




## Policy Learning with the polle Package

Andreas Nordland   
University of Copenhagen

Klaus Kähler Holst   
Novo Nordisk

---

### Abstract

The R package **polle** is a unifying framework for learning and evaluating finite stage policies based on observational data. The package implements a collection of existing and novel methods for causal policy learning including doubly robust restricted Q-learning, policy tree learning, and outcome weighted learning. The package deals with (near) positivity violations by only considering realistic policies. Highly flexible machine learning methods can be used to estimate the nuisance components, and valid inference for the policy value is ensured via cross-fitting. The library is built up around a simple syntax with four main functions `policy_data()`, `policy_def()`, `policy_learn()`, and `policy_eval()`, which are used to specify the data structure, define user-specified policies, specify policy learning methods, and evaluate (learned) policies. The functionality of the package is illustrated via extensive reproducible examples.

*Keywords:* policy learning, dynamic treatment regimes, semiparametric inference, double machine learning, R.

---

## 1. Introduction

Sequential decision problems arise in various fields. Important examples include deciding on treatment assignments in a medical application, defining equipment maintenance strategies in a military or industrial setting, or determining a sales strategy in a commercial context. Policy learning seeks to identify sequential decision strategies from observational data and to quantify the effect of implementing such a strategy using causal inference techniques. While the theoretical field has progressed substantially during the last decade based on advances in semiparametric methods, there has been a large gap in terms of generic implementations of these methods being available to practitioners.

The R package **polle** (Nordland and Holst 2026) is a unifying framework for learning optimal policies/dynamic treatment regimes from historical data based on cross-fitted doubly robust loss functions for finite horizon problems with discrete action sets. Within this scope, the

package unifies available methods from other R packages and introduces previously unavailable methods. The performance of the methods can then easily be evaluated, compared and applied to new data. As a unique feature, the package also handles a stochastic number of decision stages. In addition, the package deals with issues related to learning optimal policies from observed data under (near) positivity violations by considering *realistic policies*.

The core concept of **polle** is to use doubly robust scores/double machine learning developed from semiparametric theory when estimating the value of a policy (Robins 1986; Chernozhukov *et al.* 2018). These scores are also used to construct a doubly robust loss function for the optimal policy value (Tsiatis, Davidian, Holloway, and Laber 2019). The resulting loss function is the basis for policy value search within a restricted class of policies such as policy trees (Athey and Wager 2021). Transformations of the value loss function lead to a range of other loss functions and methods such as doubly robust  $Q$ -learning (Luedtke and Van der Laan 2016b) and outcome weighted learning based on support vector machines (Zhang, Tsiatis, Davidian, Zhang, and Laber 2012; Zhao, Zeng, Rush, and Kosorok 2012). As is customary within targeted learning and double machine learning, our policy evaluation and policy learning methods apply cross-fitting schemes, which allow for inference under weak conditions even when nuisance parameters are learned from highly flexible machine learning methods.

Recursive policy learning is closely related to estimating heterogeneous causal effects via the conditional average treatment effect; see Kennedy (2023) and Semenova and Chernozhukov (2021) for recent overviews of the field and some of the challenges regarding inference and generic error bounds. Other notable mentions include Künzel, Sekhon, Bickel, and Yu (2019) and Athey, Tibshirani, and Wager (2019). Variable importance measures formulated as projections are also closely related to policy learning; see Van der Laan (2006).

The **polle** package is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=polle>. Other packages which should be highlighted include **DynTxRegime** (Holloway, Laber, Linn, Zhang, Davidian, and Tsiatis 2025) which provides methods for estimating policies including interactive  $Q$ -learning, outcome weighted learning, and value search. However, most of the methods are only implemented for single-stage problems and the package has no cross-fitting methods for consistent policy evaluation. The **polle** package wraps efficient augmentation and relaxation learning and residual weighted learning from the **DynTxRegime** package. The package **policytree** (Sverdrup, Kanodia, Zhou, Athey, and Wager 2020, 2024) is an implementation of single-stage policy tree value search based on doubly robust scores. The **polle** package wraps this functionality and extends it to a stochastic number of stages. A related R package is **grf** (Tibshirani, Athey, Sverdrup, and Wager 2025), which implements causal forests for the conditional average treatment effect. Lastly, the R package **DTRlearn2** (Chen, Liu, Zeng, and Wang 2020) implements outcome weighted learning in a fixed number of stages. The **polle** package also wraps this functionality. Beyond R, the Python (Van Rossum *et al.* 2011) package **EconML** (Battocchi *et al.* 2019) implements a wide range of learners for the conditional average treatment effect including doubly robust estimators (equivalent to doubly robust  $Q$ -learning as formulated in **polle**), double machine learning estimators, and orthogonal random forests. The package also implements policy trees and forests. To our knowledge, **EconML** does not contain methods for cross-fitted policy evaluation. For multi-stage decision problems, the package only considers G-estimation based on specific Markov decision process structural equation models; see Lewis and Syrkanis (2021).

The available methods for policy learning in proprietary software are still very limited. A SAS (SAS Institute Inc. 2013) macro denoted PROC QLEARN performs standard  $Q$ -learning (Ertefaie, Almirall, Huang, Dziak, Wagner, and Murphy 2012). In Stata (StataCorp 2021), methods are limited to estimating average treatment effects and potential outcome means with the `teffects` function.

This article is organized as follows. In Section 2, we introduce the most important concepts of doubly robust policy learning in a simple single-stage setting. In doing so, we avoid the cumbersome notation needed for the general sequential setup as presented in Section 3. In Section 4, we give an overview of the package syntax and describe the main functions of the package. Section 5 contains four reproducible examples based on simulated data covering all aspects of the package. In Section 6, we present a complete analysis of a dataset investigating the treatment effect of a literacy intervention. Finally, in Section 7, we summarize the functionality of `polle` and discuss limitations and future developments. Examples involving random forest learners may exhibit minor differences across platforms. All results reported in this article were produced using R 4.5.2 on macOS Tahoe 26.3 (aarch64-apple-darwin25.1.0).

## 2. Concepts

In a randomized trial investigating the average treatment effect of two competing treatments, we should ask ourselves whether the treatment effect is heterogeneous or not, i.e., whether the subjects respond differently to the treatments depending on their age, sex, disease history, etc. If so, is it possible to learn a treatment policy from the observed data that will have a greater expected outcome than any of the individual treatments?

For simplicity, we consider a single-stage problem, where each subject receives a completely randomized treatment at a single time point. Let  $A$  denote the treatment variable with two levels,  $A \in \{0, 1\}$ , and let  $U$  denote the measured utility outcome. The average treatment effect is a causal parameter which can be formulated via potential outcomes (Rubin 1974; Hernán and Robins 2020). We let  $U^a$  denote the potential utility had we forced the subject to receive treatment  $A = a$ , and we refer to  $\mathbb{E}[U^a]$  as the value of the given treatment. The average treatment effect is now defined as the difference in value,  $\mathbb{E}[U^1 - U^0]$ , and due to complete randomization, the effect is identified as  $\mathbb{E}[U | A = 1] - \mathbb{E}[U | A = 0]$ . The effect is easily estimated based on a sample of  $N$  independent and identically distributed (iid) observations  $O = (A, U)$ . In the following, we will assume that treatment  $A = 1$  is recommended if the estimated effect is positive and vice versa.

Suppose now that we also collect a set of baseline covariates  $H \in \mathcal{H}$  for each subject, and that treatment randomization depends on this history by design. The treatment probability model is then given by a known function

$$g_0(h, a) = \mathbb{P}(A = a | H = h).$$

If the trial has a sensible design, we will also know that  $g_0(H, a) > 0$  almost surely for  $a \in \{0, 1\}$ , which is commonly referred to as the positivity condition. Due to confounding, the average treatment effect will no longer be identified by the mean utility in each treatment group; see Figure 1. However, it is possible to show that

$$\mathbb{E} \left[ \frac{I\{A = a\}}{g_0(H, a)} U \right] = \mathbb{E}[U^a]. \quad (1)$$

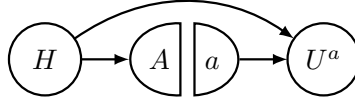


Figure 1: Single world intervention graph illustrating confounding via the history.

This equality inspires an inverse probability weighting estimator for the value of each treatment group. In an observational study, the treatment probability function  $g_0$  might not be known a priori. In that case, we instead use some appropriate estimate  $g_N$ .

Alternatively, the treatment value is identified as

$$\mathbb{E}[\mathbb{E}[U \mid A = a, H]] = \mathbb{E}[U^a], \quad (2)$$

where we define the quality function as  $Q_0(h, a) = \mathbb{E}[U \mid A = a, H = h]$ . Usually, the  $Q$ -function is not known a priori and will need to be estimated. The fit  $Q_N$  is then used to construct an outcome regression estimate of the value in each treatment group; see [Robins \(1986\)](#).

If the treatment effect is heterogeneous across the collected history, it is possible that one group of subjects benefits from treatment  $A = 1$  and that another group of subjects benefits from treatment  $A = 0$ . The researcher may even have a candidate treatment policy  $d : \mathcal{H} \rightarrow \{0, 1\}$  that is believed to improve the value. Let  $U^d$  denote the potential utility had we forced the subject to be treated in accordance to policy  $d$ . The policy value  $\mathbb{E}[U^d]$  can then be estimated using Equation 1 or 2, or a combination of the two. Define the doubly robust policy score as

$$Z(d, g, Q)(O) = Q(H, d(H)) + \frac{I\{A = d(H)\}}{g(H, A)} (U - Q(H, A)). \quad (3)$$

If either  $g = g_0$  or  $Q = Q_0$ , it holds that

$$\mathbb{E}[Z(d, g, Q)(O)] = \mathbb{E}[U^d]. \quad (4)$$

The associated empirical mean plug-in estimator of the policy value

$$\theta_N = N^{-1} \sum_{i=1}^N Z(d, g_N, Q_N)(O_i)$$

is said to be doubly robust. This estimator is a central component of the **polle** package and is implemented in the function `policy_eval()`.

Furthermore, it is possible to show that the doubly robust policy value estimator is asymptotically efficient, if the nuisance models ( $g_N$  and  $Q_N$ ) are correctly specified; see [Van der Laan and Robins \(2003\)](#). Specifically, the centralized score

$$Z(d, g, Q)(O) - \mathbb{E}[Z(d, g, Q)(O)]$$

is the efficient influence function from which we can derive the asymptotic distributions via central limit theorem arguments. Specifically,

$$N^{-1} \sum_{i=1}^N \{Z(d, g_N, Q_N)(O_i) - \theta_N\}^2$$

is a consistent estimate of  $\text{VAR}[U^d]$ . For a recent review of influence functions, see [Hines, Dukes, Diaz-Ordaz, and Vansteelandt \(2022\)](#).

By applying cross-fitting (/sample splitting) of the nuisance models in combination with the doubly robust score, the nuisance models can be estimated using flexible machine learning methods without causing asymptotic bias ([Chernozhukov et al. 2018](#)). This functionality is also implemented in `policy_eval()`. In Section 3.2 we present the estimating procedure of the policy value in detail and generalize it to multi-category policies over multiple stages.

In many situations, the aim of the researcher is not just to evaluate a given policy, but to learn the optimal policy from the data. In **polle**, this functionality is implemented in the function `policy_learn()`. The optimal treatment policy  $d_0$  is defined as the policy for which it holds that  $\text{E}[U^d] \leq \text{E}[U^{d_0}]$  for all other policies  $d$ . Thus, a direct approach to policy learning is to use Equation 4 as a loss function and perform value search:

$$\begin{aligned} d_N &= \arg \min_{d \in \mathcal{D}} \sum_{i=1}^N \tilde{L}(d)(g_N, Q_n)(O_i) \\ &= \arg \min_{d \in \mathcal{D}} (-1) \sum_{i=1}^N Z(d, g_N, Q_n)(O_i). \end{aligned}$$

In practice, the complexity of the class of candidate policies  $\mathcal{D}$  is bounded for the value search to be viable. Examples include threshold policies and policy trees; see [Athey and Wager \(2021\)](#). `policy_tree(type = "pt1")` provides a wrapper for policy tree learning in **polle**. In our experience, scalability and flexibility of the current policy tree implementation can be an issue. In the first part of Section 3.3, we present the methodology in detail and generalize it to multiple stages.

A key and rather intuitive result is that the optimal policy is also identified as

$$\begin{aligned} d_0(h) &= \arg \max_{a \in \{0, 1\}} \text{E}[U^a \mid H = h] \\ &= \arg \max_{a \in \{0, 1\}} Q_0(h, a). \end{aligned} \tag{5}$$

This result motivates  $Q$ -learning, which relies on estimating the  $Q$ -function. The fitted  $Q$ -function is then plugged into Equation 5 to obtain the associated estimated policy. The problem with  $Q$ -learning is that we put too much faith in our ability to model the  $Q$ -function. Due to confounding, in order to identify the causal optimal policy, we have to input the complete history  $H$  and estimate the  $Q$ -function consistently. However, like value search, in the non-trivial case where  $H$  is not discrete, it is impossible to construct asymptotically reasonable estimators of the  $Q$ -function without restricting the complexity of the class of candidate functions ([Semenova and Chernozhukov 2021](#); [Kennedy 2023](#); [Luedtke and Chung 2024](#); [Nie and Wager 2021](#)). In a sense,  $Q$ -learning combines two problems, a causal estimation problem and a policy estimation problem, which are hard to solve at the same time.

Luckily, it is possible to decompose the two problems via the doubly robust score. In the same process, it is also possible to restrict the input to the policy that we want to optimize. This is useful if we do not want to collect the full history in a future implementation or if we want to exclude variables which are unethical to collect or use as input to the policy.

For the purpose of restricting the input to the policy, let  $V \in \mathcal{V}$  be a subset or a function of the history  $H$ . Let  $d^V : \mathcal{V} \rightarrow \{0, 1\}$  denote a  $V$ -restricted policy. Finally, let  $\mathcal{D}^V$  denote the

class of  $V$ -restricted policies. The optimal  $V$ -restricted policy is simply defined as the policy  $d_0^V$  for which it holds that  $\mathbb{E}[U^{d^V}] \leq \mathbb{E}[U^{d_0^V}]$  for all  $d^V \in \mathcal{D}^V$ . Similarly as above, the optimal  $V$ -restricted policy is given by

$$\begin{aligned} d_0^V(v) &= \arg \max_{a \in \{0,1\}} \mathbb{E}[U^a \mid V = v] \\ &= \arg \max_{a \in \{0,1\}} QV_0(v, a) \\ &= I\{QV_0(v, 1) - QV_0(v, 0) > 0\}; \end{aligned} \tag{6}$$

see [Luedtke and Van der Laan \(2016b\)](#). The  $QV$ -function is identified in two ways:

$$\begin{aligned} QV_0(V, a) &= \mathbb{E}\left[\frac{I\{A = a\}}{g_0(H, a)}U \mid V\right] \\ &= \mathbb{E}[Q_0(H, a) \mid V]. \end{aligned}$$

Again, the nuisance models can be combined to create a doubly robust expression for the optimal  $V$ -restricted policy. Define  $Z(a, g, Q)$  as  $Z(d, g, Q)$  from Equation 3 under the static policy  $d(H) = a$ . If either  $g = g_0$  or  $Q = Q_0$ , it holds that

$$\mathbb{E}[Z(a, g, Q)(O) \mid V] = QV_0(V, a).$$

This result directly inspires a doubly robust regression type estimator for the  $QV$ -function. Specifically, we let  $QV_N$  denote the function with the lowest empirical mean squared error loss:

$$\begin{aligned} QV_N(\cdot, a) &= \arg \min_{QV} \sum_{i=1}^N L(QV)(g_N, Q_N)(O_i) \\ &= \arg \min_{QV} \sum_{i=1}^N \left\{ Z(a, g_N, Q_N)(O_i) - QV(V_i, a) \right\}^2. \end{aligned}$$

As we have detached estimation of the  $Q$ -function and the  $QV$ -function, we can truly regard the  $Q$ -function as a nuisance parameter and use flexible machine learning methods to reduce bias. At the same time we can also restrict the complexity of the learned  $QV$ -function without losing causal interpretability. For example, we can let the  $QV$ -function be a member of a Donsker class such as the class of smooth parametric models ([Luedtke and Chambaz 2020](#)).

In **polle**, doubly robust  $QV/Q$ -learning is implemented in `policy_learn(type = "drql")`. In Section 3.3, we generalize doubly robust  $QV/Q$ -learning to the multi-stage case.

For binary action sets, it is evident from Equation 6 that only the contrast between the  $QV$ -functions is relevant for learning the optimal policy. We denote this contrast as the blip, which in the single-stage case is also known as the conditional average treatment effect (CATE):

$$\begin{aligned} B_0(v) &= QV_0(v, 1) - QV_0(v, 0) \\ &= \mathbb{E}[U^1 - U^0 \mid V = v]. \end{aligned}$$

A doubly robust loss function for the blip function  $B_0$  is given by

$$L(B)(g_0, Q_0)(O) = \left\{ Z(1, g_0, Q_0)(O) - Z(0, g_0, Q_0)(O) - B(V) \right\}^2,$$

which is implemented in `policy_learn(type = "blip")`.

An advantage of doubly robust  $Q$ -learning and blip learning is that we can leverage the extensively available implementations for (regularized) regression. A downside of  $QV$ -learning and blip learning is that these methods do not target the decision boundary directly. Instead, they minimize the estimation error of the given function. This function approximation is then plugged into the threshold function to obtain an estimate of the decision boundary. An alternative direct approach, like value search, is to use the weighted classification loss function given by

$$L(d)(g_0, Q_0)(O) = |Z(1, g_0, Q_0)(O) - Z(0, g_0, Q_0)(O)| \\ \times I \{d(V) \neq I \{Z(1, g_0, Q_0)(O) - Z(0, g_0, Q_0)(O) > 0\}\},$$

which we show to be equivalent to the value loss function in Appendix B. Usually, the indicator function will be replaced by a convex surrogate to ease minimization. The **polle** package wraps the classification functionality of the **DTRlearn2** package via `policy_learn(type = "owl")`, though this is not based on the presented doubly robust score, but rather on a different augmented score; see Liu, Wang, Kosorok, Zhao, and Zeng (2018).

Since the policy learners in **polle** are based on different loss functions, we need an interpretable performance measure for a fair comparison between the policy learners. As well as evaluating user-defined policies, `policy_eval()` also allows for easy evaluation of an arbitrary policy learner by returning the value of the estimated policy; see Section 3.4 for more details. This is a key feature in **polle** which is not available in other R packages.

The final concept that we want to introduce for now is realistic policy learning. Positivity violations, or even near positivity violations, are a concern for both policy learning and evaluation (Petersen, Porter, Gruber, Wang, and Van der Laan 2012). If we in some stratum of the history do not observe both treatments, it is impossible to learn the optimal policy in the given stratum without strong structural assumptions. Thus, we introduce the set of (estimated) realistic actions at level  $\alpha$  as

$$D_N^\alpha(h) = \{a \in \mathcal{A} : g_N(h, a) > \alpha\}.$$

$Q$ -learning can then easily be adapted to the set of realistic actions as follows:

$$d_N(h) = \arg \max_{a \in D_N^\alpha(h)} Q_N(h, a).$$

For binary action sets, all of the presented policy learners can be adapted to only recommend actions which are deemed realistic at a given level.

In the next section, we formalize all of the above concepts and generalize them to multiple stages.

## 3. Setup and methods

### 3.1. General multi-stage setup

Let  $K \geq 1$  denote a fixed number of stages. Let  $B \in \mathcal{B}$  denote the baseline covariates. For a finite set  $\mathcal{A}$ , let  $A_k \in \mathcal{A}$  denote the decision or action at stage  $k \in \{1, \dots, K\}$ . For

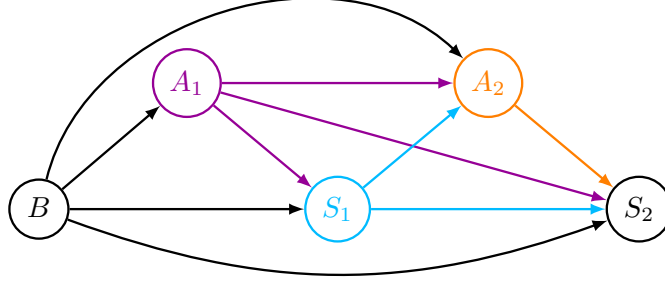


Figure 2: Graph for the observational data with two stages.  $B$  is a baseline covariate,  $A_1$  and  $A_2$  are the two decisions at stages 1 and 2, and  $S_1$  and  $S_2$  are the state variables. From each of the state variables, a reward can be derived, and the sum of these defines the utility of the decisions.

$k \in \{1, \dots, K+1\}$ , let  $S_k \in \mathcal{S}$  denote the state at stage  $k$ . The trajectory for an observation can be written as

$$O = (B, S_1, A_1, S_2, A_2, \dots, S_K, A_K, S_{K+1}),$$

as illustrated in Figure 2. Usually, we will assume to have a sample of  $N$  iid observations indexed as  $\{O_i\}_{i \in 1, \dots, N}$ . For  $k \in \{1, \dots, K+1\}$ , let  $\bar{S}_k = (S_1, \dots, S_k)$ ,  $\bar{A}_k = (A_1, \dots, A_k)$  and  $H_k = (B, \bar{S}_k, \bar{A}_{k-1}) \in \mathcal{H}_k$  define the history where  $A_0 = A_{K+1} = \emptyset$ . Using the implied ordering, the density of the data can be expressed as

$$p_0(O) = p_0(B) \left[ \prod_{k=1}^K p_{0,k}(A_k | H_k) \right] \left[ \prod_{k=1}^{K+1} p_{0,k}(S_k | H_{k-1}, A_{k-1}) \right]. \quad (7)$$

For convenience, let  $S_k = (X_k, U_k)$ , where  $U_k \in \mathbb{R}$  is the  $k$ th reward, and  $X_k$  is a state covariate/variable for  $k \in \{1, \dots, K\}$  and  $X_{K+1} = \emptyset$ . The utility is the sum of the rewards:

$$U = \sum_{k=1}^{K+1} U_k.$$

In the **polle** package, the function `policy_data()` helps the user specify the above data structure.

### 3.2. Policy value estimation

A policy is a set of rules  $d = (d_1, \dots, d_K)$ ,  $d_k : \mathcal{H}_k \rightarrow \mathcal{A}$  assigning an action in each stage. Let  $D_{0,k}(h_k) \subseteq \mathcal{A}$  denote the feasible set of decisions at stage  $k$  for history  $h_k$  under  $P_0$ , i.e.,

$$D_{0,k}(h_k) = \{a_k \in \mathcal{A} : p_{0,k}(a_k | h_k) > 0\}.$$

Define the class of feasible policies  $\mathcal{D}_0$  as all sets of rules satisfying  $d_k(h_k) \in D_{0,k}(h_k)$ .

For a feasible policy  $d$ , let  $P_0^d$  be the distribution with density

$$p_0^d(O) = p_0(B) \left[ \prod_{k=1}^K I\{A_k = d_k(H_k)\} \right] \left[ \prod_{k=1}^{K+1} p_{0,k}(S_k | H_{k-1}, A_{k-1}) \right]. \quad (8)$$

Let  $O^d$  denote the data with distribution given by Equation 8, which is identified from the observed data. Define the value of the policy as

$$\theta_0^d = \mathbb{E}[U^d].$$

Under consistency and sequential randomization, the above value will have a causal interpretation as the mean utility under an intervention given by the feasible policy.

The value of a feasible policy  $d$  can explicitly be stated via the  $Q$ -functions recursively defined as

$$\begin{aligned} Q_{0,K}(h_K, a_K) &= \mathbb{E}[U \mid H_K = h_K, A_K = a_K], \\ Q_{0,k}^{d_{k+1}}(h_k, a_k) &= \mathbb{E} \left[ Q_{0,k+1}^{d_{k+2}}(H_{k+1}, d_{k+1}(H_{k+1})) \mid H_k = h_k, A_k = a_k \right], \quad k \in \{1, \dots, K-1\} \end{aligned}$$

where  $\underline{d}_k = (d_k, \dots, d_K)$ . It is possible to show that the target parameter is identified as

$$\theta_0^d = \mathbb{E}[Q_{0,1}^d(H_1, d_1(H_1))].$$

The recursive structure of the  $Q$ -functions directly inspires a recursive regression procedure resulting in an estimate  $Q_{N,1}^d$ , based on  $N$  iid observations. The value can then be estimated as the empirical mean of  $Q_{N,1}^d(H_1, d_1(H_1))$ .

The value of a feasible policy  $d$  can also be stated via the  $g$ -functions defined as

$$g_{0,k}(h_k, a_k) = p_{0,k}(a_k \mid h_k),$$

for  $k \in \{1, \dots, K\}$ . Again, it is possible to show that

$$\theta_0^d = \mathbb{E} \left[ \left( \prod_{k=1}^K \frac{I\{A_k = d_k(H_k)\}}{g_{0,k}(H_k, A_k)} \right) U \right].$$

Given regression estimates  $g_{N,k}$ , the value can now be estimated as the weighted empirical mean of the observed utilities.

Finally, it is possible to combine the two estimation approaches. Define the doubly robust policy score at stage  $k$  as

$$\begin{aligned} Z_k(\underline{d}_k, g, Q^{\underline{d}_{k+1}})(O) &= Q_k^{\underline{d}_{k+1}}(H_k, d_k(H_k)) \\ &+ \sum_{r=k}^K \left\{ \prod_{j=k}^r \frac{I\{A_j = d_j(H_j)\}}{g_j(H_j, A_j)} \right\} \left\{ Q_{r+1}^{\underline{d}_{r+2}}(H_{r+1}, d_{r+1}(H_{r+1})) - Q_r^{\underline{d}_{r+1}}(H_r, d_r(H_r)) \right\}, \quad (9) \end{aligned}$$

where  $Q_{K+1}(H_{K+1}, d_{K+1}(H_{K+1})) = U$ . It is possible to show that  $\mathbb{E}[Z_1(d, g, Q^d)(O)] = \theta_0^d$  if either  $g = g_0$  or  $Q^d = Q_0^d$ ; see for example [Tsiatis \*et al.\* \(2019\)](#). This result directly inspires a doubly robust moment type estimator of the policy value; see [Algorithm 1](#). In **polle**, this estimator is implemented in `policy_eval(type = "dr", policy)`.

It is well known that  $\psi_0^d(O) = Z_1(d, g_0, Q_0^d)(O) - \theta_0^d$  is the efficient influence function/curve for the policy value. We assume that the absolute utility is bounded and that  $g_{0,k}(H_k, A_k) > \epsilon$

---

**Algorithm 1:** Cross-fitted doubly robust estimator of the policy value  $\theta_0^d$

---

**input :** Dataset with iid observations  $\mathcal{O} = (O_1, \dots, O_N)$   
 Feasible policy  $d$   
 Action probability regression procedure  $\hat{g}$   
 Outcome regression procedure  $\hat{Q}^d$   
**output:** Value estimate  $\theta_N^d$   
 Variance estimate  $\Sigma_N^d$

$\{\mathcal{O}_1, \dots, \mathcal{O}_M\} = \text{M-folds}(\mathcal{O})$

**foreach**  $m \in \{1, \dots, M\}$  **do**

$g_m = \hat{g}(\mathcal{O} \setminus \mathcal{O}_m)$   
 $Q_m^d = \hat{Q}^d(\mathcal{O} \setminus \mathcal{O}_m)$   
 $\mathcal{Z}_{1,m} = \{Z_1(d, g_m, Q_m^d)(O) : O \in \mathcal{O}_m\}$

**end**

$$\theta_N^d = N^{-1} \sum_{m=1}^M \sum_{Z \in \mathcal{Z}_{1,m}} Z$$

$$\Sigma_N^d = N^{-1} \sum_{m=1}^M \sum_{Z \in \mathcal{Z}_{1,m}} (Z - \theta_N^d)^2$$


---

almost surely for some  $\epsilon > 0$ . Let  $\|g\|_{P,2} = \max_{j \in \{1, \dots, K\}} \|g_j\|_{P,2}$  and  $\|Q^d\|_{P,2} = \max_{j \in \{1, \dots, K\}} \|Q_j^{d,j+1}\|_{P,2}$ . If, with probability converging to one,  $g_{m,k}(H_k, A_k) > \epsilon$  and

$$\begin{aligned} \|g_m - g_0\|_{P_0,2} &= o_{P_0}(1), \\ \|Q_m^d - Q_0^d\|_{P_0,2} &= o_{P_0}(1), \\ \|g_m - g_0\|_{P_0,2} \times \|Q_m^d - Q_0^d\|_{P_0,2} &= o_{P_0}(N^{-1/2}), \end{aligned}$$

then

$$N^{1/2}(\theta_N^d - \theta_0^d) = N^{-1/2} \sum_{i=1}^N \psi_0^d(O_i) + o_{P_0}(1).$$

Thus,  $\Sigma_N^d$  from Algorithm 1 is a good estimate of the asymptotic variance of the value estimate if the nuisance models  $\hat{g}$  and  $\hat{Q}^d$  are correctly specified. It is important to note that the convergence rate conditions are relatively weak. For example,  $Q_0^d$  and  $g_0$  may be estimated at rate  $o_{P_0}(N^{-1/4})$ , which is much lower than a parametric rate of order  $o_{P_0}(N^{-1/2})$ . This result justifies the use of adaptive and regularized nuisance models; see Chernozhukov *et al.* (2018).

### 3.3. Policy learning

One of the main objectives of **polle** is to learn the optimal policy from data. As we want to be able to control the policy input, we start by defining the optimal policy within a class of policies that are restricted to a subset of the observed history. The following result is a generalization of Van der Laan and Luedtke (2014).

Let  $V_k$  be a function (or subset) of  $H_k$ . A  $V$ -restricted policy is a set of rules  $d^V = (d_1^V, \dots, d_K^V)$ ,  $d_k^V : \bar{\mathcal{A}}_{k-1} \times \mathcal{V}_k \rightarrow \mathcal{A}$ . Let  $\mathcal{D}^V$  denote the class of  $V$ -restricted policies. Under

positivity, i.e.,  $D_{0,k}(H_k) = \mathcal{A}$  almost surely, the  $V$ -optimal policy is defined as

$$d_0^V = \arg \max_{d \in \mathcal{D}^V} \mathbb{E}[U^d].$$

The following theorem specifies the  $V$ -optimal policy in a recursive manner. A proof for the two-stage case can be found in Appendix A.

**Theorem 3.1.** *Under positivity, for any  $a = (a_1, \dots, a_K)$  and policy  $d$  define*

$$QV_{0,K}(\bar{a}_{K-1}, v_K, a_K) = \mathbb{E}[U^{\bar{a}_K} \mid V_K^{\bar{a}_{K-1}} = v_K], \quad (10)$$

$$QV_{0,k}^{\underline{d}_{k+1}}(\bar{a}_{k-1}, v_k, a_k) = \mathbb{E}[U^{\bar{a}_k, \underline{d}_{k+1}} \mid V_k^{\bar{a}_{k-1}} = v_k], \quad k \in \{1, \dots, K-1\}. \quad (11)$$

If

$$\mathbb{E}[U^a \mid V_1, \dots, V_k^{\bar{a}_{k-1}}] = \mathbb{E}[U^a \mid V_k^{\bar{a}_{k-1}}], \quad k \in \{1, \dots, K\}, \quad (12)$$

then the  $V$ -optimal policy  $d_0^V$  is recursively given by

$$d_{0,K}^V(\bar{a}_{K-1}, v_K) = \arg \max_{a_K} QV_{0,K}(\bar{a}_{K-1}, v_K, a_K),$$

$$d_{0,k}^V(\bar{a}_{k-1}, v_k) = \arg \max_{a_k} QV_{0,k}^{\underline{d}_{k+1}}(\bar{a}_{k-1}, v_k, a_k), \quad k \in \{1, \dots, K-1\}.$$

If for all  $k \in \{1, \dots, K\}$  and  $r < k$ ,  $V_r^{\bar{a}_{r-1}}$  is a function of  $V_k^{\bar{a}_{k-1}}$ , then Equation 12 holds by construction. The intuition behind the condition in Equation 12 is that it will ensure that the tower property holds for the nested conditional expectations in Equations 10 and 11. Finally, note that only future rewards affect the optimal decision at stage  $k$  since

$$\begin{aligned} \arg \max_{a_k} QV_{0,k}^{\underline{d}_{k+1}}(\bar{a}_{k-1}, v_k, a_k) \\ &= \arg \max_{a_k} \mathbb{E} \left[ U_1 + \dots + U_k^{\bar{a}_{k-1}} + U_{k+1}^{\bar{a}_k} + \dots + U_{K+1}^{\bar{a}_k, \underline{d}_{k+1}} \mid V_k^{\bar{a}_{k-1}} = v_k \right] \\ &= \arg \max_{a_k} \mathbb{E} \left[ U_{k+1}^{\bar{a}_k} + \dots + U_{K+1}^{\bar{a}_k, \underline{d}_{k+1}} \mid V_k^{\bar{a}_{k-1}} = v_k \right]. \end{aligned}$$

The basis for learning the  $V$ -restricted optimal policy is to construct an observed data loss function which identifies  $d_0^V$ , i.e., construct a function  $L$  for which  $\mathbb{E}[L(d)(O)]$  is minimized in  $d_0^V$ . Various loss functions inspire different algorithms for estimating the  $V$ -optimal policy. In the following, we present four different doubly robust loss functions: value, quality, blip, and classification loss functions.

#### Value search

For the final stage  $K$ , consider the loss function  $\tilde{L}_K(d_K)(g_K, Q_K)(O)$  in  $d_K$  given by

$$\begin{aligned} -\tilde{L}_K(d_K)(g_K, Q_K)(O) &= Z_K(d_K, g, Q)(O) \\ &= Q_K(H_k, d_K(H_k)) + \frac{I\{A_K = d_K(H_k)\}}{g_K(H_K, A_K)} \{U - Q_K(H_K, A_K)\}. \end{aligned} \quad (13)$$

**Algorithm 2:** Recursive value search

---

**input :** Dataset with iid observations  $\mathcal{O} = (O_1, \dots, O_N)$   
Class of  $V$ -restricted policies  $\mathcal{D}^V$   
Function class minimization procedure  $\hat{F}$   
Action probability regression procedure  $\hat{g}$   
Outcome regression procedure  $\hat{Q} = \{\hat{Q}_1, \dots, \hat{Q}_K\}$

**output:**  $V$ -restricted optimal policy estimate  $d_N^V$

$\{\mathcal{O}_1, \dots, \mathcal{O}_L\} = \text{L-folds}(\mathcal{O})$   
**foreach**  $l \in \{1, \dots, L\}$  **do**  
|  $g_l = \hat{g}(\mathcal{O} \setminus \mathcal{O}_l)$   
**end**  
**for**  $k = K$  **to** 1 **do**  
| **foreach**  $l \in \{1, \dots, L\}$  **do**  
| |  $Q_{l,k}^{d_{N,k+1}^V} = \hat{Q}_k^{d_{N,k+1}^V} \left( \left\{ H_k, A_k, Q_{k+1,l}^{d_{N,k+2}^V} (H_{k+1}, d_{N,k+1}^V(H_{k+1})) : O \in \mathcal{O} \setminus \mathcal{O}_l \right\} \right)$   
| **end**  
|  $d_{N,k}^V = \hat{F}_{d_k^V \in \mathcal{D}^V} \left( \sum_{l=1}^L \sum_{O \in \mathcal{O}_l} \tilde{L}(d_k^V)(\underline{d}_{k+1,N}^V, g_l, Q_l^{d_{k+1,N}^V})(O) \right)$   
**end**

---

If either  $Q_K = Q_{0,K}$  or  $g_K = g_{0,K}$ , then

$$\mathbb{E} \left[ \tilde{L}_K(d_K)(g_K, Q_K)(O) \right] = -\mathbb{E} \left[ U^{d_K} \right].$$

Thus, for a  $V$ -restricted policy

$$\mathbb{E} \left[ \tilde{L}_K(d_K^V)(g_K, Q_K)(O) \right] = -\mathbb{E} \left[ QV_{0,K} \left( \bar{A}_{K-1}, V_K, d_K^V(\bar{A}_{K-1}, V_K) \right) \right],$$

meaning that over the class of  $V$ -restricted policies  $\mathcal{D}_{0,K}^V$ , the expected loss is minimized in  $d_{0,K}^V$  by Theorem 3.1.

For stage  $k \in \{1, \dots, K-1\}$ , consider the loss function  $\tilde{L}_k(d_k)(\underline{d}_{k+1}, g, Q^{d_{k+1}})(O)$  in  $d_k$  given by

$$-\tilde{L}_k(d_k)(\underline{d}_{k+1}, g, Q^{d_{k+1}})(O) = Z_k([d_k, \underline{d}_{k+1}], g, Q^{d_{k+1}})(O).$$

If either  $Q^{d_{k+1}} = Q_0^{d_{k+1}}$  or  $g = g_0$ , then

$$\mathbb{E} \left[ \tilde{L}_k(d_k)(\underline{d}_{k+1}, g, Q^{d_{k+1}})(O) \right] = -\mathbb{E} \left[ U^{d_k, \underline{d}_{k+1}} \right],$$

and for a  $V$ -restricted policy at stage  $k$  it holds that

$$\mathbb{E} \left[ \tilde{L}_k(d_k^V)(\underline{d}_{k+1}, g, Q^{d_{k+1}})(O) \right] = -\mathbb{E} \left[ QV_{0,k}^{d_{k+1}}(\bar{A}_{k-1}, V_k, d_k^V(\bar{A}_{k-1}, V_k)) \right]$$

Thus, given  $d_{0,k+1}^V$ , the above expected loss over  $\mathcal{D}_k^V$  is again minimized in  $d_{0,k}^V$  by Theorem 3.1.

The constructed loss function directly inspires recursive value search, see Algorithm 2. Similar to Algorithm 1, the algorithm utilizes cross-fitted values of the nuisance models at each step. However, it is important to note that the  $Q$ -models are not truly cross-fitted (except for the last stage), because the fitted policy at a given stage depends on the fitted policy at later stages. A nested cross-fitting scheme would be required to make the folds (used to fit the  $Q$ -models) independent.

Algorithm 2 requires a suitable function class minimization procedure  $\hat{F}$  that imitates  $\arg \min_{d_k^V \in \mathcal{D}^V} \{\cdot\}$  at every stage. The R package **policytree** (see Sverdrup *et al.* 2020) implements such a minimization procedure where the class of policies is given by decision trees. See Zhou, Athey, and Wager (2023) for theoretical results related to this implementation. In **polle**, recursive value search using **policytree** is implemented in `policy_learn(type = "pt1")`.

### Quality learning

Under positivity, for any  $a = (a_1, \dots, a_K)$  and any policy  $d$ , let  $Z_k([a_k, \underline{d}_{k+1}], g, Q^{d_{k+1}})(O)$  be given by Equation 9 with  $d_k$  replaced by the static policy  $a_k \in \mathcal{A}$ . For the final stage  $K$ , define  $QV_{0,K}(a_K)(\bar{a}_{K-1}, v_K) = QV_{0,K}(\bar{a}_{K-1}, v_K, a_K)$  from Equation 10. If  $g_K = g_{0,K}$  or  $Q_K = Q_{0,K}$ , then

$$\mathbb{E} \left[ Z_K(a_K, g, Q)(O) \mid \bar{A}_{K-1}, V_K \right] = QV_{0,K}(a_K)(\bar{A}_{K-1}, V_K).$$

Now, a valid loss function for  $QV_{0,K}(a_K)$  over functions  $QV_K : \bar{\mathcal{A}}_{K-1} \times \mathcal{V}_K \rightarrow \mathcal{A}$  is given by

$$L_K(QV_K)(a_K, g, Q)(O) = \left\{ Z_K(a_K, g, Q)(O) - QV_K(\bar{A}_{K-1}, V_K) \right\}^2.$$

Hence, any regression type estimator which minimizes the (empirical) mean squared error can be used to estimate  $QV_{0,K}(a_K)$ . This can be repeated for every  $a_K \in \mathcal{A}$ . By Theorem 3.1, the  $V$ -optimal policy  $d_{0,K}^V$  is then identified as  $\arg \max_{a_K \in \mathcal{A}} QV_{0,K}(a_K)$ .

For  $k \in \{1, \dots, K-1\}$ , let  $QV_{0,k}^{d_{k+1}}(a_k)(\bar{a}_{k-1}, v_k) = QV_{0,k}^{d_{k+1}}(\bar{a}_{k-1}, v_k, a_k)$  from Equation 11. If  $g = g_0$  or  $Q^{d_{k+1}} = Q_0^{d_{k+1}}$ , then

$$\mathbb{E} \left[ Z_k([a_k, \underline{d}_{k+1}], g, Q^{d_{k+1}})(O) \mid \bar{A}_{k-1}, V_k \right] = QV_{0,k}^{d_{k+1}}(a_k)(\bar{A}_{k-1}, V_k),$$

and a valid loss function for  $QV_{0,k}^{d_{k+1}}(a_k)$  over functions  $QV_k$  is given by

$$L_k(QV_k)(a_k, \underline{d}_{k+1}, g, Q^{d_{k+1}})(O) = \left\{ Z_k([a_k, \underline{d}_{k+1}], g, Q^{d_{k+1}})(O) - QV_k(\bar{A}_{k-1}, V_k) \right\}^2.$$

Thus, given the future  $V$ -restricted optimal policy rules  $\underline{d}_{0,k+1}^V$ , if  $g = g_0$  or  $Q^{d_{0,k+1}} = Q_0^{d_{0,k+1}}$ , then the expected loss is minimized in  $QV_{0,k}^{d_{0,k+1}}(a_k)$ . Again, this can be repeated for each  $a_k \in \mathcal{A}$  and the  $V$ -optimal policy at stage  $k$  is identified as  $\arg \max_{a_k \in \mathcal{A}} QV_{0,k}^{d_{0,k+1}}(a_k)$ .

The constructed quality loss function directly inspires doubly robust  $V$ -restricted  $Q$ -learning; see Algorithm 3. In **polle**, this policy estimator is implemented in `policy_learn(type = "drql")`.

**Algorithm 3:** Doubly robust  $Q$ -learning

---

**input** : Dataset with iid observations  $\mathcal{O} = (O_1, \dots, O_N)$   
Action probability regression procedure  $\hat{g}$   
Outcome regression procedure  $\hat{Q} = \{\hat{Q}_1, \dots, \hat{Q}_K\}$   
Outcome regression procedure  $\widehat{QV} = \{\widehat{QV}_1, \dots, \widehat{QV}_K\}$

**output:**  $V$ -restricted optimal policy estimate  $d_N^V$

$\{\mathcal{O}_1, \dots, \mathcal{O}_L\} = \text{L-folds}(\mathcal{O})$   
**foreach**  $l \in \{1, \dots, L\}$  **do**  
|  $g_l = \hat{g}(\mathcal{O} \setminus \mathcal{O}_l)$   
**end**  
**for**  $k = K$  **to**  $1$  **do**  
| **foreach**  $l \in \{1, \dots, L\}$  **do**  
| |  $Q_{k,l}^{d_{N,k+1}^V} = \hat{Q}_k \left( \left\{ H_k, A_k, Q_{k+1,l}^{d_{N,k+2}^V} \left( H_{k+1}, d_{N,k+1}^V(H_{k+1}) \right) : O \in \mathcal{O} \setminus \mathcal{O}_l \right\} \right)$   
| | **foreach**  $a_k \in \mathcal{A}$  **do**  
| | |  $\tilde{\mathcal{O}}_{k,l}(a_k) = \left\{ \bar{A}_{k-1}, V_k, Z_k \left( [a_k, \underline{d}_{N,k+1}^V], g_l, Q_l^{d_{N,k+1}^V} \right) (O) : O \in \mathcal{O}_l \right\}$   
| | **end**  
| | **end**  
| | **foreach**  $a_k \in \mathcal{A}$  **do**  
| | |  $QV_{N,k}^{d_{N,k+1}^V}(a_k) = \widehat{QV}_k \left( \{\tilde{\mathcal{O}} \in \tilde{\mathcal{O}}_{k,l}(a_k) : l = 1, \dots, L\} \right)$   
| | **end**  
| |  $d_{N,k}^V = \arg \max_{a_k \in \mathcal{A}} QV_{N,k}^{d_{N,k+1}^V}(a_k)$   
**end**

---

*Blip learning*

As outlined in Section 2, for a binary action set  $\mathcal{A} = \{0, 1\}$ , instead of learning the  $QV$ -function for each action, it is sufficient to learn the contrast or blip between the functions. Considering Theorem 3.1, define the blips as

$$B_{0,K}(\bar{a}_{K-1}, v_K) = QV_{0,K}(\bar{a}_{K-1}, v_K, 1) - QV_{0,K}(\bar{a}_{K-1}, v_K, 0),$$

$$B_{0,k}^{d_{k+1}}(\bar{a}_{k-1}, v_k) = QV_{0,k}^{d_{k+1}}(\bar{a}_{k-1}, v_k, 1) - QV_{0,k}^{d_{k+1}}(\bar{a}_{k-1}, v_k, 0).$$

Using the blips, the  $V$ -restricted optimal policy is identified as

$$d_{0,K}^V(\bar{a}_{K-1}, v_K) = I \{B_{0,K}(\bar{a}_{K-1}, v_K) > 0\},$$

$$d_{0,k}^V(\bar{a}_{k-1}, v_k) = I \left\{ B_{0,k}^{d_{k+1}}(\bar{a}_{k-1}, v_k) > 0 \right\}, \quad k \in \{1, \dots, K-1\}.$$

Doubly robust blip learning is now almost identical to Algorithm 3, with  $\widehat{QV}$  replaced by  $\hat{B}$  and the doubly robust scores  $Z_k \left( [a_k, \underline{d}_{k+1}^V], g, Q^{d_{k+1}^V} \right) (O)$  replaced by the contrasts

$$W_k(\underline{d}_{k+1}^V, g, Q^{d_{k+1}^V})(O) = Z_k([1, \underline{d}_{k+1}^V], g, Q^{d_{k+1}^V})(O) - Z_k([0, \underline{d}_{k+1}^V], g, Q^{d_{k+1}^V})(O).$$

In **polle**, blip learning is implemented in `policy_learn(type = "blip")`.

### Weighted classification

In this section, we again consider the case where the action set is binary, i.e.,  $\mathcal{A} = \{0, 1\}$ . As shown in Appendix B, a valid weighted classification (0-1) loss function is given by

$$L_k(d_k)(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) = \left| W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) \right| \times I \left\{ d_k(\bar{A}_{K-1}, V_k) \neq I \left\{ W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) > 0 \right\} \right\}. \quad (14)$$

Given the future  $V$ -optimal policy rules  $\underline{d}_{0,k+1}^V$ , if  $g = g_0$  or  $Q^{\underline{d}_{0,k+1}^V} = Q_0^{\underline{d}_{0,k+1}^V}$ , the expected weighted classification loss function is minimized in  $d_{0,k}^V$  over  $\mathcal{D}_k^V$ .

It can be challenging to perform minimization of the weighted classification loss function due to the discontinuity of the indicator function. For this reason, it is common to use a convex surrogate of the indicator function. Let  $f_k : \bar{A}_{K-1} \times \mathcal{V}_k \rightarrow \mathbb{R}$  be some action function corresponding to  $d_k$ , i.e.,  $d_k(\bar{A}_{K-1}, V_k) = I \{ f_k(\bar{A}_{K-1}, V_k) > 0 \}$ . Then

$$L_k(f_k)(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) = \left| W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) \right| \times I \left\{ f_k(\bar{A}_{K-1}, V_k) \left[ 2I \left\{ W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) > 0 \right\} - 1 \right] \leq 0 \right\}$$

is equivalent to Equation 14. Replacing  $I \{ x \leq 0 \}$  in the above expression with a convex surrogate  $\phi : \mathbb{R} \rightarrow [0, \infty)$  differentiable at 0 with  $\phi'(0) < 0$  yields a convex loss function given by

$$L_k^\phi(f_k)(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) = \left| W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) \right| \times \phi \left( f_k(\bar{A}_{K-1}, V_k) \left[ 2I \left\{ W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) > 0 \right\} - 1 \right] \right).$$

If  $g = g_0$  or  $Q^{\underline{d}_0^V} = Q_0^{\underline{d}_0^V}$  and the non-exceptional law holds, i.e., that

$$0 < E \left[ W_k(\underline{d}_{k+1}, g, Q^{\underline{d}_{k+1}})(O) \right],$$

then the expected weighted surrogate loss function is minimized in  $f_{0,k}^V$  over the class of  $V$ -restricted action functions and  $d_{0,k}^V = I \{ f_{0,k}^V > 0 \}$ . The above result directly inspires recursive learning of the restricted optimal policy using weighted classification methods similar to Algorithm 2.

The classification perspective was first established by Zhao *et al.* (2012) and Zhang *et al.* (2012). Various methods within this approach have been implemented in the R packages **DTRlearn2** and **DynTxRegime**; see Chen *et al.* (2020) and Holloway *et al.* (2025). Generalizations to multiple actions (more than two) have also been developed; see Zhang, Chen, Fu, He, Zhao, and Liu (2020).

The **polle** package wraps the weighted classification function `owl()` from the **DTRlearn2** package, though this algorithm is not based on the presented doubly robust blip score, but on a different augmented score; see Liu *et al.* (2018). The policy learner is available via `policy_learn(type = "owl")`.

---

**Algorithm 4:** Cross-fitted doubly robust estimator of  $\theta_0^{d_N}$

---

**input** : Dataset with iid observations  $\mathcal{O} = (O_1, \dots, O_N)$   
 Policy learning procedure  $\hat{d}$   
 Action probability regression procedure  $\hat{g}$   
 Outcome regression procedure  $\hat{Q}^d$   
**output:** Cross-fitted value estimate  $\theta_N^d$   
 Cross-fitted variance estimate  $\Sigma_N^d$

$\{\mathcal{O}_1, \dots, \mathcal{O}_M\} = \mathbf{M}\text{-folds}(\mathcal{O})$

**foreach**  $m \in \{1, \dots, M\}$  **do**

$d_m = \hat{d}(\mathcal{O} \setminus \mathcal{O}_m)$   
 $g_m = \hat{g}(\mathcal{O} \setminus \mathcal{O}_m)$   
 $Q_m^{d_m} = \hat{Q}^{d_m}(\mathcal{O} \setminus \mathcal{O}_m)$   
 $\mathcal{Z}_{1,m} = \{Z_1(d_m, g_m, Q_m^{d_m})(O) : O \in \mathcal{O}_m\}$

**end**

$\theta_N^{d_N} = N^{-1} \sum_{m=1}^M \sum_{Z \in \mathcal{Z}_{1,m}} Z$

$\Sigma_N^{d_N} = N^{-1} \sum_{m=1}^M \sum_{Z \in \mathcal{Z}_{1,m}} (Z - \theta_N^{d_N})^2$

---

### 3.4. Learned policy performance

Another main objective of **polle** is to be able to compare the performance of various policy learners. For this purpose, we advocate for targeting the estimated policy’s value (Chakraborty, Laber, and Zhao 2014). Conditional on a policy estimate  $d_N$ , the target parameter is given by

$$\theta_0^{d_N} = \mathbb{E}[U^{d_N}].$$

This data-adaptive parameter is arguably more practically relevant than the true optimal policy value because the true optimal policy will never actually be implemented. Moreover, the data-adaptive policy value does not suffer from non-regularity issues related to the optimal policy value; see (Hirano and Porter 2012; Luedtke and Van der Laan 2016a; Robins and Rotnitzky 2014; Chakraborty and Moodie 2013).

Under regularity conditions similar to those stated in Section 3.2 (see Van der Laan and Luedtke 2014), if  $d_N$  has a limiting policy  $d'$ , then the efficient influence function for  $\theta_0^{d_N}$  is given by

$$Z_1(d', g_0, Q_0^{d'})(0) - \theta_0^{d'}.$$

As described in Section 3.2, this allows us to construct a doubly robust estimator for the value of the learned policy; see Algorithm 4. The algorithm also provides a variance estimate, enabling a Wald-type confidence interval. In the **polle** package, learned policy evaluation is implemented in the function `policy_eval(policy_learn)`.

### 3.5. Stochastic number of stages

The methodology developed for a fixed number of stages can be extended to handle a stochastic number of stages assuming that the maximal number of stages is finite. The key is to

modify each observation such that every observation has the same number of stages.

Let  $K^*$  denote the stochastic number of stages bounded by a maximal number of stages  $K$ . As in Section 3.1, let  $B^*$  denote the baseline data,  $(A_1^*, \dots, A_{K^*}^*)$  denote the decisions and  $(S_1^*, \dots, S_{K^*+1}^*)$  denote the stage summaries where  $S_k^* = (X_k^*, U_k^*)$  and  $X_{K^*+1}^* = \emptyset$ . The utility  $U^*$  is still the sum of the rewards,  $U^* = \sum_{k=1}^{K^*+1} U_k^*$ . We assume that the distribution of the observed data is given by  $P_0^*$  composed of conditional densities  $p_0^*(B^*)$ ,  $p_{0,k}^*(A_k^* | H_k^*)$  for  $k \in \{1, \dots, K\}$  and  $p_{0,k}^*(S_k^* | H_k^*)$  for  $k \in \{1, \dots, K+1\}$  such that the likelihood for an observation  $O^*$  is given by

$$p_0^*(O^*) = p_0^*(B^*) \left[ \prod_{k=1}^{K^*} p_{0,k}^*(A_k^* | H_k^*) \right] \left[ \prod_{k=1}^{K^*+1} p_{0,k}^*(S_k^* | H_{k-1}^*, A_{k-1}^*) \right].$$

For a feasible policy  $d^* = (d_1^*, \dots, d_K^*)$ , the distribution  $P^{*d^*}$  is defined similarly to  $P^d$  in Equation 8. Also, the value under  $P^{*d^*}$  is defined as  $\mathbb{E}[U^{*d^*}]$ .

We now construct auxiliary data such that each observation  $O^*$  has  $K$  stages. Let  $A_k = A_k^*$  for  $k \leq K^*$  and  $A_k = a^\dagger \in \mathcal{A}$  (for some default value  $a^\dagger$ ) for  $k > K^*$ . Similarly, let  $S_k = S_k^*$  for  $k \leq K^* + 1$ . Finally, let  $X_k = \emptyset$  and  $U_k = 0$  for  $k > K^* + 1$  such that  $U = U^*$ . This construction implies a partly degenerate distribution  $P_0$  over the maximal number of stages with density of the form given by Equation 7; see Goldberg and Kosorok (2012). A feasible policy  $d$  associated with  $d^*$  is given by

$$d_k(H_k) = \begin{cases} a^\dagger & \text{if } X_k = \emptyset \\ d_k^*(H_k) & \text{otherwise.} \end{cases}$$

Furthermore, it holds by construction that  $g_{0,k}(H_k, a^\dagger) = 1$  and  $Q_{0,k}^d(H_k, a^\dagger) = U$  if  $X_k = \emptyset$ . Finally, a generalization of the results in Goldberg and Kosorok (2012) yields that  $\mathbb{E}[U^d] = \mathbb{E}[U^{*d^*}]$ . Thus, the methodology developed for a fixed number of stages can be used on the augmented data.

### 3.6. Partial policy

It may occur that a small subset of the observations has numerous stages. Without further structural assumptions, information about these late stages will be sparse. Uncertain estimation of the  $Q$ -functions for the late stages can be avoided by considering partial policies. Let  $\tilde{K} < K$  and let  $\bar{d}_{\tilde{K}}$  be a given policy up till stage  $\tilde{K}$ . A partial (stochastic) policy is now given by  $(\bar{d}_{\tilde{K}}, A_{\tilde{K}+1}, \dots, A_K)$ . By setting  $Q_{0,\tilde{K}} = \mathbb{E}[U | H_{\tilde{K}} = h_{\tilde{K}}, A_{\tilde{K}} = a_{\tilde{K}}]$  and  $Q_{0,\tilde{K}+1} = U$ , the efficient influence score for the partial policy value will be equal to Equation 9 with  $K$  replaced by  $\tilde{K}$ . From a practical point of view, implementation of a partial policy requires that  $(g_{0,\tilde{K}+1}, \dots, g_{0,K})$  is known (or at least well approximated). It is easy to consider partial policies in the **polle** package by using `partial()` on a given policy data object.

### 3.7. Realistic policy learning

Positivity violations or even near positivity violations are a large concern for policy learning based on historical data. Estimation of a valid loss function will solely rely on extrapolation of the  $Q$ -functions to decisions with little or no support in the observed data; see Petersen

**Algorithm 5:** Realistic doubly robust  $Q$ -learning

---

**input** : Dataset with iid observations  $\mathcal{O} = (O_1, \dots, O_N)$   
Action probability regression procedure  $\hat{g}$   
Outcome regression procedure  $\hat{Q} = \{\hat{Q}_1, \dots, \hat{Q}_K\}$   
Outcome regression procedure  $\widehat{QV} = \{\widehat{QV}_1, \dots, \widehat{QV}_K\}$   
**output:** Realistic  $V$ -restricted optimal policy estimate  $d_N^V$

$g_N = \hat{g}(\mathcal{O})$   
 $\{\mathcal{O}_1, \dots, \mathcal{O}_L\} = \text{L-folds}(\mathcal{O})$   
**foreach**  $l \in \{1, \dots, L\}$  **do**  
|  $g_l = \hat{g}(\mathcal{O} \setminus \mathcal{O}_l)$   
**end**  
**for**  $k = K$  **to** 1 **do**  
| **foreach**  $l \in \{1, \dots, L\}$  **do**  
| |  $Q_{k,l}^{d_{N,k+1,l}^V} = \hat{Q}_k \left( \left\{ H_k, A_k, Q_{k+1,l}^{d_{N,k+2,l}^V} \left( H_{k+1}, d_{N,k+1,l}^V(H_{k+1}) \right) : O \in \mathcal{O} \setminus \mathcal{O}_l \right\} \right)$   
| | **foreach**  $a_k \in \mathcal{A}$  **do**  
| | |  $\tilde{\mathcal{O}}_{k,l}(a_k) = \left\{ \bar{A}_{k-1}, V_k, Z_k \left( [a_k, d_{N,k+1,l}^V], g_l, Q_l^{d_{N,k+1,l}^V} \right) (O) : O \in \mathcal{O}_l \right\}$   
| | **end**  
| | **end**  
| | **foreach**  $a_k \in \mathcal{A}$  **do**  
| | |  $QV_{N,k}^{d_{N,k+1}^V}(a_k) = \widehat{QV}_k \left( \{\tilde{O} \in \tilde{\mathcal{O}}_{k,l}(a_k) : l = 1, \dots, L\} \right)$   
| | **end**  
| | **foreach**  $l \in \{1, \dots, L\}$  **do**  
| | |  $d_{N,k,l}^V = \arg \max_{a_k \in D_{g_l,k}^\alpha} QV_{N,k}^{d_{N,k+1}^V}(a_k)$   
| | **end**  
| |  $d_{N,k}^V = \arg \max_{a_k \in D_{g_N,k}^\alpha} QV_{N,k}^{d_{N,k+1}^V}(a_k)$   
**end**

---

*et al.* (2012). To address this issue, we suggest restricting the set of possible interventions based on the action probability model. For a probability threshold  $\alpha > 0$ , define the set of realistic actions at stage  $k$  based on the action probability model  $g$  as

$$D_{g,k}^\alpha(h_k) = \{a_k \in \mathcal{A} : g_k(h_k, a_k) > \alpha\}.$$

It is relatively simple to modify doubly robust  $Q$ -learning to only consider realistic policies; see Algorithm 5. On the other hand, it is harder to make the same practical modification to a given value search algorithm because the structure of the candidate function class  $\mathcal{D}^V$  changes in a non-trivial way. However, in the situation that the action set is dichotomous, the recommended action can be overruled by the alternative action, if it is deemed unrealistic. In the **polle** package, realistic policy learning is implemented via `policy_learn(alpha)`.

## 4. Syntax and implementation details

The **polle** implementation is built up around four functions: `policy_data()`, `policy_def()`, `policy_eval()` and `policy_learn()`. Figure 3 provides an overview of how the functions relate and the main required inputs and outputs. `policy_data()` constructs a policy data object. The data input can be in long or wide format. Usually, the wide format is used for applications with a fixed number of stages and a possibly varying set of state covariates. Assume that the observed data has the sequential form

$$O = (B, X_1, U_1, A_1, X_2, W_2, U_2, A_2, U_3),$$

where  $B$  is a baseline covariate,  $X_1$ ,  $X_2$ , and  $W_2$  are state covariates,  $U_1$ ,  $U_2$ , and  $U_3$  are rewards, and  $A_1$  and  $A_2$  are actions. Given a `data.table`/`data.frame` denoted `data` with variable/column names `B`, `X_1`, `U_1`, `A_1`, `X_2`, `W_2`, `U_2`, `A_2` and `U_3`, we can apply `policy_data` in the following way:

```
policy_data(data, action = c("A_1", "A_2"), baseline = c("B"),
  covariates = list(X = c("X_1", "X_2"), W = c(NA, "W_2")),
  utility = c("U_1", "U_2", "U_3"), type = "wide")
```

If only the final utility  $U$  is provided, we may replace `c("U_1", "U_2", "U_3")` with `c("U")`. Note that each row in `data` corresponds to a single observation.

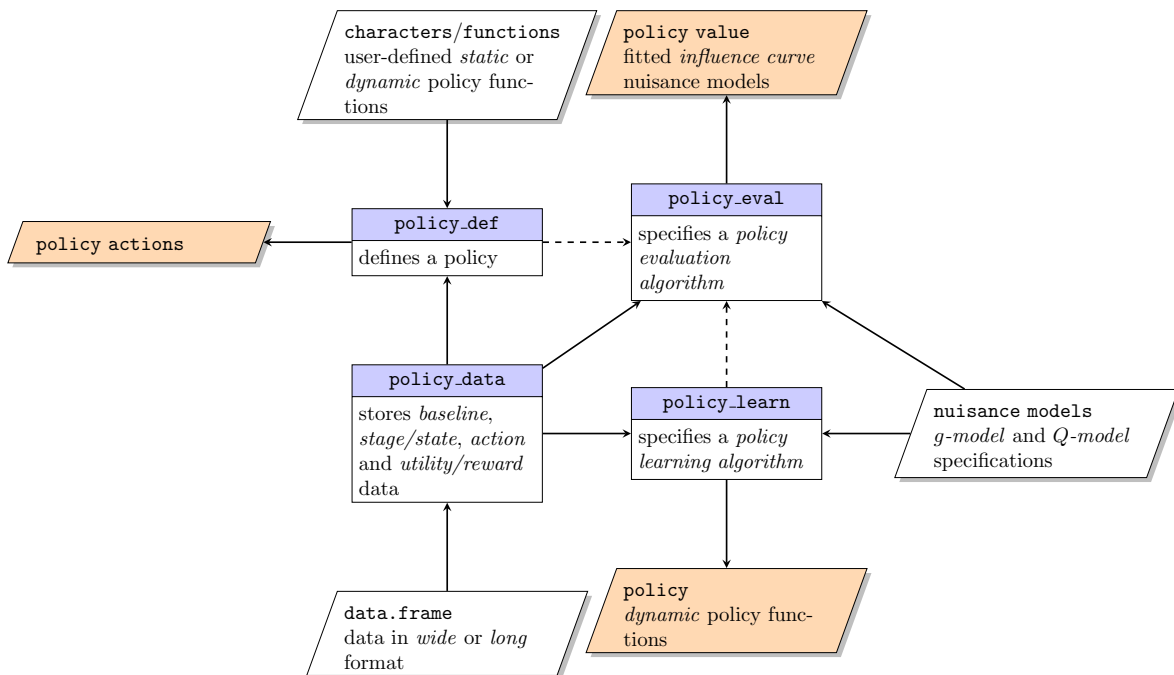


Figure 3: Overview of the four main functions of **polle**, and their arguments and return values. The starting point is to define the input data in the correct format using `policy_data()`. A policy can subsequently be defined directly by the user using `policy_def()`, or estimated with one of the algorithms described in Section 3.3 with `policy_learn()`. The value of a policy can be estimated directly using `policy_eval()`.

The long format is inspired by the data format used for survival data (Therneau 2026). This format is relevant for handling a high and possibly stochastic number of stages. Assume that the observed data has the sequential form

$$O = (B, X_1, U_1, A_1, \dots, X_{K^*}, U_{K^*}, A_{K^*}, U_{(K^*+1)}),$$

where  $K^*$  is the (stochastic) number of stages. Assume that `stage_data` is a `data.table` with variable names `id`, `stage`, `event`, `X`, `U` and `A`. `id` and `stage` denote the observation ID and stage number. The variable `event` is an event indicator which is 0 in stages 1 through  $K^*$  and 1 in stage  $(K^* + 1)$ . Also assume that `baseline_data` is a `data.table` with variable names `id` and `B`. An application of `policy_data()` is now given by:

```
policy_data(stage_data, baseline_data = baseline_data, action = "A",
  baseline = c("B"), covariates = c("X"), utility = "U", id = "id",
  stage = "stage", event = "event", type = "long")
```

Note, an observation with  $K^*$  stages spans over  $(K^* + 1)$  rows in `stage_data` and a single row in `baseline_data`.

The function `policy_def()` constructs a user-specified static or dynamic policy. The resulting policy object can be applied directly on a policy data object or as input to `policy_eval()`.

```
policy_def(policy_functions, full_history = FALSE, replicate = FALSE)
```

`policy_functions` may be a single function/character string or a list of functions/character strings defining the policy at each stage. The argument `full_history` defines the input to the policy functions. If `full_history = FALSE`, the state/Markov type history  $(B, X_k)$  is passed on to the functions with variable names `B` and `X`. If `full_history = TRUE`, the full history  $(B, X_1, A_1, \dots, X_{k-1}, A_{k-1}, X_k)$  with variable names `B`, `X_1`, `A_1`,  $\dots$ , `X_(k-1)`, `A_(k-1)`, `X_k` is passed on to the functions. As an example, `function(X) 1 * (X > 0)` in combination with `full_history = FALSE` defines the policy  $A_k = I\{X_k > 0\}$ . Similarly, at stage  $k = 2$ , `function(X_1, X_2) 1 * (X_1 > 0) * (X_2 > 0)` in combination with `full_history = TRUE` defines the policy  $A_2 = I\{X_1 > 0, X_2 > 0\}$ . The input `replicate = TRUE` will reuse the provided policy functions at each stage if possible.

`policy_learn()` specifies a policy learning algorithm which can be used directly on a policy data object or as input to `policy_eval()`. The `type` argument selects the method. Table 1 provides an overview of the method types, dependencies and limitations.

A cross-fitted doubly robust  $V$ -restricted  $Q$ -learning algorithm, see Algorithms 3 and 5, may be specified as follows:

```
policy_learn(type = "drql", control = list(qv_models = q_glm(~ X)),
  full_history = FALSE, alpha = 0.05, L = 10)
```

The control argument `qv_models` is a single model or a list of models specifying the  $QV$ -models. We will subsequently describe these models in detail. Note that a  $QV$ -model is fitted for each action in the action set. The argument `full_history` specifies the history available to the  $QV$ -models similar to `full_history` in `policy_def()`. The argument `alpha` is the probability threshold for defining the set of realistic actions. The default value is `alpha = 0`. Finally, the argument `L` is the number of folds used in the cross-fitting procedure.

type argument	Method	Imports	Limitations
"ql"	Q-learning		
"drql"	Doubly Robust Q-learning. Algorithm 3, 5.		
"blip"	Doubly Robust blip learning.		
"ptl"	Policy tree learning. Algorithm 2.	<b>policytree</b>	Realistic policy learning implemented for dichotomous action sets.
"owl"	Outcome weighted learning	<b>DTRlearn2</b>	No realistic policy learning. Fixed number of stages. Dichotomous action set. Augmentation terms are not cross-fitted.
"earl"	Efficient augmented and relaxation learning	<b>DynTxRegime</b>	Single stage. No cross-fitting. No realistic policy learning. Dichotomous action set.
"rwl"	Residual weighted learning	<b>DynTxRegime</b>	Same as "earl".

Table 1: Overview of policy learning methods and their dependencies and limitations.

Similarly, a cross-fitted doubly robust sequential value search procedure based on decision trees, see Algorithm 2, may be specified as follows:

```
policy_learn(type = "ptl", control = control_ptl(policy_vars = c("X"),
  depth, split.step, min.node.size, hybrid, search.depth),
  full_history = FALSE, alpha = 0.05, L = 10)
```

The function `control_ptl()` helps set the default control arguments for `type = "ptl"`. Similar functions are available for every policy learning type. The control argument `policy_vars` is a character vector or a list of character vectors further subsetting the history available to the decision tree model. The control arguments `depth`, `split.step`, `min.node.size`, and `search.depth` are directly passed to `policytree::policy_tree()`. Each of these arguments must be an integer or an integer vector. The control argument `hybrid` is a logical value indicating whether to use `policytree::policy_tree()` or `policytree::hybrid_policy_tree()`. The value of a user-specified policy or a policy learning algorithm can be estimated using `policy_eval()`. The evaluation can be based on inverse probability weighting or outcome regression. However, the default is to use the doubly robust value estimator given by Algorithm 4:

```
policy_eval(type = "dr", policy_data, policy, policy_learn,
  g_models = g_glm(~ X + B), g_full_history = FALSE,
  q_models = q_glm(~ A * X), q_full_history = FALSE, M = 10)
```

`g_models` and `g_full_history` specify the modeling of the  $g$ -functions. If `g_full_history = FALSE` and a single  $g$ -model is provided, a single Markov-type model across all stages is

Call	Method	Imports	Limitations
<code>g_empir</code>	Empirical (conditional) probabilities		
<code>g_glm/q_glm</code>	Generalized linear model	<code>stats</code>	<code>g_glm</code> : Dichotomous actions
<code>g_glmnet/q_glmnet</code>	Lasso and elastic-net regularized generalized linear models	<code>glmnet</code> (Friedman, Hastie, and Tibshirani 2010; Tay, Narasimhan, and Hastie 2023)	<code>g_glmnet</code> : Dichotomous actions
<code>g_rf/q_rf</code>	Random forests	<code>ranger</code> (Wright and Ziegler 2017)	
<code>g_sl/q_sl</code>	Super learner prediction algorithm	<code>SuperLearner</code> (Polley, LeDell, Kennedy, and Van der Laan 2025)	<code>g_sl</code> : Dichotomous actions

Table 2: Overview of available  $g$ -model and  $Q$ -model constructors.

fitted. In this case, a generalized linear model is fitted with a model matrix given the formula  $\sim X + B$ . If `g_full_history = TRUE` or `g_models` is a list, a  $g$ -function is fitted for each stage. Similarly, `q_models` and `q_full_history` specify the modeling of the  $Q$ -functions. A model is fitted at each stage. If `q_full_history = FALSE` and a single  $Q$ -model is provided, the model is reused at each stage with the same design. Alternatives to `g_glm()` and `q_glm()` are listed in Table 2. The models are created to save the design specifications, which is useful for cross-fitting.  $M$  is the number of folds in the cross-fitting procedure.

## 5. Examples

In this section, we go through two reproducible examples based on simulated datasets that illustrate various applications of the **polle** package.

In Section 5.1, we consider a single-stage problem. We demonstrate how the policy evaluation framework handles static policies to obtain estimates of causal effects. Furthermore, we evaluate the true optimal dynamic policy using highly adaptive nuisance models and use doubly robust blip learning to obtain an estimate of the same optimal policy. In Section 5.2, we study a problem with two fixed stages. We show how to create a policy data object from raw data in wide format and how to formulate the optimal dynamic policy over multiple stages. We use  $g$ -models and  $Q$ -models with custom data designs to evaluate the policy and showcase the use of policy trees.

Additional examples including handling a stochastic number of stages and multiple actions are available in the package vignettes for `policy_data()`, `policy_eval()`, and `policy_learn()`.

### 5.1. Single-stage problem

To illustrate the usage of the **polle** package, we first consider a single-stage problem. Here we consider data from a simulation where the optimal policy is known. We consider observed data from the directed acyclic graph (DAG) given in Figure 4.

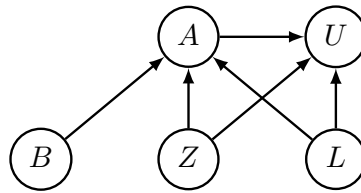


Figure 4: Single-stage problem with treatment variable  $A$ , utility  $U$ , and confounders  $B, Z, L$ .

The utility/reward/response is in this example defined as the conditional Gaussian distribution

$$U \mid Z, L, A \sim \mathcal{N}(Z + L + A \cdot \{\gamma Z + \alpha L + \beta\}, \sigma^2)$$

with independent state covariates/variables  $Z, L \sim \text{Uniform}([0, 1])$ , and treatment,  $A$ , defined by the logistic regression model

$$A \mid Z, L, B \sim \text{Bernoulli}(\text{expit}\{\kappa Z^{-2}(Z + L - 1) + \delta B\})$$

where  $B \sim \text{Bernoulli}(\pi)$  is an additional independent state covariate, and  $\text{expit}$  is the inverse logistic link function. Here we consider the choices  $\pi = 0.3, \kappa = 0.1, \Delta = 0.5, \alpha = 1, \beta = -2.5, \gamma = 3, \sigma = 1$ :

```

R> library("polle")
R> par0 <- c(p = 0.3, k = 0.1, d = 0.5, a = 1, b = -2.5, c = 3)
R> d <- sim_single_stage(n = 5e2, seed = 1, par = par0)
R> head(d, 4)

```

	Z	L	B	A	U
1	1.2879704	-1.4795962	0	1	-0.9337648
2	1.6184181	1.2966436	0	1	6.7506026
3	1.2710352	-1.0431352	0	1	-0.3377580
4	-0.2157605	0.1198224	1	0	1.4993427

The data is first transformed using `policy_data()` with instructions on which variables define the *action*, state *covariates* and the *utility*:

```

R> pd <- policy_data(d, action = "A", covariates = list("Z", "B", "L"),
+   utility = "U")
R> pd

```

Policy data with  $n = 500$  observations and maximal  $K = 1$  action stages.

Count of observed actions at a given stage:

stage	0	1	n
1	278	222	500

Baseline covariates:  
 State covariates: Z, B, L  
 Average utility: -0.98

### *Policy evaluation*

A single-stage *policy* is mapping from the history  $H = (B, Z, L)$  onto the set of actions  $\mathcal{A} = \{0, 1\}$ . It is possible to evaluate both user-defined policies as well as learning a policy from the data using **polle**. Here we first illustrate how to estimate the value of a *static policy* where all individuals are given action “1” irrespective of their covariate values. Policies are defined using `policy_def()`, which expects a function as input or, as here, a numeric vector specifying the static policy:

```
R> p1 <- policy_def(1, name = "(A=1)")
R> p1
```

```
Policy with argument(s)
policy_data
```

The policy can be applied to a `policy_data` object to get the individual actions:

```
R> p1(pd) |> head(3)
```

```
Key: <id, stage>
      id stage      d
  <int> <int> <char>
1:     1     1     1
2:     2     1     1
3:     3     1     1
```

Note that a policy applied to a `policy_data` object returns a `data.table` with keys `id` and `stage`. In this case, `policy_data()` has set the default `id` values.

The value of the policy can then be estimated using `policy_eval()`:

```
R> (pe1 <- policy_eval(pd, policy = p1))

      Estimate Std.Err   2.5% 97.5%   P-value
E[U(d)]: d=(A=1)  -2.674  0.2116 -3.089 -2.26 1.331e-36
```

This provides an estimate of the average potential outcome,  $E[U^{(a=1)}]$ . By default, a doubly robust estimator given by Algorithm 1 without cross-fitting is used to estimate the value. A logistic regression model with all main effects is used to model the  $g$ -function and a linear regression model with all interaction effects between the action and each of the state covariates is used to model the  $Q$ -function. We will later revisit how to estimate the value of the policy using flexible machine learning models and cross-fitting.

In the same way, we can estimate the value under the action “0”:

```
R> (pe0 <- policy_eval(pd, policy = policy_def(0, name = "(A=0)")))
```

```
      Estimate Std.Err   2.5%  97.5% P-value
E[U(d)]: d=(A=0) -0.02243 0.08326 -0.1856 0.1408 0.7877
```

Finally, the average treatment effect,  $ATE := E[U^{(a=1)} - U^{(a=0)}]$ , can then be estimated as:

```
R> estimate(merge(pe0, pe1), function(x) x[2] - x[1], labels = "ATE")
```

```
      Estimate Std.Err   2.5%  97.5%  P-value
ATE    -2.652  0.1737 -2.992 -2.312 1.236e-52
```

The function `lava::merge.estimate()` combines the fitted influence curve for each estimate (<https://CRAN.R-project.org/web/packages/lava/vignettes/influencefunction.html>). The influence curve matrix is available via `IC()`:

```
R> IC(merge(pe0, pe1)) |> head(3)
```

```
      E[U(d)]: d=(A=0) E[U(d)]: d=(A=1)
1      -0.08832404      0.6568576
2       2.61912976      9.6387445
3       0.27539152      1.0710632
```

The standard errors for the transformation  $f(x_1, x_2) = x_2 - x_1$  is then given by the delta method.

In this case, we know that the optimal decision boundary is defined by the hyperplane  $\gamma Z + \alpha L + \beta = 0$ . Again, we use `policy_def()` to define the optimal policy:

```
R> p_opt <- policy_def(
+   function(Z, L) 1 * ((par0["c"] * Z + par0["a"] * L + par0["b"]) > 0),
+   name = "optimal")
```

We estimate the value of the optimal policy using Algorithm 1. Specifically, we use M-fold cross-fitting and super learners for the  $g$ -function and  $Q$ -function including random forests regression and generalized additive models as implemented in the **SuperLearner** package:

```
R> set.seed(1)
R> policy_eval(pd, policy = p_opt,
+   g_models = g_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")),
+   q_models = q_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")), M = 5)
```

```
      Estimate Std.Err   2.5% 97.5%  P-value
E[U(d)]: d=optimal    0.373 0.1117 0.1539 0.592 0.0008451
```

### *Policy learning*

In most applications, the optimal policy is of course not known. Instead, we seek to estimate/learn the optimal policy from the data. The function `policy_learn()` constructs a policy learner. Here we specify a cross-fitted doubly robust blip learning algorithm almost identical to Algorithm 3:

```
R> pl <- policy_learn(type = "blip", L = 5,
+   control = control_blip(blip_models = q_glm(formula = ~ Z + L)))
```

The policy learner is restricted to  $V = (Z, L)$  given by the `formula` argument. Remember that `L` is the number of cross-fitting folds. The algorithm can be applied directly resulting in a policy object:

```
R> set.seed(1)
R> po <- pl(pd,
+   g_models = g_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")),
+   q_models = q_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")))
R> po
```

Policy object with list elements:

```
blip_functions, q_functions_cf, g_functions_cf, action_set,
stage_action_sets, alpha, threshold, K, folds, Z_1
Use 'get_policy' to get the associated policy.
```

The actions of the learned policy are available through `get_policy()` returning the associated policy function, which we can apply to a `policy_data` object:

```
R> get_policy(po)(pd) |> head(3)
```

```
Key: <id, stage>
      id stage      d
  <int> <int> <char>
1:     1     1     1
2:     2     1     1
3:     3     1     1
```

Alternatively, we can use `get_policy_functions()` to return the policy as a function of a `data.table`:

```
R> pf <- get_policy_functions(po, stage = 1)
R> pf(H = data.table(Z = c(1, -2), L = c(0.5, -1)))
```

```
[1] "1" "0"
```

The value of the learned policy can also be estimated directly via `policy_eval()`; see Algorithm 4:

```
R> set.seed(1)
R> pe <- policy_eval(pd, policy_learn = pl,
+   g_models = g_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")),
+   q_models = q_sl(SL.library = c("SL.glm", "SL.ranger", "SL.gam")), M = 5)
R> pe
```

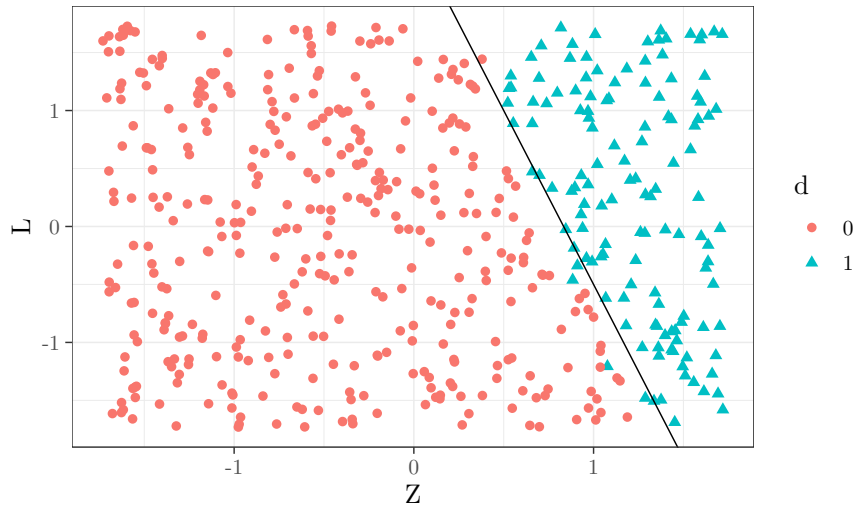


Figure 5: Fitted policy actions based on doubly robust blip learning. The black line shows the true optimal decision boundary.

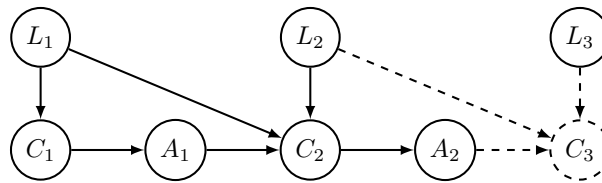


Figure 6: Two-stage problem with states  $(L_1, C_1)$ ,  $(L_2, C_2)$ ,  $L_3$  with exogenous component  $L_k$ ,  $k = 1, 2, 3$ . As the utility in this example does not depend on the last endogenous state,  $C_3$ , it can be omitted from the analysis.

	Estimate	Std.Err	2.5%	97.5%	P-value
$E[U(d)]: d=blip(\eta=0)$	0.39	0.1118	0.1708	0.6092	0.0004888

Note that the cross-fitting procedure is nested in this case, i.e.,  $M \times L$   $g$ -functions and  $Q$ -functions are fitted. The resulting policy actions are displayed in Figure 5 along with the true optimal decision boundary.

### 5.2. Two-stage problem

In this example, we consider a two-stage problem. An observation can be written as  $O := (S_1, A_1, S_2, A_2, S_3)$ , where  $S_1 = (C_1, L_1, U_1)$ ,  $S_2 = (C_2, L_2, U_2)$ , and  $S_3 = (L_3, U_3)$ . The state covariates (cost  $C_k$  and load  $L_k$ ) and action variables ( $A_k$ ) are associated with the DAG in Figure 6.

Specifically, the costs, loads and actions are given by the structural model

$$\begin{aligned}
 L_1 &\sim \mathcal{N}(0, 1) & C_1 | L_1 &\sim \mathcal{N}(L_1, 1) & A_1 | C_1 &\sim \text{Bernoulli}(\text{expit}(\beta C_1)) \\
 L_2 &\sim \mathcal{N}(0, 1) & C_2 | A_1, L_1 &\sim \mathcal{N}(\gamma L_1 + A_1, 1) & A_2 | C_2 &\sim \text{Bernoulli}(\text{expit}(\beta C_2)) \\
 L_3 &\sim \mathcal{N}(0, 1) & & & &
 \end{aligned}$$

for parameters  $\gamma, \beta \in \mathbb{R}$ . The rewards are given by

$$U_1 = L_1, \quad U_2 = A_1 \cdot C_1 + L_2, \quad U_3 = A_2 \cdot C_2 + L_3.$$

Remember that the utility is the sum of the rewards, i.e.,  $U = U_1 + U_2 + U_3$ . In this problem, we consider the parameter choices  $\gamma = 0.5, \beta = 1$ :

```
R> par0 <- c(gamma = 0.5, beta = 1)
R> d <- sim_two_stage(2e3, seed = 1, par = par0)
R> head(d[, -(1:2)], 3)
```

	L_1	C_1	A_1	L_2	C_2	A_2	L_3
	<num>	<num>	<int>	<num>	<num>	<int>	<num>
1:	0.9696772	1.711279	1	-0.7393434	2.424370	1	-0.83124340
2:	-2.1994065	-2.643124	0	0.4828756	-2.664728	0	-0.07151015
3:	1.9480938	2.061934	0	0.4803055	2.474761	1	0.40785209
	U_1	U_2	U_3				
	<num>	<num>	<num>				
1:	0.9696772	0.9719356	1.59312684				
2:	-2.1994065	0.4828756	-0.07151015				
3:	1.9480938	0.4803055	2.88261357				

The data is transformed using `policy_data()` with instructions on which variables define the *actions*, *covariates* and the *rewards* at each stage.

```
R> pd <- policy_data(d, action = c("A_1", "A_2"),
+   covariates = list(L = c("L_1", "L_2"), C = c("C_1", "C_2")),
+   utility = c("U_1", "U_2", "U_3"))
R> pd
```

Policy data with n = 2000 observations and maximal K = 2 action stages.

Count of observed actions at a given stage:

stage	action	0	1	n
1		1017	983	2000
2		819	1181	2000

Baseline covariates:

State covariates: L, C

Average utility: 0.84

### Policy evaluation

The optimal policy  $d_0 = (d_{0,1}, d_{0,2})$  is identified via the  $Q$ -functions. At stage 2, the  $Q$ -function is given by

$$\begin{aligned} Q_{0,2}(h_2, a_2) &= \mathbf{E}[U \mid H_2 = h_2, A_2 = a_2] \\ &= l_1 + a_1 c_1 + l_2 + a_2 c_2. \end{aligned}$$

Thus, the optimal policy at stage 2 is

$$\begin{aligned} d_{0,2}(h_2) &= \arg \max_{a_2 \in \{0,1\}} Q_2(h_2, a_2) \\ &= I\{c_2 > 0\}. \end{aligned}$$

At stage 1, the  $Q$ -function under the optimal policy at stage 2 is given by

$$\begin{aligned} Q_{0,1}^{d_0}(h_1, a_1) &= \mathbf{E}[Q_{0,2}(H_2, d_{0,2}(h_2)) \mid H_1 = h_1, A_1 = a_1] \\ &= l_1 + a_1 c_1 + \mathbf{E}[I\{C_2 > 0\} C_2 \mid L_1 = l_1, A_1 = a_1]. \end{aligned}$$

Let

$$\begin{aligned} \kappa(a_1, l_1) &= \mathbf{E}[I\{C_2 > 0\} C_2 \mid L_1 = l_1, A_1 = a_1] \\ &= \mathbf{E}[I\{C_2 > 0\} \mid L_1 = l_1, A_1 = a_1] \mathbf{E}[C_2 \mid L_1 = l_1, A_1 = a_1, C_2 > 0] \\ &= (1 - \Phi(-\{\gamma l_1 + a_1\})) \left( \gamma l_1 + a_1 + \frac{\phi(-\{\gamma l_1 + a_1\})}{1 - \Phi(-\{\gamma l_1 + a_1\})} \right). \end{aligned}$$

The optimal policy at stage 1 can now be written as

$$\begin{aligned} d_{0,1}(h_1) &= \arg \max_{a_1 \in \{0,1\}} Q_1(h_1, a_1) \\ &= I\{(c_1 + \kappa(1, l_1) - \kappa(0, l_1)) > 0\}. \end{aligned}$$

The basis for defining the optimal policy is the histories  $H_1 = (L_1, C_1)$  and  $H_2 = (L_1, C_1, A_1, L_2, C_2)$ , which are available via `get_history()`:

```
R> get_history(pd, stage = 1, full_history = TRUE)$H |> head(3)
```

Key: <id, stage>

	id	stage	L_1	C_1
	<int>	<num>	<num>	<num>
1:	1	1	0.9696772	1.711279
2:	2	1	-2.1994065	-2.643124
3:	3	1	1.9480938	2.061934

```
R> get_history(pd, stage = 2, full_history = TRUE)$H |> head(3)
```

Key: <id, stage>

	id	stage	A_1	L_1	L_2	C_1	C_2
	<int>	<num>	<char>	<num>	<num>	<num>	<num>
1:	1	2	1	0.9696772	-0.7393434	1.711279	2.424370
2:	2	2	0	-2.1994065	0.4828756	-2.643124	-2.664728
3:	3	2	0	1.9480938	0.4803055	2.061934	2.474761

We use the `policy_def()` function to define the optimal policy:

```
R> kappa <- function(mu) {
+   pnorm(q = -mu, lower.tail = FALSE) *
+   (mu + dnorm(-mu) / pnorm(-mu, lower.tail = FALSE))
+ }
R> p_opt <- policy_def(
+   list(function(C_1, L_1) {
+     1 * ((C_1 + kappa(par0[["gamma"]] * L_1 + 1) -
+       kappa(par0[["gamma"]] * L_1)) > 0)
+   },
+   function(C_2) {
+     1 * (C_2 > 0)
+   }),
+   full_history = TRUE, name = "optimal")
R> p_opt
```

```
Policy with argument(s)
policy_data
```

The optimal policy can be applied directly on the policy data:

```
R> p_opt(pd) |> head(3)
```

```
Key: <id, stage>
   id stage    d
<int> <int> <char>
1:    1     1    1
2:    1     2    1
3:    2     1    0
```

Doubly robust evaluation of the optimal policy requires modeling the  $g$ -functions and  $Q$ -functions. In this case, the  $g$ -function is repeated at each stage. Thus, we may combine  $(C_1, A_1)$  and  $(C_2, A_2)$  when fitting the  $g$ -function. The combined state histories and actions are available through the `get_history()` function with `full_history = FALSE`:

```
R> get_history(pd, full_history = FALSE)$H |> head(3)
```

```
Key: <id, stage>
   id stage      L      C
<int> <int> <num> <num>
1:    1     1 0.9696772 1.711279
2:    1     2 -0.7393434 2.424370
3:    2     1 -2.1994065 -2.643124
```

```
R> get_history(pd, full_history = FALSE)$A |> head(3)
```

```
Key: <id, stage>
   id stage    A
<int> <int> <char>
```

```

      <int> <int> <char>
1:      1      1      1
2:      1      2      1
3:      2      1      0

```

Similarly, when using `policy_eval()`, we can specify the structure of the used histories:

```

R> pe_opt <- policy_eval(pd, policy = p_opt,
+   g_models = g_glm(), g_full_history = FALSE,
+   q_models = list(q_glm(), q_glm()), q_full_history = TRUE)
R> pe_opt

```

```

              Estimate Std.Err  2.5% 97.5%  P-value
E[U(d)]: d=optimal   1.311 0.06578 1.182  1.44 2.067e-88

```

On closer inspection we see that a single  $g$ -model has been fitted across all stages:

```
R> get_g_functions(pe_opt)
```

```

$all_stages
$model

```

```
Call: NULL
```

```

Coefficients:
(Intercept)          L          C
      0.01591      0.03145      0.98013

```

```

Degrees of Freedom: 3999 Total (i.e. Null);  3997 Residual
Null Deviance:          5518
Residual Deviance: 4361      AIC: 4367

```

```

attr(,"full_history")
[1] FALSE

```

If `q_models` is not a list, the provided model is reused at each stage. In this case the full history is used at both stages:

```
R> get_q_functions(pe_opt)
```

```

$stage_1
$model

```

```
Call: NULL
```

```
Coefficients:
```

```
(Intercept)          A1          L_1          C_1          A1:L_1
      0.52415      0.44348      0.17462      0.09043      0.21854
      A1:C_1
      0.92899
```

```
Degrees of Freedom: 1999 Total (i.e. Null); 1994 Residual
Null Deviance:          6029
Residual Deviance: 2772          AIC: 6343
```

```
$stage_2
$model
```

```
Call: NULL
```

```
Coefficients:
```

```
(Intercept)          A1          A_11          L_1          L_2
      -0.014697      0.140420      -0.148007      -0.118571      0.002233
      C_1          C_2          A1:A_11          A1:L_1          A1:L_2
      0.124817      -0.031178      0.046876      0.171946      0.023246
      A1:C_1          A1:C_2
      -0.113314      0.944778
```

```
Degrees of Freedom: 1999 Total (i.e. Null); 1988 Residual
Null Deviance:          3580
Residual Deviance: 1881          AIC: 5579
```

```
attr(, "full_history")
[1] TRUE
```

Note that in practice only the residual value is used as input to the (residual)  $Q$ -models, i.e.,

$$Q_{0,2,\text{res}}(h_2, a_2) := \mathbb{E}[U_3 \mid H_2 = h_2, A_2 = a_2] \\ = a_2 c_2$$

$$Q_{0,1,\text{res}}^{d_0}(h_1, a_1) := \mathbb{E}[U_2 + Q_{0,2,\text{res}}(H_2, d_{0,2}(h_2)) \mid H_1 = h_1, A_1 = a_1] \\ = a_1 c_1 + \mathbb{E}[I\{C_2 > 0\} C_2 \mid L_1 = l_1, A_1 = a_1].$$

The fitted values of the  $g$ -functions and  $Q$ -functions are easily extracted using `predict()`:

```
R> predict(get_g_functions(pe_opt), pd) |> head(3)
```

```
Key: <id, stage>
  id stage    g_0    g_1
  <int> <int> <num> <num>
1:   1     1 0.15139841 0.84860159
2:   1     2 0.08557919 0.91442081
3:   2     1 0.93363059 0.06636941
```

```
R> predict(get_q_functions(pe_opt), pd) |> head(3)
```

```
Key: <id, stage>
```

	id	stage	Q_0	Q_1
	<int>	<int>	<num>	<num>
1:	1	1	1.817896	4.063055
2:	1	2	1.800290	4.233710
3:	2	1	-2.298324	-4.790946

### Policy learning

A  $V$ -restricted policy can be estimated via the `policy_learn()` function. In this case we use recursive doubly robust value search based on the **policytree** package; see Algorithm 2:

```
R> pl <- policy_learn(type = "ptl",
+   control = control_ptl(policy_vars = c("C", "L")),
+   full_history = FALSE, L = 5)
```

The policy learner is restricted to  $V = (C, L)$  given by the `policy_vars` argument. The learner can be applied directly:

```
R> po <- pl(pd, g_models = g_glm(), g_full_history = FALSE,
+   q_models = q_glm(), q_full_history = TRUE)
R> get_policy(po)(pd) |> head(3)
```

```
Key: <id, stage>
```

	id	stage	d
	<int>	<int>	<char>
1:	1	1	1
2:	1	2	1
3:	2	1	0

Or, the value of the policy learning procedure can be estimated directly using `policy_eval()`:

```
R> set.seed(1)
R> pe <- policy_eval(pd, policy_learn = pl, g_models = g_glm(),
+   g_full_history = FALSE, q_models = q_glm())
R> pe
```

	Estimate	Std.Err	2.5%	97.5%	P-value
E[U(d)]: d=ptl(eta=0)	1.385	0.0809	1.226	1.544	1.068e-65

The associated policy objects are also saved for closer inspection; see Figure 7:

```
R> po <- get_policy_object(pe)
R> po$ptl_objects
```

```

$stage_1
$stage_1$threshold_0
policy_tree object
Tree depth: 2
Actions: 1: 0 2: 1
Variable splits:
(1) split_variable: C split_value: -3.14122
    (2) split_variable: L split_value: -1.67452
        (4) * action: 1
        (5) * action: 2
    (3) split_variable: C split_value: -0.274346
        (6) * action: 1
        (7) * action: 2

$stage_2
policy_tree object
Tree depth: 2
Actions: 1: 0 2: 1
Variable splits:
(1) split_variable: C split_value: -0.747456
    (2) split_variable: C split_value: -0.811175
        (4) * action: 1
        (5) * action: 2
    (3) split_variable: C split_value: 0.0237423
        (6) * action: 1
        (7) * action: 2

```

## 6. K-2 literacy intervention

In this section, we demonstrate the application of **polle** on a real data example. To ensure reproducibility, we work with a dataset publicly available in the Harvard Dataverse at <https://dataverse.harvard.edu/>. Specifically, we use the kindergarten to second-grade literacy intervention dataset (Kim, Asher, Burkhauser, Mesite, and Leyva 2019a), funded by the Chan Zuckerberg Initiative. The dataset is documented and analyzed in Kim, Asher, Burkhauser, Mesite, and Leyva (2019b). The following analysis is conducted independently of the original study, and we take full responsibility for any misinterpretations or errors.

The dataset contains records of 273 students from kindergarten to second grade associated with 16 teachers. The study seeks to investigate the treatment effect of assigning two types of print texts (10 texts/books in each group) for the students to read over the summer break. All students received training before the summer break, and the investigators used a mobile app to engage and monitor each student. At stage 1, each classroom associated with a given teacher was randomly assigned to read either conceptually coherent texts (CCT) or leveled text (LT). At an intermediate time point during the summer break (stage 2), if a student had completed at least one activity on the mobile app, the student was classified as a responder and no further actions were initiated. However, all non-responders were subject to gamification

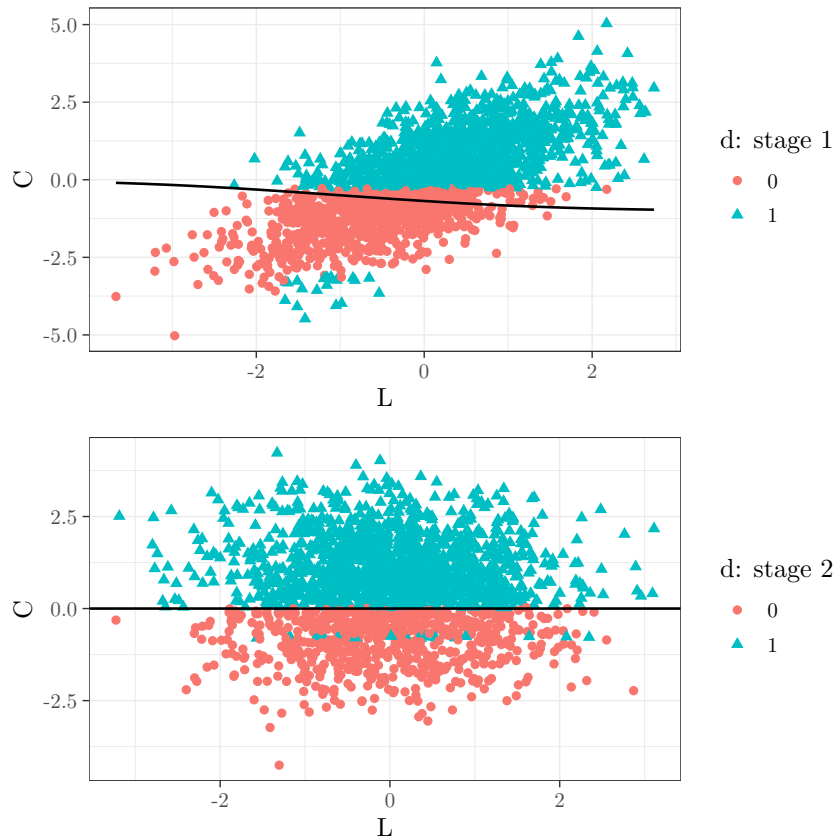


Figure 7: Fitted policy actions based on policy tree learning. The black lines show the true optimal decision boundaries.

of the app, and the parents were randomly selected to receive text messages with reminders, information, and encouragement. The main outcome of the study was the reading measure of academic progress Rasch unit (MAP RIT) score. The score was measured before and after the summer break (referred to as spring and fall). Of the 273 students enrolled in the study, 56 students have missing MAP RIT scores. As in the original study, we conduct a complete case analysis.

We start by loading the data and conducting some basic transformations, which we include for reproducibility. Note that we define the utility as the difference between the spring and fall MAP RIT scores. A description of the variables can be found in Table 3.

```
R> library("readstata13")
R> d <- readstata13::read.dta13("k2smart_public.dta")
R> d <- transform(d, utility = as.numeric(fa_maprit) - as.numeric(sp_maprit),
+   cct = as.logical(cct), responder = as.logical(responder),
+   text = as.logical(text), maprit = as.numeric(sp_maprit),
+   teacher = as.character(t_id_public),
+   attend = as.numeric(sp_pctattend_yr), trc = as.numeric(sp_trcbook_num),
+   dib = as.numeric(sp_dib_score), ell = as.logical(ell),
+   iep = as.logical(iep), grade = as.character(grade_final),
+   male = as.logical(male), familynight = as.logical(familynight))
```

Variable	Type	Description
<code>utility</code>	Numeric	Difference between the spring and fall MAP RIT scores
<code>cct</code>	Logical	If <code>TRUE</code> , the student received conceptually coherent texts (CCT). If <code>FALSE</code> , the student received leveled text (LT).
<code>responder</code>	Logical	Response indicator
<code>text</code>	Logical	If <code>TRUE</code> , the parents received text messages.
<code>maprit</code>	Numeric	Spring MAP RIT score.
<code>teacher</code>	Character string	Teacher ID.
<code>attend</code>	Numeric	Student attendance percentage.
<code>trc</code>	Numeric	Spring text reading comprehension score.
<code>dib</code>	Numeric	Spring dynamic indicators of basic early literacy skills (DIBELS) score.
<code>ell</code>	Logical	If <code>TRUE</code> , the student received English-language learner services.
<code>iep</code>	Logical	If <code>TRUE</code> , the student had an individualized education plan.
<code>grade</code>	Character string	Student grade: kindergarten (0), first grade (1), second grade (2).
<code>male</code>	Logical	If <code>TRUE</code> , the student was male.
<code>familynight</code>	Logical	If <code>TRUE</code> , the family of the student attended family night.

Table 3: Variable descriptions.

As described, we make a complete case analysis:

```
R> d <- subset(d, !is.na(utility))
```

Finally, we create a stage 1 and stage 2 treatment variable:

```
R> d <- transform(d, A_1 = ifelse(cct, "cct", "lt"),
+   A_2 = ifelse(responder, "continue", ifelse(text, "text", "nottext")))
```

The `policy_data()` function is used to create a policy data object. Note that `responder` is a stage 2 state covariate:

```
R> pd <- policy_data(d, action = c("A_1", "A_2"), utility = "utility",
+   baseline = c("maprit", "male", "ell", "iep", "attend", "trc", "dib",
+   "grade", "familynight"),
+   covariates = list(responder = c(NA, "responder")))
R> print(pd)
```

Policy data with `n = 217` observations and maximal `K = 2` action stages.

Count of observed actions at a given stage:

```
      action
stage cct continue  lt notext text  n
  1  112         0 105     0    0 217
  2   0         36  0    86   95 217
```

```
Baseline covariates: maprit, male, ell, iep, attend, trc,
dib, grade, familynight
State covariates: responder
Average utility: 2.7
```

Since student responders are not randomized at stage 2, only 4 static realistic policies exist:

```
R> actions <- list(c("cct", "text"), c("cct", "notext"),
+ c("lt", "text"), c("lt", "notext"))
R> static_policies <- lapply(actions, function(a) {
+   policy_def(list(function(...) a[1],
+     function(responder) ifelse(responder, "continue", a[2])),
+     name = paste(a, collapse = "_"))
+ })
R> head(static_policies[[1]](pd), 4)
```

```
Key: <id, stage>
      id stage      d
  <int> <int> <char>
1:     1     1    cct
2:     1     2   text
3:     2     1    cct
4:     2     2   text
```

We begin the analysis by comparing the policy value for each of the 4 static policies. First, we consider a basic inverse probability weighting estimator:

```
R> gm <- list(g_empir(~ 1), g_empir(~ responder))
R> pe_static_policies_ipw <- lapply(static_policies, function(p) {
+   policy_eval(pd, policy = p, g_models = gm, type = "ipw")
+ })
R> do.call("merge", pe_static_policies_ipw)
```

	Estimate	Std.Err	2.5%	97.5%	P-value
E[U(d)]: d=cct_text	2.920	1.159	0.64838	5.193	0.011760
-----					
E[U(d)]: d=cct_notext	1.737	1.109	-0.43707	3.912	0.117353
-----					
E[U(d)]: d=lt_text	3.573	1.120	1.37793	5.769	0.001422
-----					
E[U(d)]: d=lt_notext	2.504	1.322	-0.08728	5.095	0.058233

Note that the reported standard errors are valid because the  $g$ -models are known in a randomized trial. The  $g$ -models specified by the function `g_empir()` compute the (conditional) empirical probabilities and match them to each student:

```
R> print(get_g_functions(pe_static_policies_ipw[[1]]))
```

```

$stage_1
$tab
      A empir_prob
  <char>    <num>
1:    cct    0.516129
2:    lt     0.483871

$v
character(0)

$stage_2
$tab
Index: <A>
      A responder empir_prob
  <char>    <lgcl>    <num>
1: continue    TRUE  1.0000000
2:  notext     FALSE  0.4751381
3:    text     FALSE  0.5248619

$v
[1] "responder"

attr(,"full_history")
[1] FALSE

R> predict(get_g_functions(pe_static_policies_ipw[[1]]),
+   pd)[c(1, 2, 43, 44), ]

Key: <id, stage>
  id stage  g_cct g_continue  g_lt g_notext  g_text
  <int> <int>  <num>    <num>    <num>    <num>    <num>
1:    1    1 0.516129          0 0.483871 0.0000000 0.0000000
2:    1    2 0.000000          0 0.000000 0.4751381 0.5248619
3:   22    1 0.516129          0 0.483871 0.0000000 0.0000000
4:   22    2 0.000000          1 0.000000 0.0000000 0.0000000

```

Efficiency of the policy value estimates can be increased by using the doubly robust value scores. As we are fitting 25 sets of nuisance models, we parallelize the computations via the `future.apply` (Bengtsson 2021) package. The variable names used to specify the nuisance model are available via `get_history_names()`:

```

R> get_history_names(pd, stage = 1)

[1] "responder_1" "maprit"      "male"      "ell"
[5] "iep"         "attend"     "trc"       "dib"
[9] "grade"      "familynight"

```

```
R> get_history_names(pd, stage = 2)

[1] "A_1"          "responder_1" "responder_2" "maprit"
[5] "male"         "ell"          "iep"          "attend"
[9] "trc"          "dib"          "grade"        "familynight"
```

With this help, we can easily specify the  $Q$ -models using the **SuperLearner** package and plug them into the `policy_eval()` function:

```
R> sl_lib <- c("SL.mean", "SL.glm", "SL.gam", "SL.ranger", "SL.nnet")
R> qm <- list(q_sl(formula = ~ . - responder_1, SL.library = sl_lib),
+   q_sl(formula = ~ . - responder_1 - responder_2, SL.library = sl_lib)
+ )
R> library("future.apply")
R> plan(list(tweak("multisession", workers = 4)))
R> pe_static_policies_dr <- lapply(static_policies, function(p) {
+   set.seed(1)
+   policy_eval(pd, policy = p, g_models = gm, q_models = qm,
+     q_full_history = TRUE, type = "dr", M = 25)
+ })
R> plan("sequential")
R> print(do.call("merge", pe_static_policies_dr))
```

	Estimate	Std.Err	2.5%	97.5%	P-value
E[U(d)]: d=cct_text	2.825	0.9825	0.8995	4.751	0.004034
-----					
E[U(d)]: d=cct_notext	2.289	1.0766	0.1787	4.399	0.033504
-----					
E[U(d)]: d=lt_text	3.554	1.0959	1.4062	5.702	0.001183
-----					
E[U(d)]: d=lt_notext	1.946	1.2134	-0.4323	4.324	0.108790

So far, the reported standard errors have been overly optimistic because we ignored the random teacher/classroom effect. Luckily, when working with influence curves, it is easy to adjust for these types of dependencies. When estimating the variance, we first sum all of the influence curve terms related to each teacher. The resulting compounded influence curve terms are then independent and the variance is computed in the usual fashion. This approach is similar to computing clustered standard errors ([Liang and Zeger 1986](#)). The method is implemented in the `estimate()` function from the **lava** package ([Holst and Budtz-Jørgensen 2013](#)):

```
R> library("lava")
R> (est <- estimate(do.call("merge", pe_static_policies_dr), id = d$teacher))
```

	Estimate	Std.Err	2.5%	97.5%	P-value
E[U(d)]: d=cct_text	2.825	1.0122	0.8414	4.809	5.252e-03
E[U(d)]: d=cct_notext	2.289	0.8869	0.5506	4.027	9.856e-03
E[U(d)]: d=lt_text	3.554	0.7528	2.0788	5.030	2.340e-06
E[U(d)]: d=lt_notext	1.946	1.4342	-0.8652	4.757	1.749e-01

As is already evident, none of the static policies are statistically different in terms of marginal value. For completeness, we conduct a chi-square test:

```
R> pdiff <- function(n) lava::contr(lapply(seq(n - 1), \(x) seq(x, n)))
R> estimate(est, f = pdiff(4))
```

	Estimate	Std.Err	2.5%	97.5%	P-value
[E[U(d)]: d=cct_text]....	0.5364	0.939	-1.304	2.3769	0.5679
[E[U(d)]: d=cct_text].....1	-0.7290	1.024	-2.736	1.2783	0.4766
[E[U(d)]: d=cct_text].....2	0.8794	1.671	-2.396	4.1550	0.5988
[E[U(d)]: d=cct_notex....	-1.2653	1.020	-3.264	0.7337	0.2147
[E[U(d)]: d=cct_notex.....1	0.3430	1.659	-2.909	3.5947	0.8362
[E[U(d)]: d=lt_text] ....	1.6083	1.887	-2.091	5.3076	0.3941

Null Hypothesis:

```
[E[U(d)]: d=cct_text] - [E[U(d)]: d=cct_notext] = 0
[E[U(d)]: d=cct_text] - [E[U(d)]: d=lt_text] = 0
[E[U(d)]: d=cct_text] - [E[U(d)]: d=lt_notext] = 0
[E[U(d)]: d=cct_notext] - [E[U(d)]: d=lt_text] = 0
[E[U(d)]: d=cct_notext] - [E[U(d)]: d=lt_notext] = 0
[E[U(d)]: d=lt_text] - [E[U(d)]: d=lt_notext] = 0
```

```
chisq = 1.3022, df = 3, p-value = 0.7286
```

The function `conditional()` allows the user to easily compute the conditional policy value estimates based on categorical baseline covariates. Here, we group by the baseline covariate `male` for the static CCT text policy (`cct_text`):

```
R> estimate(conditional(pe_static_policies_dr[[1]], pd, "male"),
+   id = d$teacher)
```

	Estimate	Std.Err	2.5%	97.5%	P-value
male:FALSE	1.434	1.243	-1.003	3.870	0.248729
male:TRUE	4.484	1.602	1.344	7.623	0.005125

Even though none of the static policies have a marginal treatment effect, we may hope to find group-specific treatment effects. To investigate further, we specify a selection of doubly robust  $V$ -restricted  $Q$ -learners and estimate the cross-fitted value of the fitted policies.

We formulate simple linear  $QV$ -models using the `q_glm()` function as we do not expect to be able to find complex non-linear treatment associations in this relatively small dataset.

```
R> qvm_formulas <- list(
+   qvm_1 = list(~ 1, ~ A_1),
+   qvm_2 = list(~ maprit, ~ A_1 + maprit),
+   qvm_3 = list(~ male, ~ A_1 + male),
+   qvm_4 = list(~ grade, ~ A_1 + grade),
+   qvm_5 = list(~ male + maprit, ~ A_1 + male + maprit),
+   qvm_6 = list(~ male + grade + maprit, ~ A_1 + grade + male + maprit)
+ )
```

```
R> qvm <- lapply(qvm_formulas, function(form) {
+   list(q_glm(form[[1]]), q_glm(form[[2]]))
+ })
```

The  $Q$ -models are then passed to the controls of the `policy_learn()` function. Importantly, note that `alpha` is set to 0.01 in order to account for the degenerate structure of the data; A student responder always continue the treatment in stage 2.

```
R> pl_drql <- mapply(qvm, names(qvm), FUN = function(qv, name) {
+   policy_learn(type = "drql", control = control_drql(qv_models = qv),
+     full_history = TRUE, alpha = 0.01, L = 25,
+     cross_fit_g_models = FALSE, name = name)
+ })
```

The value of the fitted policies are cross-fitted using `policy_eval()`:

```
R> plan(list(tweak("multisession", workers = 4)))
R> set.seed(1)
R> pe_drql <- lapply(pl_drql, function(pl) {
+   set.seed(1)
+   policy_eval(pd, policy_learn = pl, g_models = gm, q_models = qm,
+     q_full_history = TRUE, type = "dr", M = 25)
+ })
R> plan("sequential")
R> estimate(do.call(what = "merge", unname(pe_drql)), id = d$teacher)
```

	Estimate	Std.Err	2.5%	97.5%	P-value
E[U(d)]: d=drql	3.063	0.7239	1.6445	4.482	2.319e-05
E[U(d)]: d=drql.1	1.228	1.0378	-0.8057	3.262	2.366e-01
E[U(d)]: d=drql.2	3.382	1.4196	0.5999	6.165	1.720e-02
E[U(d)]: d=drql.3	2.480	1.3335	-0.1335	5.094	6.291e-02
E[U(d)]: d=drql.4	1.885	1.0177	-0.1092	3.880	6.393e-02
E[U(d)]: d=drql.5	2.055	1.4455	-0.7776	4.888	1.550e-01

None of the fitted policies show a gain in value compared to the static policy `lt_text`. However, we might still want to study a possible male treatment interaction further (`qvm_3`). We fit policy learner 3 on the complete dataset and summarize the dictated actions:

```
R> set.seed(1)
R> po_drql_male <- pl_drql[["qvm_3"]](pd, g_models = gm, q_models = qm,
+   q_full_history = TRUE)
R> pa_drql_male <- get_policy(po_drql_male)(pd)
R> head(pa_drql_male, 4)
```

```
Key: <id, stage>
      id stage      d
<int> <int> <char>
```

```

1:    1    1    cct
2:    1    2    text
3:    2    1    cct
4:    2    2    text

```

```

R> pa_drql_male <- merge(pa_drql_male, get_history(pd)$H)
R> pa_drql_male[, .N, list(stage, male, d)][order(stage, male, d)]

```

```

      stage  male      d      N
      <int> <lgcl> <char> <int>
1:      1 FALSE      lt     118
2:      1  TRUE      cct     99
3:      2 FALSE continue    23
4:      2 FALSE      text    95
5:      2  TRUE continue    13
6:      2  TRUE      text    86

```

Thus, the fitted policy suggests that males receive CCT and females receive LT at stage 1, and that all non-responders get text messages at stage 2. The dictated actions are easily plotted using the `get_history()` and `get_policy()` functions, which results in Figure 8:

```

R> plot_data <- get_history(pd)$H
R> plot_data <- merge(plot_data, get_policy(po_drql_6)(pd),
+   by = c("id", "stage"))
R> library("ggplot2")
R> ggplot(plot_data) +
+   geom_point(aes(x = grade, y = maprit, color = d)) +
+   facet_wrap(~ stage + male, labeller = "label_both") +
+   theme_bw()

```

We end this analysis by emphasizing the importance of cross-fitting the policy learner because it is easy to overfit an optimal policy. We showcase this by fitting the most complex of the considered policy learners:

```

R> po_drql_6 <- pl_drql[["qvm_6"]](pd, g_models = gm, q_models = qm,
+   q_full_history = TRUE)

```

If we just plug in the resulting fitted policy, we get an overly optimistic estimate of the value.

```

R> plan(list(tweak("multisession", workers = 4)))
R> set.seed(1)
R> pe_plugin <- policy_eval(pd, policy = get_policy(po_drql_6),
+   g_models = gm, q_models = qm, q_full_history = TRUE, type = "dr",
+   M = 25, name = "qvm_6_plugin")
R> estimate(pe_plugin + pe_drql[["qvm_6"]], id = d$teacher)

```

	Estimate	Std.Err	2.5%	97.5%	P-value
qvm_6_plugin: d=drql	4.912	1.299	2.3656	7.458	0.0001562
E[U(d)]: d=drql	2.055	1.445	-0.7776	4.888	0.1550300

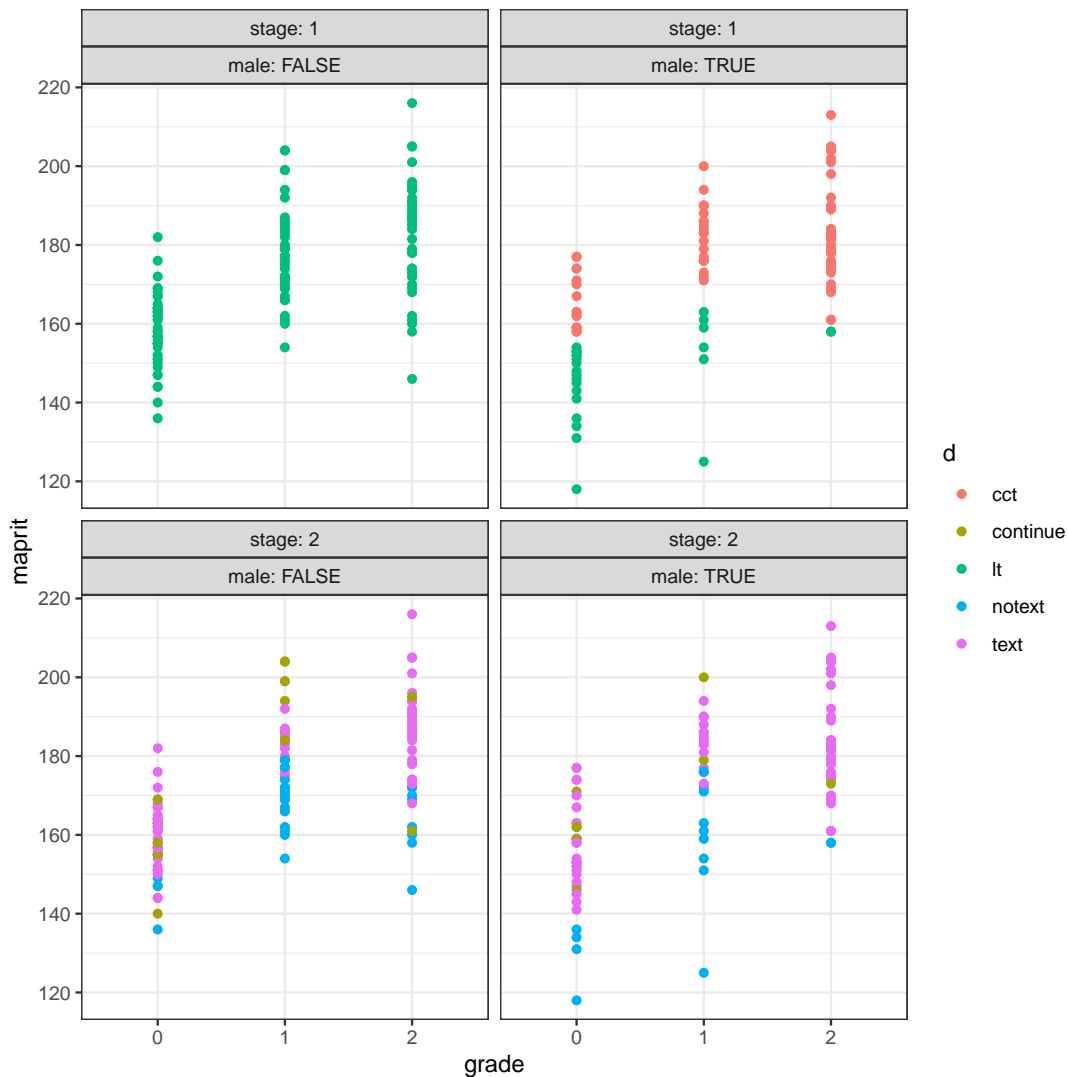


Figure 8: Plot of the fitted policy actions  $d$  for the K-2 literacy intervention example.

## 7. Summary and discussion

The **polle** package is the first unifying R package for learning and evaluating policies. The package efficiently handles cross-fitting of the nuisance models and provides protection against (near) positivity violations. Also, to our knowledge, **polle** contains the first implementation of doubly robust restricted  $Q$ -learning, which can serve as a sensible benchmark for all other learning methods.

Of course, **polle** has its limitations. Future work to be included in the package includes the handling of missing data and (right) censored observations. The **event** variable included in the policy data object can be extended to specify missing or censored data similar to that of the **Surv()** function in the **survival** package. Additional models for the censoring distribution would need to be included. Optimal policies for survival outcomes have been studied in Bai, Tsiatis, Lu, and Song (2017); Díaz, Savenkov, and Ballman (2018). See also

Cui, Kosorok, Sverdrup, Wager, and Zhu (2023); Xu, Ignatiadis, Sverdrup, Fleming, Wager, and Shah (2023); Steingrimsson, Diao, Molinaro, and Strawderman (2016).

In our work, we only consider the maximization of a scalar utility value. However, in some applications a multi-dimensional value vector may more naturally be of interest. In such cases, the set of Pareto efficient policies can be formulated. An important example would be the task of maximizing the utility subject to variance constraints in order to learn robust policies. This is closely related to introducing a penalty term to the loss function, and will be the subject of future developments to the **polle** package.

The package is available directly from the Comprehensive R Archive Network (CRAN) (Nordland and Holst 2026). We believe the package will provide practitioners with much easier access to a broad range of policy learning methods and hope that it may also serve as a framework for benchmarking as well as implementing new methods for researchers in the policy learning field. We invite collaboration on the future development of the package via pull requests to the GitHub repository <https://github.com/AndreasNordland/polle/>.

## Computational details

The results in this paper were obtained using R 4.5.2 (R Core Team 2026) with the **polle** 1.6.4 package, the **SuperLearner** 2.0-40 package, the **policytree** 1.2.3 package, the **readstata13** 0.11.0 package (Garbuszus and Jeworutzki 2025), the **future.apply** 1.20.2 package, the **lava** 1.8.2 package, the **data.table** 1.18.2.1 package (Barrett *et al.* 2026) and the **ggplot2** 4.0.2 package (Wickham 2016). R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/>.

## Acknowledgments

This work was supported by a grant from InnovationsFonden Danmark (case no. 8053-00096B).

## References

- Athey S, Tibshirani J, Wager S (2019). “Generalized Random Forests.” *The Annals of Statistics*, **47**(2), 1148–1178. doi:10.1214/18-aos1709.
- Athey S, Wager S (2021). “Policy Learning with Observational Data.” *Econometrica*, **89**(1), 133–161. doi:10.3982/ecta15732.
- Bai X, Tsiatis AA, Lu W, Song R (2017). “Optimal Treatment Regimes for Survival Endpoints Using a Locally-Efficient Doubly-Robust Estimator from a Classification Perspective.” *Lifetime Data Analysis*, **23**(4), 585–604. doi:10.1007/s10985-016-9376-x.
- Barrett T, Dowle M, Srinivasan A, Gorecki J, Chirico M, Hocking T, Schwendinger B, Krylov I (2026). **data.table: Extension of data.frame**. doi:10.32614/CRAN.package.data.table. R package version 1.18.2.1.

- Battocchi K, Dillon E, Hei M, Lewis G, Oka P, Oprescu M, Syrgkanis V (2019). **EconML**: A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. Python package version 0.16.0, URL <https://pypi.org/project/econml/>.
- Bengtsson H (2021). “A Unifying Framework for Parallel and Distributed Processing in R Using Futures.” *The R Journal*, **13**(2), 208–227. doi:10.32614/RJ-2021-048.
- Chakraborty B, Laber EB, Zhao YQ (2014). “Inference about the Expected Performance of a Data-Driven Dynamic Treatment Regime.” *Clinical Trials*, **11**(4), 408–417. doi:10.1177/1740774514537727.
- Chakraborty B, Moodie EEM (2013). *Statistical Methods for Dynamic Treatment Regimes*. Springer-Verlag, New York. doi:10.1007/978-1-4614-7428-9.
- Chen Y, Liu Y, Zeng D, Wang Y (2020). **DTRlearn2**: Statistical Learning Methods for Optimizing Dynamic Treatment Regimes. doi:10.32614/CRAN.package.dtrlearn2. R package version 1.1.
- Chernozhukov V, Chetverikov D, Demirer M, Duflo E, Hansen C, Newey W, Robins J (2018). “Double/Debiased Machine Learning for Treatment and Structural Parameters.” *The Econometrics Journal*, **21**(1), C1–C68. doi:10.1111/ectj.12097.
- Cui Y, Kosorok MR, Sverdrup E, Wager S, Zhu R (2023). “Estimating Heterogeneous Treatment Effects with Right-Censored Data via Causal Survival Forests.” *Journal of the Royal Statistical Society B*, **85**(2), 179–211. doi:10.1093/jrsssb/qkac001.
- Díaz I, Savenkov O, Ballman K (2018). “Targeted Learning Ensembles for Optimal Individualized Treatment Rules with Time-to-Event Outcomes.” *Biometrika*, **105**(3), 723–738. doi:10.1093/biomet/asy017.
- Ertefaie A, Almirall D, Huang L, Dziak JJ, Wagner AT, Murphy SA (2012). *SAS PROC QLEARN Users’ Guide (Version 1.0.0)*. University Park: The Methodology Center, Penn State. URL <http://methodology.psu.edu/>.
- Friedman J, Hastie T, Tibshirani R (2010). “Regularization Paths for Generalized Linear Models via Coordinate Descent.” *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.
- Garbuszus JM, Jeworutzki S (2025). **readstata13**: Import Stata Data Files. doi:10.32614/CRAN.package.readstata13. R package version 0.11.0.
- Goldberg Y, Kosorok MR (2012). “Q-Learning with Censored Data.” *The Annals of Statistics*, **40**(1), 529–560. doi:10.1214/12-aos968.
- Hernán MA, Robins JM (2020). *Causal Inference: What If*. Chapman & Hall/CRC, Boca Raton.
- Hines O, Dukes O, Diaz-Ordaz K, Vansteelandt S (2022). “Demystifying Statistical Learning Based on Efficient Influence Functions.” *The American Statistician*, **76**(3), 292–304. doi:10.1080/00031305.2021.2021984.

- Hirano K, Porter JR (2012). “Impossibility Results for Nondifferentiable Functionals.” *Econometrica*, **80**(4), 1769–1790. doi:10.3982/ecta8681.
- Holloway ST, Laber EB, Linn KA, Zhang B, Davidian M, Tsiatis AA (2025). **DynTxRegime: Methods for Estimating Optimal Dynamic Treatment Regimes.** doi:10.32614/CRAN.package.dynTxregime. R package version 4.16.
- Holst KK, Budtz-Jørgensen E (2013). “Linear Latent Variable Models: The **lava**-Package.” *Computational Statistics*, **28**(4), 1385–1452. doi:10.1007/s00180-012-0344-y.
- Kennedy EH (2023). “Towards Optimal Doubly Robust Estimation of Heterogeneous Causal Effects.” *Electronic Journal of Statistics*, **17**(2), 3008–3049. doi:10.1214/23-ejs2157.
- Kim JS, Asher CA, Burkhauser M, Mesite L, Leyva D (2019a). *Replication Data For: Using a Sequential Multiple Assignment Randomized Trial (SMART) to Develop an Adaptive K-2 Literacy Intervention with Personalized Print Texts and App-Based Digital Activities.* doi:10.7910/dvn/avw6kb. Harvard Dataverse, V1, UNF:6:GHFKOIIxYJ7tBJQT3vK8g==.
- Kim JS, Asher CA, Burkhauser M, Mesite L, Leyva D (2019b). “Using a Sequential Multiple Assignment Randomized Trial (SMART) to Develop an Adaptive K-2 Literacy Intervention with Personalized Print Texts and App-Based Digital Activities.” *AERA Open*, **5**(3). doi:10.1177/2332858419872701.
- Künzel SR, Sekhon JS, Bickel PJ, Yu B (2019). “Metalearners For Estimating Heterogeneous Treatment Effects Using Machine Learning.” *Proceedings of the National Academy of Sciences of the United States of America*, **116**(10), 4156–4165. doi:10.1073/pnas.1804597116.
- Lewis G, Syrgkanis V (2021). “Double/Debiased Machine Learning for Dynamic Treatment Effects.” In *Advances in Neural Information Processing Systems*, volume 34, pp. 22695–22707.
- Liang KY, Zeger SL (1986). “Longitudinal Data Analysis Using Generalized Linear Models.” *Biometrika*, **73**(1), 13–22. doi:10.1093/biomet/73.1.13.
- Liu Y, Wang Y, Kosorok MR, Zhao Y, Zeng D (2018). “Augmented Outcome-Weighted Learning for Estimating Optimal Dynamic Treatment Regimens.” *Statistics in Medicine*, **37**(26), 3776–3788. doi:10.1002/sim.7844.
- Luedtke A, Chambaz A (2020). “Performance Guarantees for Policy Learning.” *Annales De l’Institut Henri Poincaré, Probabilités et Statistiques*, **56**(3), 2162–2188. doi:10.1214/19-aihp1034.
- Luedtke A, Chung I (2024). “One-Step Estimation of Differentiable Hilbert-Valued Parameters.” *The Annals of Statistics*, **52**(4), 1534–1563. doi:10.1214/24-aos2403.
- Luedtke AR, Van der Laan MJ (2016a). “Statistical Inference for the Mean Outcome under a Possibly Non-Unique Optimal Treatment Strategy.” *The Annals of Statistics*, **44**(2), 713–742. doi:10.1214/15-aos1384.

- Luedtke AR, Van der Laan MJ (2016b). “Super-Learning of an Optimal Dynamic Treatment Rule.” *The International Journal of Biostatistics*, **12**(1), 305–332. doi:10.1515/ijb-2015-0052.
- Nie X, Wager S (2021). “Quasi-Oracle Estimation of Heterogeneous Treatment Effects.” *Biometrika*, **108**(2), 299–319. doi:10.1093/biomet/asaa076.
- Nordland A, Holst KK (2026). **polle**: *Policy Learning*. doi:10.32614/CRAN.package.polle. R package version 1.6.4.
- Petersen ML, Porter KE, Gruber S, Wang Y, Van der Laan MJ (2012). “Diagnosing and Responding to Violations in the Positivity Assumption.” *Statistical Methods in Medical Research*, **21**(1), 31–54. doi:10.1177/0962280210386207.
- Polley E, LeDell E, Kennedy C, Van der Laan MJ (2025). **SuperLearner**: *Super Learner Prediction*. doi:10.32614/CRAN.package.superlearner. R package version 2.0-40.
- R Core Team (2026). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Robins J (1986). “A New Approach to Causal Inference in Mortality Studies with a Sustained Exposure Period—Application to Control of the Healthy Worker Survivor Effect.” *Mathematical Modelling*, **7**(9), 1393–1512. doi:10.1016/0270-0255(86)90088-6.
- Robins J, Rotnitzky AG (2014). “Discussion of “Dynamic Treatment Regimes: Technical Challenges and Applications.”” *Electronic Journal of Statistics*, **8**(1), 1273–1289. doi:10.1214/14-ejs908.
- Rubin DB (1974). “Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies.” *Journal of Educational Psychology*, **66**(5), 688–701. doi:10.1037/h0037350.
- SAS Institute Inc (2013). *The SAS System, Version 9.4*. SAS Institute Inc., Cary. URL <http://www.sas.com/>.
- Semenova V, Chernozhukov V (2021). “Debiased Machine Learning of Conditional Average Treatment Effects and other Causal Functions.” *The Econometrics Journal*, **24**(2), 264–289. doi:10.1093/ectj/utaa027.
- StataCorp (2021). *Stata Statistical Software: Release 17*. StataCorp LLC, College Station. URL <https://www.stata.com/>.
- Steingrimsson JA, Diao L, Molinaro AM, Strawderman RL (2016). “Doubly Robust Survival Trees.” *Statistics in Medicine*, **35**(20), 3595–3612. doi:10.1002/sim.6949.
- Sverdrup E, Kanodia A, Zhou Z, Athey S, Wager S (2020). “**policytree**: Policy Learning via Doubly Robust Empirical Welfare Maximization over Trees.” *Journal of Open Source Software*, **5**(50), 2232. doi:10.21105/joss.02232.
- Sverdrup E, Kanodia A, Zhou Z, Athey S, Wager S (2024). **policytree**: *Policy Learning via Doubly Robust Empirical Welfare Maximization over Trees*. doi:10.32614/CRAN.package.policytree. R package version 1.2.3.

- Tay JK, Narasimhan B, Hastie T (2023). “Elastic Net Regularization Paths for All Generalized Linear Models.” *Journal of Statistical Software*, **106**(1), 1–31. doi:[10.18637/jss.v106.i01](https://doi.org/10.18637/jss.v106.i01).
- Therneau TM (2026). **survival**: *Survival Analysis*. doi:[10.32614/CRAN.package.survival](https://doi.org/10.32614/CRAN.package.survival). R package version 3.8-6.
- Tibshirani J, Athey S, Sverdrup E, Wager S (2025). **grf**: *Generalized Random Forests*. doi:[10.32614/CRAN.package.grf](https://doi.org/10.32614/CRAN.package.grf). R package version 2.5.0.
- Tsiatis AA, Davidian M, Holloway ST, Laber EB (2019). *Dynamic Treatment Regimes: Statistical Methods for Precision Medicine*. Chapman and Hall/CRC, New York. doi:[10.1201/9780429192692](https://doi.org/10.1201/9780429192692).
- Van der Laan MJ (2006). “Statistical Inference For Variable Importance.” *The International Journal of Biostatistics*, **2**(1). doi:[10.2202/1557-4679.1008](https://doi.org/10.2202/1557-4679.1008).
- Van der Laan MJ, Luedtke AR (2014). “Targeted Learning of an Optimal Dynamic Treatment, and Statistical Inference for Its Mean Outcome.” *Working Paper Series 329*, U.C. Berkeley Division of Biostatistics. URL <https://biostats.bepress.com/ucbbiostat/paper329>.
- Van der Laan MJ, Robins JM (2003). *Unified Methods for Censored Longitudinal Data and Causality*. Springer-Verlag, New York. doi:[10.1007/978-0-387-21700-0](https://doi.org/10.1007/978-0-387-21700-0).
- Van Rossum G, et al. (2011). *Python Programming Language*. URL <https://www.python.org/>.
- Wickham H (2016). **ggplot2**: *Elegant Graphics for Data Analysis*. Springer-Verlag, New York. doi:[10.1007/978-0-387-98141-3](https://doi.org/10.1007/978-0-387-98141-3).
- Wright MN, Ziegler A (2017). “**ranger**: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R.” *Journal of Statistical Software*, **77**(1), 1–17. doi:[10.18637/jss.v077.i01](https://doi.org/10.18637/jss.v077.i01).
- Xu Y, Ignatiadis N, Sverdrup E, Fleming S, Wager S, Shah N (2023). “Treatment Heterogeneity with Survival Outcomes.” In *Handbook of Matching and Weighting Adjustments for Causal Inference*, pp. 445–482. Chapman and Hall/CRC.
- Zhang B, Tsiatis AA, Davidian M, Zhang M, Laber E (2012). “Estimating Optimal Treatment Regimes from a Classification Perspective.” *Stat*, **1**(1), 103–114. doi:[10.1002/sta.411](https://doi.org/10.1002/sta.411).
- Zhang C, Chen J, Fu H, He X, Zhao YQ, Liu Y (2020). “Multicategory Outcome Weighted Margin-Based Learning for Estimating Individualized Treatment Rules.” *Statistica Sinica*, **30**, 1857–1879. doi:[10.5705/ss.202017.0527](https://doi.org/10.5705/ss.202017.0527).
- Zhao Y, Zeng D, Rush AJ, Kosorok MR (2012). “Estimating Individualized Treatment Rules Using Outcome Weighted Learning.” *Journal of the American Statistical Association*, **107**(449), 1106–1118. doi:[10.1080/01621459.2012.695674](https://doi.org/10.1080/01621459.2012.695674).
- Zhou Z, Athey S, Wager S (2023). “Offline Multi-Action Policy Learning: Generalization and Optimization.” *Operations Research*, **71**(1), 148–183. doi:[10.1287/opre.2022.2271](https://doi.org/10.1287/opre.2022.2271).

## A. Binary $V$ -optimal policy

Consider the two-stage case,  $O = (S_1, A_1, S_2, A_2, U)$ , where  $A_1$  and  $A_2$  are binary. Let  $V_1$  be a function of  $H_1 = S_1$  and let  $(A_1, V_2)$  be a function of  $H_2$ . The  $V$ -optimal policy is defined as

$$d_0 = \arg \max_{d \in \mathcal{D}} \mathbb{E}[U^d].$$

The following theorem is a corrected version of Theorem 1 in [Van der Laan and Luedtke \(2014\)](#).

**Theorem A.1.** *If  $V_1$  is a function of  $V_2$ , then the  $V$ -optimal policy  $d_0$  is given by*

$$\begin{aligned} B_{0,2}(a_1, v_2) &= \mathbb{E}[U^{a_1, a_2=1} \mid V_2^{a_1} = v_2] - \mathbb{E}[U^{a_1, a_2=0} \mid V_2^{a_1} = v_2] \\ d_{0,2}(a_1, v_2) &= I\{B_{0,2}(a_1, v_2) > 0\} \\ B_{0,1}(v_1) &= \mathbb{E}[U^{a_1=1, d_{0,2}} \mid V_1 = v_1] - \mathbb{E}[U^{a_1=0, d_{0,2}} \mid V_1 = v_1] \\ d_{0,1}(v_1) &= I\{B_{0,1}(v_1) > 0\}. \end{aligned}$$

The above statement is also true if for all  $a_1$  and  $a_2$

$$\mathbb{E}[U^{a_1, a_2} \mid V_1, V_2^{a_1}] = \mathbb{E}[U^{a_1, a_2} \mid V_2^{a_1}]. \quad (15)$$

*Proof.* Let  $V^a = (V_1, V_2^a)$ . For any policy  $d$ ,

$$\begin{aligned} \mathbb{E}[U^d] &= \mathbb{E} \left[ \sum_{a_1, a_2} U^{a_1, a_2} I\{d_2(a_1, V_2^{a_1}) = a_2\} I\{d_1(V_1) = a_1\} \right] \\ &= \sum_{a_1} \mathbb{E} \left[ \left\{ \sum_{a_2} \mathbb{E}(U^{a_1, a_2} \mid V_2^{a_1}) I\{d_2(a_1, V_2^{a_1}) = a_2\} \right\} I\{d_1(V_1) = a_1\} \right], \end{aligned}$$

where it is used that  $V_1$  is a function of  $V_2^{a_1}$  or that Equation 15 holds. For any  $a_1$  the inner sum is maximized in  $d_2$  by  $d_{0,2}$ , i.e.,  $\mathbb{E}[U^d] \leq \mathbb{E}[U^{d_1, d_{0,2}}]$ . Now,

$$\mathbb{E}[U^{d_1, d_{0,2}}] = \mathbb{E} \left[ \sum_{a_1} \mathbb{E}[U^{a_1, d_{0,2}} \mid V_1] I\{d_1(V_1) = a_1\} \right],$$

which is maximized for  $d_1 = d_{0,1}$ , i.e.,  $\mathbb{E}[U^d] \leq \mathbb{E}[U^{d_1, d_{0,2}}] \leq \mathbb{E}[U^{d_{0,1}, d_{0,2}}]$ .  $\square$

## B. Weighted classification loss function

We continue the setup from Appendix A. At the second stage, define the doubly robust blip score as

$$W_2(g, Q)(O) = Z_2(1, g, Q)(O) - Z_2(0, g, Q)(O),$$

where  $Z_2(a_2, g, Q)$  is the doubly robust score from Equation 9:

$$Z_2(a_2, g, Q)(O) = Q_2(H_2, a_2) + \frac{I\{A_2 = a_2\}}{g_2(H_2, A_2)} \{U - Q_2(H_2, A_2)\}.$$

Now, define the loss function

$$L_2(d_2)(g_0, Q_0)(O) = |W_2(g_0, Q_0)(O)| I\{d_2(A_1, V_2) \neq I\{W_2(g_0, Q_0)(O) > 0\}\}.$$

This is a valid loss function since the value loss function  $\tilde{L}_2(d_2)(g_0)(O)$  from Equation 13 is a valid loss function and

$$\begin{aligned} & -\tilde{L}_2(d_2)(g, Q)(O) \\ &= Q_2(H_2, d_2(H_2)) + \frac{I\{A_2 = d_2(A_1, V_2)\}}{g_2(H_2, A_2)} \{U - Q_2(H_2, A_2)\} \\ &= d_2(A_1, V_2) W_2(g, Q)(O) + Q_2(H_2, 0) + \frac{I\{A_2 = 0\}}{g_2(H_2, A_2)} \{U - Q_2(H_2, A_2)\} \\ &= I\{W_2(g, Q)(O) > 0\} |W_2(g, Q)(O)| + Q_2(H_2, 0) + \frac{I\{A_2 = 0\}}{g_2(H_2, A_2)} \{U - Q_2(H_2, A_2)\} \quad (16) \\ & \quad - |W_2(g, Q)(O)| I\{d_2(A_1, V_2) \neq I\{W_2(g, Q)(O) > 0\}\}. \end{aligned}$$

Importantly, note that line 16 does not depend on  $d_2$ . The last equality holds because for  $d \in \{0, 1\}$  and  $W \in \mathbb{R}$ :

$$\begin{aligned} dW &= d|W| I\{W > 0\} - d|W| I\{W \leq 0\} \\ &= |W| I\{W > 0\} - |W| \left( (1-d) I\{W > 0\} + d I\{W \leq 0\} \right) \\ &= |W| I\{W > 0\} - |W| I\{d \neq I\{W > 0\}\}. \end{aligned}$$

At the first stage, define the doubly robust blip score as

$$W_1(d_2, g, Q^{d_2}) = Z_1([1, d_2], g, Q^{d_2})(O) - Z_1([0, d_2], g, Q^{d_2})(O).$$

By similar calculations, a valid loss function for  $d_{0,1}$  is given by

$$\begin{aligned} & L_1(d_1)(d_{0,2}, g_0, Q_0^{d_{0,2}})(O) \\ &= |W_1(d_{0,2}, g_0, Q_0^{d_{0,2}})(O)| I\{d_1(V_1) \neq I\{W_1(d_{0,2}, g_0, Q_0^{d_{0,2}})(O) > 0\}\}. \end{aligned}$$

### Affiliation:

Andreas Nordland  
 Section of Biostatistics  
 University of Copenhagen  
 Øster Farimagsgade 5  
 1014 Copenhagen, Denmark  
 E-mail: [andreasnordland@gmail.com](mailto:andreasnordland@gmail.com)

Klaus Kähler Holst  
Novo Nordisk  
Vandtårnsvej 108-110  
2860 Søborg, Denmark  
E-mail: [kkzh@novonordisk.com](mailto:kkzh@novonordisk.com)