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

September 24, 2014

Abstract

A treatment regime maps observed patient characterstics to a recommended treatment. Recent technological advances have increased the quality, accessibility, and volume of patient-level data; consequently, there is a growing need for powerful and flexible estimators of an optimal treatment regime that can be used with either observational or randomized clinical trial data. The `DynTxRegime` package implements four statistical methods for estimating the optimal treatment regime: Q-learning, Interactive Q-learning (IQ-learning), doubly robust estimation from a classification perspective, and a doubly robust Augmented Inverse Probability Weighted estimator. In this paper, we briefly describe the main components of 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.

1 Introduction

In practice, clinicians and intervention scientists must adapt treatment recommendations in response to the uniquely evolving health status of each patient. Dynamic treatment regimes (DTRs) formalize this treatment process as a sequence of decision rules, one for each treatment decision, that map current and past patient information to a recommended treatment. A DTR is said to be optimal for a pre-specified desirable outcome if it yields the maximal expected outcome when applied to assign treatment to a population of interest. Deducing optimal treatment regimes using data from a clinical trial or observational study can be informed, for example, in traditional regression-based methods, by identifying patient covariates that exhibit a qualitative interaction with treatment assignments; i.e., an interaction in which the treatment effect changes direction depending on the covariates (Gunter et al. 2011).

Q- and A-learning are two main approaches for estimating the optimal dynamic treatment regime using data from a clinical trial or observational study. Q-learning (Watkins & Dayan, 1992) involves postulating at each decision point regression models for outcome as a function of patient information to that point. In A-learning (Murphy, 2003; Robins, 2004), models are posited only for the part of the regression involving contrasts among treatment and for treatment assignment at each decision point. Both are implemented through a backward recursive fitting procedure based on a dynamic programming algorithm (Bather, 2000). Under certain assumptions and correct specification of these models, Q- and A-learning lead to consistent estimation of the optimal regime. At this time, only Q-learning is implemented in `DynTxRegime` as `qLearn`.

A concern with both Q-learning and A-learning is the effect of model misspecification on the quality of the estimated optimal regime. If one attempts to circumvent this difficulty by using flexible nonparametric regression techniques, the estimated optimal rules may be complicated functions of possibly high-dimensional patient information that are difficult to interpret or implement and thus are unappealing to clinicians wary of black box approaches.

In the single decision point setting, Zhang et al. proposed a novel and general framework for estimating the optimal treatment regime within a restricted class of regimes. The method maximizes a doubly robust augmented inverse probability weighted estimator for the population mean outcome over all regimes in the class. Via the doubly robustness property, the estimated optimal regimes enjoy protection against model misspecification and comparable or superior performance than do competing methods. The problem of finding the optimal treatment regime is recast as a weighted classification problem and the optimal treatment regime is estimated using the Bayes classifier, i.e., the one that minimizes the expected weighted misclassification error. Within this framework, the class of treatment regimes does not need to be prespecified, but is identified in a data-driven manner. The approach allows for existing classification algorithms to be used without modification to estimate the optimal treatment regime. This method is implemented in `DynTxRegime` as `optimal.class`.

Q-learning and similar variants of it involve modeling nonsmooth, nonmonotone functions of the data. Nonmonotonicity complicates the regression function whereas nonsmoothness imparts nonregularity to Q-learning estimators. Interactive Q-learning (IQ-learning) was proposed to model the data before applying the necessary nonmonotone, nonsmooth operations. The method uses standard interactive model building techniques and can be used with a modified version of Q-learning. IQ-learning involves simple, well-studied, and well-understood conditional mean and variance modeling of smooth transformation of the data, resulting in better fitting and more interpretable models. The method is defined for two-stage treatment regimes with binary treatments, the steps of which are implemented in `DynTxRegime` as `iqLearnSS`, `iqLearnFS`, and `iqLearnVar`.

Zhang et al. adapted the single decision point AIPWE approach of Zhang et al. to the multiple decision point setting by recasting the problem of as one of monotone coarsening. The proposed method focuses on a restricted class of treatment regimes indexed by a finite number of parameters, where the form of the regimes in the class depend on key subsets of patient information derived from posited regression models or prespecified on the grounds

of interpretability or cost. The methods lead to estimated optimal regimes that achieve comparable performance to those derived via Q- and A-learning under correctly specified models and have the added benefit of protection against misspecification. This method is implemented as `optimal` in the `DynTxRegime` package.

In the following sections, we describe the general framework of each method as implemented in the `DynTxRegime` package. In each section, at least one illustrative example is provided.

1.1 Dataset bmiData

Many of the methods in this vignette will be illustrated using a simulated dataset called `bmiData` which is included in the `DynTxRegime` package. The data were generated to mimic a two-stage SMART of body mass index (BMI) reduction with two treatments at each stage. The variables, treatments, and outcomes in `bmiData` were based on a small subset of variables collected in a clinical trial studying the effect of meal replacements (MRs) on weight loss and BMI reduction in obese adolescents; see `?` for a complete description of the original randomized trial. Descriptions of the generated variables in `bmiData` are given in Table (1). Baseline covariates include `gender`, `race`, `parent_BMI`, and `baseline_BMI`. 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 stage two. Thus, our generated data arises from a slightly difference design than that of the original trial. In addition, some patients in the original data set were missing the final twelve month response as well as 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 (for example, using an appropriate imputation strategy).

```
options(width=70)
```

After installing **DynTxRegime**, load the package:

```
library (DynTxRegime)
```

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

```
data (bmiData)
```

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

```
dim (bmiData)
head (bmiData)
```

gender $\in \{0, 1\}$	:	patient gender, coded female (0) and male (1).
race $\in \{0, 1\}$	:	patient race, coded African American (0) or other (1).
parent_BMI $\in \mathbb{R}$	:	parent BMI measured at baseline.
baseline_BMI $\in \mathbb{R}$	:	patient BMI measured at baseline.
A1 $\in \{-1, 1\}$	:	first-stage randomized treatment, coded so that A1 = 1 corresponds to meal replacement (MR) and A1 = -1 corresponds to conventional diet (CD).
month4_BMI $\in \mathbb{R}$	:	patient BMI measured at month 4.
A2 $\in \{-1, 1\}$	:	second-stage randomized treatment, coded so that A2 = 1 corresponds to meal replacement (MR) and A2 = -1 corresponds to conventional diet (CD).
month12_BMI $\in \mathbb{R}$	:	patient BMI measured at month 12.

Table 1: Description of variables in bmiData.

Recode treatments Meal Replacement (MR) and Conventional Diet (CD) as 1 and -1, respectively.

```
bmiData$A1[which (bmiData$A1=="MR")] = 1
bmiData$A1[which (bmiData$A1=="CD")] = -1
bmiData$A2[which (bmiData$A2=="MR")] = 1
bmiData$A2[which (bmiData$A2=="CD")] = -1
bmiData$A1 = as.numeric (bmiData$A1)
bmiData$A2 = as.numeric (bmiData$A2)
```

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

```
y <- -100*(bmiData$month12_BMI -
           bmiData$baseline_BMI)/bmiData$baseline_BMI
```

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

1.2 Defining models

The DynTxRegime package uses the modelObj package to define models and regression methods to be used for analysis. This choice is intended to generalize the methods available in the DynTxRegime package to any available regression method. The reader is referred to the vignette for the modelObj package for details. In general, the model, regression method, and prediction method are specified as follows:

```
moMain <- modelObj(model = ~ gender + parent_BMI + month4_BMI,
                   solver.method = 'glm',
                   solver.args = list(family='binomial'),
                   predict.method = 'predict.glm',
                   predict.args = list(type='response'))
```

where `model` is a formula object, `solver.method` can be any available `R` regression method, `solver.args` is a list of additional arguments to be passed to `solver.method`, `predict.method` is its corresponding prediction method, and `predict.args` is a list of additional arguments to be passed to `predict.method`. If `solver.args` and `predict.args` are not specified, default values are used. Note that a prediction method **must** be available for the specified regression method.

In our illustrations, we skip some of the typical exploratory techniques that a careful analyst would employ to find the best-fitting models. These steps would not be meaningful with the `bmiData` dataset because it was simulated with linear working models and would only detract 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. Another consequence of using generated data is that we will not interpret any coefficients or comment on model fit. All models and decision rules estimated in the following sections are strictly illustrative. In addition, when the `bmiData` dataset is used, the results are not representative of the results of the original meal replacement study.

2 General notation and problem specification.

We begin my developing a common notation and vocabulary with which we 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.

Assume that there are K prespecified, ordered decision points and an outcome of interest, Y, with larger values preferred. At each decision $k = 1, \ldots, K$, there are t k-specific treatments in the set of options $\mathcal{A}_k$, each element of $\mathcal{A}_k$ is denoted as a_k. For example, for a binary treatment option, $t = 2$, $a_k \in \{0, 1\}$ or $a_k \in \{-1, 1\}$. A possible treatment history up to and including decision k is denoted as $\bar{a}_k = (a_1, \ldots, a_k) \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_k = \bar{\mathcal{A}}_k$.

We will use a potential outcomes framework. For a randomly chosen patient, let X_1 denote baseline covariates recorded prior to the first decision. 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 $\bar{a}_{k-1} (k = 2, \ldots, K)$, taking 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, which is always observed and thus is not strictly a potential outcome, in W, and write $\bar{X}_k^*(\bar{a}_{k-1}) = \{X_1, X_2^*(a_1), \ldots, X_k^*(\bar{a}_{k-1})\}$ and $\bar{x}_k = (x_1, \ldots, x_k)$ for $k = 1, \ldots, K$, where then $\bar{x}_k \in \bar{\mathcal{X}}_k = \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$.

A dynamic treatment regime $g = (g_1, \ldots, g_K)$ is an ordered set of decision rules, where $g_k(\bar{x}_k, \bar{a}_{k-1})$ corresponding to the kth decision 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 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 pa-

tients with certain histories. For example, if treatments are binary, $\Phi_k(\bar{x}_k, \bar{a}_{k-1}) \subseteq \{0, 1\}$ or $\Phi_k(\bar{x}_k, \bar{a}_{k-1}) \subseteq \{-1, 1\}$. Thus, a feasible treatment regime must satisfy $g_k(\bar{x}_k, \bar{a}_{k-1}) \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})(k = 1, \ldots, K)$. Denote the class of all feasible regimes as $\mathcal{G}$.

For $g \in \mathcal{G}$, writing $\bar{g}_k = (g_1, \ldots, g_k)$ for $k = 1, \ldots, K$ and $\bar{g}_K = g$, 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 decision $k-1$ 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.

Define an optimal treatment regime $g^{opt} = (g_1^{opt}, \ldots, g_K^{opt}) \in \mathcal{G}$ as satisfying

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

That is, g^{opt} is a regime that maximizes the expected outcome were all patients in the population to follow it. The optimal regime, g^{opt} may be determined via dynamic programming, also referred to as backward induction. At the Kth decision point, for any $\bar{x}_K \in \mathcal{X}_K, \bar{a}_{k-1} \in \mathcal{A}_{K-1}$, define

$$g_K^{opt}(\bar{x}_K, \bar{a}_{K-1}) = \arg \max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{k-1})} E\{Y^*(\bar{a}_{K-1}, a_K) | \bar{X}_K^*(\bar{a}_{K-1}) = \bar{x}_K\}. \tag{2}$$

$$V_K(\bar{x}_K, \bar{a}_{K-1}) = \max_{a_K \in \Phi_K(\bar{x}_K, \bar{a}_{k-1})} E\{Y^*(\bar{a}_{K-1}, a_K) | \bar{X}_K^*(\bar{a}_{K-1}) = \bar{x}_K\}. \tag{3}$$

For $k = K-1, \ldots, 2$ and any $\bar{x}_k \in \bar{\mathcal{X}}_k, \bar{a}_{k-1} \in \bar{\mathcal{A}}_{k-1}$ define

$$g_k^{opt}(\bar{x}_k, \bar{a}_{k-1}) = \arg \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} E\{V_{k+1}\{\bar{x}_k, X_{k+1}^*(\bar{a}_{k-1}, a_k), \bar{a}_{k-1}, a_k\} | \bar{X}_k^*(\bar{a}_{k-1}) = \bar{x}_k\}.$$

$$V_k(\bar{x}_k, \bar{a}_{k-1}) = k \max_{a_k \in \Phi_k(\bar{x}_k, \bar{a}_{k-1})} E\{V_{k+1}\{\bar{x}_k, X_{k+1}^*(\bar{a}_{k-1}, a_k), \bar{a}_{k-1}, a_k\} | \bar{X}_k^*(\bar{a}_{k-1}) = \bar{x}_k\}.$$

And for $k = 1, x_1 \in \mathcal{X}_1$,

$$g_1^{opt}(x_1) = \arg \max_{a_1 \in \Phi_1(x_1)} E\left[V_2\{x_1, X_2^*(a_1), a_1\} | X_1 = x_1\right]$$

and

$$V_1(x_1) = \max_{a_1 \in \Phi_1(x_1)} E\left[V_2\{x_1, X_2^*(a_1), a_1\} | X_1 = x_1\right].$$

Thus, g_K^{opt} yields the treatment option at decision K that maximizes the expected potential outcome given prior covariate and treatment history. At decisions $k = K-1, \ldots, 1, g_k^{opt}$ dictates the option that maximizes the expected potential outcome that would be achieved if the otpimal rules were followed in the future. An argument that g^{opt}, so defined, satisfies Eq. (1) is given in a 2013 unpublished report by P.J. Schulte, A. A. Tsiatis, E. B. Laber and M. Davidian available from the last author.

This definition of an optimal regime is intuitively given in terms of potential outcomes. In practice, with the exception of X_1, W cannot be observed for any patient. Rather, a patient is observed to experience only a single treatment history. Let A_k 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 $k = 1 \ldots, K$ to decision 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 the following standard assumptions, g^{opt} may equivalently be expressed in terms of the observed data. The consistency assumption states that

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

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. The stable unit treatment value assumption (Rubin, 1978) implies that a patient's covariates and outcome are not influenced by treatments received by other patients. A version of the sequential randomization assumption (Robins, 2004) states that W is independent of A_k condition on $(\bar{X}_k, \bar{A}_{k-1})$. This is satisfied by default from a sequentially randomized clinical trial (Murphy, 2005), but is not verifiable from data from an observational study. It is reasonable to believe that decisions made in an observational study are based on a patient's covariate and treatment history; however, all such information associated with treatment assignment and outcome must be recorded in the $\bar{X}_k$ to validate the assumption.

Under these assumptions, it can be shown that $E\{Y^*(\bar{a}_{K-1} = \bar{x}_K\} = E(Y|\bar{X}_K = \bar{x}_K, \bar{A}_K = \bar{a}_K)$. To ease notation, we define a history, $\mathcal{H}_k = \{\bar{X}_k, \bar{A}_{k-1}\}$ denoting all available patient information up to decision point k, where $\mathcal{H}_1 = \{X_1\}$. Thus, letting $Q_K(\bar{x}_K, \bar{a}_K) = Q_K(h_K, a_K) = E(Y|\bar{X}_K = \bar{x}_K, \bar{A}_K = \bar{a}_K)$, Eqs. (2) and (3) become

$$g_K^{opt}(h_K) = \arg \max_{a_K \in \Phi_K(h_k)} Q_K(h_K, a_K),$$

$$V_K(h_K) = \max_{a_K \in \Phi_K(h_K)} Q_K(h_K, a_K),$$

Similarly, for $k = K, \ldots, 2$,

$$Q_k(h_k, a_k) = E(V_{k+1}(h_{k+1})|\bar{X}_k = \bar{x}_k, \bar{A}_k = \bar{a}_k)(k = K-1, \ldots, 1),$$

$$g_k^{opt}(h_k) = \arg \max_{a_k \in \Phi_k(h_k)} Q_k(h_k, a_k), (k = K-1, \ldots, 2)$$

$$V_k(h_k) = \max_{a_k \in \Phi_k(h_k)} Q_k(h_k, a_k), (k = K-1, \ldots, 2),$$

and

$$g_1^{opt}(x_1) = \arg \max_{a_1 \in \Phi_1(x_1)} Q_1(x_1, a_1),$$

$$V_1(x_1) = \max_{a_1 \in \Phi_1(x_1)} Q_1(x_1, a_1),$$

The $Q_k(h_k, a_k)$ and $V_k(h_k)$ are referred to as Q-functions and value functions, respectively, and are derived from the distribution of the observed data.

2.1 Q-learning

Q-learning is based on the developments in the preceding section. Linear or non-linear models $Q_k(h_k, a_k; \beta_k)$ in a finite-dimensional parameter β_k may be posited and estimators $\hat{\beta}_k$ obtained via a backward iterative process for $k = K, \ldots, 1$.

In the two decision points setting, the Q-functions are defined as:

$$\begin{aligned} Q_2(h_2, a_2) &\triangleq \mathbb{E}(Y | \mathcal{H}_2 = h_2, A_2 = a_2) \\ Q_1(h_1, a_1) &\triangleq \mathbb{E}\left(\max_{a_2 \in \Phi_2(h_2)} Q_2(h_2, a_2) | \mathcal{H}_1 = h_1, A_1 = a_1 \right). \end{aligned}$$

The Q-function at stage two measures the **Q**uality of assigning a_2 to a patient presenting with history h_2. Similarly, Q_1 measures the quality of assigning a_1 to a patient with h_1, assuming an optimal decision rule will be followed at stage two. Were the Q-functions known, dynamic programming gives the optimal solution, $g^{opt} = (g_1^{opt}, g_2^{opt})$. Because the underlying distribution of the patient histories is not known, the conditional expectations that define the Q-functions are unknown and must be approximated. Q-learning approximates the Q-functions with regression models: $Q_k(h_k, a_k) = \mu_k(h_k; \gamma_k) + a_k C_k(h_k; \beta_k),\ k = 1, 2$, where $\mu_k(h_k; \gamma)$ models the main effects and $C_k(h_k; \beta)$ models the contrasts among treatments. Define $\eta_k \triangleq (\gamma_k^\top, \beta_k^\top)^\top$. The Q-learning algorithm is given below.

Q-learning Algorithm:

Q1.	Modeling:	Regress Y on $\mathcal{H}_2$ and A_2 to obtain $\widehat{Q}_2(h_2, a_2; \widehat{\eta}_2) = \mu_2(h_2; \hat{\gamma}_2) + A_2 C_2(h_2; \hat{\beta}_2)$.
	Maximization:	Define $V_2(h_2) \triangleq \max_{a_2 \in \Phi_2(h_2)} \widehat{Q}_2(h_2, a_2; \widehat{\eta}_2)$. $V_2(h_2)$ is the predicted future outcome assuming the optimal decision is made at stage two.
		Define $g_2^{opt}(h_2) \triangleq \arg\max_{a_2 \in \Phi_2(h_2)} \widehat{Q}_2(h_2, a_2; \widehat{\eta}_2)$. $g_2^{opt}(h_2)$ is the predicted optimal decision at stage two for each observation.
Q2.	Modeling:	Regress $V_2(h_2)$ on $\mathcal{H}_1$ and A_1 to obtain $\widehat{Q}_1(h_1, a_1; \widehat{\eta}_1) = \mu_1(h_1; \hat{\gamma}_1) + A_1 C_1(h_1; \hat{\beta}_1)$.
	Maximization:	Define $g_1^{opt}(h_1) \triangleq \arg\max_{a_1 \in \Phi_1(h_1)} \widehat{Q}_1(h_1, a_1; \widehat{\eta}_1)$. $g_1^{opt}(h_1)$ is the predicted optimal decision at stage one for each observation.

The k^{th}-stage optimal decision rule then assigns the treatment a_k that maximizes the estimated Q_k-function,

$$\widehat{g}_k^{opt}(\boldsymbol{h}_k) = \arg\max_{a_k} \widehat{Q}_k(\boldsymbol{h}_k, a_k; \widehat{\eta}_k).$$

2.2 Q-learning using `DynTxRegime`

Function `qLearn()` implements a single regression step of the Q-learning algorithm.

```
qLearn(moMain, moCont, data, txName, iter=0,
       object=NULL, response=NULL, fSet=NULL)
```

`moMain` and `moCont` are objects of class `modelObj` used to communicate the models $\mu_k(h_k; \gamma_k)$ and $C_k(h_k; \beta_k)$ as well as the regression and prediction methods. The main effect model, $\mu_k(h_k; \gamma_k)$, is defined by `moMain`. Similarly, `moCont` defines the contrast model, $C_k(h_k; \beta_k)$. `data` is a data.frame of covariates and treatments. `txName` is a character string indicating the column header of `data` containing the treatment variable for the stage under analysis. `iter` is a numeric value indicating how the main effect and contrast models are to be fit. Specifically, if the `R` methods and method arguments specified in `moMain` and `moCont` are the same, the user can request that the main effect and contrast models be combined into a single formula and fit in a single step by specifying `iter`=0. If different `R` methods are required, the two models will be fit iteratively, and `iter` must be >= 1, indicating the maximum number of iterations allowed to attain convergence. `object` contains the results of a previous Q-learning step and will be discussed in detail below. `response` is the response vector. And, `fSet` is a variable for defining the feasible treatment options, $\Phi_k(h_k)$; this variable will be discussed in detail in the next subsection.

We will use the bmiData dataset for this example. For the final-stage regression (second-stage in our example), we will use a linear model for the main effect and another for the contrast. The model objects are created by calling modelObj as follows:

```
moMain <- modelObj(model = ~ gender + parent_BMI + month4_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

moCont <- modelObj(model = ~ parent_BMI + month4_BMI,
                   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 variables 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 main effect and contrast models can be fit in a single regression step; in the call to qLearn(), we will specify iter=0. And 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 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 always explicitly set the scale of the predictions if possible.

For the final stage regression, the final outcome variable is passed to qLearn() through input response.

```
fitQ2 <- qLearn(moMain = moMain,
                moCont = moCont,
                data = bmiData,
                txName = 'A2',
                response = y,
                iter = 0)
fitQ2
```

Method qLearn() returns an object of class qLearn. There are several methods available for objects of class qLearn, including: coef(), contrast(), getFit(), main(), optTx(), plot(), residuals(), summary(), and ValueFuncs(). Methods coef() and plot() are applied to the fit objects returned by the specified R regression method, and thus must be defined by the regression method(s) specified in moMain and/or moCont. Method getFit() retrieves the value object returned by the regression method(s), and thus all functionality available through said method can be accessed. If an iterative fit is specified, getFit() returns a list containing the main effect value object $mainEffects and the contrast value

object `$contrast`. Methods `contrast()` and `main()` return the fitted contrast function and main effect function, respectively. Method `optTx()` returns the recommended stage treatment for all patients in the study, or if data for a new patient is provided, the recommended treatment. Finally, method `ValueFuncs()` returns the fitted Q-function for each treatment option.

For all other stages of the analysis, the previous-stage `qLearn` object is passed through input `object`; `response` is provided only for the final stage analysis. In our example, the first-stage analysis follows:

```
moMain <- modelObj(model = ~ gender + race + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

moCont <- modelObj(model = ~ gender + parent_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

fitQ1 <- qLearn(moMain = moMain,
                moCont = moCont,
                data = bmiData,
                txName = 'A1',
                object = fitQ2,
                iter = 0)
fitQ1
```

Again, an object of class `qLearn` is returned. See above for available methods for objects of this class.

Recommend treatment with optTx()

To recommend the Q-learning estimated optimal treatments for a new patient based on observed histories, method `optTx()` is available. It requires the observed history vector for a new patient. For the first-stage optimal treatment:

```
newdata <- data.frame(1, 1, 30, 35)
colnames(newdata) <- c("gender","race","parent_BMI","baseline_BMI")
optQ1 <- optTx(x=fitQ1, newdata=newdata)
optQ1
```

As displayed above, a list is returned by `optTx()` that includes `$valueFuncs`, the vector of the first-stage Q-functions at each treatment value, as well as the recommended first-stage treatment for that patient, `$optTx`.

The recommended second-stage optimal treatment is similarly obtained:

```
newdata <- data.frame(1, 30, 45)
colnames(newdata) <- c("gender","parent_BMI","month4_BMI")
optQ2 <- optTx(x = fitQ2,
               newdata = newdata)
optQ2
```

Feasible Treatment Sets

In some trial settings, the treatment options available to a patient at each decision point, $\Phi_k(h_k)$, 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, for patients randomized to CD in stage one, the set of feasible treatment options at the second-stage is restricted to $\{CD\}$ or $\{-1\}$. For all other patients, the set of feasible treatment options at the second-stage is $\{MR, CD\}$ or $\{-1, 1\}$. 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 each patient.

The method `qLearn()` has an optional input variable, `fSet`, which allows the user to specify the available treatments, based on a patient history. These rules are defined by the user as a function. The function takes as input `data`, the covariates and treatments of a single patient, and returns a vector of treatment options available to that patient. For the second stage of the original study, this rule would take the following form

```
fSet <- function(data){
          if(data$A1 == -1){
            return(-1)
          } else {
            return(c(-1,1))
          }
        }
```

The second-stage Q-learning analysis would be

```
moMain <- modelObj(model = ~ gender + parent_BMI + month4_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

moCont <- modelObj(model = ~ parent_BMI + month4_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

fitQ2 <- qLearn(moMain = moMain,
```

```
                moCont = moCont,
                data = bmiData,
                txName = 'A2',
                response = y,
                fSet = fSet,
                iter = 0)
fitQ2
```

Note, a `qLearn` object is again returned, to which the previously discussed methods can be applied. However, the vectors or matrices returned by `contrast()`, `main()`, and `ValueFuncs()` will include NAs for the patients not included in the regression analysis (i.e., have only one treatment option available).

For the first-stage analysis, all treatments are available to all patients, and the input variable `fSet` is not required. Here, the analysis is initiated as was previously shown. The `qLearn` object returned by the stage two analysis is provided as input and includes all necessary information regarding previous stage treatment rules.

```
moMain <- modelObj(model = ~ gender + race + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

moCont <- modelObj(model = ~ gender + parent_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

fitQ1 <- qLearn(moMain = moMain,
                moCont = moCont,
                data = bmiData,
                txName = 'A1',
                object = fitQ2,
                iter = 0)
fitQ1
```

When treatment recommendations are obtained using `optTx`, the feasible set of treatment options defined in the analysis is used to determine the optimal treatment.

```
newdata = data.frame(1, 1, 32, 35)
colnames(newdata) <- c("gender","race","parent_BMI","baseline_BMI")
optQ1 = optTx(x=fitQ1, newdata=newdata);
optQ1

newdata = data.frame(1, 30, 45,optQ1$optTx)
```

```
colnames(newdata) <- c("gender","parent_BMI","month4_BMI","A1")
optQ2 = optTx(x = fitQ2,
              newdata = newdata);
optQ2
```

Note that only one value function is returned by `optTx()` for the second-stage treatment recommendation in this example because the first-stage treatment was -1.

```
newdata = data.frame(1, 1, 28, 35)
colnames(newdata) <- c("gender","race","parent_BMI","baseline_BMI")
optQ1 = optTx(x=fitQ1, newdata=newdata);
optQ1
```

```
newdata = data.frame(1, 30, 45,optQ1$optTx)
colnames(newdata) <- c("gender","parent_BMI","month4_BMI","A1")
optQ2 = optTx(x = fitQ2,
              newdata = newdata);
optQ2
```

In addition to specifying feasible treatment sets for each patient, the `qLearn()` method allows for unique models to be specified for each subset of treatments. For further information regarding this feature, please see the documentation for `subsetModel`.

2.3 Interactive Q-learning

The first modeling step in the Q-learning algorithm described in the previous section 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 (Q2) requires modeling the conditional expectation of $\widetilde{V}_2$. This can be written as

$$\begin{aligned} Q_1(\boldsymbol{H}_1, A_1) &= \mathbb{E}(V_2(h_2)|\mathcal{H}_1, A_1) \\ &= \mathbb{E}(\mu_2(h_2;\gamma_2) + |C_2(h_2;\beta_2)| \mid \boldsymbol{H}_1, A_1). \end{aligned} \tag{4}$$

where we are assuming a binary treatment option $\{-1, 1\}$. Due to the absolute value function, $V_2(h_2)$ is a nonsmooth, nonmonotone transformation of
$mathcal H_2$. Thus, the model in step Q2 is generally misspecified. In addition, the nonsmooth, nonmonotone max operator in step Q1 leads to difficult nonregular inference for the parameters that index the first stage Q-function (**????**). IQ-learning was developed as an alternative to Q-learning that addresses the applied problem of building good models for the first-stage Q-function 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 $\{-1, 1\}$.

IQ-learning differs from Q-learning in the order in which the maximization step is performed. We demonstrate how the maximization step can be 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 more accurately estimates the first-stage Q-function, resulting in a higher-quality estimated decision rule (**?**). Another advantage of IQ-learning is that in many cases, conditional mean and variance modeling techniques (**?**) offer a nice framework for the necessary modeling steps. These mean and variance models are interpretable, and the coefficients indexing them enjoy normal limit theory. Thus, they are better suited to inform clinical practice than the misspecified first-stage model in Q-learning whose indexing parameters are nonregular. However, the mean-variance modeling approach we advocate here is not necessary and other modeling techniques may be applied as needed. Indeed, a major advantage and motivation for IQ-learning is the ability for the seasoned applied statistician to build high-quality models using standard interactive techniques for model diagnosis and validation.

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

$$Q_1(\boldsymbol{h}_1, a_1) \triangleq \mathbb{E}(\mu(\boldsymbol{H}_2; \beta_2)|\boldsymbol{H}_1 = \boldsymbol{h}_1, A_1 = a_1) + \int |z| f(z \mid \boldsymbol{h}_1, a_1) dz, \tag{5}$$

where $f(\cdot \mid \boldsymbol{h}_1, a_1)$ is the conditional distribution of the contrast function $\Delta(\boldsymbol{H}_2; \beta_2)$ given $\boldsymbol{H}_1 = \boldsymbol{h}_1$ and $A_1 = a_1$. In fact, Equation 5 is equivalent to the representation of Q_1 in Equation 4, only 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 Equation 4 directly, IQ-learning separately models $\mathbb{E}(\mu(\boldsymbol{H}_2; \beta_2)|\boldsymbol{H}_1 = \boldsymbol{h}_1, A_1 = a_1)$ and $f(\cdot \mid \boldsymbol{h}_1, a_1)$. Although IQ-learning trades one modeling step for two, splitting up the conditional expectation in Equation 4 is advantageous because the terms that require modeling are now smooth, monotone functionals of the data. The maximization occurs when the integral in Equation 5 is computed, which occurs after the conditional density $f(\cdot \mid \boldsymbol{h}_1, a_1)$ has been estimated. The IQ-learning algorithm is described next.

IQ-learning Algorithm:

IQ1. Modeling:	Regress Y on $\mathcal{H}_2, A_2$ to obtain $\widehat{Q}_2^{IQ}(\mathcal{H}_2, A_2; \widehat{\beta}_2) = \mu_2(h_2; \gamma_2) + A_2 C_2(h_2; \beta_2)$.
IQ2. Modeling:	Regress observed data $\mu_2(\mathcal{H}_{2,i}; \widehat{\gamma}_2)_{i=1}^n$ on $\{\mathcal{H}_{1,i}, A_{1,i}\}_{i=1}^n$ to obtain an estimator $\widehat{\ell}(\mathcal{H}_1, A_1)$ of $\mathbb{E}(\mu_2(\mathcal{H}_2; \gamma_2)\|\mathcal{H}_1, A_1)$.
IQ3. Modeling:	Use $\{C_2(\mathcal{H}_{2,i}; \widehat{\beta}_2), \mathcal{H}_{1,i}, A_{1,i})\}_{i=1}^n$ to obtain an estimator $\widehat{f}(\cdot \mid \mathcal{H}_1, A_1)$ of $f(\cdot \mid \mathcal{H}_1, A_1)$.
IQ4. Maximization:	Combine the above estimators to form $\widehat{Q}_1^{IQ}(\mathcal{H}_1, A_1) = \widehat{\ell}(\mathcal{H}_1, A_1) + \int \|z\| \widehat{f}(z \mid \mathcal{H}_1, A_1) dz$.

The IQ-learning estimated optimal DTR assigns the treatment at stage k as the maximizer of the estimated stage-k Q-function $\widehat{g}_k^{\text{IQ-opt}}(h_k) = \arg\max_{a_k} \widehat{Q}_k^{IQ}(h_k, a_k; \widehat{\eta}_k)$.

Remark about density estimation in IQ3

Step IQ3 in the IQ-learning algorithm requires estimating a one-dimensional conditional density. In **?** we accomplish this using mean-variance, location-scale estimators of $f(\cdot \mid h_1, a_1)$ of the form

$$\widehat{f}(z \mid h_1, a_1) = \frac{1}{\widehat{\sigma}(h_1, a_1)} \widehat{\phi} \left(\frac{z - \widehat{C_2}(h_1, a_1)}{\widehat{\sigma}(h_1, a_1)} \right),$$

where $\widehat{C_2}(\boldsymbol{h}_1, a_1)$ is an estimator of $C_2(\boldsymbol{h}_1, a_1) \triangleq \mathbb{E}\{\Delta(\boldsymbol{H}_2; \beta_2) \mid \boldsymbol{H}_1 = \boldsymbol{h}_1, A_1 = a_1\}$, $\widehat{\sigma}^2(\boldsymbol{h}_1, a_1)$ is an estimator of $\sigma^2(\boldsymbol{h}_1, a_1) \triangleq \mathbb{E}\{(\Delta(\boldsymbol{H}_2; \beta_2) - C_2(\boldsymbol{h}_1, a_1))^2 \mid \boldsymbol{H}_1 = \boldsymbol{h}_1, A_1 = a_1\}$, and $\widehat{\phi}$ is an estimator of the density of the standardized residuals $\{\Delta(\boldsymbol{H}_2; \beta_2) - \mu(\boldsymbol{h}_1, a_1)\} / \sigma(\boldsymbol{h}_1, a_1)$, say ϕ_{h_1, a_1}, which we assume does not depend on the history $\boldsymbol{h}_1$ or the treatment a_1. Mean-variance function modeling tools are well-studied and applicable in many settings (**?**). Currently, `DynTxRegime` implements mean-variance modeling steps to estimate $f(\cdot \mid \boldsymbol{h}_1, a_1)$ with the option of using a standard normal density or empirical distribution estimator for $\widehat{\phi}$.

3 Using the `DynTxRegime` Package for IQ-learning

3.1 IQ-learning functions

The current version of the `DynTxRegime` package allows specification of models linear in the treatment variable at all modeling steps. An advantage of IQ-learning over Q-learning is that for a large class of generative models, models are correctly specified at each modeling step (Laber et al., 2013). In general, this is not true for Q-learning at the first-stage.

STEP IQ1: second-stage regression

The first step in the IQ-learning algorithm is to model the response as a function of second-stage history variables and treatment. We model the second-stage Q-function as a linear function of `gender`, `parent_BMI`, `month4_BMI`, and `A2`, fitting the model using least squares. All methods in the `DynTxRegime` package require that the model be specified as a main effects (ME) model and a contrast (c) model. These models can be fit separately using an iterative method or can be combined as ME + tx:C. Below, the ME and C models are combined and fit as a single model (iter = 0).

```
moMain <- modelObj(model = ~ gender + parent_BMI + month4_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
moCont <- modelObj(model = ~ parent_BMI + month4_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
fitIQ2 = iqLearnSS(moMain = moMain,
                   moCont = moCont,
                   data = bmiData,
                   response = y,
                   txName = 'A2',
                   iter = 0)
```

In this example, the function `iqLearnSS()` creates an object of class `iqLearnSS` that contains the object returned by the user specified regression method (e.g., 'lm') in addition to several other components. The user can specify any model, but it must include the main effect of treatment (moMain) and at least one treatment interaction term (moCont). If exploratory work suggests there are no treatment-by-covariate interactions at the second stage, IQ-learning has no advantage over Q-learning, and it would be appropriate to model the conditional expectation of $\widetilde{Y}$ directly at the first stage. In addition to the two modelObjs, the function `iqLearnSS()` requires as input: `data` a data.frame of patient covariates; `response` the response vector of interest; `txName` a character string specifying the name of the treatment variable in the provided dataset; and `iter`. The main effect and contrast models can be fitted as a single model (`iter`=0) or using an iterative method (`iter` ¿ 0), where `iter` is the maximum number of iterations to attain convergence.

There are several methods available for objects of class `iqLearnSS`, including: `coef()`, `contrast()`, `getFit()`, `main()`, `optTx()`, `plot()`, `residuals()` and `summary()`. Methods `coef()` and `plot()` must be defined by the regression method specified by the user. Method `getFit()` retrieves the value object returned by the regression method, and thus all functionality available through said method can be accessed. Methods `contrast()` and `main()` return the fitted contrast function and main function, respectively. Method `optTx()` returns the recommended second-stage treatment for all patients in the study.

To illustrate some of these methods: to print the regression output we can call a `summary(iqLearnSS)` or retrieve the fit object and calls its summary methood.

```
summary(fitIQ2)
fitObj <- getFit(fitIQ2)
summary(fitObj)
```

The `plot()` function can be used to obtain fit diagnostics. Figure 1 shows the residual diagnostics for this example. The `coef()` function can be used to retrieve the estimated

```
plot(fitIQ2,sub.caption="Second-Stage Regression")
```

Figure 1: Residual diagnostic plots from the second-stage regression in IQ-learning.

coefficients,

```
coef(fitIQ2)
coef(fitObj)
```

STEP IQ2: main effect function regression

The next step in the IQ-learning algorithm is to model the conditional expectation of the main effect term given first-stage history variables and treatment. We accomplish this by regressing $\{\boldsymbol{H}_{20,i}^{\intercal}\widehat{\beta}_{20}\}_{i=1}^{n}$ on a linear function of $\{\boldsymbol{H}_{1,i}, A_{1,i}\}_{i=1}^{n}$ using the function `iqLearnFS()` with argument `step`='M', which creates an object of class `iqLearnFS.ME`. The `iqLearnFS()` function extracts the estimated vector of main effect terms from the `iqLearnSS` object to use as the response variable in the regression.

```
moMain <- modelObj(model = ~ gender + race + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
moCont <- modelObj(model = ~ gender + parent_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
fitIQ1main = iqLearnFS(moMain = moMain,
                       moCont = moCont,
                       data = bmiData,
                       step = 'M',
                       object = fitIQ2,
                       txName = 'A1',
                       iter = 0)
summary (fitIQ1main);
```

The user can specify any model, but it must include the main effect of treatment `A1`. If no treatment interactions are desired, `moCont` should be specified as `NULL`. In addition to

the two modelObjs (or a single modelObj and NULL, the function iqLearnFS() requires as input: data a data.frame of patient covariates; step one of c('M','C') indicating which term of the second-stage regression is the response variable; object an object of class iqLearnSS, which contains the second-stage regression; txName a character string specifying the name of the treatment variable in the provided dataset; and iter. The main effect and contrast models can be fitted as a single model (iter=0) or using an iterative method (iter ¿ 0), where iter is the maximum number of iterations to attain convergence. Again, there are several methods available for objects of class iqLearnFS.ME, including: coef(), contrast(), getFit(), main(), plot(), residuals() and summary(). See description of second-stage regression for further details on these methods.

In our example, plot() gives residual diagnostic plots from the fitted regression model, shown in Figure 2.

```
plot(fitIQ1main)
```

Figure 2: Residual diagnostic plots from the regression model for the main effect term.

STEP IQ3: contrast function density modeling

The final modeling step in IQ-learning is to model the conditional density of the contrast function given first-stage history variables and treatment. We will accomplish this by considering the class of location-scale density models and employing standard conditional mean and variance modeling techniques. Thus, we begin by modeling the conditional mean of the contrast function using iqLearnFS() with step = 'C'.

```
moMain <- modelObj(model = ~ gender + race + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
moCont <- modelObj(model = ~ gender + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
fitIQ1cm = iqLearnFS(moMain = moMain,
                        moCont = moCont,
                        data = bmiData,
                        step = 'C',
                        object = fitIQ2,
                        txName = 'A1',
                        iter = 0)
summary (fitIQ1cm)
```

The user can specify any model, but it must include the main effect of treatment A1. If no treatment interactions are desired, moCont should be specified as NULL. In addition to

the two `modelObj`s (or a single `modelObj` and `NULL`, the function `iqLearnFS()` requires as input: `data` a data.frame of patient covariates; `step` one of c('M','C') indicating which term of the second-stage regression is the response variable; `object` an object of class `iqLearnSS`, which contains the second-stage regression; `txName` a character string specifying the name of the treatment variable in the provided dataset; and `iter`. The main effect and contrast models can be fitted as a single model (`iter`=0) or using an iterative method (`iter` ¿ 0), where `iter` is the maximum number of iterations to attain convergence. Again, there are several methods available for objects of class `iqLearnFS.C`, including: `coef()`, `contrast()`, `getFit()`, `main()`, `plot()`, `residuals()` and `summary()`. See description of second-stage regression for further details on these methods.

Figure (3) displays the residual diagnostics produced by `plot()`.

```
plot (fitIQ1cm)
```

Figure 3: Residual diagnostic plots from the linear regression model for the contrast function mean.

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

```
fitIQ1var = iqLearnVar (fitIQ1cm)
```

is called to estimate the common standard deviation. An object of class `iqLearnFS.VHom` is returned, which contains the estimated common standard deviation of the contrast mean fit residuals (`stdDev`), the vector of standardized residuals for each patient in the dataset (`residuals`), among others.

If the variance is thought to be non-constant across histories $\boldsymbol{H}_1$ and/or treatment A_1, a log-linear model for the squared residuals can be used. As before, the formula should be only right-hand sided and must include the main effect of treatment `A1`.

```
moMain <- modelObj(model = ~ gender + race + parent_BMI + baseline_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))
moCont <- modelObj(model = ~ parent_BMI,
                   solver.method = 'lm',
                   predict.method = 'predict.lm',
                   predict.args = list(type='response'))

fitIQ1var = iqLearnVar(object=fitIQ1cm,
                       moMain = moMain,
                       moCont = moCont,
                       data = bmiData,
```

```
                        iter = 0)
summary (fitIQ1var)
```

In addition to the two `modelObj`s (or a single `modelObj` and `NULL`, the function `iqLearnFS()` requires as input: `data` a data.frame of patient covariates; `object` an object of class `iqLearnFS.C`; and `iter` the maximum number of iterations to attacin convergence. If `moMain` and `moCont` are to be combined into a single fit, `iter` = 0. Again, there are several methods available for objects of class `iqLearnFS.VHet`, including: `coef()`, `contrast()`, `getFit()`, `main()`, `plot()`, `residuals()` and `summary()`. See description of second-stage regression for further details on these methods.

Figure (4) displays the residual diagnostics produced by `plot()`.

```
plot (fitIQ1var)
```

Figure 4: Residual diagnostic plots from the log-linear variance model.

The final step in the conditional density modeling process is to choose between the normal and empirical density estimators. Based on empirical experiments (see **?**), we recommend choosing the empirical estimator by default, as not much is lost when the true density is normal.

!!! However, `plot()` with formal argument `y`='d' can be used to inform the choice of density estimator. An object of type `iqLearnVar.VHet` can be plotted to obtain a normal QQ-plot of the standardized residuals, displayed in Figure 5. If the observations deviate from the line, `dens='nonpar'` should be used in the final IQ-learning step, IQ4.

```
plot (fitIQ1var,'d')
```

Figure 5: Normal QQ-plot of the standardized residuals obtained from the contrast mean and variance modeling steps.

STEP IQ4: combine first-stage estimators

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.

```
fitIQ1 = optTx (x=fitIQ1main, y=fitIQ1cm, z=fitIQ1var, dens="nonpar")
```

A vector of estimated optimal first-stage treatments for patients in the study is returned.

Recommend treatment with optTx()

After estimating the optimal regime using the IQ-learning algorithm, the function `optTx()` can be used to recommend treatment for future patients. To determine the recommended first-stage treatment for a patient with observed data in `newdata`

```
newdata = data.frame(1, 1, 30, 35)
colnames(newdata) <- c("gender","race","parent_BMI","baseline_BMI")
optIQ1 = optTx(x = fitIQ1main,
               y = fitIQ1cm,
               z = fitIQ1var,
               dens = "nonpar",
               newdata = newdata)
optIQ1
```

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

For a patient with observed data in `newdata`, the recommended second-stage optimal treatment is obtained as

```
newdata = data.frame(1, 30, 45)
colnames(newdata) <- c("gender","parent_BMI","month4_BMI")
optIQ2 = optTx(x = fitIQ2,
               newdata = newdata);
optIQ2
```

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

3.2 Estimating Regime Value

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 $V^{\boldsymbol{\pi}}$ is

$$\widehat{V}^{\boldsymbol{\pi}} \triangleq \frac{\sum_{i=1}^{n} Y_i \mathbb{1}\{A_{1i} = \pi_1(\boldsymbol{h}_{1i})\}\mathbb{1}\{A_{2i} = \pi_2(\boldsymbol{h}_{2i})\}}{\sum_{i=1}^{n} \mathbb{1}\{A_{1i} = \pi_1(\boldsymbol{h}_{1i})\}\mathbb{1}\{A_{2i} = \pi_2(\boldsymbol{h}_{2i})\}},$$

where Y_i is the i^{th} patient's response, (A_{1i}, A_{2i}) the randomized treatments and $(\boldsymbol{h}_{1i}, \boldsymbol{h}_{2i})$ the observed histories. This estimator is a weighted average of the outcomes observed from patients in the trial who received treatment in accordance with the regime $\boldsymbol{\pi}$. It is more commonly known as the Horvitz-Thompson estimator (**?**). The function `value()` estimates the value of a regime using the plug-in estimator and also returns value estimates corresponding to four non-dynamic regimes: `$valPosPos` ($\pi_1 = 1, \pi_2 = 1$); `$valPosNeg` ($\pi_1 = 1, \pi_2 = -1$); `$valNegPos` ($\pi_1 = -1, \pi_2 = 1$); and `$valNegNeg` ($\pi_1 = -1, \pi_2 = -1$). `value()` takes as

input `optTx1`, a vector of first-stage treatments assigned by the regime of interest; `optTx2`, a vector of second-stage treatments assigned by the regime of interest; `response`, the response vector; `tx1`, the vector of first-stage randomized treatments received by patients in the trial; and `tx2`, the vector of second-stage randomized treatments.

```
fitIQ1 = optTx(x = fitIQ1main,
               y = fitIQ1cm,
               z = fitIQ1var,
               dens = "nonpar")

fitIQ2 = optTx(x = fitIQ2)

estVal = value (optTx1 = fitIQ1,
                optTx2 = fitIQ2,
                response = y,
                tx1 = bmiData$A1,
                tx2 = bmiData$A2)
estVal
```

4 Conclusion

We have demonstrated how to estimate an optimal two-stage DTR using the IQ-learning functions and tools in the R package `DynTxRegime`. As indicated by its name, Interactive Q-learning allows the analyst to interact with the data at each step of the IQ-learning process to build models that fit the data well and are interpretable. At each model building step, the IQ-learning functions in `DynTxRegime` encourage the use of standard statistical methods for exploratory analysis, model selection, and model diagnostics.

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.