Dynamic Treatment Regimes in R using *DynTxRegime*:

Shannon T. Holloway, Marie Davidian, Eric B. Laber, Kristin A. Linn,
Leonard A. Stefanski, Anastasios Tsiatis, Baqun Zhang, and Min Zhang

March 24, 2015

Abstract

A goal of personalized medicine is to use the unique characteristics of individual patients to customize treatments and therapies and thereby optimize long-term clinical outcomes. A dynamic treatment regime formalizes this process as a sequence of decision rules that map current and past patient information to a recommended treatment or therapy. As the interest in personalized medicine has grown in recent years, so has the need for powerful and flexible estimators of optimal treatment regimes that can be used with either observational or randomized clinical trial data. The *DynTxRegime* package implements several statistical methods for estimating an optimal treatment regime: Q-learning, Interactive Q-learning (IQ-learning), value-search estimators from a missing data perspective, value-search estimators from a classification perspective, and value-search estimators from a coarsened data perspective. In this vignette, we briefly describe the main structure behind each method and discuss in detail their implementation in the *DynTxRegime* package.

Keywords: Interactive Q-learning; Q-learning; Dynamic Treatment Regimes; Dynamic Programming; doubly robust; augmented inverse probability weighted estimator; inverse probability weighted estimator.

1 Introduction

In practice, clinicians and intervention scientists must adapt treatment recommendations in response to the evolving health status of each patient. A dynamic treatment regime (DTR) formalizes the treatment process as a sequence of decision rules that map patient information to a recommended treatment. For a prespecified outcome, a DTR is said to be optimal if it yields the maximal expected outcome when applied to assign treatment to a population of interest.

Q-learning (Watkins and Dayan, 1992; Murphy, 2005; Schulte et al., 2012) is one approach used to estimate an optimal DTR using data from a clinical trial or observational study. At each decision point, regression models based on available patient information are postulated for the outcome. The method is implemented through a backward-recursive fitting procedure based on a dynamic programming algorithm (Bather, 2000). Under certain assumptions and correct specification of the models, Q-learning leads to a consistent estimation of the optimal regime. Q-learning is implemented in *DynTxRegime* through *qLearn*().

Q-learning involves modeling nonsmooth, nonmonotone functions of the data. Nonmonotonicity complicates the regression function, while nonsmoothness imparts nonregularity to Q-learning estimators. Interactive Q-learning (IQ-learning) was proposed by Linn et al. (2014) to model the data before applying the necessary nonmonotone, nonsmooth operations. The method uses standard interactive model building techniques that involve conditional mean and variance modeling of smooth transformations of the data. As such, this method often results in better fitting and more interpretable models than are obtained using Q-learning. In *DynTxRegime*, the method is defined for two-stage treatment regimes with binary treatments, the steps of which are implemented as *iqLearnSS*(), *iqLearnFSM*(), *iqLearnFSC*(), and *iqLearnFSV*().

In the single-decision-point setting, Zhang et al. (2012b) proposed an approach to the estimation of an optimal treatment regime that maximizes an estimator for the so-called "value," the population mean outcome if all patients were to receive treatment according to the regime. The value estimator is derived from a missing data analogy. The approach focuses on a restricted class of treatment regimes indexed by a finite number of parameters, where the forms of the regimes in the class depend on key subsets of patient information derived from posited regression models or are prespecified on the grounds of interpretability or cost. The method leads to estimated optimal regimes that achieve comparable performance to those derived via Q-learning under correctly specified models and has the added benefit of protection against misspecification if a doubly robust outcome estimator is used. In Zhang et al. (2013) the authors adapted the single-treatment-stage method to the multiple-decision-point setting by reformulating the problem as one of monotone coarsening. Both of these methods are implemented as *optimalSeq*() in the *DynTxRegime* package.

Zhang et al. (2012a) developed a general framework for estimating the optimal treatment regime for single-decision-point analyses. The procedure for determining the optimal treatment regime is recast as a weighted classification problem in which the optimal treatment minimizes the expected weighted misclassification error. Within this framework, a variety of outcome estimators can be used, including the doubly robust augmented inverse probability weighted estimator (AIPWE) of Zhang et al. (2012b). In addition, the class of treatment regimes does not need to be prespecified but is identified in a data-driven manner. This method is implemented in *DynTxRegime* as *optimalClass*().

In the following sections, we describe the general framework of each method as implemented in the *DynTxRegime* package and provide illustrative examples of the key capabilities.

1.1 Text Formatting Conventions

Throughout this manuscript, we will use the following conventions to differentiate between package names, class names, functions, and arguments:

package
`class`
function()
argument

Table 1: Text formatting conventions.

1.2 Defining Regression Steps

All of the statistical methods implemented in *DynTxRegime* rely on at least one postulated regression model. The choice of linear or non-linear models is not an inherent limitation for any of these methods. As such, hard-coding a specific regression algorithm into the `R` procedures would artificially limit the applicability of the implementation. Thus, we make use of `R` package *modelObj*. This tool affords users the freedom to completely define each regression step of an analysis. Though we briefly review the framework below, the reader is referred to the vignette of *modelObj* for details of its use.

modelObj provides a "model object" framework for the development of new `R` packages, through which the details of a regression step and subsequent predictions are defined by the user. Specifically, a "model object" contains the model (**model**), the `R` function to be used for the regression (**solver.method**), control parameters to be passed to the fitting function (**solver.args**), the `R` function to be used to obtain predictions (**predict.method**), and control parameters to be passed to the prediction function (**predict.args**). The only requirement of this framework is that the specified fitting function must have a corresponding prediction method.

The following illustrates how to create a so-called "model object."

```
moExample <- buildModelObj(model = ~ x,
                           solver.method = 'glm',
                           solver.args = list('family'='binomial'),
                           predict.method = 'predict.glm',
                           predict.args = list('type'='response'))
```

moExample contains all of the information needed to perform a regression analysis and to make predictions based on that analysis. The model is defined to be $\sim x$. In this example,

the regression analysis is to be completed using the `R` *glm*() function. With the exception of **family**, all formal arguments of *glm*() will use their default settings. The specification of **family**='binomial' in **solver.args** indicates that **family** will be set to 'binomial' rather than its default value of 'gaussian.' Predictions will be obtained using *predict.glm*(). The **predict.args** specification **type**='response' indicates that any predictions will be on the scale of the response. For most methods of *DynTxRegime*, this scaling for predictions is required.

1.3 Standard Regression-Analysis Tools

In the illustrative examples provided in this vignette, we skip the usual exploratory techniques that an analyst would employ to find the best-fitting models. These steps would only distract from our main focus, which is to present the steps of the methods implemented in *DynTxRegime*. Analysts who use these methods should employ standard data exploration techniques. All models and decision rules estimated in the following sections are strictly illustrative.

In an effort to facilitate responsible model-building, some standard regression-analysis tools have been extended to the objects returned by all methods of *DynTxRegime*, i.e., objects of class `DynTxRegime`. Most of these tools simply extend those defined for the class of object returned by the model fitting function, e.g., *coef*() and *plot*().

The method *coef*(signature(object = "DynTxRegime")) extends the *coef*() method available for most `R` regression implementations. The structure of the extracted model coefficients depends on the regression method. However, for standard model fitting classes it will be a named numeric vector.

Method *fitObject*(signature(object = "DynTxRegime")) extends the *fitObject*() method of *modelObj* and returns the value object as defined for the fitting function, e.g., an object of class `lm` or `glm`. The structure of the returned object depends on the regression method. This function is useful when there are methods available for a model fitting class that cannot be accessed through the `DynTxRegime` object directly, e.g., *fitted.values*() for class `lm` objects.

If defined, *plot*(signature(x = "DynTxRegime")) utilizes the *plot*() method available for the `R` class of objects returned by the fitting function used to obtain parameter estimates.

The *residuals*(signature(object = "DynTxRegime")) method does not extend the *residuals*() method available through the fitting class. Rather, the function returns a numeric vector of the residuals as defined for the *DynTxRegime* method.

Finally, *summary*(signature(object = "DynTxRegime")) uses the *summary*() method available for the `R` regression implementation. The exact structure of the summary information depends on the fitting function.

gender $\in \{0, 1\}$	:	patient gender; female (0) and male (1).
race $\in \{0, 1\}$	:	patient race; African American (0) or other (1).
parentBMI $\in \mathbb{R}$	:	parent BMI measured at baseline.
baselineBMI $\in \mathbb{R}$	:	patient BMI measured at baseline.
A1 $\in \{CD, MR\}$	:	first-stage randomized treatment; meal replacement (MR) and conventional diet (CD).
month4BMI $\in \mathbb{R}$	:	patient BMI measured at month 4.
A2 $\in \{CD, MR\}$	:	second-stage randomized treatment; meal replacement (MR) and conventional diet (CD).
month12BMI $\in \mathbb{R}$	:	patient BMI measured at month 12.

Table 2: Description of covariates in **bmiData**.

1.4 Dataset bmiData

The methods in this vignette will be illustrated using a simulated dataset called **bmiData** which is included in *DynTxRegime*. The data were generated to mimic a two-stage sequential, multiple assignment, randomized trial (SMART) of body mass index (BMI) reduction with two treatments at each stage. The arguments, treatments, and outcomes in **bmiData** are based on a small subset of arguments collected in a clinical trial that studied the effect of meal replacements (MRs) on weight loss and BMI reduction in obese adolescents; see Berkowitz et al. (2010) for a complete description of the original randomized trial. Descriptions of the generated arguments in **bmiData** are given in Table (2). Baseline covariates include **gender**, **race**, **parentBMI**, and **baselineBMI**. Four- and twelve-month patient BMI measurements were also included to reflect the original trial design. In the generated data, treatment was randomized to meal replacement (MR) or conventional diet (CD) at both stages, each with probability 0.5. In the original study, patients randomized to CD in stage one remained on CD with probability one in the second stage. Thus, our generated data arise from a slightly different design than that of the original trial. In addition, some patients in the original dataset were missing the final twelve month response and various first- and second-stage covariates. Our generated data is complete, and the illustrations that follow are presented under the assumption that missing data have been addressed prior to using these methods.

After installing *DynTxRegime*, load the package:

```
> library( DynTxRegime )
```

Next, load **bmiData** into the workspace with

```
> data( bmiData )
```

The dataset is a `data.frame` with 210 rows corresponding to patients and 8 columns corresponding to covariates, BMI measurements, and assigned treatments.

```
> dim( bmiData )
```

```
[1] 210   8
```

```
> head( bmiData )
```

```
  gender race parentBMI baselineBMI month4BMI month12BMI A1 A2
1      0    1  31.59683    35.84005  34.22717   34.27263 CD MR
2      1    0  30.17564    37.30396  36.38014   36.38401 CD MR
3      1    0  30.27918    36.83889  34.42168   34.41447 MR CD
4      1    0  27.49256    36.70679  32.52011   32.52397 CD CD
5      1    1  26.42350    34.84207  33.72922   33.73546 CD CD
6      0    0  29.30970    36.68640  32.06622   32.15977 MR MR
```

We will use the negative percent change in BMI at month 12 from baseline as our final outcome:

```
> y <- -100*(bmiData$month12BMI -
+            bmiData$baselineBMI)/bmiData$baselineBMI
```

Thus, higher values indicate greater BMI loss, a desirable clinical outcome.

2 General notation and problem specification

We begin by developing a common notation and vocabulary with which we will describe each method. The following will be developed in the framework of a multiple-decision-points setting with an unspecified number of treatment options at each decision point. Most of the details of the theory will be omitted, and only the main results will be presented. Users are referred to the original manuscripts for details.

Assume that there are K prespecified, ordered decision points and an outcome of interest, Y, measured after decision point K. It is assumed that larger values of Y are preferred. At each stage $k = 1, \ldots, K$, the set of treatment options is denoted as $\mathcal{A}_k$. We write a_k to denote an element of $\mathcal{A}_k$. For example, if the treatment option is binary at decision point k, then $a_k \in \mathcal{A}_k \equiv \{0, 1\}$. We will use an overline to denote a history, e.g., $\bar{a}_k = (a_1, \ldots, a_k)$.

We will use a potential outcomes framework. For a randomly chosen patient, let X_1 denote the baseline covariates recorded prior to the first decision. For $k = 2, \ldots, K$, let $X_k^*(\bar{a}_{k-1})$ be the covariate information that would accrue between decisions $(k-1)$ and k were the patient to receive treatment history $\bar{a}_{k-1}$. $X_k^*(\bar{a}_{k-1})$ takes values $x_k \in \mathcal{X}_k$. Let $Y^*(\bar{a}_K)$ be the outcome that would result were the patient to receive full treatment history $\bar{a}_K$. Then, define the potential outcomes (Robins, 1986) as

$$W = \{X_1, X_2^*(a_1), \ldots, X_K^*(\bar{a}_{K-1}), Y^*(\bar{a}_K) \text{ for all } \bar{a}_K \in \bar{\mathcal{A}}_K \}.$$

For convenience, we include X_1 in W. However, X_1 is always observed and thus is not strictly a potential outcome.

A dynamic treatment regime (DTR), $g = \{g_1(x_1), \ldots, g_K(\bar{x}_K, \bar{a}_{k-1})\}$, is an ordered set of decision rules. Decision rule $g_k(\bar{x}_k, \bar{a}_{k-1})$ corresponds to the k^{th} decision and takes as input a patient's realized covariate and treatment history up to decision k and outputs a treatment option, $a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1}) \subseteq \mathcal{A}_k$. In general, $\Phi_k(\bar{x}_k, \bar{a}_{k-1})$ is the set of feasible treatment options at decision k for a patient with realized history $(\bar{x}_k, \bar{a}_{k-1})$, allowing that some options in $\mathcal{A}_k$ may not be possible for patients with certain histories. Thus, a feasible treatment rule must satisfy $g_k(\bar{x}_k, \bar{a}_{k-1}) \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})$ for all $\bar{x}_k, \bar{a}_{k-1}$. We denote the class of all feasible regimes as $\mathcal{G}$.

For a specific $g \in \mathcal{G}$, define the potential outcomes associated with g to be

$$W_g = \{X_1, X_2^*(g_1), \ldots, X_K^*(\bar{g}_{K-1}), Y^*(g)\},$$

where $X_k^*(\bar{g}_{k-1})$ is the covariate information that would be seen between decisions $(k-1)$ and k were a patient to receive the treatments dictated sequentially by the first $(k-1)$ rules in g, and $Y^*(g)$ is the outcome if the patient were to receive the K treatments determined by g. Thus, W_g is an element of W.

An optimal treatment regime $g^{opt} = (g_1^{opt}, \ldots, g_K^{opt}) \in \mathcal{G}$ satisfies

$$\mathbb{E}\{Y^*(g^{opt})\} \geq \mathbb{E}\{Y^*(g)\}, g \in \mathcal{G}. \tag{1}$$

That is, g^{opt} is a regime that maximizes the expected potential outcome were all patients in the population to follow it.

This definition of an optimal regime is intuitively given in terms of potential outcomes. In practice, a patient is observed to experience only a single treatment history. Thus, with the exception of X_1, W cannot be observed for any patient. To be useful in practice, an optimal regime must be defined in terms of the observed data. To this end, define A_k to be the observed treatment received at decision k, and let $\bar{A}_k = (A_1, \ldots, A_k)$ be the observed treatment history up to decision k. Let X_k be the covariate information observed between decision $(k-1)$ and k under the observed treatment history $\bar{A}_{k-1}$ $(k = 2, \ldots, K)$, with history $\bar{X}_k = (X_1, \ldots, X_k)$ for $k = 1, \ldots, K$. Let Y be the observed outcome under $\bar{A}_K$. The observed data on a patient are $(\bar{X}_K, \bar{A}_K, Y)$, and the data available from a clinical

trial or observational study involving n subjects are independent and identically distributed $(\bar{X}_{Ki}, \bar{A}_{Ki}, Y_i)$ for $i = 1, \ldots, n$.

Under standard assumptions, g^{opt} may equivalently be expressed in terms of the observed data, where

$$X_k = X_k^*(\bar{A}_{k-1}) = \sum_{\bar{a}_{k-1} \in \bar{\mathcal{A}}_{k-1}} X_k^*(\bar{a}_{k-1}) I(\bar{A}_{k-1} = \bar{a}_{k-1}) \text{ for } k = 1, \ldots, K,$$

and

$$Y = Y^*(\bar{A}_K) = \sum_{\bar{a}_K \in \bar{\mathcal{A}}_K} Y^*(\bar{a}_K) I(\bar{A}_K = \bar{a}_K);$$

that is, a patient's observed covariates and outcome are the same as the potential ones s/he would exhibit under the treatment history actually received.

3 Outcome Regression Methods

Both Q-learning and Interactive Q-learning are outcome regression methods. The methods employ dynamic programming, also referred to as backward induction, to estimate g^{opt}. One begins at the Kth decision point and defines

$$Q_K(\bar{x}_K, \bar{a}_K) \;=\; \mathbb{E}(Y \mid \bar{X}_K = \bar{x}_K, \bar{A}_K = \bar{a}_K),$$

Thus, the optimal treatment at the K^{th} stage is

$$g_K^{opt}(\bar{x}_K, \bar{a}_K) \;=\; \arg \max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{K-1})} Q_K(\bar{X}_K, \bar{A}_{K-1}, a_K),$$

Thus, g_K^{opt} yields the treatment option at decision K that maximizes the potential outcome given prior covariate and treatment history. We further define

$$V_K(\bar{X}_K, \bar{A}_{K-1}) \;=\; \max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{K-1})} Q_K(\bar{X}_K, \bar{A}_{K-1}, a_K).$$

For $k = K-1, \ldots, 1$, define

$$\begin{aligned} Q_k(\bar{x}_k, \bar{a}_k) &= \mathbb{E}\{V_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k) \mid \bar{X}_k = \bar{x}_k, \bar{A}_k = \bar{a}_k\} \\ g_k^{opt}(\bar{x}_k, \bar{a}_k) &= \arg \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} Q_k(\bar{X}_k, \bar{A}_{k-1}, a_k), \\ V_k(\bar{x}_k, \bar{a}_{k-1}) &= \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} Q_k(\bar{X}_k, \bar{A}_{k-1}, a_k). \end{aligned}$$

And,

$$g_k^{opt}(\bar{x}_k, \bar{a}_k) \;=\; \arg \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} Q_k(\bar{X}_k, \bar{A}_{k-1}, a_k),$$

which dictates the option that maximizes the potential outcome if the optimal rules were followed in the future.

The $Q_k(\bar{x}_k, \bar{a}_k)$ are referred to as the "Q-functions," viewed as measuring the "quality" associated with using treatment a_k at decision k given the history up to that decision and then following the optimal regime thereafter. The "value functions," $V_k(\bar{x}_k, \bar{a}_{k-1})$, reflect the "value" of a patient's history assuming that optimal decisions are made in the future.

3.1 Q-learning

Q-learning approximates the Q-functions with linear or non-linear regression models $Q_k(\bar{x}_k, \bar{a}_k; \beta)$. In the *DynTxRegime* implementation, the regression models are defined in terms of the main effects of treatment and the interactions between treatment a_k and $\{\bar{x}_k, \bar{a}_{k-1}\}$, the covariate and treatment history:

$$Q_k(\bar{x}_k, \bar{a}_{k-1}, a_k; \beta_k) = \mu_k(\bar{x}_k, \bar{a}_{k-1}; \gamma_k) + a_k \, \mathcal{C}_k(\bar{x}_k, \bar{a}_{k-1}; \eta_k), \quad k = 1, \ldots, K,$$

where $\mu_k(\bar{x}_k, \bar{a}_{k-1}; \gamma_k)$ models the main effects of treatment, $\mathcal{C}_k(\bar{x}_k, \bar{a}_{k-1}; \eta_k)$ models the contrast functions, and $\beta_k = (\gamma_k^\top, \eta_k^\top)^\top$. The Q-learning algorithm is given below.

Q-learning Algorithm		
QK.	Modeling:	Regress Y on $\bar{X}_K$ and $\bar{A}_K$ to obtain $Q_K(\bar{x}_K, \bar{a}_{K-1}, a_K; \hat{\beta}_K) = \mu_K(\bar{x}_K, \bar{a}_{K-1}; \hat{\gamma}_K) + a_K \; \mathcal{C}_K(\bar{x}_K, \bar{a}_{K-1}; \hat{\eta}_K)$.
	Maximization:	Define $V_K(\bar{x}_K, \bar{a}_{K-1}; \hat{\beta}_K) = \max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{K-1})} Q_K(\bar{x}_K, \bar{a}_{K-1}, a_K; \hat{\beta}_K)$. $g_K^{opt}(\bar{x}_K, \bar{a}_{K-1}; \hat{\beta}_K) = \arg\max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{K-1})} Q_K(\bar{x}_K, \bar{a}_{K-1}, a_K; \hat{\beta}_K)$.
For $k = K-1, \ldots, 1$		
Qk.	Modeling:	Regress $V_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k; \hat{\beta}_{k+1})$ on $\bar{X}_k$ and $\bar{A}_k$ to obtain $Q_k(\bar{x}_k, \bar{a}_{k-1}, a_k; \hat{\beta}_k) = \mu_k(\bar{x}_k, \bar{a}_{k-1}; \hat{\gamma}_k) + a_k \; \mathcal{C}_k(\bar{x}_k, \bar{a}_{k-1}; \hat{\eta}_k)$.
	Maximization:	Define $V_k(\bar{x}_k, \bar{a}_{k-1}; \hat{\beta}_k) = \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} Q_k(\bar{x}_k, \bar{a}_{k-1}, a_k; \hat{\beta}_k)$. $g_k^{opt}(\bar{x}_k, \bar{a}_{k-1}; \hat{\beta}_k) = \arg\max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} Q_k(\bar{x}_k, \bar{a}_{k-1}, a_k; \hat{\beta}_k)$.

3.1.1 The qLearn Function

Function *qLearn*() implements a single step of the Q-learning algorithm and is called for each step of the analysis.

```
qLearn(..., moMain, moCont, data, response, txName, fSet = NULL, iter = 0L, suppress = F
```

Complete input argument names are required, the meaning of which follow.

- **moMain**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the main effects term of the Q-function, $\mu_k(\bar{x}_k, \bar{a}_{k-1}; \gamma_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **moCont**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the contrast functions of the Q-function, $\mathcal{C}_k(\bar{x}_k, \bar{a}_{k-1}; \eta_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **data**: an object of class `data.frame` containing the observed covariate and treatment histories. Treatments can be factors or integers.

- **response**: an object of class `vector` or an object of class `DynTxRegime` as returned by a prior call to *qLearn*(). For the first step of the Q-learning algorithm (the final-stage analysis), **response** is a `vector` containing the final outcome of interest. For all other steps of the Q-learning algorithm, **response** is an object of class `DynTxRegime`, the value object returned from the previous step of the Q-learning algorithm. This argument is discussed in detail below.

- **txName**: an object of class `character` specifying the column header of **data** that corresponds to the treatment variable for the stage under analysis.

- **fSet**: an object of class `function`. A user defined function specifying the rules for determining the feasible treatment options, $\Phi_k(\bar{x}_k, \bar{a}_{k-1})$, for an individual based on their covariate and treatment history. This argument will be discussed in detail in Subsection **??**.

- **iter**: an object of class `integer`. If **iter** $= 0$, the regression analyses for the main effects and contrast functions will be combined into a single regression step, i.e., the models specified in input arguments **moMain** and **moCont** will be combined into a single model for $Q_k(\bar{x}_k, \bar{a}_k; \beta_k)$, and parameter estimates will be obtained simultaneously. By default, the parameter estimates will be obtained using the regression tools specified in **moMain**. If **moMain** = NULL, the methods specified in **moCont** will be used. If **iter** ≥ 1, the **moMain** and **moCont** regression analyses will be performed separately using an iterative algorithm. The iterative algorithm is as follows:

  (1) $Y = Y_{main} + Y_{cont}$
  (2) $\hat{Y}_{cont} = 0$
  (3) $Y_{main} = Y - \hat{Y}_{cont}$
  (4) fit $Y_{main} \sim moMain$
  (5) $Y_{cont} = Y - \hat{Y}_{main}$
  (6) fit $Y_{cont} \sim A * moCont$
  (7) Repeat steps $(3) - (6)$
  until convergence or a maximum number of iterations.

  iter is the maximum number of iterations to be used to attain convergence.

- **suppress**: an object of class `logical` indicating if the final screen prints are to suppressed.

3.1.2 Illustrative Example

We will use the **bmiData** dataset for this example. We assume that the code in Section 1.4 has been executed and that the data is loaded into the working environment. Before

starting the Q-learning analysis, we must recast the treatment arguments as either integers or factors. If the treatment vector is not provided as such, it will be coerced to integer values during execution.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <- 1
> data$A1[ bmiData$A1=="CD" ] <- 0
> data$A2[ bmiData$A2=="MR" ] <- 1
> data$A2[ bmiData$A2=="CD" ] <- 0
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
```

3.1.2.1 Second-stage regression First, we must specify the model for $Q_2(\bar{x}_2, a_1, a_2) = \mu_2(\bar{x}_2, a_1) + a_2\ \mathcal{C}_2(\bar{x}_2, a_1)$. One can choose any combination of linear or non-linear models for which appropriate R methods are available. For simplicity, we choose linear models for $\mu_2(\bar{x}_2, a_1)$ and $\mathcal{C}_2(\bar{x}_2, a_1)$, where we define $X_{2,m} = (1, gender, parentBMI, month4BMI)$ and $X_{2,c} = (1, parentBMI, month4BMI)$ and

$$Q_2(\bar{x}_2, a_1, a_2; \beta_2) = X_{2,m}\gamma_2 + A2X_{2,c}\eta_2\ .$$

The model objects are created using *buildModelObj*() of *modelObj* as follows:

```
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

Note that only the right-hand-side of the formula object is required; any left-hand-side arguments will be ignored. In addition, because we do not specify any additional arguments for the regression method, *lm*(), default values will be used. For both models, the same regression method, prediction method, and method arguments are specified. Thus, the Q-function can be fit in a single regression step; doing so will be more efficient. In the call to *qLearn*(), we can specify **iter**= 0 to combine the regression steps into one. Finally, predictions are specified to be given on the scale of the response, **predict.args** = list(type='response'). This scaling is required by method *qLearn*(). For *predict.lm*(), this setting is the default and thus does not technically need to be provided by the user. However, because this setting is critical to the method, we recommend that users explicitly set the scale of the predictions as a means to ensure that this requirement has been considered.

For convenience, if the full Q-function is to be fit in a single analysis (**iter**= 0), users can provide the complete model (including treatment interactions) through **moMain** as a `modelObj` and specify **moCont**=NULL. For our example, we could also have defined the model objects as follows:

```
> moMainEx <- buildModelObj(model = ~ gender + A2*(parentBMI + month4BMI),
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContEx <- NULL
```

We will only provide examples using two model objects in the remainder of this vignette.

For the final stage regression, **response** is the final outcome, y. The second-stage Q-learning analysis is initiated as follows:

```
> fitQ2 <- qLearn(moMain = moMainSS, moCont = moContSS,
+                 data = data, txName = 'A2', response = y)
```

```
Step  1  of Q-learning algorithm.
```

```
qLearn(moMain = moMainSS, moCont = moContSS, data = data, response = y,
    txName = "A2")
```

```
 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    48.67554      -0.64891      -0.35732      -0.84883     -14.77418
parentBMI:A2  month4BMI:A2
     0.40447       0.05632

Mean of Value Function:  7.646356
```

qLearn() returns an object that inherits from class `DynTxRegime`.

```
> is(fitQ2, "DynTxRegime")
```

```
[1] TRUE
```

Note that the entire Q-learning step, $Q2$, has been performed. No additional steps are required for the second-stage analysis.

3.1.2.2 First-stage regression As before, we start by defining the regression models and methods to be used to estimate $Q_1(x_1, a_1)$. We assume a linear model and define $X_{1,m} = (1, gender, race, parentBMI, baselineBMI)$ and $X_{1,c} = (1, gender, parentBMI)$. Thus,

$$Q_1(x_1, a_1; \beta_1) = X_{1,m}\gamma_1 + A1X_{1,c}\eta_1 \ .$$

The model objects are defined as

```
> moMainFS <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContFS <- buildModelObj(model = ~ gender + parentBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

qLearn() is used for all steps of the Q-learning algorithm. The regression stage is determined by the object passed through **response**. For the final-stage, **response** was the final outcome. For all other stages, **response** is the object returned by the previous step analysis. The first-stage analysis follows:

```
> fitQ1 <- qLearn(moMain = moMainFS, moCont = moContFS,
+                 data = data, txName = 'A1', response = fitQ2,
+                 iter = 100L)
```

```
Step  2  of Q-learning algorithm.
```

```
qLearn(moMain = moMainFS, moCont = moContFS, data = data, response = fitQ2,
    txName = "A1", iter = 100L)
```

```
 *** moMain Fit ***

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI,
    data = data)

Coefficients:
(Intercept)       gender         race    parentBMI  baselineBMI
   34.28319     -1.02733      0.01416     -0.11703     -0.57426


 *** moCont Fit ***

Call:
lm(formula = YinternalY ~ A1 + A1:gender + A1:parentBMI - 1,
    data = data)

Coefficients:
          A1     A1:gender  A1:parentBMI
      9.0968        0.6378       -0.3002

Mean of Value Function:  8.380706
```

Throughout the vignette, we will use the iterative method for all first-stage analyses. This is not necessary as we limit our examples to linear models, which are solved more efficiently in a single regression step. However, the structure of the objects returned by most methods depends on the regression algorithm, and we wish to highlight these differences.

Above, notice that two regression analyses are shown: moMain Fit and moCont Fit. These results correspond to the final step of the iterative algorithm.

3.1.2.3 Post-Analysis Tools Common tools used to asses a regression analysis have been extended to objects of class `DynTxRegime` and were described previously in Section 1.3. The objects returned by *qLearn*() can be passed directly to those tools.

Single-decision-point objects of class `DynTxRegime`, such as those returned by *qLearn*(), may contain more than one regression analysis (e.g., if **iter** > 0). To ensure that the regression steps are clearly identifiable, the tools described in Section 1.3 return named lists. The names of the list elements depend on the original call to the *DynTxRegime* method. All possible combinations are described in Table **??**.

To illustrate, recall that we used the iterative algorithm to obtain parameter estimates for the first-stage analysis, i.e., **moMain** was of class `modelObj`, **moCont** was of class `modelObj`,

Table 3: Structure of values returned by standard regression-analysis tools.

moMain class	moCont class	iter	Returned List length	Returned List key(s)
modelObj	modelObj	0	1	$Combined
modelObj	modelObj	>0	2	$MainEffect $Contrast
modelObj	NULL	–	1	$moMain
NULL	modelObj	–	1	$moCont
modelObjSubset List of length n	modelObjSubset List of length n	0	n	named list, each element of which contains $Combined
modelObjSubset List of length n	modelObjSubset List of length n	>0	$2n$	named list, each element of which contains $MainEffect $Contrast
modelObjSubset List of length n	NULL	–	n	named list, each element of which contains $moMain
NULL	modelObjSubset List of length n	–	n	named list, each element of which contains $moCont

and **iter**>0. Therefore, the returned object is a list

```
> fitObjQ1 <- fitObject(fitQ1)
> is(fitObjQ1, 'list')
```

```
[1] TRUE
```

each element of which corresponds to a component of the Q-function $Q_1(x_1, a_1)$.

```
> names(fitObjQ1)
```

```
[1] "MainEffect" "Contrast"
```

```
> fitObjQ1
```

```
$MainEffect

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI,
    data = data)
```

```
Coefficients:
(Intercept)       gender         race    parentBMI  baselineBMI
   34.28319     -1.02733      0.01416     -0.11703     -0.57426


$Contrast

Call:
lm(formula = YinternalY ~ A1 + A1:gender + A1:parentBMI - 1,
    data = data)

Coefficients:
          A1     A1:gender  A1:parentBMI
      9.0968        0.6378       -0.3002
```

Notice that for the iterative algorithm, the treatment variable has been explicitly included in the model of **moCont**.

For the second stage, we combined **moMain** and **moCont** into a single regression analysis; the returned object is a list

```
> fitObjQ2 <- fitObject(fitQ2)
> is(fitObjQ2, 'list')


[1] TRUE
```

with one named element.

```
> names(fitObjQ2)


[1] "Combined"


> fitObjQ2


$Combined

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
```

```
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    48.67554      -0.64891       -0.35732      -0.84883     -14.77418
parentBMI:A2  month4BMI:A2
     0.40447       0.05632
```

For *plot*(), the keys indicated in Table 3 will be appended to the title of the generated plots. This additional information can be suppressed using **suppress**=TRUE in the call to *plot*().

As was noted in Section 1.3, the value returned by *residuals*() depends on the *DynTxRegime* method. For Q-learning, *residuals*() returns a numeric vector of the residuals for the full/combined model.

Function *optTx*() can be used to retrieve the estimated optimal second-stage treatment for the training set, i.e., **data**:

```
> optQ2_testSet <- optTx(x=fitQ2)
> head(optQ2_testSet$qFunctions)


             0        1
[1,]  8.332204 8.265574
[2,]  6.363610 5.843406
[3,]  7.989007 7.400385
[4,] 10.598840 8.776027
[5,]  9.954517 7.767399
[6,] 10.983728 9.870325


> head(optQ2_testSet$optimalTx)


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

A list is returned. Element $qFunctions is a matrix of the estimated Q-functions. The i^{th} row corresponds to the i^{th} observation in **data**. Each column corresponds to a treatment value as indicated in the column header. $optimalTx contains a vector of the estimated optimal treatment. The class of $optimalTx can be either `factor` or `integer`, and is dictated by the class of the treatment variable in **data**.

To retrieve only the matrix of the estimated second-stage Q-functions for each treatment option,

```
> qFuncsQ2 <- qFuncs(object = fitQ2)
> head(qFuncsQ2)
```

```
             0        1
[1,]  8.332204 8.265574
[2,]  6.363610 5.843406
[3,]  7.989007 7.400385
[4,] 10.598840 8.776027
[5,]  9.954517 7.767399
[6,] 10.983728 9.870325
```

Similarly, for the first-stage treatment:

```
> optQ1_testSet <- optTx(x = fitQ1)
> head(optQ1_testSet$qFunctions)
```

```
             0         1
[1,] 10.018201  9.628950
[2,]  8.302373  8.977623
[3,]  8.557323  9.201486
[4,]  8.959302 10.440069
[5,] 10.169398 11.971122
[6,]  9.785684 10.083081
```

```
> head(optQ1_testSet$optimalTx)
```

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

```
> qFuncsQ1 <- qFuncs(object = fitQ1)
> head(qFuncsQ1)
```

```
             0         1
[1,] 10.018201  9.628950
[2,]  8.302373  8.977623
[3,]  8.557323  9.201486
[4,]  8.959302 10.440069
[5,] 10.169398 11.971122
[6,]  9.785684 10.083081
```

To recommend a k^{th}-stage optimal treatment for a new patient, both the k^{th}-stage `DynTxRegime` object and a `data.frame` of new patient covariates are passed to $optTx()$. Consider a new patient with the following baseline covariate information:

```
> newpatient <- data.frame("gender" = 1,
+                          "race" = 1,
+                          "parentBMI" = 40,
+                          "baselineBMI" = 35)
```

The recommended first-stage optimal treatment based on our Q-learning analysis is:

```
> optQ1 <- optTx(x = fitQ1, newdata = newpatient)
> optQ1

$qFunctions
            0        1
[1,] 8.489848 6.215605

$optimalTx
[1] 0
```

As shown for the training set, a list is returned, which includes $qFunctions, a matrix of the first-stage Q-functions for each treatment option, and the recommended first-stage treatment for the new patient, $optimalTx.

Assume that our new patient is given the recommended first-stage treatment, and at month 4, the patient's BMI is measured to be 25.

```
> newpatient <- cbind(newpatient, "A1" = optQ1$optimalTx, "month4BMI" = 25)
```

The recommended second-stage optimal treatment is:

```
> optQ2 <- optTx(x = fitQ2, newdata = newpatient)
> optQ2

$qFunctions
            0        1
[1,] 12.51293 15.32544

$optimalTx
[1] 1
```

3.2 More Complex Q-Learning Examples

3.2.1 Feasible Treatment Sets

In some trial settings, the set of treatment options available to a patient at each decision point, $\Phi_k(\bar{x}_k, \bar{a}_{k-1})$, depends on the patient history and/or prior treatments. For example, in the original study upon which our **bmiData** is based, patients randomized to CD in stage one remained on CD with probability one in stage two. Thus, $\Phi_2(\bar{x}_2, A_1 = CD) = \{CD\}$ and $\Phi_2(\bar{x}_2, A_1 = MR) = \{CD, MR\}$. Patients that have only one treatment option available should not be included in the regression analysis. And, the maximization step must be taken over only the treatments available to a patient.

qLearn() has an optional input argument, **fSet**, which allows the user to specify the available treatments based on a patient's history. These rules are defined by the user as a function. The function must take as input **data**, the covariates and treatment history of a single patient, and return a list. The first element of the list is a nickname for the subset. It can take any character value that is consistent with R's standard naming convention. The second element of the list is the vector of treatment options. For the second stage of the original study, this rule would take the following form:

```
> fSet <- function(data){
+           if( data$A1 == 0L ) return( list("A", c(0L)) )
+           if( data$A1 == 1L ) return( list("B", c(0L,1L)) )
+         }
```

where we have nicknamed subset $\{0\}$ "A" and subset $\{0, 1\}$ "B."

The *qLearn*() call for the second-stage analysis differs from the previous example only in the presence of the **fSet** input argument as defined above:

```
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> fitQ2_FS <- qLearn(moMain = moMainSS, moCont = moContSS,
+                    data = data, txName = 'A2', response = y,
+                    fSet = fSet)
```

```
Step  1  of Q-learning algorithm.


qLearn(moMain = moMainSS, moCont = moContSS, data = data, response = y,
    txName = "A2", fSet = fSet)


 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
     54.5703       -1.1334       -0.8292       -0.5936       -2.1309
parentBMI:A2  month4BMI:A2
      0.3149       -0.2357

Mean of Value Function:  6.757812
```

For the first-stage analysis, all treatments are available to all patients, and input argument **fSet** is not required. Thus, the *qLearn*() call for the first-stage is the same as the previous example, with the exception that fitQ2_FS is the **response**.

```
> moMainFS <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContFS <- buildModelObj(model = ~ gender + parentBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> fitQ1_FS <- qLearn(moMain = moMainFS, moCont = moContFS,
+                    data = data, txName = 'A1', response = fitQ2_FS,
+                    iter = 100L)


Step  2  of Q-learning algorithm.


qLearn(moMain = moMainFS, moCont = moContFS, data = data, response = fitQ2_FS,
    txName = "A1", iter = 100L)
```

```
 *** moMain Fit ***

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI,
    data = data)

Coefficients:
(Intercept)       gender         race    parentBMI  baselineBMI
    6.31835     -0.43027     -0.41583      0.04144      0.01956


 *** moCont Fit ***

Call:
lm(formula = YinternalY ~ A1 + A1:gender + A1:parentBMI - 1,
    data = data)

Coefficients:
          A1     A1:gender  A1:parentBMI
     34.1322        0.2623       -1.0934

Mean of Value Function:  9.437163
```

When treatment recommendations for new patients are obtained using $optTx()$, the rules for determining the feasible set of treatment options are respected. Consider again our new patient:

```
> newpatient


  gender race parentBMI baselineBMI
1      1    1        40          35
```

In this analysis, the recommended first-stage treatment is:

```
> optQ1_FS <- optTx(x = fitQ1_FS, newdata = newpatient)
> optQ1_FS


$qFunctions
          0         1
```

```
[1,] 7.8143 -1.525916

$optimalTx
[1] 0
```

Because the recommended first-stage treatment is $A_1 = 0$, the only treatment option available to the patient at the second stage is $A_2 = 0$. Again, assume that the patient is given the recommended first-stage treatment and that the patient's month 4 BMI is measured to be 25.

```
> newpatient <- cbind(newpatient, "A1" = optQ1_FS$optimalTx, "month4BMI" = 25)
```

The recommended second-stage optimal treatment is:

```
> optQ2_FS <- optTx(x = fitQ2_FS, newdata = newpatient)
> optQ2_FS


$qFunctions
            0  1
[1,] 5.428271 NA

$optimalTx
[1] 0
```

Notice that the Q-function returned for the second-stage treatment $A_2 = 1$ is NA indicating that this treatment was not in the feasible set of treatments for the patient.

3.2.2 Subset Modeling

In addition to defining rules for feasible treatment sets, the *qLearn*() method allows for unique models to be specified for subsets of patients. For example, the design of a clinical trial could be such that prior to the second-stage treatment, a response argument, $R \in \{0, 1\}$, is measured. One can specify a model for the subset of patients with $R = 0$ and another for the subset of patients with $R = 1$. Our **bmiData** dataset can be manipulated to illustrate this scenario.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <- 1
```

```
> data$A1[ bmiData$A1=="CD" ] <- 0
> data$A2[ bmiData$A2=="MR" ] <- 1
> data$A2[ bmiData$A2=="CD" ] <- 0
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
> data$R <- rbinom(nrow(data),1,0.5)
```

Above, we have randomly assigned a response argument to each patient. Though the set of feasible treatments available to both subsets is the same, we now provide **fSet** to define how the data is to be subset.

```
> fSet <- function(data){
+           if( data$R == 0L ) return(list("A",c(0L,1L)))
+           if( data$R == 1L ) return(list("B",c(0L,1L)))
+         }
```

Unlike previous examples, we want to specify a unique model for each subset, A and B. *buildModelObjSubset*() extends *buildModelObj*() of *modelObj* by allowing the user to specify the subset for which the model is defined. The second-stage Q-learning model objects will now be provided as lists and are defined as:

```
> moMain <- list()
> moCont <- list()
> #Models for subset data$R=0
> moMain[[1]] <- buildModelObjSubset(model = ~ gender + parentBMI + month4BMI,
+                                    subset = "A",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> moCont[[1]] <- buildModelObjSubset(model = ~ parentBMI + month4BMI,
+                                    subset = "A",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> #Models for subset data$R=1
> moMain[[2]] <- buildModelObjSubset(model = ~ gender + parentBMI + month4BMI,
+                                    subset = "B",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> moCont[[2]] <- buildModelObjSubset(model = ~ parentBMI + month4BMI,
+                                    subset = "B",
```

```
+                                        solver.method = 'lm',
+                                        predict.method = 'predict.lm',
+                                        predict.args = list(type='response'))
```

The call for the second-stage Q-learning analysis is:

```
> fitQ2_SM <- qLearn(moMain = moMain, moCont = moCont,
+                     data = data, txName = 'A2', response = y,
+                     fSet = fSet)


Step  1  of Q-learning algorithm.


qLearn(moMain = moMain, moCont = moCont, data = data, response = y,
    txName = "A2", fSet = fSet)

Subset:  A

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    49.57915      -0.47500       -0.18722      -1.01217     -14.57257
parentBMI:A2  month4BMI:A2
     0.47846      -0.03674

Subset:  B

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
     46.5895       -0.3985        -0.6161       -0.5830      -13.7254
parentBMI:A2  month4BMI:A2
      0.3315        0.1145
```

```
Mean of Value Function:  7.61751
```

Notice the structure of the object returned by the statistical analysis tools of Section 1.3. To illustrate, let's consider *coef*().

```
> cfs <- coef(fitQ2_SM)
```

A named list is returned,

```
> is(cfs,'list')
```

```
[1] TRUE
```

the elements of which indicate the subset.

```
> names(cfs)
```

```
[1] "A" "B"
```

For each subset, there is a second list giving the estimated coefficients for the combined model ($Combined).

```
> cfs[["A"]]
```

```
$Combined
 (Intercept)       gender    parentBMI    month4BMI           A2
 49.57914908  -0.47500399  -0.18721514  -1.01217159 -14.57257077
parentBMI:A2 month4BMI:A2
  0.47845928  -0.03673911
```

All methods described in Section 1.3 will follow this format when the model objects are built using *buildModelObjSubset*().

The structure of the first-stage analysis does not change:

```
> moMainFS <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContFS <- buildModelObj(model = ~ gender + parentBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> fitQ1_SM <- qLearn(moMain = moMainFS, moCont = moContFS,
+                    data = data, txName = 'A1', response = fitQ2_SM,
+                    iter = 100L)
```

where we have chosen not to evaluate the expression in the vignette.

3.3 Interactive Q-learning

The first modeling step in the Q-learning algorithm is a standard multiple regression problem to which common model building and model checking techniques can be applied to find a parsimonious, well-fitting model. The second modeling step requires modeling the conditional expectation of the value function, $V_{k+1}(\bar{x}_{k+1}, \bar{a}_k)$. If we assume binary treatment options coded as $\{-1, 1\}$, this expectation can be written as

$$\begin{aligned} Q_k(\bar{x}_k, \bar{a}_{k-1}) &= \mathbb{E}(V_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k) \mid \bar{X}_k = \bar{x}_k, \bar{A}_k = \bar{a}_k) \\ &= \mathbb{E}(\mu_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k; \hat{\gamma}_{k+1}) + |\mathcal{C}_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k; \hat{\eta}_{k+1})| \mid \bar{X}_k = \bar{x}_k, \bar{A}_k = \bar{a}_k). \end{aligned}$$

Due to the absolute value function, $V_{k+1}(\bar{X}_k, X_{k+1}, \bar{A}_k)$ is a nonsmooth, nonmonotone transformation of $\mathcal{C}_{k+1}(\bar{X}_{k+1}, \bar{A}_k)$. Thus, the model is generally a complex, nonlinear function of $\bar{X}_k, \bar{A}_k$. In addition, the nonsmooth, nonmonotone max operator leads to difficult nonregular inference for the parameters that index the first stage Q-function (Robins, 2004; Chakraborty et al., 2010; Laber et al., 2010; Song et al., 2011). IQ-learning was developed as an alternative to Q-learning that addresses the applied problem of building good models and avoids model misspecification for a large class of generative models. The IQ-learning methods implemented in *DynTxRegime* are valid only for two-decision-point settings with binary treatment options coded as $\{-1, 1\}$. Though the choice of treatment coding $\{-1, 1\}$ vs. $\{0, 1\}$ is one of convenience, the choice of $\{-1, 1\}$ allows one to define the optimal treatment as the sign of the contrast component. This choice is required for the IQ-learning implementation.

IQ-learning differs from Q-learning in the order in which the maximization step is performed. In IQ-learning, the maximization step is delayed, enabling all modeling to be performed *before* this nonsmooth, nonmonotone transformation. This reordering of modeling and maximization steps facilitates the use of standard, *interactive* model-building techniques because all terms to be modeled are smooth and monotone transformations of the data. For a large

class of generative models, IQ-learning consistently estimates the first-stage Q-function, resulting in a higher-quality estimated decision rule (Linn et al., 2014).

Both IQ- and Q-learning implement the same second-stage regression. In the IQ-learning framework, the first-stage Q-function is defined as

$$Q_1(x_1, a_1) = \mathbb{E}(\mu_2(X_1, X_2, A_1)|X_1 = x_1, A_1 = a_1) + \int |z| f(z \mid X_1 = x_1, A_1 = a_1) dz, \quad (2)$$

where $f(\cdot \mid X_1 = x_1, A_1 = a_1)$ is the conditional distribution of the contrast function $\mathcal{C}_2(X_1, X_2, A_1)$ given $X_1 = x_1, A_1 = a_1$. Equation (2) is equivalent to the representation of Q_1 in Eq. (2); the conditional expectation has been split into two separate expectations and the second has been written in integral form. Instead of modeling the conditional expectation in Eq. (2)directly, IQ-learning separately models $\mathbb{E}(\mu_2(X_1, X_2, A_1|X_1 = x_1, A_1 = a_1)$ and $f(\cdot \mid X_1 = x_1, A_1 = a_1)$. Although IQ-learning trades one modeling step for two, splitting up the conditional expectation in Eq. (2) is advantageous because the terms that require modeling are now smooth, monotone functionals of the data. The maximization occurs when the integral in Eq. (2) is computed, which occurs after the conditional density $f(\cdot \mid X_1 = x_1, A_1 = a_1)$ has been estimated. The IQ-learning algorithm is described next.

IQ-learning Algorithm		
IQ1.	Modeling:	Regress Y on $\bar{X}_2, \bar{A}_2$ to obtain $Q_2(\bar{x}_2, \bar{a}_2; \hat{\beta}_2) = \mu_2(\bar{x}_2, a_1; \hat{\gamma}_2) + a_2\, \mathcal{C}_2(\bar{x}_2, a_1; \hat{\eta}_2)$.
IQ2.	Modeling:	Regress observed data $\left\{\mu_2(\bar{X}_{2,i}, A_{1,i}; \widehat{\gamma}_2)\right\}_{i=1}^n$ on $\{X_{1,i}, A_{1,i}\}_{i=1}^n$ to obtain an estimator $L(x_1, a_1; \hat{\beta}_{1M})$ of $\mathbb{E}(\mu_2(X_1, X_2, A_1; \hat{\gamma}_2)|X_1 = x_1, A_1 = a_1)$. $L(x_1, a_1; \hat{\beta}_{1M}) = \mu_{1M}(x_1; \hat{\gamma}_{1M}) + a_1\, \mathcal{C}_{1M}(x_1; \hat{\beta}_{1M})$.
IQ3.	Modeling:	Use $\{\mathcal{C}_2(\bar{X}_{2,i}, A_{1,i}; \widehat{\beta}_2), X_{1,i}, A_{1,i})\}_{i=1}^n$ to obtain an estimator $F(z, x_1, a_1; \hat{\beta}_{1\sigma}, \hat{\beta}_{1C})$ of $f(z \mid X_1 = x_1, A_1 = a_1)$.
IQ4.	Maximization:	Combine the above estimators to form $Q_1(x_1, a_1; \hat{\beta}_1) = L(x_1, a_1; \hat{\beta}_1^M) + \int |z| F(z, x_1, a_1; \hat{\beta}_{1\sigma}, \hat{\beta}_{1C}) dz$.

Remark about density estimation in IQ3

Step IQ3 in the IQ-learning algorithm requires estimating a one-dimensional conditional density. Linn et al. (2014) accomplish this using mean-variance, location-scale estimators of

$f(\cdot \mid X_1 = x_1, A_1 = a_1)$ of the form

$$F(z, x_1, a_1; \hat{\beta}_{1\sigma}, \hat{\beta}_{1C}) = \frac{1}{\sigma(x_1, a_1; \hat{\beta}_{1\sigma})} \widehat{\phi} \left(\frac{z - C(x_1, a_1; \hat{\beta}_{1C})}{\sigma(x_1, a_1; \hat{\beta}_{1\sigma})} \right), \tag{3}$$

where $C(x_1, a_1; \hat{\beta}_{1C}) = \mu_1^C(x_1; \hat{\gamma}_{1C}) + a_1 \mathcal{C}_1^C(x_1; \hat{\eta}_{1C})$ is an estimator of $C(x_1, a_1) = \mathbb{E}\left\{\mathcal{C}_2(\bar{X}_2, A_1) \mid X_1 = x_1, A_1 = a_1\right\}$, $\sigma^2(x_1, a_1; \beta_\sigma)$ is an estimator of $\sigma^2(x_1, a_1) = \mathbb{E}\left\{(\mathcal{C}_2(\bar{X}_2, A_1) - C(X_1, A_1))^2 \mid X_1 = x_1, A_1 = a_1\right\}$, and $\widehat{\phi}$ is an estimator of the density of the standardized residuals $\left\{\mathcal{C}_2(\bar{x}_2, a_1; \hat{\beta}_2) - C(x_1, a_1)\right\} / \sigma(x_1, a_1)$. Currently, *DynTxRegime* implements mean-variance modeling steps (Carroll and Ruppert, 1988) to estimate $f(\cdot \mid X_1 = x_1, A_1 = a_1)$ with the option of using a standard normal density or empirical distribution estimator for $\widehat{\phi}$.

3.3.1 IQ-learning functions

There are four IQ-learning functions in the *DynTxRegime* package. The structures of the calls are very similar and are presented in combination below.

```
iqLearnSS(..., moMain, moCont, data, response, txName,
          iter = 0L, suppress = FALSE)
iqLearnFSM(..., moMain, moCont, data, response, txName,
          iter = 0L, suppress = FALSE)
iqLearnFSC(..., moMain, moCont, data, response, txName,
          iter = 0L, suppress = FALSE)
iqLearnFSV(object, ..., moMain=NULL, moCont=NULL, data=NULL,
          iter = 0L, suppress = FALSE)
```

Complete input argument names are required, the meaning of which follow.

- **moMain**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the main effects term of the Q-function, $\mu_k(\bar{x}_k, \bar{a}_{k-1}; \gamma_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **moCont**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the contrast functions of the Q-function, $\mathcal{C}_k(\bar{x}_k, \bar{a}_{k-1}; \eta_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **object**: an object of class `DynTxRegime`. The object returned by *iqLearnFSC*()

- **data**: an object of class `data.frame` containing the observed covariate and treatment histories. Treatments must be of class `integer` coded as $\{-1, 1\}$.

- **response**: an object of class `vector` or an object of class `DynTxRegime` as returned by a prior call to *iqLearnSS*(). For the first step of the *IQ*-learning algorithm (the final-stage analysis), **response** is a `vector` containing the final outcome of interest. For the first-stage functions *iqLearnFSM*() and *iqLearnFSC*(), **response** is an object of class `DynTxRegime`, the value object returned from *iqLearnSS*().

- **txName**: an object of class `character` specifying the column header of **data** that corresponds to the treatment variable for the stage under analysis.

- **iter**: an object of class `integer`. If **iter** $= 0$, the regression analyses for the main effects and contrast functions will be combined into a single regression step, i.e., the models specified in input arguments **moMain** and **moCont** will be combined into a single model for $Q_k(\bar{x}_k, \bar{a}_k; \beta_k)$, and parameter estimates will be obtained simultaneously. By default, the parameter estimates will be obtained using the regression tools specified in **moMain**. If **moMain** = NULL, the methods specified in **moCont** will be used. If **iter** ≥ 1, the **moMain** and **moCont** regression analyses will be performed separately using an iterative algorithm. The iterative algorithm is as follows:

  (1) $Y = Y_{main} + Y_{cont}$
  (2) $\hat{Y}_{cont} = 0$
  (3) $Y_{main} = Y - \hat{Y}_{cont}$
  (4) fit $Y_{main} \sim moMain$
  (5) $Y_{cont} = Y - \hat{Y}_{main}$
  (6) fit $Y_{cont} \sim A * moCont$
  (7) Repeat steps $(3) - (6)$
  until convergence or a maximum number of iterations.

  iter is the maximum number of iterations to be used to attain convergence.

- **suppress**: an object of class `logical`. If TRUE, final screen prints will be suppressed.

3.3.2 Illustrative Example

We will use the **bmiData** dataset for the examples in this section. Previously, we imported the dataset into the working environment and defined the outcome of interest: y, the negative percent change in BMI at month 12 from baseline. Before starting the *IQ*-learning analysis, we need to recast the treatment arguments as integers in the set $\{-1, 1\}$.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <-  1
```

```
> data$A1[ bmiData$A1=="CD" ] <- -1
> data$A2[ bmiData$A2=="MR" ] <-  1
> data$A2[ bmiData$A2=="CD" ] <- -1
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
```

3.3.2.1 Second-stage regression (Step IQ1) As for Q-learning, the first step in the IQ-learning algorithm is to model the response as a function of second-stage history arguments and treatment. For simplicity, we will use the same model objects as those assumed in the Q-learning section. Namely,

```
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

Again, we explicitly indicate that the predictions must be returned on the scale of the response (type='response'), as is required for the IQ-learning method.

For the second-stage regression, **response** is is the final outcome of interest, y, and the analysis is initiated as follows:

```
> fitIQ2 = iqLearnSS(moMain = moMainSS, moCont = moContSS,
+                    data = data, response = y, txName = 'A2')


 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    41.28845      -0.64891       -0.15509      -0.82067      -7.38709
parentBMI:A2  month4BMI:A2
     0.20223       0.02816
```

```
Mean of Value Function:  7.646356
```

An object that inherits from class `DynTxRegime` is returned.

```
> is(fitIQ2,'DynTxRegime')
```

```
[1] TRUE
```

3.3.2.2 Regression of second-stage main effects (Step IQ2) The next step in the *IQ*-learning algorithm is to model the conditional expectation of the main effects term given the first-stage covariates and treatment. We will accomplish this by regressing $\left\{\mu_2(\bar{X}_{2,i}, A_{1,i}; \widehat{\gamma}_2)\right\}_{i=1}^n$ on a linear function of $\{(X_{1,i}, A_{1,i})\}_{i=1}^n$ using the function *iqLearnFSM*(). If desired, the estimated main effects of the second-stage, $\left\{\mu_2(\bar{X}_{2,i}, A_{1,i}; \widehat{\gamma}_2)\right\}_{i=1}^n$, can be retrieved by *fittedMain*(); though, this is not necessary for the algorithm.

```
> head(fittedMain(fitIQ2))
```

```
[1]  8.298889  6.103508  7.694696  9.687434  8.860958 10.427026
```

The **response** is the object returned by the second-stage analysis, *iqLearnSS*(), and step IQ2 of the *IQ*-learning algorithm is initiated as:

```
> moMainFS <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContFS <- buildModelObj(model = ~ gender + parentBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> fitIQ1main <- iqLearnFSM(moMain = moMainFS, moCont = moContFS,
+                          data = data, response = fitIQ2, txName = 'A1',
+                          iter = 100L)
```

```
 *** moMain Fit ***

Call:
```

```
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI,
    data = data)

Coefficients:
(Intercept)       gender         race    parentBMI  baselineBMI
    40.2974      -0.6288      -0.1418      -0.3708      -0.5477


 *** moCont Fit ***

Call:
lm(formula = YinternalY ~ A1 + A1:gender + A1:parentBMI - 1,
    data = data)

Coefficients:
          A1     A1:gender  A1:parentBMI
      5.0536        0.1846       -0.1638
```

where we have assume the same first-stage model objects as those of the Q-learning example (Section 3.1.2. An object inheriting from class `DynTxRegime` is returned.

```
> is(fitIQ1main,'DynTxRegime')


[1] TRUE
```

3.3.2.3 Conditional density of second-stage contrast (STEP IQ3) The final modeling step in IQ-learning is to model the conditional density of the contrast function given the first-stage covariates and treatment. We begin by modeling the conditional mean of the contrast function using *iqLearnFSC*(); $C(x_1, a_1; \beta_{1C}) = \mu_{1C}(x_1; \gamma_{1C}) + a_1 \mathcal{C}_{1C}(x_1; \eta_{1C})$. The estimated contrast function can be retrieved using *fittedCont*(); though it is not necessary for the analysis.

```
> head(fittedCont(fitIQ2))


[1] -0.03331512 -0.26010187 -0.29431120 -0.91140685 -1.09355900
[6] -0.55670159
```

The conditional mean is modeled as

```
> moMainFSc <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                            solver.method = 'lm',
+                            predict.method = 'predict.lm',
+                            predict.args = list(type='response'))
> moContFSc <- buildModelObj(model = ~ gender + parentBMI + baselineBMI,
+                            solver.method = 'lm',
+                            predict.method = 'predict.lm',
+                            predict.args = list(type='response'))
```

And the regression analysis is obtained using:

```
> fitIQ1cm <- iqLearnFSC(moMain = moMainFSc, moCont = moContFSc,
+                        data = data, response = fitIQ2, txName = 'A1',
+                        iter = 100L)


 *** moMain Fit ***

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI,
    data = data)

Coefficients:
(Intercept)       gender         race    parentBMI  baselineBMI
  -7.328790    -0.004459     0.007200     0.209414     0.018306


 *** moCont Fit ***

Call:
lm(formula = YinternalY ~ A1 + A1:gender + A1:parentBMI + A1:baselineBMI -
    1, data = data)

Coefficients:
            A1       A1:gender    A1:parentBMI  A1:baselineBMI
     -0.052074       -0.009003        0.006662       -0.004076
```

where **response** is the object returned by the second-stage analysis. An object inheriting from class `DynTxRegime` is returned.

```
> is(fitIQ1cm,'DynTxRegime')


[1] TRUE
```

After fitting the model for the conditional mean of the contrast function, we must specify a model for the conditional variance of the residuals, σ^2. Standard approaches can be used to determine if a constant variance fit is sufficient. If so,

```
> fitIQ1var = iqLearnFSV(fitIQ1cm)
```

```
iqLearnVarHom(object = object)

Standard Deviation: 0.0565326
```

estimates the common standard deviation, which can be retrieved using *stdDev*().

```
> stdDev(fitIQ1var)
```

```
[1] 0.0565326
```

If the variance is thought to be non-constant across histories X_1 and/or treatment A_1, a log-linear model for the squared residuals can be used.

```
> moMainFSv <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                            solver.method = 'lm',
+                            predict.method = 'predict.lm',
+                            predict.args = list(type='response'))
> moContFSv <- buildModelObj(model = ~ parentBMI + baselineBMI,
+                            solver.method = 'lm',
+                            predict.method = 'predict.lm',
+                            predict.args = list(type='response'))
> fitIQ1var <- iqLearnFSV(object = fitIQ1cm,
+                         moMain = moMainFSv, moCont = moContFSv,
+                         data = data)
```

```
 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI +
    A1 + parentBMI:A1 + baselineBMI:A1, data = data)

Coefficients:
   (Intercept)          gender            race       parentBMI
```

```
     -8.241606        0.077415        0.075925       -0.002661
   baselineBMI              A1    parentBMI:A1  baselineBMI:A1
      0.036738        1.921779       -0.053469       -0.002528
```

```
> is(fitIQ1var,'DynTxRegime')
```

```
[1] TRUE
```

The final step in the conditional density modeling process is to choose between the normal and empirical density estimators, $\hat{\phi}$. Based on empirical experiments, Linn et al. (2014) recommend choosing the empirical estimator by default, as not much is lost when the true density is normal. *qqPlot*() can be used to inform the choice of density estimator. The object returned by the log-linear *iqLearnFSV*() function can be plotted to obtain a normal QQ-plot of the standardized residuals, displayed in Figure 1. If the observations deviate from the line, **dens**='nonpar' should be used in the final IQ-learning step, IQ4.

3.3.2.4 Estimate optimal treatment (Step IQ4) For the first-stage IQ-learning, the function *optTx*() has four inputs: the previous three first-stage objects and the method to use for the density estimator, either 'norm' or 'nonpar'. It combines all of the first-stage modeling steps to estimate the first-stage optimal decision rule.

```
> optIQ1_testSet = optTx( x = fitIQ1main, y = fitIQ1cm, z = fitIQ1var, dens = "nonpar"
> head(optIQ1_testSet$qFunctions)
```

```
            -1         1
[1,]  8.993008  8.724504
[2,]  8.071947  8.687005
[3,]  8.294638  8.870605
[4,]  9.511798 11.036698
[5,] 10.863851 12.738011
[6,]  9.596739 10.114861
```

```
> head(optIQ1_testSet$optimalTx)
```

```
[1] -1  1  1  1  1  1
```

A list is returned. Element $qFunctions is a matrix of the estimated first-stage Q-functions. The i^{th} row corresponds to the i^{th} observation in **data**. Each column corresponds to a

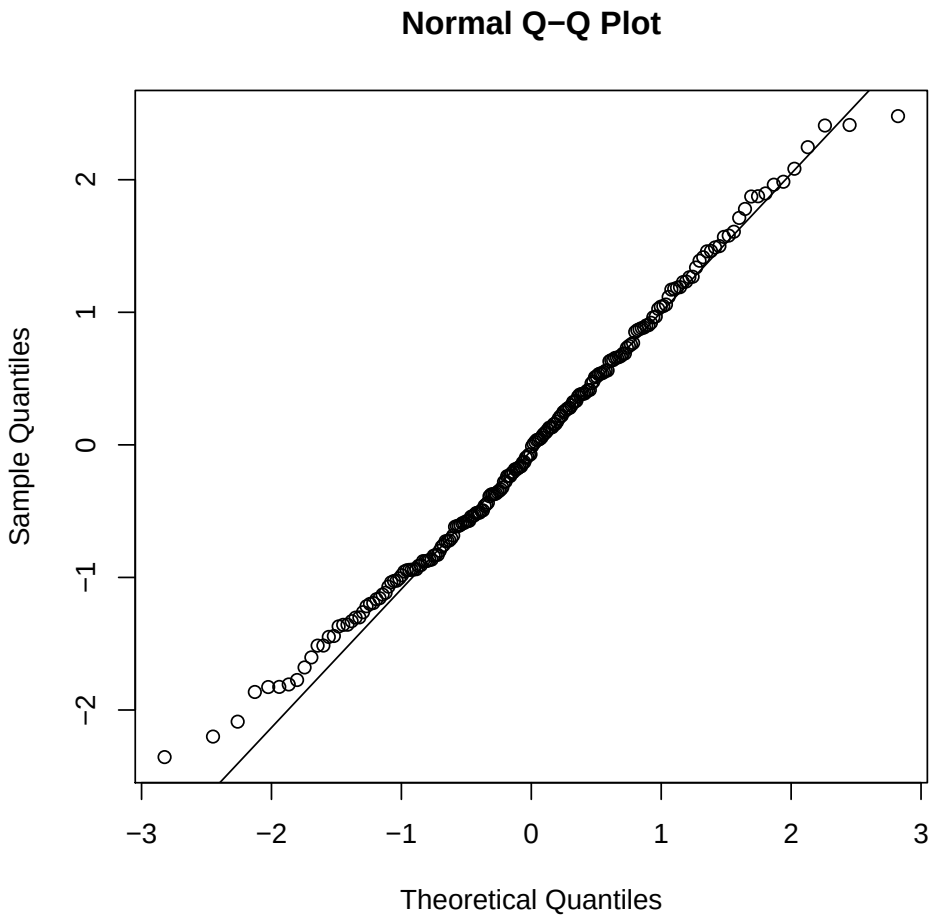

Figure 1: Normal QQ-plot of the standardized residuals obtained from the contrast mean and variance modeling steps.

treatment value as indicated in the column header. $optimalTx contains a vector of the estimated optimal treatment.

To retrieve the estimated second-stage optimal treatments, only the second-stage object is required.

```
> optIQ2_testSet = optTx( x = fitIQ2 )
> head(optIQ2_testSet$qFunctions)
```

```
           -1        1
[1,]  8.332204 8.265574
[2,]  6.363610 5.843406
[3,]  7.989007 7.400385
```

```
[4,] 10.598840 8.776027
[5,]  9.954517 7.767399
[6,] 10.983728 9.870325
```

```
> head(optIQ2_testSet$optimalTx)
```

```
[1] -1 -1 -1 -1 -1 -1
```

All of the IQ-learning functions return objects of class `DynTxRegime`, to which the standard analysis tools discussed in Section **??** can be applied. For the structure of the objects returned by those methods, see Table 3.

After estimating the optimal regime using the IQ-learning algorithm, the function $optTx()$ can be used to recommend treatment for new patients. To determine the recommended first-stage treatment for our previously defined new patient

```
> newpatient
```

```
  gender race parentBMI baselineBMI
1      1    1        40          35
```

```
> optIQ1 = optTx(x = fitIQ1main, y = fitIQ1cm, z = fitIQ1var, dens = "nonpar", newdata
> optIQ1
```

```
$qFunctions
          -1        1
[1,] 8.467605 5.965503

$optimalTx
[1] -1
```

As displayed above, a list is returned by $optTx()$ that includes the values of the first-stage Q-function when $A_1 = 1$ and $A_1 = -1$ (**$qFunctions**) as well as the recommended first-stage treatment for that patient, **$optimalTx**.

As before, assume that our new patient is given the recommended first-stage treatment and the patient's BMI is measured at month 4 to be 25.

```
> newpatient <- cbind(newpatient, "A1" = optIQ1$optimalTx, "month4BMI" = 25)
```

The recommended second stage treatment is

```
> optIQ2 = optTx(x = fitIQ2, newdata = newpatient)
> optIQ2


$qFunctions
           -1        1
[1,] 12.51293 15.32544

$optimalTx
[1] 1
```

Again, a list is returned by *optTx*() that includes the value of the second-stage Q-function when $A_2 = 1$ and $A_2 = -1$ (**$qFunctions**) as well as the recommended second-stage treatment for that patient, **$optimalTx**.

We may wish to compare our estimated optimal regime to a standard of care or constant regime that recommends one treatment for all patients. One way to compare regimes is to estimate the value function. A plug-in estimator for $V^{\boldsymbol{g}}$ is

$$\widehat{V}^{\boldsymbol{g}} \triangleq \frac{\sum_{i=1}^{n} Y_i \mathbb{1}\{A_{1i} = g_1(x_{1i})\}\mathbb{1}\{A_{2i} = g_2(\bar{x}_{2i}, a_{1i})\}}{\sum_{i=1}^{n} \mathbb{1}\{A_{1i} = g_1(x_{1i})\}\mathbb{1}\{A_{2i} = g_2(\bar{x}_{2i}, a_{1i})\}}, \tag{4}$$

where Y_i is the i^{th} patient's response, (a_{1i}, a_{2i}) the randomized treatments and (x_{1i}, x_{2i}) the observed covariates. This estimator is a weighted average of the outcomes observed from patients in the trial who received treatment in accordance with the regime $g = (g_1, g_2)$. It is more commonly known as the Horvitz-Thompson estimator (Horvitz and Thompson, 1952). Function *plugInValue*() estimates the value of a regime using the plug-in estimator and also returns value estimates corresponding to all non-dynamic regimes **$fixedReg**.

```
plugInValue(optTx1, optTx2, response, tx1, tx2)
```

- **optTx1**an object of class `vector`. The first-stage treatments assigned by the regime of interest.
- **optTx2**an object of class `vector`. The second-stage treatments assigned by the regime of interest.
- **response** an object of class `vector`. The final outcome of interest.
- **tx1**an object of class `vector`. The first-stage treatments received by patients in the trial.

- **tx2**an object of class `vector`. The second-stage treatments received by patients in the trial.

```
> estVal = plugInValue(optTx1 = optIQ1_testSet$optimalTx, optTx2 = optIQ2_testSet$optim
+                      response = y, tx1 = data$A1, tx2 = data$A2)
> estVal


$value
[1] 9.060113

$fixedReg
         tx2=-1    tx2=1
tx1=-1 6.201568 3.523643
tx1=1  8.063114 7.917462
```

4 Value Search Methods

In Q-learning and its variants, the optimal treatment is predicted using the postulated models for the Q-functions, thus the resulting estimated regime may be far from g^{opt} if these models are misspecified.

Zhang et al. (2012b) and Zhang et al. (2013) considered the posited regression model as a mechanism for defining a class of induced treatment regimes. These methods estimate the optimal regime within a prespecified class by directly maximizing an estimator of the population mean outcome across all regimes in the class. Both the inverse probability weighted estimator (IPWE) and the doubly robust augmented inverse probability weighted estimator (AIPWE) are implemented as described in Sections 4.1 and 4.3.

Zhang et al. (2012a) proposed a general framework for estimating the optimal treatment regime that recasts the original problem of finding the optimal treatment regime as a weighted classification problem. Within this framework, the class of treatment regimes does not need to be prespecified and is identified in a data-driven way by minimizing an expected weighted mis-classification error. The estimation of mean outcome under a regime is separate from the optimization for identifying the form of the treatment regime. Though the framework does not limit the form of the estimator, only the IPWE and the AIPWE presented in Zhang et al. (2012b) are implemented in *optimalClass*().

4.1 Estimation from a Missing Data Persepctive

In this method, the problem of estimating the optimal treatment regime in the single-decision-point setting is recast as a missing data problem. A class of regimes parameterized in ν is defined as $g(X, \nu)$. As discussed previously in Section 2, we can define the set of potential outcomes associated with g to be $W_g = \{X, Y^*(g_\nu)\}$, where X is the baseline covariate information and $Y^*(g_\nu)$ the outcome if the patient were to receive treatment $g(X, \nu)$. For fixed ν, the "missingness" is quantified as C_ν, where $C_\nu = \sum_{a \in \Phi(X)} I\{A - g(X, \nu)\}$. When $C_\nu = 1$, the potential outcome under treatment regime $g(X, \nu)$ is observed. If $C_\nu = 0$, $Y^*(g_\nu)$ is "missing." In this way, one can conceive of "full data" $W_g = \{X, Y^*(g_\nu)\}$ and "observed data" $\{C_\nu, C_\nu Y^*(g_\nu), X\} = \{C_\nu, C_\nu Y, X\}$. Let $\pi(a, X) = \text{pr}(A = a \mid X)$ denote the propensity score for treatment a. It is then straightforward to obtain $\text{pr}(C_\nu = 1 \mid X) = \pi(g(X, \nu), X)$.

Parametric models, $\pi(a, X; \gamma)$, are posited for the propensity of treatment, e.g., using logistic regression.

Following the missing data analogy, a simple estimator for $\mathbb{E}\{Y^*(g_\nu)\}$ for fixed ν is the inverse probability weighted estimator (IPWE) given by

$$IPWE(\nu) = n^{-1} \sum_{i=1}^{n} \frac{C_{\nu,i} Y_i}{\pi(g(X_i, \nu), X_i; \hat{\gamma})}. \tag{5}$$

This estimator is consistent for $\mathbb{E}\{Y^*(g_\nu)\}$ if $\pi(A, X; \gamma)$ is correctly specified, but may not be otherwise.

Following Robins et al. (1994) and Cao et al. (2009), an alternative estimator that offers protection against such misspecification and improved efficiency is the doubly robust augmented inverse probability weighted estimator (AIPWE)

$$AIPWE(\nu) = n^{-1} \sum_{i=1}^{n} \left\{ \frac{C_{\nu,i} Y_i}{\pi(g(X_i, \nu), X_i; \hat{\gamma})} - \frac{C_{\nu,i} - \pi(g(X_i, \nu), X_i; \hat{\gamma})}{\pi(g(X_i, \nu), X_i; \hat{\gamma})} m(g(X_i, \nu), X_i; \hat{\beta}) \right\}. \tag{6}$$

In Eq. (6), $m(A, X; \hat{\beta})$ is a model for $\mathbb{E}(Y \mid A, X)$, and $\hat{\beta}$ is an appropriate estimator for β. This estimator is consistent for $\mathbb{E}\{Y^*(g_\nu)\}$ if either of $\pi(A, X; \gamma)$ or $m(A, X; \beta)$, but not both, is misspecified.

The algorithm for this value search method is as follows:

Value Search Missing Data Perspective Algorithm		
MD1.	Modeling:	Regress A on X to obtain $\pi(A, X; \hat{\gamma})$.
MD2	Optimization	
For a fixed estimate of ν		
	MD2a. Modeling:	Regress Y on X and A to obtain $m(a, x; \hat{\beta}) = \mu(x; \hat{\gamma}) + a\ \mathcal{C}(x; \hat{\eta})$.
	MD2b. Estimator:	Use $m(g(X, \nu), X; \hat{\beta})$ and $\pi(g(X, \nu), X; \hat{\gamma})$ to estimate $AIPWE(\nu)$ or $IPWE(\nu)$
	MD2c. Update:	Update ν, repeat MD2a-MD2c; terminate at convergence of estimator.

Both the IPWE and AIPWE estimators are non-smooth functions in ν; accordingly, the use of traditional optimization methods to maximize these quantities in ν can be problematic. In *DynTxRegime*, the parameters of the prespecified treatment regime are optimized using a genetic algorithm, *rgenoud* (Mebane and Sekhon, 2011).

4.1.1 The optimalSeq function

Function *optimalSeq*() implements the value search method from a missing data perspective as developed by Zhang et al. (2012b).

```
optimalSeq(..., moPropen, moMain, moCont, data, response, txName, regimes,
                fSet = NULL, refit = FALSE, iter = 0L, suppress = FALSE)
```

Complete input argument names are required, the meaning of which follow.

- **moPropen**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the propensity for treatment, $\pi(A, X; \gamma)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **moMain**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the main effects term of the outcome, $\mu(X; \gamma)$.

When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **moCont**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the contrasts of the outcome, $\mathcal{C}(X;\eta)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **data**: an object of class `data.frame` containing the observed covariate and treatment histories. Treatments can be factors or integers.

- **response**: an object of class `vector`.T he final outcome of interest.

- **txName**: an object of class `character` specifying the column header of **data** that corresponds to the treatment variable.

- **regimes**: an object of class `function`. A user defined function specifying the parameterized treatment regime, $g(X,\nu)$.

- **fSet**: ignored in single-decision-point analyses.

- **refit**: ignored in single-decision-point analyses.

- **iter**: an object of class `integer`. If **iter** = 0, the regression analyses for the main effects and contrast functions will be combined into a single regression step, i.e., the models specified in input arguments **moMain** and **moCont** will be combined into a single model, and parameter estimates will be obtained simultaneously. By default, the parameter estimates will be obtained using the regression tools specified in **moMain**. If **moMain** = NULL, the methods specified in **moCont** will be used. If **iter** ≥ 1, the **moMain** and **moCont** regression analyses will be performed separately using an iterative algorithm. The iterative algorithm is as follows:

  (1) $Y = Y_{main} + Y_{cont}$
  (2) $\hat{Y}_{cont} = 0$
  (3) $Y_{main} = Y - \hat{Y}_{cont}$
  (4) fit $Y_{main} \sim moMain$
  (5) $Y_{cont} = Y - \hat{Y}_{main}$
  (6) fit $Y_{cont} \sim A * moCont$
  (7) Repeat steps $(3) - (6)$
  until convergence or a maximum number of iterations.

  iter is the maximum number of iterations to be used to attain convergence.

- **suppress**: an object of class `logical` indicating if the final screen prints are to suppressed.

- **...** Additional arguments to be passed to *genoud*().

4.1.2 Illustrative Example

To illustrate this method, we will consider only the second-stage of our **bmiData** dataset. Specifically, the treatment argument is $A =$**A2** and the baseline covariates are $X = \{$**gender**, **race**, **parentBMI**, **baselineBMI**, **month4BMI**, **A1**$\}$.

The first step is to specify the model for the propensity for treatment, $\pi(A, X; \gamma)$. In this example, we will make the treatment argument categorical:

```
> data <- bmiData
> data$A1 <- as.factor(data$A1)
> data$A2 <- as.factor(data$A2)
```

and we specify a logistic regression model. The model object for $\pi(A, X; \gamma)$ must be created using *buildModelObj*() of *modelObj*.

```
> moPropen <- buildModelObj(model = ~1,
+                           solver.method = 'glm',
+                           solver.args = list('family'='binomial'),
+                           predict.method = 'predict.glm',
+                           predict.args = list('type'='response'))
```

Note that the default scale of the prediction for *predict.glm*() is on the scale of the linear predictors, not the response argument. Thus, argument **type** must be changed from its default setting.

Next, we specify the model objects for the outcome regression, $m(A, X; \beta)$. As for Q-learning and IQ-learning, the model is specified as two components, one for the main effects of treatment, **moMain**, and a second for the contrast functions, **moCont**.

```
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

regimes is the parameterized treatment regime and is provided to *optimalSeq*() as a user defined function. The formal arguments must be the parameter names followed by "data."

The function must return the vector of treatment assignments for the input data. The treatment regime in this example is taken from the form of the regression model.

```
> regimes <- function(v1, v2, v3, data){
+   temp <- logical(nrow(data))
+   temp[data$A1 == "MR"] <- ((v1 + v2*data$month4BMI[data$A1 == "MR"] + v3) > 0)
+   temp[data$A1 == "CD"] <- ((v1 + v2*data$month4BMI[data$A1 == "CD"]) > 0)
+
+   tx <- rep("CD",nrow(data))
+   tx[temp] <- "MR"
+   return(tx)
+ }
```

The function call for the genetic algorithm is *genoud*(). This method requires several additional pieces of information that are passed through the ellipsis of the call to *optimalSeq*(). At a minimum the information should include the search space for the parameters of the regimes

```
> c1 <- rep(-10,3)
> c2 <- rep( 10,3)
> Domains <- cbind(c1,c2)
```

Starting values for the parameters

```
> sv <- rep(0,3)
```

and a population size

```
> psize <- 100
```

We have opted for a VERY small population in this example to expedite the vignette. It is recommended that a much larger population be used, ≥ 1000.

The analysis is carried out as follows:

```
> fitSeq <- optimalSeq(moPropen = moPropen,
+                      moMain = moMainSS, moCont = moContSS,
+                      data = data, response = y, txName = "A2",
+                      regimes = regimes,
+                      pop.size = psize, starting.values = sv, Domains = Domains)
```

```
Augmented Inverse Probability Weighted Estimator

Genetic Algorithm
$value
[1] 7.639943

$par
[1] -9.4258511  0.2544200  0.4635112

$gradients
[1] NA NA NA

$generations
[1] 28

$peakgeneration
[1] 17

$popsize
[1] 100

$operators
[1] 15 12 12 12 12 12 12 12  0


Propensity for Treatment

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            291.1
Residual Deviance: 291.1         AIC: 293.1

Outcome Regression

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)
```

```
Coefficients:
   (Intercept)          gender       parentBMI       month4BMI
      48.67554        -0.64891         -0.35732         -0.84883
          A2MR  parentBMI:A2MR  month4BMI:A2MR
     -14.77418         0.40447          0.05632


Regime Parameters:
[1] -9.4258511  0.2544200  0.4635112


Estimated Value: 7.639943
```

optimalSeq() returns an object that inherits from class `DynTxRegime`.

The complete algorithm is performed with the single call to *optimalSeq*().

If **moMain** and/or **moCont** are provided in the call to *optimalSeq*(), the AIPWE estimator is used. If both **moMain**=NULL and **moCont**=NULL, the IPWE estimator is used.

```
> fitSeqIPWE <- optimalSeq(moPropen = moPropen,
+                          moMain = NULL, moCont = NULL,
+                          data = data, response = y, txName = "A2",
+                          regimes = regimes,
+                          pop.size = psize, starting.values = sv, Domains = Domains)


Inverse Probability Weighted Estimator

Genetic Algorithm
$value
[1] 7.65518

$par
[1] -4.505170  0.339819 -5.666066

$gradients
[1] NA NA NA

$generations
[1] 24

$peakgeneration
[1] 13
```

```
$popsize
[1] 100

$operators
[1] 15 12 12 12 12 12 12 12  0


Propensity for Treatment

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            291.1
Residual Deviance: 291.1         AIC: 293.1

Regime Parameters:
[1] -4.505170  0.339819 -5.666066


Estimated Value: 7.65518
```

Standard regression analysis tools were discussed in Section 1.3. Additional methods specific to the *optimalSeq*() procedure follow.

Function *regimeCoef*(signature(object = "DynTxRegime")) retrieves the estimated parameters of the regime.

```
> est <- regimeCoef(fitSeq)
> est <- est/sqrt(est %*% est)
> est
```

```
[1] -0.99843036  0.02694936  0.04909728
```

The estimated mean potential outcome for the treatment regime can be retrieved using *estimator*(signature(object = "DynTxRegime")).

```
> estimator(fitSeq)
```

```
[1] 7.639943
```

Function *genetic*(signature(object = "DynTxRegime")) retrieves the value object returned by the genetic algorithm *genoud*().

```
> genetic(fitSeq)
```

```
$value
[1] 7.639943

$par
[1] -9.4258511  0.2544200  0.4635112

$gradients
[1] NA NA NA

$generations
[1] 28

$peakgeneration
[1] 17

$popsize
[1] 100

$operators
[1] 15 12 12 12 12 12 12 12  0
```

The regression analyses for the outcome are retrieved using *outcome*(signature(object = "DynTxRegime")). A list is returned, the structure of which depends on the choice of input parameters; see Table 3 for details. This method is useful when analysis tools are available for the fitting class that cannot be accessed from the `DynTxRegime` object

```
> fitO <- outcome(fitSeq)
> names(fitO)
```

```
[1] "Combined"
```

```
> head(fitted.values(fitO[[ "Combined" ]]))
```

```
        1         2         3         4         5         6
 8.265574  5.843406  7.989007 10.598840  9.954517  9.870325
```

Function *propen*(signature(object = "DynTxRegime")) retrieves the regression analysis for the propensity for treatment.

```
> fitP <- propen(fitSeq)
> head(fitted.values(fitP))
```

```
  1   2   3   4   5   6
0.5 0.5 0.5 0.5 0.5 0.5
```

The estimated optimal treatments for the training data is returned using *optTx*(signature(x = "DynTxRegime", newdata = "missing")).

```
> head(optTx(x=fitSeq))
```

```
[1] 1 1 1 1 1 1
```

A new dataset can be included in the call to estimate the optimal treatment for new patients: *optTx*(signature(x = "DynTxRegime", newdata = "data.frame")).

```
> optSeq <- optTx(x = fitSeq,newdata = newpatient)
> optSeq
```

```
     dp= 1
[1,] "CD"
```

The information returned by *optTx*() differs from that returned for the Q-learning and IQ-learning methods. For the *optimalSeq*() method, *optTx*() returns only a matrix of estimated optimal treatments. Each row corresponds to an observation; each column corresponds to a treatment decision point.

4.2 Estimation from a Classification Perspective

The value search method described in the previous subsection requires that a class of treatment regimes be prespecified. Zhang et al. (2012a) proposed a classification framework wherein the optimal classifier corresponds to the optimal treatment regime. Within this framework, the class of treatment regimes does not need to be prespecified and can instead be identified in a data-driven way by minimizing an expected weighted misclassification error. This method is developed in *DynTxRegime* only for single-decision-point scenarios with binary treatment coded as $\{0, 1\}$.

It can be shown that the IPWE and AIPWE estimators given in the previous subsection are equivalent to

$$\begin{aligned} IPWE(\nu) &= n^{-1}\sum_{i=1}^{n}\left\{g(X_i,\nu)\widehat{C}_{IPWE}(X_i)\right\} \\ &+ n^{-1}\sum_{i=1}^{n}\left\{\frac{1-A_i}{\pi(0,X_i;\hat{\gamma})}Y_i\right\} \end{aligned} \tag{7}$$

and

$$\begin{aligned} AIPWE(\nu) &= n^{-1}\sum_{i=1}^{n}\left\{g(X_i,\nu)\widehat{C}_{AIPWE}(X_i)\right\} \\ &+ n^{-1}\sum_{i=1}^{n}\left\{\frac{1-A_i}{\pi(0,X_i;\hat{\gamma})}Y_i - \frac{A_i-\pi(1,X_i;\hat{\gamma})}{\pi(0,X_i;\hat{\gamma})}m(0,X_i;\hat{\beta})\right\} \end{aligned} \tag{8}$$

where

$$\widehat{C}_{IPWE}(X_i) = \frac{A_i}{\pi(1,X_i;\hat{\gamma})}Y_i - \frac{1-A_i}{\pi(0,X_i;\hat{\gamma})}Y_i \tag{9}$$

and

$$\widehat{C}_{AIPWE}(X_i) = \frac{A_i}{\pi(1,X_i;\hat{\gamma})}Y_i - \frac{1-A_i}{\pi(0,X_i;\hat{\gamma})}Y_i \tag{10}$$

$$- \frac{A_i-\pi(1,X_i;\hat{\gamma})}{\pi(1,X_i;\hat{\gamma})}m(1,X_i;\hat{\gamma}) - \frac{A_i-\pi(1,X_i;\hat{\gamma})}{\pi(0,X_i;\hat{\gamma})}m(0,X_i;\hat{\gamma}) \tag{11}$$

For these estimators, optimizing $\mathbb{E}\{Y^*(g)\}$ is equivalent to optimizing $n^{-1}\sum_{i=1}^{n}\left\{g(X_i,\nu)\widehat{C}_{AIPWE}(X_i)\right\}$ or $n^{-1}\sum_{i=1}^{n}\left\{g(X_i,\nu)\widehat{C}_{IPWE}(X_i)\right\}$.

Estimating the optimal treatment regime in the class $\mathcal{G}$ can be separated into two steps: constructing an estimator $\widehat{C}(X_i)$ of the contrast function $C(X_i)$ for $i = 1, \ldots, n$, and subsequently estimating g^{opt} by $\hat{g}^{opt} = \arg\max n^{-1}\sum_{i=1}^{n}\left\{g(X_i,\nu)\widehat{C}(X_i)\right\}$, where the maximization is across all regimes in the class considered.

Zhang et al. (2012a), showed that the optimal treatment regime thus defined can be rewritten as a weighted classification problem, where

$$g^{opt} = \arg\max_{g \in \mathcal{G}} \left[\mathbb{E} \left\{ |C(X)| \left[I \left\{ C(X) > 0 \right\} - g(X) \right]^2 \right\} \right], \tag{12}$$

where $W = |C(X)|$ is a weight and $Z = I\{C(X) > 0\}$ is the class, or treatment, to which the subject is assigned.

The algorithm for the doubly robust classification method is as follows:

Value Search Algorithm – Classification Perspective		
C1.	Modeling:	Regress A on X to obtain $\pi(A, X; \hat{\gamma})$.
C2.	Modeling:	Regress Y on X and A to obtain $m(A, X; \hat{\beta}) = \mu(x; \hat{\gamma}_2) + A\, \mathcal{C}(x; \hat{\eta})$.
C3.	Estimator:	Use $m(A, X; \hat{\beta})$ and $\pi(A, X; \hat{\gamma})$ to estimate $\hat{C}_{AIPWE}$ ($\hat{C}_{IPWE}$)
C4.	Classification:	Construct class labels $\hat{Z}_i = I\left\{\hat{C}(X_i) > 0\right\}$ and weights $\hat{W}_i = \lvert\hat{C}(X_i)\rvert$ for each subject. Perform classification regression.

4.2.1 The optimalClass function

Function *optimalClass*() implements the single-decision-point classification method of Zhang et al. (2012a). It is appropriate for single-decision-point analyses with binary treatments coded as $\{0, 1\}$.

```
optimalClass(..., moPropen, moMain, moCont, moClass, data, response, txName,
             iter = 0L, suppress = FALSE)
```

Complete input argument names are required, the meaning of which follow.

- **moPropen**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the propensity for treatment, $\pi(A, X; \gamma)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **moMain**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the main effects term of the outcome, $\mu(X;\gamma)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **moCont**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the regression analysis for the contrasts of the outcome, $\mathcal{C}(X;\eta)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.

- **moClass**: an object of class `modelObj` created by *buildModelObj*() of *modelObj*. This object defines the classification regression analysis. When defining the modeling object, the prediction method must return predictions on the scale of the classifier (i.e., { 0, 1}).

- **data**: an object of class `data.frame` containing the observed covariate and treatment histories. Treatments must be coded as integers $\{0, 1\}$

- **response**: an object of class `vector`.T he final outcome of interest.

- **txName**: an object of class `character` specifying the column header of **data** that corresponds to the treatment variable.

- **iter**: an object of class `integer`. If **iter** = 0, the regression analyses for the main effects and contrast functions will be combined into a single regression step, i.e., the models specified in input arguments **moMain** and **moCont** will be combined into a single model, and parameter estimates will be obtained simultaneously. By default, the parameter estimates will be obtained using the regression tools specified in **moMain**. If **moMain** = NULL, the methods specified in **moCont** will be used. If **iter** ≥ 1, the **moMain** and **moCont** regression analyses will be performed separately using an iterative algorithm. The iterative algorithm is as follows:

  (1) $Y = Y_{main} + Y_{cont}$
  (2) $\hat{Y}_{cont} = 0$
  (3) $Y_{main} = Y - \hat{Y}_{cont}$
  (4) fit $Y_{main} \sim moMain$
  (5) $Y_{cont} = Y - \hat{Y}_{main}$
  (6) fit $Y_{cont} \sim A * moCont$
  (7) Repeat steps $(3) - (6)$
  until convergence or a maximum number of iterations.

  iter is the maximum number of iterations to be used to attain convergence.

- **suppress**: an object of class `logical` indicating if the final screen prints are to suppressed.

4.2.2 Illustrative Example

To illustrate this method, we consider only the second-stage of our bmiData dataset. Specifically, the treatment argument is $A = A2$ and the baseline covariates are $X = \{\mathbf{gender}, \mathbf{race}, \mathbf{parentBMI}, \mathbf{baselineBMI}, \mathbf{month4BMI}, \mathbf{A1}\}$.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <- 1
> data$A1[ bmiData$A1=="CD" ] <- 0
> data$A2[ bmiData$A2=="MR" ] <- 1
> data$A2[ bmiData$A2=="CD" ] <- 0
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
```

and we specify a logistic regression model. The model object for $\pi(A, X; \gamma)$ must be created using *buildModelObj*() of *modelObj*.

```
> moPropen <- buildModelObj(model = ~1,
+                           solver.method = 'glm',
+                           solver.args = list('family'='binomial'),
+                           predict.method = 'predict.glm',
+                           predict.args = list('type'='response'))
```

As before, the default scale of the prediction for *predict.glm*() is on the scale of the linear predictors, not the response argument. Thus, argument **type** must be changed from its default setting.

Next, we specify the model objects for the outcome regression, $m(A, X; \beta)$. The model is specified as two components, one for the main effects of treatment, **moMain**, and a second for the contrasts among treatments, **moCont**.

```
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

We will use *rpart*() to perform the classification step. Any R classification method can be used, provided that it accepts as input **weights** and that predictions can be returned on the

scale of the class.

```
> library(rpart)
> moClass <- buildModelObj(model = ~ parentBMI + month4BMI,
+                          solver.method = 'rpart',
+                          solver.args = list(method="class"),
+                          predict.args = list(type='class'))
```

Note that *rpart* makes use of the generic *predict*(), and thus **predict.method** was not specified.

The analysis is completed in a single call as follows:

```
> fitCl <- optimalClass(moPropen = moPropen,
+                       moMain = moMainSS, moCont = moContSS,
+                       moClass = moClass,
+                       data = data, response = y, txName = "A2")
```

```
Augmented Inverse Probability Weighted Estimator
Classification
n= 210

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 210 0.460505500 1 (0.42150071 0.57849929)
    2) parentBMI< 39.19166 169 0.344959700 0 (0.48902886 0.51097114)
      4) month4BMI< 38.63 144 0.256919700 0 (0.53721054 0.46278946)
        8) parentBMI>=32.80268 38 0.059588370 0 (0.69066148 0.30933852)
         16) parentBMI< 37.19687 27 0.027260260 0 (0.80224385 0.19775615) *
         17) parentBMI>=37.19687 11 0.026307350 1 (0.40989570 0.59010430) *
        9) parentBMI< 32.80268 106 0.193526500 1 (0.45567227 0.54432773)
         18) parentBMI< 31.67436 96 0.150995900 0 (0.51626702 0.48373298)
           36) month4BMI< 30.16994 17 0.013115070 0 (0.76528760 0.23471240) *
           37) month4BMI>=30.16994 79 0.137880800 0 (0.46197050 0.53802950)
             74) parentBMI>=29.865 20 0.021942710 0 (0.61957938 0.38042062) *
             75) parentBMI< 29.865 59 0.096828780 1 (0.41619326 0.58380674)
              150) parentBMI< 29.21239 44 0.065615300 0 (0.51978788 0.48021212)
                300) parentBMI>=28.46493 17 0.020031260 0 (0.65149568 0.34850432) *
                301) parentBMI< 28.46493 27 0.039335560 1 (0.42415572 0.57584428)
                  602) month4BMI< 33.27589 18 0.020848640 0 (0.58169884 0.41830116) *
                  603) month4BMI>=33.27589 9 0.005369976 1 (0.15633955 0.84366045) *
```

```
             151) parentBMI>=29.21239 15 0.013623590 1 (0.18770933 0.81229067) *
          19) parentBMI>=31.67436 10 0.004733414 1 (0.08020476 0.91979524) *
       5) month4BMI>=38.63 25 0.037385140 1 (0.26603638 0.73396362) *
     3) parentBMI>=39.19166 41 0.073730110 1 (0.24443661 0.75556339)
       6) parentBMI< 43.64256 32 0.070802980 1 (0.31291090 0.68708910)
        12) month4BMI>=38.46362 17 0.037101120 0 (0.52826709 0.47173291) *
        13) month4BMI< 38.46362 15 0.022129020 1 (0.16497807 0.83502193) *
       7) parentBMI>=43.64256 9 0.002927133 1 (0.03884157 0.96115843) *
Propensity for Treatment

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            291.1
Residual Deviance: 291.1         AIC: 293.1
Outcome Regression

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    48.67554      -0.64891      -0.35732      -0.84883     -14.77418
parentBMI:A2  month4BMI:A2
     0.40447       0.05632

Estimated Value: 8.124419
```

and returns an object that inherits from class `DynTxRegime`.

```
> is(fitCl,'DynTxRegime')

[1] TRUE
```

If **moMain** and/or **moCont** are provided in the call to *optimalClass*(), the AIPWE estimator is used. If both **moMain**=NULL and **moCont**=NULL, the IPWE estimator is

used.

```
> fitClIPWE <- optimalClass(moPropen = moPropen,
+                           moMain = NULL, moCont = NULL,
+                           moClass = moClass,
+                           data = data, response = y, txName = "A2")


Inverse Probability Weighted Estimator
Classification
n= 210

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 210 0.455019100 1 (0.4107605 0.5892395)
   2) month4BMI< 37.22051 133 0.343508200 0 (0.4655247 0.5344753)
     4) month4BMI>=33.01012 63 0.081056040 0 (0.6265283 0.3734717)
       8) parentBMI>=32.77801 26 0.018983580 0 (0.8080958 0.1919042) *
       9) parentBMI< 32.77801 37 0.062072460 0 (0.4744600 0.5255400)
        18) parentBMI< 30.50098 26 0.025682520 0 (0.6475175 0.3524825) *
        19) parentBMI>=30.50098 11 0.010611710 1 (0.1958014 0.8041986) *
     5) month4BMI< 33.01012 70 0.195484900 1 (0.3834343 0.6165657)
      10) parentBMI< 27.75467 22 0.060121750 0 (0.5717984 0.4282016)
        20) month4BMI< 31.74998 12 0.026998730 0 (0.6918174 0.3081826) *
        21) month4BMI>=31.74998 10 0.023566080 1 (0.3726578 0.6273422) *
      11) parentBMI>=27.75467 48 0.099328580 1 (0.2907221 0.7092779)
        22) parentBMI< 32.61048 38 0.088915130 1 (0.3321983 0.6678017)
          44) parentBMI>=28.9742 21 0.056173220 0 (0.4631555 0.5368445)
            88) month4BMI< 31.35797 11 0.018668140 0 (0.6700832 0.3299168) *
            89) month4BMI>=31.35797 10 0.012631610 1 (0.2194823 0.7805177) *
          45) parentBMI< 28.9742 17 0.030870870 1 (0.2168914 0.7831086) *
        23) parentBMI>=32.61048 10 0.010413460 1 (0.1407131 0.8592869) *
   3) month4BMI>=37.22051 77 0.096672200 1 (0.2860311 0.7139689)
     6) parentBMI< 30.69162 7 0.002672736 0 (0.8013759 0.1986241) *
     7) parentBMI>=30.69162 70 0.083756680 1 (0.2602260 0.7397740) *
Propensity for Treatment

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
```

```
Null Deviance:            291.1
Residual Deviance: 291.1         AIC: 293.1
Estimated Value: 10.40926
```

Available standard regression analysis tools were discussed in Subsection 1.3. Additional methods specific to the *optimalClass*() procedure follow.

Function *estimator*(signature(object = "DynTxRegime")) retrieves the estimated mean potential outcome for the treatment regime.

```
> estimator(fitCl)
```

```
[1] 8.124419
```

The value object returned by the classification method can be retrieved using *classif*(signature(object = "DynTxRegime")).

```
> classif(fitCl)
```

```
n= 210

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 210 0.460505500 1 (0.42150071 0.57849929)
    2) parentBMI< 39.19166 169 0.344959700 0 (0.48902886 0.51097114)
      4) month4BMI< 38.63 144 0.256919700 0 (0.53721054 0.46278946)
        8) parentBMI>=32.80268 38 0.059588370 0 (0.69066148 0.30933852)
         16) parentBMI< 37.19687 27 0.027260260 0 (0.80224385 0.19775615) *
         17) parentBMI>=37.19687 11 0.026307350 1 (0.40989570 0.59010430) *
        9) parentBMI< 32.80268 106 0.193526500 1 (0.45567227 0.54432773)
         18) parentBMI< 31.67436 96 0.150995900 0 (0.51626702 0.48373298)
           36) month4BMI< 30.16994 17 0.013115070 0 (0.76528760 0.23471240) *
           37) month4BMI>=30.16994 79 0.137880800 0 (0.46197050 0.53802950)
             74) parentBMI>=29.865 20 0.021942710 0 (0.61957938 0.38042062) *
             75) parentBMI< 29.865 59 0.096828780 1 (0.41619326 0.58380674)
              150) parentBMI< 29.21239 44 0.065615300 0 (0.51978788 0.48021212)
                300) parentBMI>=28.46493 17 0.020031260 0 (0.65149568 0.34850432) *
                301) parentBMI< 28.46493 27 0.039335560 1 (0.42415572 0.57584428)
                  602) month4BMI< 33.27589 18 0.020848640 0 (0.58169884 0.41830116) *
```

```
              603) month4BMI>=33.27589 9 0.005369976 1 (0.15633955 0.84366045) *
           151) parentBMI>=29.21239 15 0.013623590 1 (0.18770933 0.81229067) *
        19) parentBMI>=31.67436 10 0.004733414 1 (0.08020476 0.91979524) *
      5) month4BMI>=38.63 25 0.037385140 1 (0.26603638 0.73396362) *
    3) parentBMI>=39.19166 41 0.073730110 1 (0.24443661 0.75556339)
      6) parentBMI< 43.64256 32 0.070802980 1 (0.31291090 0.68708910)
       12) month4BMI>=38.46362 17 0.037101120 0 (0.52826709 0.47173291) *
       13) month4BMI< 38.46362 15 0.022129020 1 (0.16497807 0.83502193) *
      7) parentBMI>=43.64256 9 0.002927133 1 (0.03884157 0.96115843) *
```

To retrieve the value objects returned by the outcome regression model fitting functions use *outcome*(signature(object = "DynTxRegime")). A list is returned, the structure of which depends on the choice of input parameters; see Table 3 for details. This method is useful when analysis tools are available for the regression method that cannot be accessed from the `DynTxRegime` object

```
> fitO <- outcome(fitCl)
> head(fitted.values(fitO[[ "Combined" ]]))
```

```
        1         2         3         4         5         6
 8.265574  5.843406  7.989007 10.598840  9.954517  9.870325
```

Similarly, *propen*(signature(object = "DynTxRegime")) retrieves the value object returned by the propensity for treatment model fitting function. This method is useful when analysis tools are available for the fitting class that cannot be accessed from the `DynTxRegime` object

```
> fitP <- propen(fitCl)
> head(fitted.values(fitP))
```

```
  1   2   3   4   5   6
0.5 0.5 0.5 0.5 0.5 0.5
```

To obtain the estimated optimal treatments for the training data, use *optTx*(signature(x = "DynTxRegime", newdata = "missing")).

```
> head(optTx(x=fitCl))
```

```
[1] 0 0 0 0 1 1
Levels: 0 1
```

To recommend an optimal treatments for a new patient, their covariate and treatment histories are provided to $optTx()$ as a `data.frame`.

```
> newpatient
```

```
  gender race parentBMI baselineBMI A1 month4BMI
1      1    1        40          35  0        25
```

```
> optCl <- optTx(x=fitCl, newdata=newpatient)
> optCl
```

```
[1] 1
```

4.3 Estimation from a Coarsened Data Perspective

In Zhang et al. (2013), the methods of Section 4.1 were extended to the multiple-decision-points scenario. The problem of estimating the optimal treatment regime is recast as a coarsened data problem.

As before, let K denote the number of decision points under analysis. The treatment regimes parameterized in ν are defined as $g(\bar{X}_K, \bar{A}_{K-1}, \nu) = \{g_1(X_1, \nu_1), \ldots, g_K(\bar{X}_K, \bar{A}_{K-1}, \nu_K)\}$, where $\nu = (\nu_1^T, \ldots, \nu_K^T)^T$. For notational convenience, we will denote $g_{k,\nu} = g_k(\bar{X}_k, \bar{A}_{k-1}, \nu_k)$ and $\bar{g}_{k,\nu} = (g_{1,\nu}, \ldots, g_{k,\nu})$.

Let, C_ν be a discrete coarsening argument taking values $1, \ldots, K, \infty$, corresponding to the $K+1$ levels of coarsening. This argument reflects the extent to which the observed treatments received are consistent with those dictated by $\bar{g}_{K,\nu}$. For example, if the observed first-stage treatment is not the treatment dictated by the first-stage treatment regime ($A_1 \neq g_{1,\nu}$), $C_\nu = 1$. In this case, we observe only X_1, and all other covariates are "missing." Similarly, if treatments $A_1, \ldots, A_{k-1}$ are consistent with the treatment regime, but $A_k \neq g_{k,\nu}$, $C_\nu = k$, and we observe only $X_1, \ldots, X_k$. If all observed treatments are consistent with the treatment regime, $C_\nu = \infty$, and we observe the "full data."

Define the coarsening hazard function, $\lambda_{k,\nu}(\bar{X}_k, \bar{A}_k) = \text{pr}(A_k \neq g_{k,\nu} \mid \bar{X}_k, \bar{A}_{k-1} = \bar{g}_{k-1,\nu})$. Thus, $\text{pr}(C_k > k \mid \bar{X}_k, \bar{A}_k) = K_{k,\nu}(\bar{X}_k, \bar{A}_k) = \prod_{k'=1}^{k}\{1 - \lambda_{k',\nu}(\bar{X}_{k'}, \bar{A}_{k'})\}$.

With these developments, the form of the estimator is taken to be

$$\begin{aligned}\mathbb{E}\{Y^*(g)\} =& n^{-1}\sum_{i=1}^{n}\left\{\frac{I(\mathcal{C}_i=\infty)}{K_K(\bar{X}_{ki},\bar{A}_{ki})}Y_i\right.\\ &\left.+\sum_{k=1}^{K}\frac{I(\mathcal{C}_i=k)-\lambda_{k,\nu}(\bar{X}_{ki},\bar{A}_{ki})I(\mathcal{C}_i\geq k)}{K_k(\bar{X}_{ki},\bar{A}_{ki})}L_k(\bar{X}_{ki})\right\} \qquad (13)\end{aligned}$$

where $L_k(\bar{X}_k)$ are arbitrary functions of $\bar{X}_k$.

To implement Eq. (13), one must specify $\pi_k(a,\bar{X}_k,\bar{A}_{k-1})$ and $L_k(\bar{X}_{ki})$. The first follow from specifying $\pi_1(a_1,x_1)=\text{pr}(A_1=a_1\mid X_1=x_1),\pi_k(a_k,\bar{x}_k,\bar{a}_{k-1})=\text{pr}(A_k=a_k\mid \bar{X}_k=\bar{x}_k,\bar{A}_{k-1}=\bar{a}_{k-1})$ for $k=K,\ldots,2$. It is assumed that these are not known, and the user must posit models $\pi_1(a_1,x_1;\gamma_1)$, $\pi_k(a_k,\bar{x}_k,\bar{a}_{k-1};\gamma_k)$ for $k=2,\ldots,K$ and estimate γ_k by $\hat{\gamma}_k$, e.g., using logistic regression. This implies that $\lambda_{1,\nu}(X_1;\gamma_1)=1-\pi_1(g_{1,\nu},x_1;\gamma_1)$ and $\lambda_{k,\nu}(\bar{X}_k,\bar{A}_k)=(1-\pi_k(g_{k,\nu},\bar{X}_k,\bar{g}_{k-1,\nu}))$

In *DynTxRegime*, two options exist for specification of the $L_k(\bar{X}_k)$. The simplest is $L_k(\bar{X}_k)\equiv 0$, yielding the IPWE:

$$IPWE(\nu)=n^{-1}\sum_{i=1}^{n}\frac{I(\mathcal{C}_i=\infty)}{K_K(\bar{X}_{ki},\bar{A}_{ki};\gamma_k)}Y_i. \qquad (14)$$

To take greatest advantage of the potential for improved efficiency through the augmentation term in Eq. (13), one can posit and fit parametric models approximating the conditional expectations $L_k^{opt}(\bar{x}_k)=\mathbb{E}\{Y^*(g)\mid \bar{X}_k^*(\bar{g}_{k-1})=\bar{x}_k\}$ and substitute these into Eq. 13. To this end, let

$$\begin{aligned} m_K(\bar{x}_K,\bar{a}_K) &= \mathbb{E}(Y\mid \bar{X}_K=\bar{x}_K,\bar{A}_K=\bar{a}_K) & (15)\\ f_K(\bar{x}_K,\bar{a}_{k-1}) &= m_K(\bar{x}_K,\bar{a}_{K-1},g_{K,\nu}) & (16)\end{aligned}$$

Then define iteratively, for $k=K-1,\ldots,2$, the quantities

$$\begin{aligned} m_k(\bar{x}_k,\bar{a}_k) &= \mathbb{E}\{f_{k+1}(\bar{x}_k,X_{k+1},\bar{a}_k)\mid \bar{X}_k=\bar{x}_k,\bar{A}_k=\bar{a}_k\} & (17)\\ f_k(\bar{x}_k,\bar{a}_{k-1}) &= m_k(\bar{x}_k,\bar{a}_{k-1},g_{k\nu}) & (18)\end{aligned}$$

for $k=1$, $m_1(x_1,a_1)=\mathbb{E}\{f_2(x_1,X_2,a_1)\mid X_1=x_1,A_1=a_1\}$, $f_1(x_1)=m_1(x_1,g_{1,\nu})$.

The user must specify models $m_k(\bar{x}_k,\bar{a}_k;\beta_k)$, for $k=1,\ldots,K$. The fitted $m_k(\bar{x}_k,\bar{a}_k;\hat{\beta}_k)$ are then used to approximate $L_k^{opt}(\bar{x}_k)$ in Eq. (13).

$$\begin{aligned}AIPWE(\nu) =& n^{-1}\sum_{i=1}^{n}\left\{\frac{I(\mathcal{C}_i=\infty)}{K_K(\bar{X}_{ki},\bar{A}_{ki};\gamma)}Y_i\right.\\ &\left.+\sum_{k=1}^{K}\frac{I(\mathcal{C}_i=k)-\lambda_{k,\nu}(\bar{X}_{ki},\bar{A}_{ki};\gamma)I(\mathcal{C}_i\geq k)}{K_k(\bar{X}_{ki},\bar{A}_{ki};\gamma)}m_k(\bar{X}_{ki},\bar{g}_{k,\nu};\hat{\beta}_k)\right\}. \qquad (19)\end{aligned}$$

The algorithm for this method is as follows:

Value Search Algorithm - Coarsened Data Perspective		
For each decision point, k:		
CD1.	Modeling:	Regress A_k on $\bar{X}_k$, $\bar{A}_{k-1}$ to obtain $\pi_k(A_k, \bar{X}_k, \bar{A}_{k-1}; \hat{\gamma}_k)$.
CD2	Optimization	
For an estimate of ν		
	CD2a. Modeling:	Regress Y on $\bar{X}_K$ and $\bar{A}_K$ to obtain $m_K(\bar{x}_K, \bar{a}_K; \hat{\beta}_K) = \mu_K(\bar{x}_K, \bar{a}_{K-1}; \hat{\gamma}_K) + a_K \; \mathcal{C}_K(\bar{x}_K, \bar{a}_{K-1}; \hat{\eta}_K)$.
		Define $f_K(\bar{x}_K, \bar{a}_{K-1}) = m_K(\bar{x}_K, \bar{a}_{K-1}, g_{K,\nu}; \hat{\beta}_K)$
	For k=$K-1, \ldots, 1$	
	CD2b. Modeling:	Regress $f_{k+1}(\bar{X}_{k+1}, \bar{A}_k)$ on $\bar{X}_k$ and $\bar{A}_k$ to obtain $m_k(\bar{x}_k, \bar{a}_k; \hat{\beta}_k) = \mu(\bar{x}_k, \bar{a}_{k-1}; \hat{\gamma}_k) + a_k \; \mathcal{C}(\bar{x}_k, \bar{a}_{k-1}; \hat{\eta}_k)$.
		Define $f_k(\bar{x}_k, \bar{a}_{k-1}) = m_k(\bar{x}_k, \bar{a}_{k-1}, g_{k,\nu}; \hat{\beta}_k)$
	CD2c. Estimator:	Use $m_k(\bar{X}_k, \bar{g}_{k,\nu}; \hat{\beta}_k)$ and $\pi_k(g_{k,\nu}, \bar{X}_k, \bar{g}_{k-1}; \hat{\gamma}_k)$ $k = 1, \ldots, K$ to estimate $AIPWE(\nu)$ or $IPWE(\nu)$
	CD2d. Update:	Update ν
repeat CD2a-CD2d; terminate at convergence of estimator.		

Both the IPWE and AIPWE estimators are non-smooth functions in ν; accordingly, the use of traditional optimization methods to maximize these quantities in ν can be problematic. In *DynTxRegime*, the parameters of the pre-specified treatment regime are optimized using a genetic algorithm, *rgenoud* **?**.

Notice that the model $m(\bar{X}, g(\bar{X}_k, \nu); \beta)$ must be refit for each iteration in the optimization of ν. A practical alternative when the regime is derived from a model is to fit $m_k(\bar{X}_k, \bar{A}_{k-1}; \beta)$ using Q-learning and to hold $\hat{\beta}$ fixed during the optimization of ν. Both options are implemented in *DynTxRegime*.

4.3.1 The optimalSeq function

For multiple decision points, the call to *optimalSeq*() is very similar to that of the single-decision-point analysis. The primary difference between the calls is the use of lists rather than single objects.

```
optimalSeq(..., moPropen, moMain, moCont, data, response, txName, regimes,
           fSet = NULL, refit = FALSE, iter = 0L, suppress = FALSE)
```

Complete input argument names are required, the meaning of which follow.

- **moPropen**: a list of objects of class `modelObj` created by *buildModelObj*() of *modelObj*. The object in the k^{th} element of **moPropen** defines the regression step for the k^{th} propensity for treatment, $\pi_k(A, X; \gamma)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **moMain**: a list of objects of class `modelObj` created by *buildModelObj*() of *modelObj*. The object in the k^{th} element of **moMain** defines the regression step for the main effects component of the k^{th} outcome regression model, $\mu_k(\bar{X}_k, \bar{A}_{k-1}; \gamma_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **moCont**: a list of objects of class `modelObj` created by *buildModelObj*() of *modelObj*. The object in the k^{th} element of **moCont** defines the regression step for the contrasts component of the k^{th} outcome regression model, $\mathcal{C}_k(\bar{X}_k, \bar{A}_{k-1}; \eta_k)$. When defining the modeling object, the prediction method must return predictions on the scale of the response.
- **data**: an object of class `data.frame` containing the observed covariate and treatment histories. Treatments can be factors or integers.
- **response**: an object of class `vector`.T he final outcome of interest.
- **txName**: a vector of objects of class `character` specifying the column header of **data** that corresponds to the treatment variables.
- **regimes**: a list of objects of class `function`. The k^{th} element of this list is a user defined function that defines the k^{th} parameterized treatment regime, $g(\bar{X}_k, \bar{A}_{k-1}, \nu)$.
- **fSet**: a list of objects of class `function`. The k^{th} element of this list is a user defined function that specifies the rules for determining the feasible treatment options, $\Phi_k(\bar{x}_k, \bar{a}_{k-1})$, for an individual or modeling subset based on their covariate and treatment history.

- **refit**: an object of class `logical`. If **refit**=TRUE, the outcome regression model, $m(A, X; \beta)$ is refit at each iteration of the treatment regime optimization algorithm. If **refit**=FALSE, Q-learning is used.

- **iter**: an object of class `integer`. If **iter** = 0, the regression analyses for the main effects and contrast functions will be combined into a single regression step, i.e., the models specified in input arguments **moMain** and **moCont** will be combined into a single model, and parameter estimates will be obtained simultaneously. By default, the parameter estimates will be obtained using the regression tools specified in **moMain**. If **moMain** = NULL, the methods specified in **moCont** will be used. If **iter** ≥ 1, the **moMain** and **moCont** regression analyses will be performed separately using an iterative algorithm. The iterative algorithm is as follows:

  (1) $Y = Y_{main} + Y_{cont}$
  (2) $\hat{Y}_{cont} = 0$
  (3) $Y_{main} = Y - \hat{Y}_{cont}$
  (4) fit $Y_{main} \sim moMain$
  (5) $Y_{cont} = Y - \hat{Y}_{main}$
  (6) fit $Y_{cont} \sim A * moCont$
  (7) Repeat steps $(3) - (6)$
  until convergence or a maximum number of iterations.

  iter is the maximum number of iterations to be used to attain convergence.

- **suppress**: an object of class `logical` indicating if the final screen prints are to suppressed.

- **...** Additional arguments to be passed to *genoud*().

4.3.2 Illustrative Example

To illustrate this method, we will use the full **bmiData** dataset.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <- 1
> data$A1[ bmiData$A1=="CD" ] <- 0
> data$A2[ bmiData$A2=="MR" ] <- 1
> data$A2[ bmiData$A2=="CD" ] <- 0
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
```

The first step is to specify the model for the propensity for treatment, $\pi_k(\bar{A}_k, \bar{X}_k; \gamma_k)$. For both treatment stages, the treatment argument is binary, coded as $\{0, 1\}$, and we specify a logistic regression model. The model object for $\pi_k(\bar{A}_k, \bar{X}_k; \gamma)$ must be created using *buildModelObj*() of package *modelObj*.

```
> moPropen <- list()
> moPropen[[1]] <- buildModelObj(model = ~1,
+                                solver.method = 'glm',
+                                solver.args = list(family='binomial'),
+                                predict.method = 'predict.glm',
+                                predict.args = list(type='response'))
> moPropen[[2]] <- buildModelObj(model = ~1,
+                                solver.method = 'glm',
+                                solver.args = list(family='binomial'),
+                                predict.method = 'predict.glm',
+                                predict.args = list(type='response'))
```

Note that the default scale of the prediction for *predict.glm*() is on the scale of the linear predictors, not the response argument. Thus, variable **type** must be changed from its default setting.

Next, we specify the model objects for the outcome regression, $m_k(\bar{A}_k, \bar{X}_k; \beta_k)$. The model is specified as two components, one for the main effects of treatment, **moMain**, and a second for the contrasts among treatments, **moCont**.

```
> #
> #Models for first-stage regression
> ######
> moMainFS <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moContFS <- buildModelObj(model = ~ gender + parentBMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> #
> #Models for second-stage regression
> ######
> moMainSS <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
```

```
> moContSS <- buildModelObj(model = ~ parentBMI + month4BMI,
+                           solver.method = 'lm',
+                           predict.method = 'predict.lm',
+                           predict.args = list(type='response'))
> moMain <- list(moMainFS, moMainSS)
> moCont <- list(moContFS, moContSS)
```

regimes is the parameterized treatment regime and is provided to *optimalSeq*() as a function. The formal arguments must be the parameter names followed by "data." The function must return the vector of treatment assignments for the input data. Our treatment regime in this example is derived from the outcome regression models.

```
> regimes <- list()
> regimes[[1]] <- function(a, b, d, data){
+                   as.numeric(a + b*data$parentBMI + d*data$baselineBMI > 0 )
+                 }
> regimes[[2]] <- function(a, b, d, data){
+                   as.numeric(a + b*data$month4BMI + d*data$A1 > 0 )
+                 }
```

The function call for the genetic algorithm is *genoud*(). This method requires several additional pieces of information that are passed through the ellipsis of the call to *optimalSeq*(). At a minimum the information should include the search space for the parameters of the regimes

```
> c1 <- rep(-1000,6)
> c2 <- rep( 1000,6)
> Domains <- cbind(c1,c2)
```

Starting values for the parameters

```
> sv <- (1:6)/10
```

And a population size

```
> psize <- 100
```

We have opted for a VERY small population in this example to expedite the vignette. It is recommended that a much larger population be used, ≥ 1000.

The analysis is executed as follows:

```
> fitSeq2 <- optimalSeq(moPropen = moPropen,
+                       moMain = moMain, moCont = moCont,
+                       data = data, response = y, txName = c("A1", "A2"),
+                       regimes = regimes,
+                       pop.size = psize, starting.values = sv, Domains = Domains)


Augmented Inverse Probability Weighted Estimator

Genetic Algorithm
$value
[1] 10.07928

$par
[1] -542.74422 -624.75364  513.50143  -93.84508 -688.57458 -317.63630

$gradients
[1] NA NA NA NA NA NA

$generations
[1] 13

$peakgeneration
[1] 2

$popsize
[1] 100

$operators
[1] 15 12 12 12 12 12 12 12  0


Propensity for Treatment
Decision Point  1

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.07623

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            290.8
Residual Deviance: 290.8        AIC: 292.8
Decision Point  2
```

```
Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            291.1
Residual Deviance: 291.1         AIC: 293.1

Outcome Regression
Decision Point  1

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + race + parentBMI + baselineBMI +
    A1 + gender:A1 + parentBMI:A1, data = data)

Coefficients:
 (Intercept)        gender          race     parentBMI   baselineBMI
    34.28319      -1.02733       0.01416      -0.11703      -0.57426
          A1     gender:A1  parentBMI:A1
     9.09682       0.63783      -0.30022

Decision Point  2

 *** Combined Fit ***

Call:
lm(formula = YinternalY ~ gender + parentBMI + month4BMI + A2 +
    parentBMI:A2 + month4BMI:A2, data = data)

Coefficients:
 (Intercept)        gender     parentBMI     month4BMI            A2
    48.67554      -0.64891      -0.35732      -0.84883     -14.77418
parentBMI:A2  month4BMI:A2
     0.40447       0.05632


Regime Parameters:
[[1]]
[1] -542.7442 -624.7536  513.5014
```

```
[[2]]
[1]  -93.84508 -688.57458 -317.63630



Estimated Value: 10.07928
```

optimalSeq() returns an object that inherits from class `DynTxRegime`.

```
> is(fitSeq2,'DynTxRegime')


[1] TRUE
```

The complete value search algorithm for a multi-stage trial is performed with the single call to *optimalSeq*().

If **moMain**=NULL and **moCont**=NULL, the IPWE estimator is used.

```
> fitSeq2IPWE <- optimalSeq(moPropen = moPropen,
+                           moMain = NULL, moCont = NULL,
+                           data = data, response = y, txName = c("A1", "A2"),
+                           regimes = regimes,
+                           pop.size = psize, starting.values = sv, Domains = Domains)


Inverse Probability Weighted Estimator

Genetic Algorithm
$value
[1] 10.10778

$par
[1]  606.30643 -463.87915  354.84625  729.76665  305.16885   73.57075

$gradients
[1] NA NA NA NA NA NA

$generations
[1] 13

$peakgeneration
```

```
[1] 2

$popsize
[1] 100

$operators
[1] 15 12 12 12 12 12 12 12  0


Propensity for Treatment
Decision Point  1

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
   -0.07623

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            290.8
Residual Deviance: 290.8        AIC: 292.8
Decision Point  2

Call:  glm(formula = YinternalY ~ 1, family = "binomial", data = data)

Coefficients:
(Intercept)
          0

Degrees of Freedom: 209 Total (i.e. Null);  209 Residual
Null Deviance:            291.1
Residual Deviance: 291.1        AIC: 293.1

Regime Parameters:
[[1]]
[1]  606.3064 -463.8792  354.8463

[[2]]
[1] 729.76665 305.16885  73.57075



Estimated Value: 10.10778
```

Available standard regression analysis tools were discussed in Subsection 1.3. Additional methods specific to the *optimalSeq*() procedure follow.

All methods are the same as discussed in Section 4.1. However, most returned objects will be lists; the k^{th} element of the list corresponds to the results for the k^{th} decision point. For example,

```
> est <- regimeCoef(fitSeq2)
> est[[1]] <- est[[1]]/sqrt(est[[1]] %*% est[[1]])
> est[[2]] <- est[[2]]/sqrt(est[[2]] %*% est[[2]])
> est
```

```
[[1]]
[1] -0.5572628 -0.6414660  0.5272377

[[2]]
[1] -0.1228193 -0.9011686 -0.4157049
```

A list is returned. The contents of the k^{th} element of the list are the parameters for the k^{th}-stage regime.

The only exception to this structure is the estimated optimal treatment, where a matrix is returned; the k^{th} column corresponds to the k^{th} decision point.

```
> head(optTx(x = fitSeq2))
```

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

```
> optTx(x = fitSeq2, newdata = newpatient)
```

```
     dp= 1
[1,]    0
```

4.3.3 Feasible Treatments and Subset Modeling

As was illustrated for Q-learning, the *optimalSeq*() method allows feasible treatment subsets to be defined and for unique models to be specified for patient subsets. We will combine both of these features into a single example. Consider a clinical trial design in which a response

argument, R, is measured prior to second stage randomization. If $R = 0$, patients receive treatment (1-A1) at the second-stage. If $R = 1$, patients are randomized to one of two treatments $\{0, 1\}$ at the second-stage. In addition, we want to fit the first-stage regression for patients with **baselineBMI** < 37 using one model, and those with **baselineBMI** ≥ 37 using another.

Our earlier working example can be manipulated to illustrate this scenario.

```
> data <- bmiData
> data$A1[ bmiData$A1=="MR" ] <- 1
> data$A1[ bmiData$A1=="CD" ] <- 0
> data$A2[ bmiData$A2=="MR" ] <- 1
> data$A2[ bmiData$A2=="CD" ] <- 0
> data$A1 <- as.integer(data$A1)
> data$A2 <- as.integer(data$A2)
> data$R <- rbinom(nrow(data),1,0.5)
> data$A2[data$R==0] <- 1L - data$A1[data$R==0]
```

To create the dataset, we randomly generated a binary response argument with equal probability. For patients with $R = 0$, the second-stage treatment was set to $1 - A_1$. All patients have the same subset of treatments available to them. However, because we were defining a unique model for subsets of the data, **fSet** is used to define the subsets.

```
> fSet <- list()
> fSet[[1]] <- function(data){
+                 if( data$baselineBMI < 37 ) return(list("g1", c(0,1) ) )
+                 if( data$baselineBMI >= 37 )return(list("g2", c(0,1) ) )
+               }
> fSet[[2]] <- function(data){
+                 if( data$R == 0L ) return(list("g3", c(0,1) ) )
+                 if( data$R == 1L ) return(list("g4", c(0,1) ) )
+               }
```

We want to specify a unique model for each subset of patients in the first-stage regression. *buildModelObjSubset*() extends the *buildModelObj*() function of *modelObj* by allowing the user to specify the subset for which the model is defined. Because all models are communicated in a single call, we must add the optional **dp** argument to the call to *buildModelObjSubset*(). The model objects will now be lists and are defined as:

```
> moMain <- list()
> moCont <- list()
> #Models for second-stage outcome
```

```
> moMain[[1]] <- buildModelObjSubset(model = ~ gender + parentBMI + month4BMI + A1,
+                                    dp = 2,
+                                    subset = c("g3","g4"),
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> moCont[[1]] <- buildModelObjSubset(model = ~ month4BMI + A1,
+                                    dp = 2,
+                                    subset = c("g3","g4"),
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> #Models for first-stage outcome for subset g1
> moMain[[2]] <- buildModelObjSubset(model = ~ gender + race + parentBMI +
+                                                baselineBMI,
+                                    dp = 1,
+                                    subset = "g1",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> moCont[[2]] <- buildModelObjSubset(model = ~ parentBMI + baselineBMI,
+                                    dp = 1,
+                                    subset = "g1",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> #Models for first-stage outcome for subset g2
> moMain[[3]] <- buildModelObjSubset(model = ~ gender + race + parentBMI +
+                                                baselineBMI,
+                                    dp = 1,
+                                    subset = "g2",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
> moCont[[3]] <- buildModelObjSubset(model = ~ parentBMI + baselineBMI,
+                                    dp = 1,
+                                    subset = "g2",
+                                    solver.method = 'lm',
+                                    predict.method = 'predict.lm',
+                                    predict.args = list(type='response'))
```

Though the second-stage model is for all patients, it must be created using *buildModelObjSubset*(). All elements of a list passed to **moCont**, **moMain**, or **moPropen** must be of the same class.

We can also define subset models for the propensity of treatment. For example

```
> library(nnet)
> moPropen <- list()
> moPropen[[1]] <- buildModelObjSubset(model = ~1,
+                                      dp = 1,
+                                      subset = "g1",
+                                      solver.method = 'multinom',
+                                      predict.method = 'predict',
+                                      predict.args = list(type='probs'))
> moPropen[[2]] <- buildModelObjSubset(model = ~1,
+                                      dp = 1,
+                                      subset = "g2",
+                                      solver.method = 'multinom',
+                                      predict.method = 'predict',
+                                      predict.args = list(type='probs'))
> moPropen[[3]] <- buildModelObjSubset(model = ~1,
+                                      dp = 2,
+                                      subset = c("g3","g4"),
+                                      solver.method = 'multinom',
+                                      predict.method = 'predict',
+                                      predict.args = list(type='probs'))
```

The regime rules are:

```
> regimes <- list()
> regimes[[1]] <- function(a, b, d, data){
+                   as.numeric(a + b*data$parentBMI + d*data$baselineBMI > 0 )
+                 }
> regimes[[2]] <- function(a, b, d, data){
+                   res <- numeric(nrow(data))
+                   tst <- data$R < 0.5
+                   res[tst] <- 1-data$A1[tst]
+                   res[!tst] <- (a + b*data$month4BMI[!tst] + d*data$A1[!tst] > 0 )
+                   return(res)
+                 }
```

The function call is the same, with the exception of **fSet**. Because the calculation and screen print are lengthy, it is not executed within the vignette.

```
> fitSeq2_FS <- optimalSeq(moPropen = moPropen,
+                          moMain = moMain, moCont = moCont,
```

```
+                           fSet = fSet,
+                           data = data, response = y, txName = c("A1", "A2"),
+                           regimes = regimes, suppress = TRUE,
+                           pop.size = psize, starting.values = sv, Domains = Domains)
```

```
# weights:  2 (1 variable)
initial  value 69.314718
final  value 69.234697
converged
# weights:  2 (1 variable)
initial  value 76.246190
final  value 75.590340
converged
# weights:  2 (1 variable)
initial  value 145.560908
final  value 145.322723
converged
```

5 Conclusion

We have demonstrated how to estimate an optimal DTR using Q-learning, IQ-learning, and value search methods in the `R` package *DynTxRegime*.

Acknowledgments

The authors would like to thank Dr. Reneé Moore for discussions about meal replacement therapy for obese adolescents that informed the data generation model.

References

Bather, J. (2000). *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions.* Chichester: Wiley.

Berkowitz, R. I., Wadden, T. A., Gehrman, C. A., Bishop-Gilyard, C. T., Moore, R. H., Womble, L. G., Cronquist, J. L., Trumpikas, N. L., Katz, L. E. L., and Xanthopoulos, M. S. (2010). Meal replacements in the treatment of adolescent obesity: A randomized controlled trial. *Obesity*, 19(6):1193–1199.

Cao, W., Tsiatis, A. A., and Davidian, M. (2009). Improving efficiency and robustness of the doubly robust estimator for population mean with incomplete data. *Biometrika*, 96:723–732.

Carroll, R. J. and Ruppert, D. (1988). *Transformation and Weighting in Regression*. New York: Chapman and Hall.

Chakraborty, B., Murphy, S. A., and Strecher, V. J. (2010). Inference for non-regular parameters in optimal dynamic treatment regimes. *Statistical Methods in Medical Research*, 19(3):317–343.

Horvitz, D. G. and Thompson, D. J. (1952). A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685.

Laber, E. B., Lizotte, D. J., Qian, M., Pelham, W. E., and Murphy, S. A. (2010). Statistical inference in dynamic treatment regimes. *arXiv preprint arXiv:1006.5831*.

Linn, K. A., Laber, E. B., and Stefanski, L. (2014). Interactive model building for q-learning. *Biometrika*, page in press.

Murphy, S. A. (2005). A generalization error for q-learning. *Journal of Machine Learning Research*, 6(7):1073 – 1097.

Robins, J. M. (1986). A new approach to causal inference in mortality studies with sustained exposure periods – applications to control of the healthy worker survivor effect. *Mathematical Modeling*, 7:1393–1512.

Robins, J. M. (2004). Optimal structural nested models for optimal sequential decisions. In *Proceedings of the Second Seattle Symposium in Biostatistics*, pages 189–326. NY: Springer-Verlag.

Robins, J. M., Rotnitzky, A., and Zhao, L. P. (1994). Estimation of regression coefficients when some regressors are not always observed. *Journal of the American Statistical Association*, 89:846–866.

Schulte, P. J., Tsiatis, A. A., Laber, E. B., and Davidian, M. (2012). Q- and a-learning methods for estimating optimal dynamic treatment regimes. *arXiv:1202.4177 [stat.ME]*.

Song, R., Wang, W., Zeng, D., and Kosorok, M. R. (2011). Penalized q-learning for dynamic treatment regimes. *arXiv preprint arXiv:1108.5338*.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Zhang, B., Tsiatis, A. A., Davidian, M., Zhang, M., and Laber, E. B. (2012a). Estimating optimal treatment regimes from a classification perspective. *Stat*, 1(1):103–114.

Zhang, B., Tsiatis, A. A., Laber, E. B., and Davidian, M. (2012b). A robust method for estimating optimal treatment regimes. *Biometrics*, 68(4):1010–1018.

Zhang, B., Tsiatis, A. A., Laber, E. B., and Davidian, M. (2013). Robust estimation of optimal dynamic treatment regimes for sequential treatment decisions. *Biometrika*, 100:681–694.