global macro containing the name of the variable "cate_new"
. global T `e(cate_new)'

* Select only the "new data"
. keep if _train_new_index=="new"

* Drop "my_cate_train" as in the new dataset treatment assignment
* and outcome performance are unknown
. drop my_cate_train $w $y

* Run "opl_tb" to find the optimal thresholds
. opl_tb , xlist($z) cate($T)

* Display the optimal threshold values
. di e(best_c1)
.60000002

. di e(best_c2)
.79999999
```

We can see that the optimal (or best) value for `c1` is 0.60, while the optimal (or best) value for `c2` is 0.79.

5.3 Example using `opl_tb_c`

Given the best thresholds `c1` and `c2` estimated in the previous section, we can use `opl_tb_c` to provide the welfare maximization and the actual best individuals to treat in the upcoming new policy round. As said above, `opl_tb_c` is a command implementing ex-ante treatment assignment using as policy class a threshold-based (or quadrant) approach at specific threshold values `c1` and `c2` for respectively the selection variables *var1* and *var2*. We can run `opl_tb_c` at *var1*=`age`, *var2*=`mostrn`, `c1`=0.60, and `c2`=0.79. The code with the main output is displayed below.

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training) and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)

* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"

* Set the selection variables
. global z "age mostrn"

* Run "make_cate" and generate training (old policy) and
* testing (new policy) CATE predictions
. make_cate $y $x , treatment($w) model("linear") new_cate("my_cate_new") ///
  train_cate("my_cate_train") new_data("jtrain_test")

* Generate a global macro containing the name of the variable "cate_new"
. global T ‘e(cate_new)’

* Select only the "new data"
. keep if _train_new_index=="new"
```

```
* Drop "my_cate_train" as in the new dataset treatment
* assignment and outcome performance are unknown
. drop my_cate_train $w $y

* Run "opl_tb" to find the optimal thresholds
. opl_tb , xlist($z) cate($T)

* Save the optimal threshold values into two global macros
. global c1_opt=e(best_c1)
. global c2_opt=e(best_c2)

* Run "opl_tb_c" at optimal thresholds and generate the graph
. opl_tb_c , xlist($z) cate($T) c1($c1_opt) c2($c2_opt) graph depvar("re78")

------------------------------------------------------------------------------
Policy class: Threshold-based

Main results

Learner = Regression adjustment                Target variable = re78
N. of units = 178                              Selection variables = age mostrn
Threshold value c1 = .60000002                 Threshold value c2 = .79999999
Average unconstrained welfare = 2.0673337      Average constrained welfare = 2.885844
Percentage of treated = 1.1                    N. of treated = 2
N. of untreated = 176
------------------------------------------------------------------------------

* Tabulate the variable "_units_to_be_treated"
. tab _units_to_be_treated , mis

1 = unit to |
 treat; 0 = |
unit not to |
     treat  |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        176       98.88       98.88
          1 |          2        1.12      100.00
------------+-----------------------------------
      Total |        178      100.00
```

The main results are reported in the above table. We see that the learner used to learn the policy is the regression adjustment, the target variable is `re78`, the number of units is 178, the employed selection variables are `age` and `mostrn`. Also, we see that the optimal threshold value `c1` is equal to 0.60, while the optimal threshold value `c2` is equal to 0.79. In terms of welfare, the maximum average unconstrained welfare is

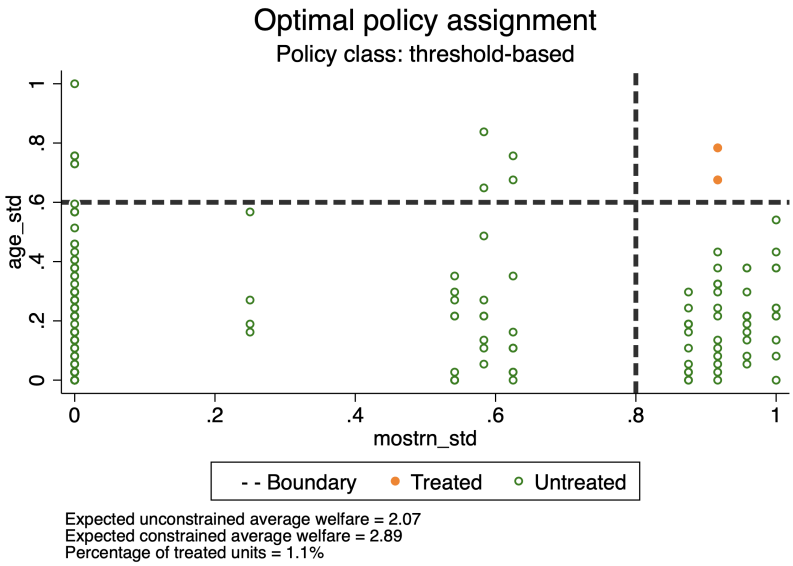

Figure 2: Optimal treatment assignment: Threshold-based policy class.

found to be 2.06, while the maximum average constrained welfare is 2.88. Notably, at these optimal values, the percentage of treated individuals is relatively low at 1.1%, comprising 2 treated units and 176 untreated units.

Finally, Figure 2 presents the graphical representation of the optimal solution within the quadrant frame defined by the two selection variables. The two individuals identified as optimal for treatment are depicted as fully filled points, and the optimal decision boundary is also illustrated.

5.4 Example using `opl_lc` and `opl_lc_c`

For brevity, in this section, we directly examine the output of the `opl_lc_c` command, which utilizes the optimal linear combination parameters provided by `opl_lc`. Here, we focus on optimal ex-ante treatment assignment, employing a policy class represented by a linear combination of the variables `age` and `mostrn`: `c1` $\times$ `age` + `c2` $\times$ `mostrn` = `c3`. The corresponding Stata code is illustrated below.

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training) and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)
```

```
* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"

* Set the selection variables
. global z "age mostrn"

* Run "make_cate" and generate training (old policy) and
* testing (new policy) CATE predictions
. make_cate $y $x , treatment($w) model("linear") new_cate("my_cate_new") ///
  train_cate("my_cate_train") new_data("jtrain_test")

* Generate a global macro containing the name of the variable "cate_new"
. global T ‘e(cate_new)’

* Select only the "new data"
. keep if _train_new_index=="new"

* Drop "my_cate_train" as in the new dataset treatment assignment
* and outcome performance are unknown
. drop my_cate_train $w $y

* Run "opl_lc" to find the optimal linear-combination parameters
. opl_lc , xlist($z) cate($T)

* Save the optimal linear-combination parameters into three global macros
. global c1_opt=e(best_c1)
. di $c1_opt
.59999999

. global c2_opt=e(best_c2)
. di $c2_opt
.45000001

. global c3_opt=e(best_c3)
. di $c3_opt
.8
```

```
* Run "opl_lc_c" at optimal linear-combination parameters and generate the graph
. opl_lc_c , xlist($z) cate($T) c1($c1_opt) c2($c2_opt) c3($c3_opt) ///
  graph depvar("re78")

------------------------------------------------------------------------------
Policy class: Linear-combination

Main results

Learner = Regression adjustment                Target variable = re78
N. of units = 178                              Selection variables = age mostrn
Lin. comb.parameter c1 = .59999999             Lin. comb.parameter c2 = .45000001
Lin. comb.parameter c3 = .8                    Average unconstrained welfare = 2.07
Average constrained welfare = 2.885844         Percentage of treated = 1.1
N. of treated = 2                              N. of untreated = 176
------------------------------------------------------------------------------

* Tabulate the variable "_units_to_be_treated"
. tab _units_to_be_treated , mis

1 = unit to |
 treat; 0 = |
unit not to |
      treat |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        176       98.88       98.88
          1 |          2        1.12      100.00
------------+-----------------------------------
      Total |        178      100.00
```

The initial segment of the provided code mirrors the one utilized for `opl_tb`. However, deviations occur upon executing `opl_lc`, which yields three optimal parameters stored in the scalars `e(best_c1)` (0.59), `e(best_c2)` (45), and `e(best_c3)` (0.8).

Subsequently, I execute the code with the aforementioned optimal linear-combination parameters and generate the optimal beneficiaries' graph. In this scenario, the average unconstrained welfare stands at 2.07, while the average constrained welfare is 2.88. The percentage of treated individuals remains consistent at 1.1%, with 2 treated and 176 untreated individuals.

Figure 3 shows the optimal beneficiaries of the prospective new policy round. As in the case of the threshold-based policy, only two people are selected.

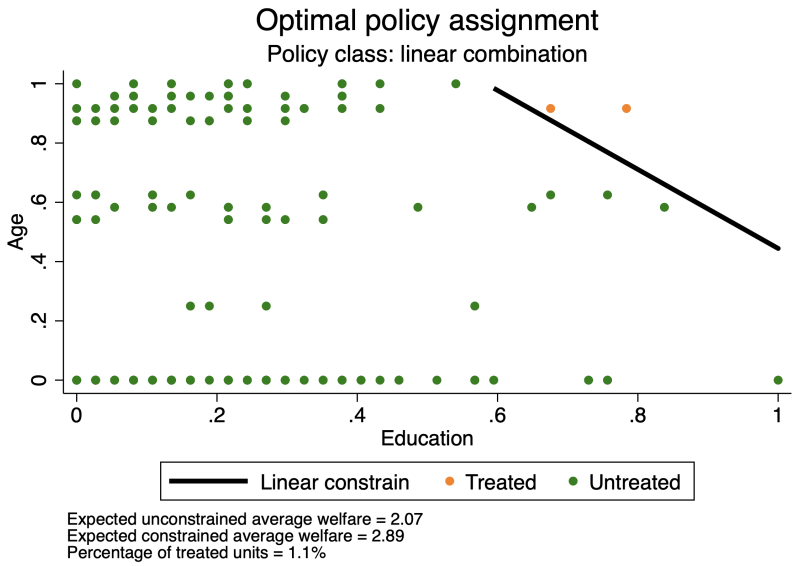

Figure 3: Optimal treatment assignment: Linear-combination policy class.

5.5 Example using `opl_dt` and `opl_dt_c`

In this section, I consider optimal ex-ante treatment assignment using as policy class a 2-layer fixed-depth decision-tree, at optimal specific splitting variables and optimal threshold values. The Stata code is displayed below.

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training) and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)

* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"
```

```
* Set the selection variables
. global z "age mostrn"

* Run "make_cate" and generate training (old policy)
* and testing (new policy) CATE predictions
. make_cate $y $x , treatment($w) model("linear") new_cate("my_cate_new") ///
  train_cate("my_cate_train") new_data("jtrain_test")

* Generate a global macro containing the name of the variable "cate_new"
. global T `e(cate_new)'

* Select only the "new data"
. keep if _train_new_index=="new"

* Drop "my_cate_train" as in the new dataset treatment
* assignment and outcome performance are unknown
. drop my_cate_train $w $y

* Run "opl_dt" to find the optimal linear-combination parameters
. opl_dt , xlist($z) cate($T)

* Save the optimal splitting variables into three global macros
. global x1_opt `e(best_x1)'
. global x2_opt `e(best_x2)'
. global x3_opt `e(best_x3)'

* Save the optimal splitting thresholds into three global macros
. global c1_opt=e(best_c1)
. global c2_opt=e(best_c2)
. global c3_opt=e(best_c3)

* Run "opl_dt_c" at optimal splitting variables and
* corresponding thresholds and generate the graph
. opl_dt_c , xlist($z) cate($T) c1($c1_opt) c2($c2_opt) c3($c3_opt) ///
  x1($x1_opt) x2($x2_opt) x3($x3_opt) graph depvar("re78")

------------------------------------------------------------------------------
Policy class: Fixed-depth decision-tree

Main results

Learner = Regression adjustment                Target variable = re78
N. of units = 178                              Selection variables = age mostrn
Threshold first splitting var. = .69999999     Threshold second splitting var. = .89
```

```
Threshold third splitting var. =  = .60000002  Average unconstrained welfare = 2.06
Average constrained welfare = 4.2417823        Percentage of treated = 1.7
N. of treated = 3                              N. of untreated = 175
First splitting variable x1 = age              Second splitting variable x2 = age
Third splitting variable x3 = age
------------------------------------------------------------------------------

* Tabulate the variable "_units_to_be_treated"
. tab _units_to_be_treated , mis

1 = unit to |
 treat; 0 = |
unit not to |
      treat |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        175       98.31       98.31
          1 |          3        1.69      100.00
------------+-----------------------------------
      Total |        178      100.00
```

The initial segment of the code aligns with the structure used for both the threshold-based and linear-combination policy classes. However, when employing the decision-tree policy class, distinctions arise upon executing `opl_dt`. This command yields three optimal splitting variables (stored as local macros: `e(best_x1)`, `e(best_x2)`, and `e(best_x3)`), along with the optimal thresholds for each variable (stored as scalars: `e(best_c1)`, `e(best_c2)`, and `e(best_c3)`).

The core of the code lies in the execution of `opl_dt_c`, taking as arguments the primary returns from `opl_dt`. Examining the results' table, it is evident that the first optimal splitting variable is `age`, the second is `age`, and the third is `age` once again.

The thresholds for these splitting variables are 0.69, 0.89, and 0.60, respectively. The resulting average unconstrained welfare is 2.06, the average constrained welfare is 4.24, and the percentage of treated individuals is 1.7%, corresponding to 3 treated units and 175 untreated units.

Finally, figure 4 shows the optimal beneficiaries of the prospective new policy round when the decision-tree policy class is employed. As said, in this case, it is optimal to treat three people.

6 Scenario building

Optimal policy learning can be conceived as a specialized form of pre-implementation policy impact assessment, focusing on evaluating the potential consequences of a policy before its enactment. Within this framework, scenario building becomes pivotal, involving the creation and analysis of diverse future scenarios to anticipate various potential

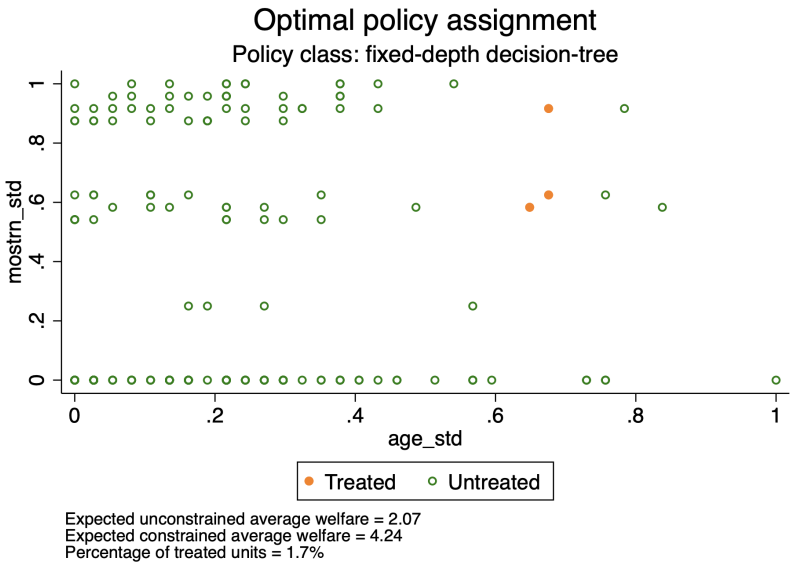

Figure 4: Optimal treatment assignment: Fixed-depth decision tree policy class.

policy effects. This method offers a systematic approach to comprehend and prepare for the potential impacts of policy decisions under different circumstances.

In the context of an OPL exercise, scenario building proves particularly valuable due to the potential presence of *angle* solutions arising from the monotonic nature of selection variable(s). Monotonicity occurs when the welfare consistently increases or decreases as at least one of the selection variables increases or decreases. For instance, in certain social education programs, maximizing empirical welfare might lead to selecting individuals with the highest educational attainment. This scenario is undesirable, as it would result in only individuals with the highest education level being chosen as policy beneficiaries, potentially comprising a small set of individuals, or even a single individual. Conversely, an optimal solution might imply treating everyone, which is either impractical or undesirable.

A viable solution to such issues is the adoption of a *menu* strategy Cerulli (2023). In a nutshell, this strategy involves the analyst considering different values of the selection variable affected by monotonicity, computing and presenting alternative situations in terms of welfare obtained, percentage of beneficiaries, etc. Consequently, policymakers can make decisions based on a range of choices, weighing the pros and cons of each potential selection strategy.

Typically, alternative scenarios with distinct policy parameters involve a trade-off between the magnitude of the policy effect and the number of units to be treated. For example, for a threshold-based policy class, as the threshold of the selection variable

affected by monotonicity increases, the average welfare tends to increase as well. Within budget constraints, policymakers can strategically choose from these scenarios, aligning with their objectives for an optimal pre-implementation re-programming of policy treatment assignments.

The following Stata code outlines scenario building using the OPL package, with a focus on the threshold-based policy class for the sake of brevity. The code unfolds as follows:

```
* Load initial dataset
. sysuse JTRAIN2, clear

* Split the original data into a "old" (training) and "new" (testing) dataset
. get_train_test, dataname(jtrain) split(0.60 0.40) split_var(svar) rseed(101)

* Use the "old" dataset (i.e. policy) for training
. use jtrain_train , clear

* Set the outcome
. global y "re78"

* Set the features
. global x "re74 re75 age agesq nodegree"

* Set the treatment variable
. global w "train"

* Set the selection variables
. global z "age educ"

* Run "make_cate" and generate training (old policy)
* and testing (new policy) CATE predictions
. qui make_cate $y $x , treatment($w) model("linear")  ///
new_cate("my_cate_new") train_cate("my_cate_train") new_data("jtrain_test")

* Generate a global macro containing the name of the variable "cate_new"
. global T ‘e(cate_new)’

* Select only the "new data"
. keep if _train_new_index=="new"

* Drop "my_cate_train" as in the new dataset treatment
* assignment and outcome performance are unknown
. drop my_cate_train $w $y

* Run "opl_tb" to find the optimal thresholds
```

```
. opl_tb , xlist($z) cate($T)

* Save the optimal threshold values into two global macros
. global c1_opt=e(best_c1)
. global c2_opt=e(best_c2)

* Mute graphs
. set graph off

* Generate a grid of thresholds for "educ" with "age" set to optimal threshold
. global R1 "$c1_opt" // age
. global R2 "0.3 0.4 0.5 0.6 0.7 0.9" // educ

* Run "opl_tb_c" by looping over R1 and R2, and save the graphs
. local j=1
foreach k of global R1{
foreach h of global R2{
opl_tb_c , xlist($z) cate($T) c1(‘k’) c2(‘h’) graph depvar("re78")
graph save  Graph G_‘j’.gph , replace
local j=‘j’+1
}
}

* Put the graphs names into a global macro named "G"
. global K: word count $R2
. global G ""
forvalues k = 1/$K{
global G $G G_‘k’.gph
}

* Unmute the graphs
. set graph on

* Combine the graphs into a unique scenario graph
. graph combine $G , iscale(*0.8)

* Export the scenario graph
. graph export ScenarioGraph.png , as(png) replace
```

The code begins with the loading of the initial dataset. To ensure meaningful model evaluation, the original data is then strategically partitioned into two subsets: the *old* dataset designated for policy training purposes, and the *new* dataset earmarked for testing.

The training process unfolds by leveraging the old dataset, also referred to as the policy training dataset. In this phase, critical components are established, including

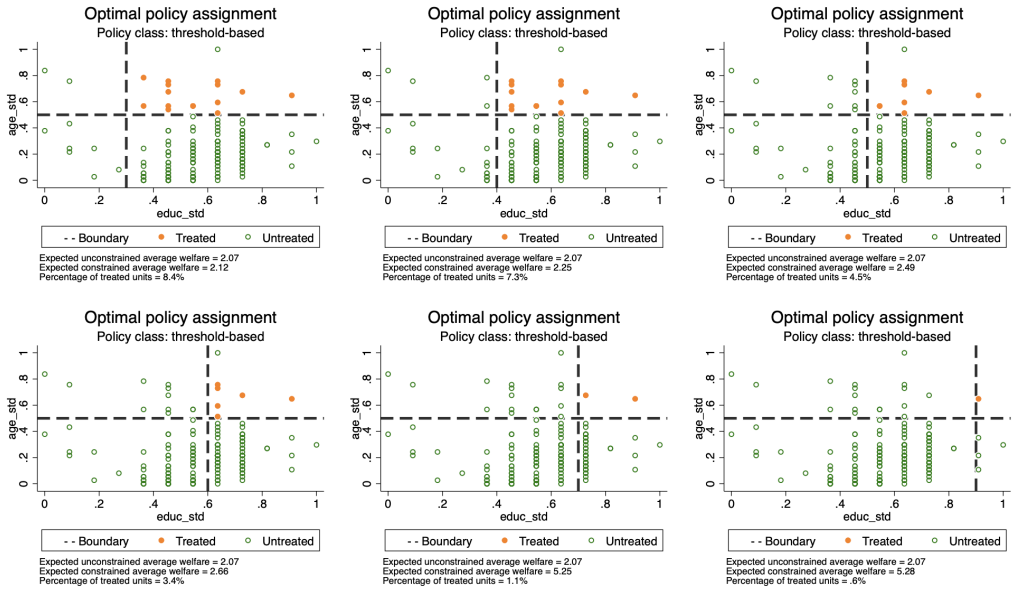

Figure 5: Scenario analysis: threshold-based policy class.

the specification of the desired outcome, identification of pertinent features, and the definition of the treatment variable. Concurrently, the selection variables are set as the tools the policy maker can maneuver.

As seen in previous examples, a pivotal step in the process involves the execution of the `make_cate` command, generating both training and testing Conditional Average Treatment Effects (CATE) predictions. These predictions are encapsulated within a global macro, featuring the variable name `cate_new`.

The focus then narrows down to the *new* data subset, where the variable `my_cate_train` is systematically dropped.

To optimize the model's threshold, the `opl_tb` command is executed, leading to the identification of the optimal threshold values. These values are stored in two global macros for subsequent use.

The subsequent stage involves the creation of a threshold grid for the variable `educ`, the selection variable affected by monotonicity, with the optimal threshold for `age` being assumed. This grid sets the stage for the implementation of `opl_tb_c`, where a loop over the grids `R1` and `R2` is executed to produce insightful graphs. These graphs are assigned names, consolidated within a global macro named `G`.

The final stride in this code involves the amalgamation of individual scenario graphs into a comprehensive and visually impactful representation. This is the main result of

this exercise.

Examining Figure 5, we can discern interesting patterns. The `age` threshold, as said, is strategically positioned at its optimal level, and each graph shows the dynamics concerning welfare and the percentage of treated individuals as the (standardized) variable `educ` increases from 0.3 to 0.9. Notably, a clear trade-off emerges between welfare level and the percentage of treated individuals: at the minimum level of `educ`, welfare reaches its minimum, and the percentage of treated units its maximum; conversely, welfare reaches its maximum at the largest value of `educ`, whereas the proportion of treated units exhibits its minimum in this case. This underscores the presence of monotonicity concerning the selection variable `educ`, as discussed above.

In essence, the figure functions as a comprehensive menu for policymakers, offering a spectrum of alternative choices. Policymakers can make selections based on their preferences regarding the number of treated individuals and the achieved welfare. Importantly, these choices may extend beyond the immediate context and involve considerations such as budgetary constraints or the overall scale of the policy that the policymaker intends to implement. Thus, the figure serves as a valuable tool for decision-makers to align policy choices with their overarching goals and constraints.

7 Conclusion

This article introduced the Stata package `OPL`, dedicated to facilitating *optimal policy learning* within the Stata environment. The significance of ex-ante evaluation of policy impacts, informed by econometric outcomes derived from prior ex-post evaluations, is highlighted as a crucial aspect of policymaking. Referred to as *policy learning*, *optimal policy assignment*, or *empirical welfare maximization*, this approach addresses the need for evidence-based ex-ante decision-making.

Despite theoretical advancements in this domain, the practical implementation of policy learning algorithms has been limited, leaving a notable gap in applied analysis. This paper aimed to bridge this void by presenting three widely-used policy learning algorithms coded in Stata, aligning with three common policy classes: threshold-based (TB), linear-combination (LC), and decision tree (DT).

Beginning with an overview of the OPL literature, the paper established the statistical foundations of the OPL framework and outlined the implementation protocol for the three policy classes. The proposed OPL commands' syntax was presented, followed by practical demonstrations using an illustrative training program database.

The final part addressed scenario development, emphasizing *angle solutions* resulting from welfare *monotonicity*. This phenomenon, observed for specific selection variables chosen by policymakers, was discussed in the context of real-world policy scenarios.

The provided detailed exploration of the OPL algorithms, along with the empirical application, underscored the importance of evidence-based ex-ante policy evaluations in informing effective and impactful policy-making. By means of the `OPL` package, this

task can be effectively accomplished using Stata.

8 References

[1] Athey, S., & Wager, S. (2021). Policy Learning with Observational Data. *Econometrica*, 89(1), 133–161.

[2] Bhattacharya, D., & Dupas, P. (2012). Inferring Welfare Maximizing Treatment Assignment under Budget Constraints. *Journal of Econometrics*, 167(1), 168-196.

[3] Cerulli, G. (2023). Optimal treatment assignment of a threshold-based policy: empirical protocol and related issues. *Applied Economics Letters*, 30(8), 1010-1017.

[4] Dehejia, R. & Wahba, S. (1999). Causal Effects in Non-Experimental Studies: Re-Evaluating the Evaluation of Training Programs. *Journal of the American Statistical Association*, 94(448), 1053-1062.

[5] Dehejia, R. (2005). Program Evaluation as a Decision Problem. *Journal of Econometrics*, 125(1-2), 141-173.

[6] Kitagawa, T., & Tetenov, A. (2018). Who Should Be Treated? Empirical Welfare Maximization Methods for Treatment Choice. *Econometrica*, 86(2), 591–616.

[7] LaLonde, R. (1986). Evaluating the Econometric Evaluations of Training Programs. *American Economic Review*, 76(4), 604-620.

[8] Manski, C. F. (2004). Statistical Treatment Rules for Heterogeneous Populations. *Econometrica*, 72(4), 1221-1246.

[9] Mbakop, E., & Tabord-Meehan, M. (2018). Model Selection for Treatment Choice: Penalized Welfare Maximization. *arXiv preprint*, arXiv:1609-03167.

[10] Nie, X., Brunskill, E., & Wager, S. (2019). Learning When-to-Treat Policies. *arXiv preprint*, arXiv:1905.09751.

[11] Zhou, Z., Athey, S., & Wager, S. (2023). Offline Multi-Action Policy Learning: Generalization and Optimization. *Operations Research*, 71(1).

About the authors

Giovanni Cerulli is a senior researcher at the CNR-IRCrES, Research Institute on Sustainable Economic Growth, National Research Council of Italy, Rome. His research interest is in applied econometrics, with a special focus on causal inference and machine learning. He has developed original causal inference models and provided several implementations. He is currently editor in chief of the *International Journal of Computational Economics and Econometrics*.