Package 'DynTxRegime'

May 22, 2017

Type Package

Title Methods for Estimating Optimal Dynamic Treatment Regimes

Version 3.01

Date 2017-04-21

Author S. T. Holloway, E. B. Laber, K. A. Linn, B. Zhang, M. Davidian, and A. A. Tsiatis

Maintainer Shannon T. Holloway <sthollow@ncsu.edu>

Description Methods to estimate dynamic treatment regimes using Interactive Q-Learning, Q-Learning, weighted learning, and value-search methods based on Augmented Inverse Probability Weighted Estimators and Inverse Probability Weighted Estimators.

License GPL-2

Depends methods, modelObj, stats

Suggests MASS, rpart, nnet

Imports kernlab, rgenoud, dfoptim

NeedsCompilation no

Repository CRAN

Date/Publication 2017-05-21 23:04:11 UTC

R topics documented:

`DynTxRegime-package` *Methods for Estimating Optimal Dynamic Treatment Regimes*

Description

Implementations of Interactive Q-Learning, Q-Learning, value-search methods based on augmented inverse probability weighted estimators and inverse probability weighted estimators, outcome weighted learning (OWL), residual weighted learning (RWL), backward outcome weighted learning (BOWL), and efficient augmentation and relaxation learning (EARL).

Details

Package:	DynTxRegime
Type:	Package
Version:	3.01
Date:	2017-05-21
License:	GPL-2

Depends:	methods, modelObj, stats
Suggests:	MASS, rpart, nnet
Imports:	kernlab, rgenoud, dfoptim

See the references below for details of each method implemented.

Author(s)

Marie Davidian, Eric B. Laber, Kristin A. Linn, Leonard A. Stefanski, Anastasios A. Tsiatis, Baqun Zhang, Min Zhang, and Shannon T. Holloway
Maintainer: Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E. B., Linn, K. A., and Stefanski, L. A. (2014). Interactive model building for Q-learning. Biometrika, 101, 831–847.

Zhang, B., Tsiatis, A. A., Davidian, M., Zhang, M., and Laber, E. B. (2012). Estimating Optimal Treatment Regimes from a Classification Perspective. Stat, 1, 103–114

Zhang, B., Tsiatis, A. A., Laber, E. B., and Davidian, M. (2012). A Robust Method for Estimating Optimal Treatment Regimes. Biometrics, 68, 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.

Mebane, W. and Sekhon, J. S. (2011). Genetic Optimization Using Derivatives : The rgenoud package for R. Journal of Statistical Software, 42, 1–26.

Zhao, Y-Q., Laber, E. B., Saha, S., and Sands, B. E. (2016+). Efficient Augmentation and Relaxation Learning for Treatment Regimes Using Observational Data. in press.

Zhou, X., Mayer-Hamblett, N., Kham, U., and Kosorok, M. R. (2016+). Residual Weighted Learning for Estimating Individualized Treatment Rules. Journal of the American Statistical Association, in press.

Zhao, Y-Q., Zeng, D., Rush, A. J., and Kosrok, M. R. (2012). Estimating Individualized Treatment Rules Using Outcome Weighted Learning. Journal of the American Statistical Association, 107, 1106–1118.

Zhao, Y-Q., Zeng, D., Laber, E. B., and Kosorok, M. R. (2015). New Statistical Learning Methods for Estimating Optimal Dynamic Treatment Regimes. Journal of the American Statistical Association, 110, 583–598.

See Also

`bowl`, `earl`, `iqLearnFSC`, `iqLearnFSM`, `iqLearnFSV`, `iqLearnSS`, `optimalClass`, `optimalSeq`, `owl`, `qLearn`, and `rwl`,

bmiData *Adolescent BMI dataset (generated toy example)*

Description

A dataset generated to mimic data from a two-stage randomized clinical trial that studied the effect of meal replacement shakes on adolescent obesity. The dataset contains the following covariates collected at the start of the first stage: "gender," "race," "parentBMI," and "baselineBMI." At the second-stage, "month4BMI" was collected. Variables "A1" and "A2" are the randomized treatments at stages one and two, and "month12BMI" is the primary outcome collected at the end of stage two.

Usage

```
bmiData
```

Format

A matrix with rows corresponding to patients.

Source

Generated by Kristin A. Linn in R

bowl *Backward Outcome Weighted Learning*

Description

A statistical learning method for estimating the optimal dynamic treatment regime (DTR) termed backward outcome weighted learning (BOWL). This approach converts individualized treatment selection into a sequential classification problem. The method directly maximizes over all DTRs a nonparametric estimator of the expected long-term outcome. This function implements a single step of a BOWL analysis. A complete BOWL analysis requires repeated calls to this function. Note that this method is limited to binary treatment regimes.

Usage

```
bowl(..., moPropen, data, reward, txName, regime, BOWLObj = NULL,
     lambdas = 2.0, cvFolds = 0L, kernel = "linear", kparam = NULL,
     fSet = NULL, verbose = TRUE)
```

Arguments

...	ignored. Included to require named input.
moPropen	An object of class "modelObj" or an object of class "list" containing objects of class "modelObjSubset." This input specifies the model(s) for the propensity score regression, Pr(A=a|X), and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). See moPropen for additional information.
data	An object of class "data.frame." The covariates and treatment history.
reward	An object of class "numeric." The reward for the decision point under analysis.
txName	An object of class "character." The column header of the stage treatment variable in input data. Treatment must be binary and will be recoded to -1/+1 if not provided as such.
regime	An object of class "formula" or an object of class "list" containing objects of class "formula." The formula defines the decision rule, i.e., the covariates to be included in the kernel. If regime is a single "formula," all patients are used to obtain parameter estimates. If regime is a list, subsets of patients are used to fit each decision rule. When using subset models, input fSet must be defined, and the name of each element of the list must correspond to a subset defined by fSet. See fSet for further details.
BOWLObj	An object of class "BOWL" or NULL. For the first step of the BOWL algorithm (the final decision point), this input must be NULL. For all other steps, it is the value object returned by the previous call to bowl().
lambdas	An object of class "numeric." This input is the tuning parameter to avoid over fitting. If more than one value is given and input cvFolds is greater than zero, cross-validation will be used to select an optimal value from among those provided.
cvFolds	An object of class "integer." If cross-validation is to be used to find an optimal lambda and/or kernel parameter, the number of folds to use in the cross-validation procedure.
kernel	An object of class "character." In conjunction with input kparam, this input specifies the kernel function to be used. Must be one of {'linear', 'poly', or 'radial'}. If 'linear,' the linear kernel; kparam is ignored. If 'poly,' the polynomial kernel; kparam must be specified. If 'radial,' the Gaussian radial basis function kernel; kparam must be specified.
kparam	An object of class "numeric." If input kernel = 'linear', this input is ignored. If input kernel = 'poly', this input is the order of the polynomial. If input kernel = 'radial', this input is sigma; i.e., $$K(x, y) = exp(\|\|x - y\|\|^2/(2 * sigma^2)).$$ For kernel = 'radial', a vector of kernel parameters can be provided, and cross-validation will be used to determine the optimal of those provided. Note that input cvFolds must be > 0.

fSet	An object of class "function" or NULL. If a function, fSet defines (i) the conditions under which only a subset of treatment options is available to a patient or (ii) the conditions under which patients are to be subset and the subsets modeled uniquely. The form of this input has been extended from the original release of **DynTxRegime**. See fSet for additional information.
verbose	An object of class "logical." If FALSE, screen prints will be suppressed.

Value

Returns a "BOWL" object, which inherits directly from class "DynTxRegime."

Methods

coef signature(object = "BOWL"): Retrieve parameter estimates for all regression steps.

cvInfo signature(object = "BOWL"): Retrieve cross-validation results.

DTRstep signature(object = "BOWL"): Retrieve description of method used to create object.

estimator signature(x = "BOWL"): Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject signature(object = "BOWL"): Retrieve value object returned by regression methods.

optimObj signature(object = "BOWL"): Retrieve value object returned by optimization routine.

optTx signature(x = "BOWL", newdata = "missing"): Retrieve the estimated optimal treatment regime for training data set.

optTx signature(x = "BOWL", newdata = "data.frame"): Estimate the optimal treatment regime for newdata.

plot signature(x = "BOWL"): Generate plots for regression analyses.

print signature(object = "BOWL"): Print main results of analysis.

propen signature(x = "BOWL"): Retrieve value object returned by propensity score regression methods.

regimeCoef signature(object = "BOWL"): Retrieve the estimated decision function parameter estimates.

show signature(object = "BOWL"): Show main results of analysis.

summary signature(object = "BOWL"): Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhao, Y-Q., Zeng, D., Laber, E. B., and Kosorok, M. R. (2015). New Statistical Learning Methods for Estimating Optimal Dynamic Treatment Regimes. Journal of the American Statistical Association, 110, 583–598.

Examples

```
# Load and process data set
data(bmiData)

# define the negative 12 month change in BMI from baseline
y12 <- -100*(bmiData[,6L] - bmiData[,4L])/bmiData[,4L]

# define the negative 4 month change in BMI from baseline
y4 <- -100*(bmiData[,5L] - bmiData[,4L])/bmiData[,4L]

# reward for second stage
rewardSS <- y12 - y4

#### Second-stage regression

# Constant propensity model
moPropen <- buildModelObj(model = ~1,
                          solver.method = 'glm',
                          solver.args = list('family'='binomial'),
                          predict.method = 'predict.glm',
                          predict.args = list(type='response'))

fitSS <- bowl(moPropen = moPropen,
              data = bmiData, reward = rewardSS,  txName = 'A2',
              regime = ~ parentBMI + month4BMI)

##Available methods

  # Coefficients of the propensity score regression
  coef(fitSS)

  # Description of method used to obtain object
  DTRstep(fitSS)

  # Estimated value of the optimal treatment regime for training set
  estimator(fitSS)

  # Value object returned by propensity score regression method
  fitObject(fitSS)

  # Summary of optimization routine
  optimObj(fitSS)

  # Estimated optimal treatment for training data
  optTx(fitSS)

  # Estimated optimal treatment for new data
  optTx(fitSS, bmiData)

  # Plots if defined by propensity regression method
  dev.new()
  par(mfrow = c(2,4))
```

```
  plot(fitSS)
  plot(fitSS, suppress = TRUE)

  # Value object returned by propensity score regression method
  propen(fitSS)

  # Parameter estimates for decision function
  regimeCoef(fitSS)

  # Show main results of method
  show(fitSS)

  # Show summary results of method
  summary(fitSS)

#### First-stage regression

# Constant propensity model
fitFS <- bowl(moPropen = moPropen,
              data = bmiData, reward = y4,  txName = 'A1',
              regime = ~ gender + parentBMI,
              BOWLObj = fitSS)
```

`buildModelObjSubset` *Create Model Objects for Subsets of Data.*

Description

Extends the buildModelObj() function of package **modelObj**. Here, the returned model object includes a specification of the decision point and subset of the data to which the model is to be applied.

Usage

```
buildModelObjSubset(..., model,
                    solver.method, solver.args = NULL,
                    predict.method = NULL, predict.args = NULL,
                    dp = 1L, subset = NA)
```

Arguments

`...`	ignored. Included to require named input.
`model`	An object of class `"formula."` The symbolic description of the model to be fitted. If the regression method specified in `solver.method` accepts as input a `"formula"` object, `model` is passed to the solver.method. If the regression method instead accepts a matrix of covariates as the model to fit, `model` is used to obtain the model matrix that is passed to solver.method.

`solver.method` An object of class `"character."` The name of the R function to be used to obtain parameter estimates, e.g., `lm`, `glm`, or `rpart`. The specified function MUST have a corresponding predict method, which can be the generic predict() function.

`solver.args` An object of class `"list."` Additional arguments to be sent to the function specified in `solver.method`. This argument must be provided as a named list, where the name of each element matches a formal argument of the function specified in `solver.method`. For example, if a logistic regression using `glm` is desired,

```
solver.method = "glm"
solver.args = list("family"=binomial)
```

See Details section for further information.

`predict.method`

An object of class `"character."` The name of the R function to be used to obtain predictions, e.g., `predict.lm`, `predict`, or `predict.glm`. If no function is explicitly given, the generic `predict()` is assumed. For many regression methods, the generic `predict()` method is appropriate.

`predict.args` An object of class `"list."` Additional arguments to be sent to the function specified in `predict.method`. This argument must be provided as a named list, where the name of each element matches a formal argument of the function specified in `predict.method`. For example, if a logistic regression using `glm` was used to fit the model and predictions on the scale of the response are desired,

```
predict.method = "predict.glm"
predict.args = list("type"="response").
```

See Details section for further information.

`dp` An object of class `"integer."` The decision point for which this model and subset are defined.

`subset` An object of class `"character."` A nickname for the subset for which model and methods are to be used. This argument will be used by the methods of **DynTxRegime** to "link" input arguments. In the event that a model is to be fit using more than 1 subset, collapse the subset names into a single character string separating each with a comma. For example, if the model is to be fit using patients in both subsets "a" and "b," the subset nickname should be "a,b" (no space).

Details

In some settings, an analyst may want to use different models for unique subsets of the data. `buildModelObjSubset()` provides a mechanism for users to define models for such subset. Specifically, models are specified in connection with the decision point and subset to which they are to be applied.

It is assumed that if the method specified in `solver.method` uses a formula then `formula` and `data` are the formal arguments for the `"formula"` object and `"data.frame,"` respectively. However, if solver.method does not use formal argument `formula` and/or `data`, the appropriate names can be provided through `solver.args`. For example, the input list for `solver.args` would include the

element "x"="formula" if the "formula" object is passed through input argument x or it would include element "df"="data" if the "data.frame" object is passed through input argument df.

If the R method specified in solver.method instead uses a model matrix to define the model, it is assumed to have formal arguments x and y. If the solver.method does not use x for the model matrix and/or y as the response, solver.args must explicitly indicate the variable names used for these inputs. For example, the list passed through solver.args would include element "X"="x" if the model matrix is passed through input argument X or element "Y"="y" if the response is passed through input argument Y.

It is also assumed that the function specified in predict.method has formal arguments object and newdata through which the value returned by the solver.method and the "data.frame" objects are provided. If the predict.method does not use these formal arguments, predict.args must explicitly indicate the argument names used for these inputs. For example, the names list passed through predict.args would include element "x"="object" if the object returned by the function specified in solver.method is passed through input argument x or it would include element "ndf"="newdata" if the "data.frame" object is passed through input argument ndf.

Value

An object of class "ModelObjSubset," which contains a complete description of the conditions under which a model is to be used and the R methods to be used to obtain parameter estimates and predictions.

Examples

```
# Consider a 2 decision point trial. At the 1st decision point, the subset of
# treatment options available to each patient is always set "set1."
# At the 2nd decision point, some patients are eligible to receive
# treatment from set "set2a" and others from set "set2b." The outcome
# for these subsets will be modeled as ~ x1 + x2 and ~ x2 + x3, respectively.
#
# All parameter estimates are to be obtained used lm and predictions obtained using predict.
#
# The following illustrates how to build these model objects.

  model <- list()

  model[[1]] <- buildModelObjSubset(dp = 1, subset = "set1",
                                    model = ~ x1 + x2 + x3, solver.method = 'lm')

  model[[2]] <- buildModelObjSubset(dp = 2, subset = "set2a",
                                    model = ~ ~ x1 + x2, solver.method = 'lm')

  model[[3]] <- buildModelObjSubset(dp = 2, subset = "set2b",
                                    model = ~ x2 + x3, solver.method = 'lm')
```

classif *Retrieve Classification Value Object*

Description

Retrieve the value object returned by the classification method used to estimate an optimal treatment regime when viewed as a weighted classification problem, i.e., method optimalClass().

Usage

```
classif(object, ...)
```

Arguments

object	An object of class "DynTxRegime".
...	ignored.

Details

The exact value and structure of the returned object is determined by the method specified for input moClass of optimalClass(). Because the value object of the modeling function is returned, any methods developed for said object can be applied to the object returned. For example, if the classification method is rpart, the object returned by classif() can be passed as input to residuals.rpart(), path.rpart(), etc. without modification.

Value

For objects returned by optimalClass(), see Details section. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

coef *Retrieve Model Coefficients*

Description

Retrieve model coefficients from regression analyses.

Usage

```
coef(object,...)
```

Arguments

object	An object of class "DynTxRegime".
...	Passed to coef() defined by fitting function.

Details

The exact structure of the returned list will depend on the statistical method used to estimate an optimal treatment regime and value.

For methods that employ outcome regression, the list will have one element named 'outcome' in which all coefficient information for the outcome regression step(s) will be given.

For methods that use propensity score modeling, the list will have one element named 'propensity' in which all coefficient information for the propensity score regression step(s) will be given.

The 'outcome' and 'propensity' elements may also be lists. For example, for multiple-decision-points, the sub-list will have an element for each decision point named with 'dp=x' where x takes the value of the decision point. For example, [['outcome']][['dp=1']], [['outcome']][['dp=2']], etc.

Subset modeling will also result in sub-lists, the names of which are 'Subset=x' where x takes the values of the subset names as defined by the user in the input.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

---

cvInfo *Retrieve Cross-Validation Results*

Description

Retrieve cross-validaton results.

Usage

```
cvInfo(object)
```

Arguments

object	An object of class "DynTxRegime" originating from a weighted learning method or EARL.

Value

For objects returned by owl(), bowl(), rwl(), and earl(), a matrix of values at each parameter. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

DTRstep *Identify Statistical Method of DynTxRegime Object*

Description

Retrieves a description of the method used to create the object.

Usage

```
DTRstep(object)
```

Arguments

object	An object of class "DynTxRegime."

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

DynTxRegime-class *Class* "DynTxRegime"

Description

A VIRTUAL class from which all methods implemented in **DynTxRegime** inherit.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

call: An object of class "CallOrNull." The matched call.

estimatedValue: An object of class "LogicalOrNumeric." The estimated value of the estimated optimal treatments for the training data.

optimalTx: An object of class "MatrixOrVector." The estimated optimal treatments for the training data.

Methods

classif `signature(object = "DynTxRegime")`: Retrieve value object from classification step. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by optimalClass().

coef `signature(object = "DynTxRegime")`: Retrieve parameter estimates for all regression steps.

cvInfo `signature(object = "DynTxRegime")`: Retrieve cross-validation results. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by earl(), owl(), rwl(), and bowl().

DTRstep `signature(object = "DynTxRegime")`: Retrieve description of method used to create object.

estimator `signature(x = "DynTxRegime")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "DynTxRegime")`: Retrieve value object returned by regression methods.

fittedCont `signature(object = "DynTxRegime")`: Retrieve estimated contrast component of outcome regression. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by iqLearnSS().

fittedMain `signature(object = "DynTxRegime")`: Retrieve estimated main effects component of outcome regression. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by iqLearnSS().

genetic `signature(object = "DynTxRegime")`: Retrieve value object returned by genoud(). NA for all objects returned by **DynTxRegime** statistical methods except for those returned by optimalSeq().

optimObj `signature(object = "DynTxRegime")`: Retrieve value object returned by optimization routine. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by earl(), owl(), rwl(), and bowl().

optTx `signature(x = "DynTxRegime", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "DynTxRegime", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "DynTxRegime")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "DynTxRegime")`: Generate plots for regression analyses.

print `signature(object = "DynTxRegime")`: Print main results of analysis.

propen `signature(x = "DynTxRegime")`: Retrieve value object returned by propensity score regression methods. NA for objects returned by qLearn(), iqLearnSS, iqLearnFSM(), iqLearnFSC(), and iqLearnFSV().

qqPlot `signature(x = "DynTxRegime")`: Generate a qq-plot. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by iqLearnFSV().

regimeCoef `signature(object = "DynTxRegime")`: Retrieve the estimated decision function parameter estimates. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by optimalSeq(), earl(), owl(), rwl(), and bowl().

residuals `signature(object = "DynTxRegime")`: Retrieve residuals used by statistical methods. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by iqLearnFSC(), iqLearnFSV(), and rwl().

sd `signature(x = "DynTxRegime")`: Retrieve the standard deviation of residuals. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by iqLearnFSC().

show `signature(object = "DynTxRegime")`: Show main results of analysis.

summary `signature(object = "DynTxRegime")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

Examples

```
showClass("DynTxRegime")
```

`earl` *Efficient Augmentation and Relaxation Learning*

Description

Estimation of optimal treatment regime using efficient augmentation and relaxation learning (EARL). The method is limited to single-decision-point scenarios with binary treatment options.

Usage

```
earl(..., moPropen, moMain, moCont, data, response, txName, regime,
     iter = 0L, lambdas = 0.5, cvFolds = 0L, surrogate = "hinge",
     guess = NULL, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moPropen`	An object of class `"modelObj."` This object specifies the model of the propensity score regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). See moPropen for further information.
`moMain`	An object of class `"modelObj"` or NULL. If a `"modelObj,"` this input specifies the model of the main effects component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.

moCont	An object of class "modelObj" or NULL. If a "modelObj," this input specifies the model of the contrasts component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
data	An object of class "data.frame." The covariates and treatment history.
response	An object of class "vector." The outcome of interest.
txName	An object of class "character." The column header of the stage treatment variable as given in input data. Not that treatment must be binary and will be recoded as -1/+1 if not provided as such.
regime	An object of class "formula." A formula defining the decision rule.
iter	An object of class "integer." If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
lambdas	An object of class "numeric." The tuning parameter to avoid overfitting. If a vector of values is given and cvFolds > 0, cross-validation will be used to select an optimal value from among those provided.
cvFolds	An object of class "integer." If cross-validation is to be used to find an optimal lambda parameter, the number of folds to use in the cross-validation procedure.
surrogate	An object of class "character." Specification of the surrogate for the 0-1 loss function. Must be one of {'hinge', 'logit', 'exp', 'sqhinge'} indicating the hinge, logistic, exponential, and squared-hinge functions, respectively
guess	An object of class "numeric" or NULL. Starting parameter values for optimization method.
verbose	An object of class "logical." If FALSE, screen prints will be suppressed.

Details

The inverse propensity weighted estimator is calculated if both moMain and moCont are NULL; otherwise the augmented inverse propensity weighted estimator is calculated.

Value

Returns an object of class "EARL" that inherits directly from class "DynTxRegime."

Methods

coef signature(object = "EARL"): Retrieve parameter estimates for all regression steps.

cvInfo signature(object = "EARL"): Retrieve cross-validation results.

DTRstep signature(object = "EARL"): Retrieve description of method used to create object.

estimator signature(x = "EARL"): Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject signature(object = "EARL"): Retrieve value object returned by regression methods.

optimObj signature(object = "EARL"): Retrieve value object returned by optimization routine.

optTx `signature(x = "EARL", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "EARL", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "EARL")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "EARL")`: Generate plots for regression analyses.

print `signature(object = "EARL")`: Print main results of analysis.

propen `signature(x = "EARL")`: Retrieve value object returned by propensity score regression methods.

regimeCoef `signature(object = "EARL")`: Retrieve the estimated decision function parameter estimates.

show `signature(object = "EARL")`: Show main results of analysis.

summary `signature(object = "EARL")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhao, Y-Q., Laber, E. B., Saha, S., and Sands, B. E. (2017+). Efficient Augmentation and Relaxation Learning for Treatment Regimes Using Observational Data. in press.

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

  bmiData$y <- bmiData$y - min(bmiData$y) + 0.001

# Constant propensity model
  moPropen <- buildModelObj(model = ~1,
                            solver.method = 'glm',
                            solver.args = list('family'='binomial'),
                            predict.method = 'predict.glm',
                            predict.args = list(type='response'))

# IPWE
  earlRes <- earl(moPropen = moPropen, moMain = NULL, moCont = NULL,
                  data = bmiData, response = bmiData$y, txName = "A2",
                  regime = ~ parentBMI + month4BMI)

#Available methods
```

```
# Coefficients of the propensity score regression
coef(earlRes)

# Description of method used to obtain object
DTRstep(earlRes)

# Estimated value of the optimal treatment regime for training set
estimator(earlRes)

# Value object returned by propensity score regression method
fitObject(earlRes)

# Summary of optimization routine
optimObj(earlRes)

# Estimated optimal treatment for training data
optTx(earlRes)

# Estimated optimal treatment for new data
optTx(earlRes, bmiData)

# Plots if defined by propensity regression method
dev.new()
par(mfrow = c(2,4))

plot(earlRes, suppress = FALSE)
plot(earlRes, suppress = TRUE)

# Value object returned by propensity score regression method
propen(earlRes)

# Parameter estimates for decision function
regimeCoef(earlRes)

# Show main results of method
show(earlRes)

# Show summary results of method
summary(earlRes)



# Augmented IPWE
  # Create modeling object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method='lm')

  # Create modeling object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method='lm')

  earlResA <- earl(moPropen = moPropen, moMain = moMain, moCont = moCont,
```

```
                    data = bmiData, response = bmiData$y, txName = "A2",
                    regime = ~ parentBMI + month4BMI)

#Available methods

# Coefficients of the propensity score and outcome regressions
coef(earlResA)

# Description of method used to obtain object
DTRstep(earlResA)

# Estimated value of the optimal treatment regime for training set
estimator(earlResA)

# Value object returned by propensity score and outcome regression methods
fitObject(earlResA)

# Summary of optimization routine
optimObj(earlResA)

# Estimated optimal treatment for training data
optTx(earlResA)

# Estimated optimal treatment for new data
optTx(earlResA, bmiData)

# Value object returned by outcome regression method
outcome(earlResA)

# Plots if defined by propensity score and outcome regressionmethod
dev.new()
par(mfrow = c(2,4))
plot(earlResA, suppress = FALSE)

dev.new()
par(mfrow = c(2,4))
plot(earlResA, suppress = TRUE)

# Value object returned by propensity score regression method
propen(earlResA)

# Parameter estimates for decision function
regimeCoef(earlResA)

# Show main results of method
show(earlResA)

# Show summary results of method
summary(earlResA)
```

`estimator` *Estimated Value of Estimated Optimal Regime*

Description

Retrieve the estimated value of the estimated optimal treatment regime for the training data.

Usage

```
estimator(x,...)
## S4 method for signature 'DynTxRegime'
estimator(x)
## S4 method for signature 'IQLearnFS'
estimator(x, w = NULL, y = NULL, z = NULL, dens = NULL)
```

Arguments

`x`	Object of class `"DynTxRegime"`.
`...`	Ignored.
`w`	Object of class `"IQLearnFS"` or `"IQLearnSS"`.
`y`	Object of class `"IQLearnFS"` or `"IQLearnSS"`.
`z`	Object of class `"IQLearnFS"` or `"IQLearnSS"`.
`dens`	`"character"` indicating density approximation. Must be one of {'nonpar', 'norm'}.

Details

For the IQ-learning algorithm, additional inputs are required to obtain the estimated value of the first-stage treatment. Function estimator() required inputs `w`, `x`, `y`, and `z` are the three first-stage and single second-stage objects (in no specific order).

For all other methods, no information beyond the value object returned by the method is needed.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

fitObject *Modeling Function Value Objects*

Description

Retrieve the value objects returned by the regression method(s). Extends methods defined in **modelObj**.

Usage

```
fitObject(object, ...)
```

Arguments

object	An object of class "DynTxRegime".
...	Ignored.

Details

The exact structure of the returned list will depend on the statistical method.

For methods that employ outcome regression, the list will have one element named 'outcome' in which all fit information for the outcome regression step(s) will be given.

For methods that use propensity score modeling, the list will have one element named 'propensity' in which all fit information for the propensity regression step(s) will be given.

The 'outcome' and 'propensity' elements may also be lists. For example, if multiple decision points, the sub-list will have an element for each decision point named with 'dp=x' where x takes the value of the decision point. For example, [['outcome']][['dp=1']], [['outcome']][['dp=2']], etc.

Subset modeling will also result in sub-lists, the names of which are 'Subset=x' where x takes the values of the subset names as defined by the user in the input.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

fittedCont *Extract Fitted Contrast Component of Outcome*

Description

Retrieve the fitted contrast component of the outcome regression analysis for the second-stage decision point. Only available for the IQ-Learning algorithm.

Usage

```
fittedCont(object, ...)
```

Arguments

object	An object of class "DynTxRegime"
...	Ignored.

Value

For objects returned by iqLearnSS(), a vector. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

---

fittedMain *Extract Fitted Main Effects of Outcome*

Description

Retrieve the fitted main effects component of the outcome regression analysis for the second-stage decision point. Only available for the IQ-Learning algorithm.

Usage

```
fittedMain(object, ...)
```

Arguments

object	An object of class "DynTxRegime"
...	Ignored.

Value

For objects returned by iqLearnSS(), a vector. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

`fSet` *Defining the fSet input variable*

Description

Several of the statistical methods implemented in package **DynTxRegime** allow for subset modeling or limiting of feasible treatment options. This section details how this input is to be defined.

Details

In general, input `fSet` is used to define subsets of patients within an analysis. These subsets can be specified to (1) limit available treatments, (2) use different models for the propensity score and/or outcome regressions, and/or (3) use different decision function models for each subset of patients. The combination of inputs `moPropen`, `moMain`, `moCont`, `fSet`, and/or `regimes` determines which of these scenarios is being considered. We cover some common situations below.

Regardless of the purpose for specifying `fSet`, it must be a function that returns a list. There are two options for defining the function. Version 1 is that of the original **DynTxRegime** package. In this version, `fSet` defines the rules for determining the subset of treatment options for an INDIVIDUAL. The first element of the returned list is a character, which we term the subset 'nickname.' This nickname is for bookkeeping purposes and is used to link models to subsets. The second element of the returned list is a vector of available treatment options for the subset. The formal arguments of the function must include (i) 'data' or (ii) individual covariate names as given by the column headers of `data`. An example using the covariate name input form is

```
fSet <- function(a1) {
  if(a1 > 1){
    subset <- list('subA',c(1,2))
  } else {
    subset <- list('subB',c(3,4) )
  }
  return(subset)
}
```

This function indicates that if an individual has covariate a1 > 1, they are a member of subset 'subA' and their feasible treatment options are {1,2}. If a1 <= 1, they are a member of subset 'subB' and their feasible treatment options are {3,4}.

A more efficient implementation for `fSet` is now accepted. In the second form, `fSet` defines the subset of treatment options for the full DATASET. It is again a function with formal arguments (i) 'data' or (ii) individual covariate names as given by the column headers of `data`. The function returns a list containing two elements: 'subsets' and 'txOpts.' Element 'subsets' is a list comprising all treatment subsets; each element of the list contains the nickname and treatment options for a single subset. Element 'txOpts' is a character vector indicating the subset of which each individual is a member. In this new format, the equivalent definition of `fSet` as that given above is:

```
fSet <- function(a1) {
  subsets <- list(list('subA', c(1,2)),
```

```
                  list('subB', c(3,4)))
  txOpts <- rep('subB', length(a1))
  txOpts[a1 > 1] <- 'subA'

  return(list("subsets" = subsets,
              "txOpts" = txOpts))
}
```

Though a bit more complicated, this version is much more efficient as it processes the entire dataset at once rather than each individual separately.

The simplest scenario involving `fSet` is to define feasible treatment options and the rules that dictate how those treatment options are determined. For example, responder/non-responder scenarios are often encountered in multiple-decision-point settings. An example of this scenario is: patients that respond to the first stage treatment remain on the original treatment; those that do not respond to the first stage treatment have all treatment options available to them at the second stage. In this case, the propensity score models for the second stage are fit using only 'non-responders' for whom more than 1 treatment option is available.

An example of an appropriate `fSet` function for the second-stage is

```
fSet <- function(data) {
   if(data$responder  == 0L){
     subset <- list('subA',c(1L,2L))
   } else if(data$tx1 == 1L) {
     subset <- list('subB',c(1L) )
   } else if(data$tx1 == 2L) {
     subset <- list('subC',c(2L) )
   }
   return(subset)
}
```

for version 1 or for version 2

```
fSet <- function(data) {
  subsets <- list(list('subA', c(1L,2L)),
                  list('subB', c(1L)),
                  list('subC', c(2L)))
  txOpts <- character(nrow(data))
  txOpts[data$tx1 == 1L] <- 'subB'
  txOpts[data$tx1 == 2L] <- 'subC'
  txOpts[data$responder == 0L] <- 'subA'

  return(list("subsets" = subsets,
              "txOpts" = txOpts))
}
```

The functions above specify that patients with covariate responder = 0 receive treatments from subset 'subA,' which comprises treatments A = (1,2). Patients with covariate responder = 1 receive treatment from subset 'subB' or 'subC' depending on the first stage treatment received. If `fSet` is

specified in this way, moPropen must be a "modelObj"; the propensity model will be fit using only those patients with responder = 0. If outcome regression is used by the method, moMain and moCont can be either objects of class "modelObj" if all all patients are to be used to obtain parameter estimates or lists of objects of class "ModelObjSubset" if subsets are to be analyzed individually.

For a scenario where all patients have the same set of treatment options available, but subsets of patients are to be analyzed using different models. We cane define fSet as

```
fSet <- function(data) {
   if(data$a1 == 1){
     subset <- list('subA',c(1L,2L))
   } else {
     subset <- list('subB',c(1L,2L) )
   }
   return(subset)
}
```

for version 1 or in the format of version 2

```
fSet <- function(data)
{
  subsets <- list(list('subA', c(1L,2L)),
                  list('subB', c(1L,2L)))
  txOpts <- rep('subB', nrow(data))
  txOpts[data$a1 == 1L] <- 'subA'

  return(list("subsets" = subsets,
              "txOpts" = txOpts))
}
```

where all patients have the same treatment options available, A = (1,2), but different regression models will be fit for each subset (case 2 above) and/or different decision function models (case 3 above) for each subset. If different propensity score models are used, moPropen must be a list of objects of class "modelObjSubset." Perhaps,

```
  propenA <- buildModelObjSubset(model = ~1,
                                 solver.method = 'glm',
                                 solver.args = list('family'='binomial'),
                                 predict.method = 'predict.glm',
                                 predict.args = list(type='response'),
                                 subset = 'subA')

  propenB <- buildModelObjSubset(model = ~1,
                                 solver.method = 'glm',
                                 solver.args = list('family'='binomial'),
                                 predict.method = 'predict.glm',
                                 predict.args = list(type='response'),
                                 subset = 'subB')

  moPropen <- list(propenA, propenB)
```

If different decision function models are to be fit, `regimes` would take a form similar to

```
regimes <- list( 'subA' = ~x1 + x2,
                 'subB' = ~x2 )
```

Notice that the names of the elements of `regimes` and the subsets passed to buildModelObjSubset() correspond to the names defined by `fSet`, i.e., 'subA' or 'subB.' These nicknames are used for bookkeeping and link subsets to the appropriate models.

For a single-decision-point analysis, `fSet` is a single function. For multiple-decision-point analyses, `fSet` is a list of functions where each element of the list corresponds to the decision point (1st element <- 1st decision point, etc.)

`genetic` *Retrieve the Result of the Genetic Algorithm Optimization*

Description

Retrieve the value object returned by the rgenoud algorithm. Only available for objects returned by optimalSeq(). See ?genoud for further information.

Usage

```
genetic(object, ...)
```

Arguments

`object`	An object of class `"DynTxRegime"`.
`...`	Ignored.

Value

For objects returned by optimalSeq(), a list, the structure of which is defined by rgenoud(). For all other `"DynTxRegime"` objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

iqLearnFSC *IQ-Learning: Regression of Estimated Second-Stage Contrasts*

Description

Estimate an optimal dynamic treatment regime using the Interactive Q-learning (IQ-learning) algorithm when the data has been collected from a two-stage randomized trial with binary treatments. iqLearnFSC implements the regression of the estimated second-stage contrasts in the IQ-Learning algorithm (IQ3).

Usage

```
iqLearnFSC(..., moMain, moCont, data, response, txName, iter = 0L, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moMain`	An object of class `"modelObj."` This object specifies the main effects component of the model for the regression of the estimated second-stage contrasts and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	An object of class `"modelObj."` This object specifies the contrasts component of the model for the regression of the estimated second-stage contrasts and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`response`	An object of class `"IQLearnSS."` The object returned by a prior call to `iqLearnSS()`.
`txName`	An object of class `"character."` The column header of the stage treatment variable in input `data`. The treatment variable must be binary and will be recoded as -1/+1 if not provided as such.
`iter`	An object of class `"integer."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Value

Returns an object of class `"IQLearnFS_C"` that inherits directly from class `"DynTxRegime."`

Methods

coef `signature(object = "IQLearnFS_C")`: Retrieve parameter estimates for all regression steps.

DTRstep `signature(object = "IQLearnFS_C")`: Retrieve description of method used to create object.

estimator `signature(x = "IQLearnFS_C")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "IQLearnFS_C")`: Retrieve value object returned by regression methods.

optTx `signature(x = "IQLearnFS_C", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "IQLearnFS_C", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "IQLearnFS_C")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "IQLearnFS_C")`: Generate plots for regression analyses.

print `signature(object = "IQLearnFS_C")`: Print main results of analysis.

residuals `signature(object = "IQLearnFS_C")`: Retrieve residuals used by statistical methods.

sd `signature(x = "IQLearnFS_C")`: Retrieve the standard deviation of residuals.

show `signature(object = "IQLearnFS_C")`: Show main results of analysis.

summary `signature(object = "IQLearnFS_C")`: Retrieve summary information from regression analyses.

Note

The implementation of IQ-Learning utilizes methods `optTx()` and `estimator()` in a different way than other methods implemented in **DynTxRegime**. For the first-stage component of the IQ-Learning method, all first-stage steps must be complete before an optimal treatment or value can be estimated. Therefore, `optTx()` requires the objects returned by first stage methods `iqLearnFSM()`, `iqLearnFSC()`, and `iqLearnVar()` if using a log-linear model of the variance or `iqLearnFSM()` and `iqLearnFS()` if using a constant model of the variance. Function `estimator()` requires the first-stage objects and the second stage object returned by `iqLearnSS()`. In addition, the density must be specified. See `optTx` and `estimator` for further details.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E. B., Linn, K. A., and Stefanski, L.A. (2014). Interactive model building for Q-learning. Biometrika, 101, 831–847.

See Also

`iqLearnSS`, `iqLearnFSM`, `iqLearnFSV`

Examples

```
#### Full IQ-Learning Algorithm

## Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

## Second-stage regression (IQ1)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

  iqSS <- iqLearnSS(moMain = moMain, moCont = moCont,
                    data = bmiData, response = bmiData$y, txName = "A2",
                    iter = 0L)

## Model conditional mean of main effects function (IQ2)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSM <- iqLearnFSM(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
                      iter = 0L)


## Model conditional mean of contrast function (IQ3)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSC <- iqLearnFSC(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
                      iter = 0L)

## Estimated optimal treatment and value when variance assumed constant
  # optimal treatment
```

```
  ot <- optTx(iqFSM, y = iqFSC, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, dens = 'nonpar')

## Log-Linear Variance Modeling (IQ4)

  # heteroskedastic variance
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + baselineBMI,
                          solver.method = 'lm')


  iqFSV <- iqLearnFSV(object = iqFSC,
                      moMain = moMain, moCont = moCont,
                      data = bmiData, txName = "A1",
                      iter = 0L)

## Estimated optimal treatment and value with log-linear variance modeling
  # optimal treatment
  ot <- optTx(iqFSM, y = iqFSC, z = iqFSV, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, w = iqFSV, dens = 'nonpar')

## Available methods for contrast step

  # Coefficients of the outcome regression
  coef(iqFSC)

  # Description of method used to obtain object
  DTRstep(iqFSC)

  # Value object returned by outcome regression method
  fitObject(iqFSC)

  # Value object returned by outcome regression method
  outcome(iqFSC)

  # Plots if defined by outcome regression method
  dev.new()
  par(mfrow = c(2,4))
  plot(iqFSC)
  plot(iqFSC, suppress = TRUE)

  # Residuals of regression
  residuals(iqFSC)

  # Standard deviation of residuals
```

```
sd(iqFSC)

# Show main results of method
show(iqFSC)

# Show summary results of method
summary(iqFSC)
```

---

iqLearnFSM *IQ-Learning: Regression of Estimated Second-Stage Main Effects*

Description

Estimate an optimal dynamic treatment regime using the Interactive Q-learning (IQ-learning) algorithm when the data has been collected from a two-stage randomized trial with binary treatments. iqLearnFSM implements the regression of the estimated second-stage main effects (IQ2).

Usage

```
iqLearnFSM(..., moMain, moCont, data, response, txName, iter = 0L, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moMain`	An object of class `"modelObj."` This object specifies the main effects component of the model for the regression of the estimated second-stage main effects and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	An object of class `"modelObj."` This object specifies the contrasts component of the model for the regression of the estimated second-stage main effects and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`response`	An object of class `"IQLearnSS."` The object returned by a prior call to `iqLearnSS()`.
`txName`	An object of class `"character."` The column header of the stage treatment variable in input `data`. The treatment variable must be binary and will be recoded as -1/+1 if not provided as such.
`iter`	An object of class `"integer."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Value

Returns an object of class "IQLearnFS_ME" that inherits directly from class "DynTxRegime."

Methods

coef signature(object = "IQLearnFS_ME"): Retrieve parameter estimates for all regression steps.

DTRstep signature(object = "IQLearnFS_ME"): Retrieve description of method used to create object.

estimator signature(x = "IQLearnFS_ME"): Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject signature(object = "IQLearnFS_ME"): Retrieve value object returned by regression methods.

optTx signature(x = "IQLearnFS_ME", newdata = "missing"): Retrieve the estimated optimal treatment regime for training data set.

optTx signature(x = "IQLearnFS_ME", newdata = "data.frame"): Estimate the optimal treatment regime for newdata.

outcome signature(x = "IQLearnFS_ME"): Retrieve value object returned by outcome regression methods.

plot signature(x = "IQLearnFS_ME"): Generate plots for regression analyses.

print signature(object = "IQLearnFS_ME"): Print main results of analysis.

show signature(object = "IQLearnFS_ME"): Show main results of analysis.

summary signature(object = "IQLearnFS_ME"): Retrieve summary information from regression analyses.

Note

The implementation of IQ-Learning utilizes methods optTx() and estimator() in a different way than other methods implemented in **DynTxRegime**. For the first-stage component of the IQ-Learning method, all first-stage steps must be complete before an optimal treatment or value can be estimated. Therefore, optTx() requires the objects returned by first stage methods iqLearnFSM(), iqLearnFSC(), and iqLearnVar() if using a log-linear model of the variance or iqLearnFSM() and iqLearnFS() if using a constant model of the variance. Function estimator() requires the first-stage objects and the second stage object returned by iqLearnSS(). In addition, the density must be specified. See optTx and estimator for further details.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E. B., Linn, K. A., and Stefanski, L.A. (2014). Interactive model building for Q-learning. Biometrika, 101, 831–847.

See Also

iqLearnSS, iqLearnFSC, iqLearnFSV

Examples

```
#### Full IQ-Learning Algorithm

## Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

## Second-stage regression (IQ1)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

  iqSS <- iqLearnSS(moMain = moMain, moCont = moCont,
                    data = bmiData, response = bmiData$y, txName = "A2",
                    iter = 0L)

## Model conditional mean of main effects function (IQ2)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSM <- iqLearnFSM(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
                      iter = 0L)


## Model conditional mean of contrast function (IQ3)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSC <- iqLearnFSC(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
```

```
                      iter = 0L)

## Estimated optimal treatment and value when variance assumed constant
  # optimal treatment
  ot <- optTx(iqFSM, y = iqFSC, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, dens = 'nonpar')

## Log-Linear Variance Modeling (IQ4)

  # heteroskedastic variance
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + baselineBMI,
                          solver.method = 'lm')


  iqFSV <- iqLearnFSV(object = iqFSC,
                      moMain = moMain, moCont = moCont,
                      data = bmiData, txName = "A1",
                      iter = 0L)

## Estimated optimal treatment and value with log-linear variance modeling
  # optimal treatment
  ot <- optTx(iqFSM, y = iqFSC, z = iqFSV, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, w = iqFSV, dens = 'nonpar')

## Available methods for main effects step

  # Coefficients of the outcome regression
  coef(iqFSM)

  # Description of method used to obtain object
  DTRstep(iqFSM)

  # Value object returned by outcome regression method
  fitObject(iqFSM)

  # Value object returned by outcome regression method
  outcome(iqFSM)

  # Plots if defined by outcome regression method
  dev.new()
  par(mfrow = c(2,4))
  plot(iqFSM)
  plot(iqFSM, suppress = TRUE)
```

```
# Show main results of method
show(iqFSM)

# Show summary results of method
summary(iqFSM)
```

---

iqLearnFSV *IQ-Learning: Variance of the Regression of the Estimated Second-Stage Contrast (IQ3)*

Description

Estimates the variance function of the regression of the estimated second-stage contrast by fitting a log-linear model to the residuals.

Usage

```
iqLearnFSV( ..., object, moMain, moCont, data = NULL, iter = 0, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`object`	An object of class `"IQLearnFS_C."` The value object returned from a previous call to `iqLearnFSC()`
`moMain`	An object of class `"modelObj."` This object specifies the model for the main effects component for the regression of the residual and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	An object of class `"modelObj."` This object specifies the model for the contrast component for the regression of the residual and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`iter`	An object of class `"numeric."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Value

Returns an object of class `"IQLearnFS_VHet"` that inherits directly from class `"DynTxRegime."`

Methods

coef `signature(object = "IQLearnFS_VHet")`: Retrieve parameter estimates for all regression steps.

DTRstep `signature(object = "IQLearnFS_VHet")`: Retrieve description of method used to create object.

estimator `signature(x = "IQLearnFS_VHet")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "IQLearnFS_VHet")`: Retrieve value object returned by regression methods.

optTx `signature(x = "IQLearnFS_VHet", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "IQLearnFS_VHet", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "IQLearnFS_VHet")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "IQLearnFS_VHet")`: Generate plots for regression analyses.

print `signature(object = "IQLearnFS_VHet")`: Print main results of analysis.

qqPlot `signature(x = "IQLearnFS_VHet")`: Generate a qq-plot.

residuals `signature(object = "IQLearnFS_VHet")`: Retrieve standardized fitted residuals.

show `signature(object = "IQLearnFS_VHet")`: Show main results of analysis.

summary `signature(object = "IQLearnFS_VHet")`: Retrieve summary information from regression analyses.

Note

The implementation of IQ-Learning utilizes methods `optTx()` and `estimator()` in a different way than other methods implemented in **DynTxRegime**. For the first-stage component of the IQ-Learning method, all first-stage steps must be complete before an optimal treatment or value can be estimated. Therefore, `optTx()` requires the objects returned by first stage methods `iqLearnFSM()`, `iqLearnFSC()`, and `iqLearnVar()` if using a log-linear model of the variance or `iqLearnFSM()` and `iqLearnFS()` if using a constant model of the variance. Function `estimator()` requires the first-stage objects and the second stage object returned by `iqLearnSS()`. In addition, the density must be specified. See `optTx` and `estimator` for further details.

In addition to the standard methods available to all objects of class `DynTxRegime`, objects of class `"IQLearnFS_VHet"` have qqPlot(x) available. The plot can be used to determine if the normal or empirical density estimator is appropriate. If the observations deviate from the line, den='nonpar' should be used in the final IQ-Learning step.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E. B., Linn, K. A., and Stefanski, L.A. (2014). Interactive model building for Q-learning. Biometrika, 101, 831–847.

See Also

iqLearnFSM, iqLearnFSC, iqLearnSS

Examples

```
#### Full IQ-Learning Algorithm

## Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

## Second-stage regression (IQ1)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

  iqSS <- iqLearnSS(moMain = moMain, moCont = moCont,
                    data = bmiData, response = bmiData$y, txName = "A2",
                    iter = 0L)

## Model conditional mean of main effects function (IQ2)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSM <- iqLearnFSM(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
                      iter = 0L)


## Model conditional mean of contrast function (IQ3)
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  iqFSC <- iqLearnFSC(moMain = moMain, moCont = moCont,
                      data = bmiData, response = iqSS, txName = "A1",
```

```
                      iter = 0L)

## Estimated optimal treatment and value when variance assumed constant
  # optimal treatment
  ot <- optTx(iqFSM, y = iqFSC, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, dens = 'nonpar')

## Log-Linear Variance Modeling (IQ4)

  # heteroskedastic variance
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + baselineBMI,
                          solver.method = 'lm')


  iqFSV <- iqLearnFSV(object = iqFSC,
                      moMain = moMain, moCont = moCont,
                      data = bmiData, txName = "A1",
                      iter = 0L)

## Estimated optimal treatment and value with log-linear variance modeling
  # optimal treatment
  ot <- optTx(iqFSM, y = iqFSC, z = iqFSV, dens = 'nonpar')

  # estimated value
  est <- estimator(x = iqSS, y = iqFSM, z = iqFSC, w = iqFSV, dens = 'nonpar')

## Available methods for variance step

  # Coefficients of the outcome regression
  coef(iqFSV)

  # Description of method used to obtain object
  DTRstep(iqFSV)

  # Value object returned by outcome regression method
  fitObject(iqFSV)

  # Value object returned by outcome regression method
  outcome(iqFSV)

  # Plots if defined by outcome regression method
  dev.new()
  par(mfrow = c(2,4))
  plot(iqFSV)
  plot(iqFSV, suppress = TRUE)
```

```
# qq-plot
dev.new()
qqplot(iqFSV)

# Show main results of method
show(iqFSV)

# Show summary results of method
summary(iqFSV)
```

`iqLearnSS` *IQ-Learning: Second-Stage Regression*

Description

Estimate an optimal dynamic treatment regime using the Interactive Q-learning (IQ-learning) algorithm when the data has been collected from a two-stage randomized trial with binary treatments. iqLearnSS implements the second-stage regression step of the IQ-Learning algorithm (IQ1).

Usage

```
iqLearnSS(..., moMain, moCont, data, response, txName, iter = 0L, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moMain`	An object of class `"modelObj."` This object specifies the main effects component of the model for the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	An object of class `"modelObj."` This object specifies the contrasts component of the model for the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`response`	An object of class `"vector."` The outcome of interest.
`txName`	An object of class `"character."` The column header of the stage treatment variable as given in input `data`. The treatment variable must be binary and will be recoded as -1/+1 if not provided as such.
`iter`	An object of class `"integer."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Value

Returns an object of class "IQLearnSS" that inherits directly from class "DynTxRegime."

Methods

coef signature(object = "IQLearnSS"): Retrieve parameter estimates for all regression steps.

DTRstep signature(object = "IQLearnSS"): Retrieve description of method used to create object.

estimator signature(x = "IQLearnSS"): Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject signature(object = "IQLearnSS"): Retrieve value object returned by regression methods.

fittedCont signature(object = "IQLearnSS"): Retrieve estimated contrast component of outcome regression.

fittedMain signature(object = "IQLearnSS"): Retrieve estimated main effects component of outcome regression.

optTx signature(x = "IQLearnSS", newdata = "missing"): Retrieve the estimated optimal treatment regime for training data set.

optTx signature(x = "IQLearnSS", newdata = "data.frame"): Estimate the optimal treatment regime for newdata.

outcome signature(x = "IQLearnSS"): Retrieve value object returned by outcome regression methods.

plot signature(x = "IQLearnSS"): Generate plots for regression analyses.

print signature(object = "IQLearnSS"): Print main results of analysis.

show signature(object = "IQLearnSS"): Show main results of analysis.

summary signature(object = "IQLearnSS"): Retrieve summary information from regression analyses.

Note

The implementation of IQ-Learning utilizes methods optTx() and estimator() in a different way than other methods implemented in **DynTxRegime**. For the first-stage component of the IQ-Learning method, all first-stage steps must be complete before an optimal treatment or value can be estimated. Therefore, optTx() requires the objects returned by first stage methods iqLearnFSM(), iqLearnFSC(), and iqLearnVar() if using a log-linear model of the variance or iqLearnFSM() and iqLearnFS() if using a constant model of the variance. Function estimator() requires the first-stage objects and the second stage object returned by iqLearnSS(). In addition, the density must be specified. See optTx and estimator for further details.

In addition to the standard methods available to all objects of class "DynTxRegime", the following methods can be applied to objects of class "IQLearnSS"

fittedCont(object) returns the estimated contrast component of the second-stage regression. This is the 'outcome' regressed during the iqLearnFSC() step of the IQ-Learning algorithm.

fittedMain(object) returns the estimated main effects component of the second-stage regression. This is the 'outcome' regressed during the iqLearnFSM() step of the IQ-Learning algorithm.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E. B., Linn, K. A., and Stefanski, L.A. (2014). Interactive model building for Q-learning. Biometrika, 101, 831–847.

See Also

iqLearnFSM, iqLearnFSC, iqLearnFSV

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

# Second-stage regression - Single Regression Analysis
  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

  iqSS <- iqLearnSS(moMain = moMain, moCont = moCont,
                    data = bmiData, response = bmiData$y, txName = "A2",
                    iter = 0L)

## Available methods for second stage step

  # Coefficients of the outcome regression
  coef(iqSS)

  # Description of method used to obtain object
  DTRstep(iqSS)

  # Value object returned by outcome regression method
  fitObject(iqSS)

  # Contrast component of outcome regression
  fittedCont(iqSS)

  # Main effects component of outcome regression
  fittedMain(iqSS)

  # Value object returned by outcome regression method
```

```
outcome(iqSS)

# Plots if defined by outcome regression method
dev.new()
par(mfrow = c(2,4))
plot(iqSS)
plot(iqSS, suppress = TRUE)

# Show main results of method
show(iqSS)

# Show summary results of method
summary(iqSS)
```

`iter` *Defining the iter input variable*

Description

Several of the statistical methods implemented in package **DynTxRegime** allow for an iterative algorithm when completing an outcome regression. This section details how this input is to be defined.

Details

Outcome regression models are specified by the main effects components (`moMain`) and the contrasts component (`moCont`). Assuming that the treatment is denoted as binary A, the full regression model is: moMain + A*moCont. There are two ways to fit this model: (i) in the full model formulation (moMain + A*moCont) or (ii) each component, `moMain` and `moCont`, is fit separately. `iter` specifies if (i) or (ii) should be used.

`iter` >= 1 indicates that `moMain` and `moCont` are to be fit separately using an iterative algorithm. `iter` is the maximum number of iterations. Assume Y = Ymain + Ycont; the iterative algorithm is as follows:

(1) hat(Ycont) = 0;

(2) Ymain = Y - hat(Ycont);

(3) fit Ymain ~ moMain;

(4) set Ycont = Y - hat(Ymain)

(5) fit Ycont ~ A*moCont;

(6) Repeat steps (2) - (5) until convergence or a maximum of iter iterations.

This choice allows the user to specify, for example, a linear main effects component and a non-linear contrasts component.

`iter` <= 0 indicates that the full model formulation is to be used. The components `moMain` and `moCont` will be combined in the package and fit as a single object. Note that if `iter` <= 0, all non-model components of `moMain` and `moCont` must be identical. Specifically, the regression method and any non-default arguments should be identical. By default, the specifications in `moMain` are used.

moPropen *Defining the moPropen input variable*

Description

Several of the statistical methods implemented in package **DynTxRegime** use propensity score modeling. This section details how this input is to be defined.

Details

For input moPropen, the method specified to obtain predictions MUST return the prediction on the scale of the probability, i.e., predictions must be in the range (0,1). In addition, moPropen differs from standard "modelObj" objects in that an additional element may be required in predict.args. Recall, predict.args is the list of control parameters passed to the prediction method. An additional control parameter, propen.missing can be included. propen.missing takes value "smallest" or "largest". It will be required if the prediction method returns predictions for only a subset of the treatment data; e.g., predict.glm(). propen.missing indicates if it is the smallest or the largest treatment value that is missing from the returned predictions.

For example, fitting a binary treatment (A in {0,1}) using

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

returns only P(A=1). P(A=0) is "missing," and thus

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

If the dimension of the value returned by the prediction method is less than the number of treatment options and no value is provided in propen.missing, it is assumed that the smallest valued treatment option is missing. Here, 'smallest' indicates the lowest value integer if treatment is an integer, or the 'base' level if treatment is a factor.

`optimalClass` *Classification Based Estimation of Optimal Dynamic Treatment Regimes*

Description

Uses an Augmented Inverse Propensity Weighted Estimator (AIPWE) or an Inverse Propensity Weighted Estimator (IPWE) of the contrast function to define a weighted classification problem. The method is limited to single decision point binary treatment regimes.

Usage

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

Arguments

`...`	ignored. Included to require named input.
`moPropen`	An object of class `"modelObj."` This object specifies the model of the propensity score regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). See moPropen for further information.
`moMain`	An object of class `"modelObj."` This object specifies the model of the main effects component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	An object of class `"modelObj."` This object specifies the models of the contrasts component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable.
`moClass`	An object of class `"modelObj."` This object defines the classification model (covariates to be used) and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the predicted category.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`response`	An object of class `"vector."` A vector of the outcome of interest.
`txName`	An object of class `"character."` The column header of the stage treatment variable as given in input `data`. Treatment must be binary and will be recoded as 0/1 if not provided as such.
`iter`	An object of class `"numeric."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Details

The user can opt for the IPWE estimator by specifying `moMain` and `moCont` as NULL or not providing them in the named input.

The classification method must accept "weights" as a formal argument.

Value

Returns an object of class `"OptimalClass"` that inherits directly from class `"DynTxRegime."`

Methods

classif `signature(object = "OptimalClass")`: Retrieve value object from classification step. NA for all objects returned by **DynTxRegime** statistical methods except for those returned by optimalClass().

coef `signature(object = "OptimalClass")`: Retrieve parameter estimates for all regression steps.

DTRstep `signature(object = "OptimalClass")`: Retrieve description of method used to create object.

estimator `signature(x = "OptimalClass")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "OptimalClass")`: Retrieve value object returned by regression methods.

optTx `signature(x = "OptimalClass", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "OptimalClass", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "OptimalClass")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "OptimalClass")`: Generate plots for regression analyses.

print `signature(object = "OptimalClass")`: Print main results of analysis.

propen `signature(x = "OptimalClass")`: Retrieve value object returned by propensity score regression methods.

show `signature(object = "OptimalClass")`: Show main results of analysis.

summary `signature(object = "OptimalClass")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhang, B., Tsiatis, A. A., Davidian, M., Zhang, M., and Laber, E. B. (2012). Estimating Optimal Treatment Regimes from a Classification Perspective. Stat, 1, 103–114

Examples

```
library(rpart)

# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

  # Define the propensity for treatment model and methods.
  moPropen <- buildModelObj(model =  ~ 1,
                            solver.method = 'glm',
                            solver.args = list('family'='binomial'),
                            predict.method = 'predict.glm',
                            predict.args = list(type='response'))

  # Define the classification model.
  moClass <- buildModelObj(model = ~parentBMI + month4BMI,
                           solver.method = 'rpart',
                           solver.args = list(method="class"),
                           predict.args = list(type='class'))

# IPWE estimator
  estIPWE <- optimalClass(moPropen = moPropen, moClass = moClass,
                          data = bmiData, response = bmiData$y, txName = "A2",
                          iter = 0L)

  # Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

  # Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

# AIPWE estimator
  estAIPWE <- optimalClass(moPropen = moPropen, moMain = moMain,
                           moCont = moCont, moClass = moClass,
                           data = bmiData, response = bmiData$y, txName = "A2",
                           iter = 0L)

## Available methods

  # Value object returned by classification method
  classif(estAIPWE)

  # Coefficients of the propensity score and outcome regression
  coef(estAIPWE)

  # Description of method used to obtain object
```

```
DTRstep(estAIPWE)

# Estimated value of estimated optimal treatment for training data
estimator(estAIPWE)

# Value object returned by outcome regression method
fitObject(estAIPWE)

# Estimated optimal treatment for training data
optTx(estAIPWE)

# Estimated optimal treatment for new data
optTx(estAIPWE, newdata = bmiData)

# Value object returned by outcome regression method
outcome(estAIPWE)

# Plots if defined by outcome regression method
dev.new()
par(mfrow = c(2,4))
plot(estAIPWE)

dev.new()
par(mfrow = c(2,4))
plot(estAIPWE, suppress = TRUE)

# Value object returned by propensity score regression method
propen(estAIPWE)

# Show main results of method
show(estAIPWE)

# Show summary results of method
summary(estAIPWE)
```

`optimalSeq` *Regression Based Value-Search Estimation of Optimal Dynamic Treatment Regimes*

Description

A doubly robust Augmented Inverse Propensity Weighted Estimator (AIPWE) or Inverse Propensity Weighted Estimator (IPWE) for population mean outcome is optimized over a restricted class of regimes. Methods are available for both single-decision-point and multiple-decision-point regimes. This method requires the **rgenoud** package.

Usage

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

Arguments

...	additional arguments required by rgenoud. At a minimum, this should include Domains, pop.size and starting.values. See ?rgenoud for more information.
moPropen	An object of class "modelObj," an object of class "list" containing objects of class "modelObj," or an object of class "list" containing objects of class "modelObjSubset." This object specifies the model(s) of the propensity score regression and the R methods to be used to obtain parameter estimates and predictions. The method(s) specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). If performing a single-decision-point analysis, moPropen is an object of class "modelObj" or a list of objects of class "modelObjSubset." If performing a multiple-decision-point analysis, moPropen is a list of objects of class "modelObj" or a list of objects of class "modelObjSubset." See moPropen for further information.
moMain	An object of class "modelObj," an object of class "list" containing objects of class "modelObj," or an object of class "list" containing objects of class "modelObjSubset." This object specifies the model(s) of the main effects component of the outcome regression(s) and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable. If performing a single-decision-point analysis, moMain is an object of class "modelObj" or a list of objects of class "modelObjSubset." If performing a multiple-decision-point analysis, moMain is a list of objects of class "modelObj" or a list of objects of class "modelObjSubset."
moCont	An object of class "modelObj," an object of class "list" containing objects of class "modelObj," or an object of class "list" containing objects of class "modelObjSubset." This object specifies the model(s) of the contrasts component of the outcome regression(s) and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable. If performing a single-decision-point analysis, moCont is an object of class "modelObj" or a list of objects of class "modelObjSubset." If performing a multiple-decision-point analysis, moCont is a list of objects of class "modelObj" or a list of objects of class "modelObjSubset."
data	An object of class "data.frame." The covariates and treatment histories.
response	An object of class "vector." A vector of the outcome of interest.
txName	An object of class "character." For a single-decision-point analysis, the column header of the stage treatment variable as given in input data. For multiple-decision-point analyses, a vector, the ith element of which gives the column header of data containing the treatment variable for the ith stage.
regimes	An object of class "function" or an object of class "list" containing objects of class "function." Function(s) defining the class of decision rule to be considered. See Details section for further information
fSet	An object of class "function," an object of class "list" containing objects of class "function," or NULL. See fSet section for further information.

refit	No longer used.
iter	An object of class "numeric." If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
verbose	An object of class "logical." If FALSE, screen prints will be suppressed.

Details

The IPWE estimator can be chosen by specifying moMain and moCont as NULL.

The method uses a genetic algorithm (R package **rgenoud**) to maximize the AIPWE/IPWE estimator over the class of treatment regimes specified by the treatment rules.

A regimes function defines the class of the decision rule to be considered. The formal arguments of each function must include the parameters to be estimated and the data.frame. NOTE: THE LAST ARGUMENT OF THE FUNCTION MUST BE THE DATA FRAME. For example, for

$$d_i = I(eta_1 > x1),$$

```
regimes <- function(a,data){
  as.numeric(a > data$x1)
}
```

For a single-decision-point analysis, regimes is a single function. For multiple-decision-point analyses, regimes is a list of functions where each element of the list corresponds to the decision point (1st element <- 1st decision point, etc.)]

Value

Returns an object of class "OptimalSeq" that inherits directly from class "DynTxRegime."

Methods

coef signature(object = "OptimalSeq"): Retrieve parameter estimates for all regression steps.

DTRstep signature(object = "OptimalSeq"): Retrieve description of method used to create object.

estimator signature(x = "OptimalSeq"): Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject signature(object = "OptimalSeq"): Retrieve value object returned by regression methods.

genetic signature(object = "OptimalSeq"): Retrieve value object returned by genoud().

optTx signature(x = "OptimalSeq", newdata = "missing"): Retrieve the estimated optimal treatment regime for training data set.

optTx signature(x = "OptimalSeq", newdata = "data.frame"): Estimate the optimal treatment regime for newdata.

outcome signature(x = "OptimalSeq"): Retrieve value object returned by outcome regression methods.

plot signature(x = "OptimalSeq"): Generate plots for regression analyses.

print `signature(object = "OptimalSeq")`: Print main results of analysis.

propen `signature(x = "OptimalSeq")`: Retrieve value object returned by propensity score regression methods.

show `signature(object = "OptimalSeq")`: Show main results of analysis.

summary `signature(object = "OptimalSeq")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhang, B., Tsiatis, A. A., Laber, E. B., and Davidian, M. (2012). A Robust Method for Estimating Optimal Treatment Regimes. Biometrics, 68, 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.

Mebane, W. and Sekhon, J. S. (2011). Genetic Optimization Using Derivatives : The rgenoud package for R. Journal of Statistical Software, 42, 1–26.

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

# Define the propensity for treatment model and methods.
  moPropen <- buildModelObj(model =  ~ 1,
                            solver.method = 'glm',
                            solver.args = list('family'='binomial'),
                            predict.method = 'predict.glm',
                            predict.args = list(type='response'))


# Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')

# Create modelObj object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method = 'lm')

# treatment regime rules at each decision point.
  regimes <- function(a,b,c,data){
              tst <- a + b*data$parentBMI + c*data$month4BMI > 0
              res <- character(nrow(data))
```

```
                res[tst] <- "MR"
                res[!tst] <- "CD"
                return(res)
              }

  # genoud requires some additional information
    c1 <- c(-1,-1,-1)
    c2 <- c( 1, 1, 1)
    Domains <- cbind(c1,c2)
    starts <- c(0,0,0)

  #!! A LARGER VALUE FOR POP.SIZE IS RECOMMENDED
  #!! THIS VALUE WAS CHOSEN TO MINIMIZE RUN TIME OF EXAMPLES
    pop.size <- 50
  ## Not run:
    ft <- optimalSeq(moPropen = moPropen, moMain = moMain, moCont = moCont,
                     data = bmiData,  response = bmiData$y,  txName = "A2",
                     regimes = regimes, iter = 0L,
                     pop.size = pop.size, starting.values = starts,
                     Domains = Domains, solution.tolerance = 0.0001)

  ## Available methods

    # Coefficients of the propensity score and outcome regression
    coef(ft)

    # Description of method used to obtain object
    DTRstep(ft)

    # Estimated value of estimated optimal treatment for training data
    estimator(ft)

    # Value object returned by outcome regression method
    fitObject(ft)

    # Value object returned by genetic algorithm
    genetic(ft)

    # Estimated optimal treatment for training data
    optTx(ft)

    # Estimated optimal treatment for new data
    optTx(ft, newdata = bmiData)

    # Value object returned by outcome regression method
    outcome(ft)

    # Plots if defined by outcome regression method
    dev.new()
    par(mfrow = c(2,4))
    plot(ft)

    dev.new()
```

```
  par(mfrow = c(2,4))
  plot(ft, suppress = TRUE)

  # Value object returned by propensity score regression method
  propen(ft)

  # Estimated decision function parameters
  regimeCoef(ft)

  # Show main results of method
  show(ft)

  # Show summary results of method
  summary(ft)


## End(Not run)
```

---

optimObj *Retrieve Optimization Value Object*

Description

Retrieve the value object returned by the optimization method for the weighted learning and EARL statistical methods.

Usage

```
optimObj(object, ...)
```

Arguments

object	An object of class "DynTxRegime" originating from a weighted learning method or EARL.
...	Ignored.

Value

For objects returned by owl(), bowl(), rwl(), and earl(), a list, the exact structure of which depends on the optimization algorithm used by method. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

optTx *Methods to Retrieve Estimated Optimal Treatment or to Predict Optimal Treatment*

Description

Retrieves the estimated or predicted optimal treatment.

Usage

```
optTx(x, newdata, ...)
## S4 method for signature 'DynTxRegime,data.frame'
optTx(x, newdata)
## S4 method for signature 'DynTxRegime,missing'
optTx(x, newdata)
## S4 method for signature 'IQLearnFS,data.frame'
optTx(x, newdata, ..., y = NULL, z = NULL, dens = NULL)
## S4 method for signature 'IQLearnFS,missing'
optTx(x, newdata, ..., y = NULL, z = NULL, dens = NULL)
```

Arguments

x	An object of class "DynTxRegime".
newdata	"data.frame" of covariates
...	Ignored. Used to required named input
y	An object of class IQLearnFS.
z	An object of class IQLearnFS.
dens	"character" indicating density estimator. One of {'nonpar','norm'}.

Value

For outcome regression methods Q-Learning and IQ-Learning and weighted learning methods OWL, BOWL, RWL, and EARL, a list with elements: 'decisionFunc', a matrix of estimated Q-Functions and 'optimalTx', a vector of estimated optimal treatments. For value-search methods optimalClass() and optimalSeq() a matrix of estimated optimal treatments.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

`outcome` *Retrieve Regression Objects for the Outcome Regression Models*

Description

Retrieve the value object(s) returned by the outcome regression method(s).

Usage

```
outcome(object,...)
```

Arguments

`object`	An object of class `"DynTxRegime"`.
`...`	Ignored.

Details

The exact structure of the returned object will depend on the statistical method.

If outcome regression models were fit in a single step, the list returned will be one of {'Combined', 'moMain', 'moCont'} indicating that the main effects and contrast models were combined, that only a main effects model was provided, or that only a contrast model was provided. If the iterative algorithm was used, the list will have two elements: 'moMain' and 'moCont.'

If the method included multiple decision points, the list will have an element for each decision point named with 'dp=x' where x takes the value of the decision point. For example, [['dp=1']], [['dp=2']], etc.

Subset modeling will also result in sub-lists, the names of which are 'Subset=x' where x takes the values of the subset names as defined by the user in the input.

Value

A list. See details section for further information.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

owl *Outcome Weighted Learning*

Description

Estimation of an optimal treatment regime using outcome weighted learning. The convex surrogate of the 0-1 loss function is taken to be the hinge function. The method is for single-decision-point analysis with binary treatment options.

Usage

```
owl(..., moPropen, data, reward, txName, regime, lambdas = 2.0, cvFolds = 0L,
    kernel = "linear", kparam = NULL, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moPropen`	An object of class `"modelObj."` This object specifies the model of the propensity for treatment regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). See moPropen for further information.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`reward`	An object of class `"vector."` A vector of the outcome of interest.
`txName`	An object of class `"character."` The column header of the stage treatment variable as given in input `data`. Treatment must be binary and will be recoded as -1/+1 if not provided as such.
`regime`	An object of class `"formula."` The formula defines the decision rule, i.e., the covariates to be included in the kernel.
`lambdas`	An object of class `"numeric."` Tuning parameter to avoid overfitting. If more than one is given and `cvFolds` > 0, cross-validation will be used to select an optimal from among those provided.
`cvFolds`	An object of class `"integer."` If cross-validation is to be used to find an optimal lambda and/or kernel parameter, the number of folds to use in the CV.
`kernel`	An object of class `"character."` In conjunction with input `kparam`, specification of the kernel function to be used. Must be one of {'linear', 'poly', or 'radial'}. If 'linear,' the linear kernel; `kparam` is ignored. If 'poly,' the polynomial kernel; `kparam` must be specified. If 'radial,' the Gaussian radial basis function kernel; `kparam` must be specified.
`kparam`	An object of class `"numeric."` If input `kernel` = 'linear', this input is ignored. If input `kernel` = 'poly', this input is the order of the polynomial. If input `kernel` = 'radial', this input is sigma; i.e., $$K(x, y) = exp(\|\|x - y\|\|^2/(2 * sigma^2)).$$

For `kernel = 'radial'`, a vector of kernel parameters can be provided and cross-validation will be used to determine the optimal of those provided. Note that input `cvFolds` must be > 0.

`verbose` An object of class `"logical."`. If FALSE, screen prints will be suppressed.

Value

Returns an object of class `"OWL"` that inherits directly from class `"DynTxRegime."`

Methods

coef `signature(object = "OWL")`: Retrieve parameter estimates for all regression steps.

cvInfo `signature(object = "OWL")`: Retrieve cross-validation results.

DTRstep `signature(object = "OWL")`: Retrieve description of method used to create object.

estimator `signature(x = "OWL")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "OWL")`: Retrieve value object returned by regression methods.

optimObj `signature(object = "OWL")`: Retrieve value object returned by optimization routine.

optTx `signature(x = "OWL", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "OWL", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

plot `signature(x = "OWL")`: Generate plots for regression analyses.

print `signature(object = "OWL")`: Print main results of analysis.

propen `signature(x = "OWL")`: Retrieve value object returned by propensity score regression methods.

regimeCoef `signature(object = "OWL")`: Retrieve the estimated decision function parameter estimates.

show `signature(object = "OWL")`: Show main results of analysis.

summary `signature(object = "OWL")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhao, Y-Q., Zeng, D., Rush, A. J., and Kosrok, M. R. (2012). Estimating Individualized Treatment Rules Using Outcome Weighted Learning. Journal of the American Statistical Association, 107, 1106–1118.

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]


# Constant propensity model
  moPropen <- buildModelObj(model = ~1,
                            solver.method = 'glm',
                            solver.args = list('family'='binomial'),
                            predict.method = 'predict.glm',
                            predict.args = list(type='response'))

  owlRes <- owl(moPropen = moPropen,
                data = bmiData, reward = bmiData$y,  txName = "A2",
                regime = ~ parentBMI + month4BMI)

##Available methods

  # Coefficients of the propensity score regression
  coef(owlRes)

  # Description of method used to obtain object
  DTRstep(owlRes)

  # Estimated value of the optimal treatment regime for training set
  estimator(owlRes)

  # Value object returned by propensity score regression method
  fitObject(owlRes)

  # Summary of optimization routine
  optimObj(owlRes)

  # Estimated optimal treatment for training data
  optTx(owlRes)

  # Estimated optimal treatment for new data
  optTx(owlRes, bmiData)

  # Plots if defined by propensity regression method
  dev.new()
  par(mfrow = c(2,4))

  plot(owlRes)
  plot(owlRes, suppress = TRUE)

  # Value object returned by propensity score regression method
  propen(owlRes)
```

```
# Parameter estimates for decision function
regimeCoef(owlRes)

# Show main results of method
show(owlRes)

# Show summary results of method
summary(owlRes)
```

---

plot *Generate Standard Plots*

Description

Generate standard plots. This method uses the plot function as defined by the regression method used to obtain parameter estimates.

Usage

```
plot(x, y, ...)
## S4 method for signature 'DynTxRegime'
plot(x, suppress = FALSE, ...)
```

Arguments

x	An object of class "DynTxRegime".
y	Ignored.
suppress	Logical indicating if identifiers should be added to titles.
...	Additional arguments passed to plot()

Details

Titles and/or subtitles will automatically be appended with a character string indicating the regression model being plotted. This additional information can be suppressed using optional argument "suppress = TRUE" in the call to plot

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

plugInValue *Estimate Plug-in Value*

Description

Estimate the plug-in value of a fixed treatment regime.

Usage

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

Arguments

optTx1	Object of class "vector." First-stage treatments corresponding to the first-stage decision rule of the proposed regime.
optTx2	Object of class "vector." Second-stage treatments corresponding to the second-stage decision rule of the proposed regime.
response	Object of class "vector." Outcome of interest.
tx1	Object of class "vector." First-stage randomized treatments.
tx2	Object of class "vector." Second-stage randomized treatments.

Details

The formula for the plug-in value estimate is

$$\frac{\sum_i Y_i * ind1_i * ind1_i}{\sum_i ind1_i * ind2_i}$$

where $ind1$ and $ind2$ are indicators that the first- and second-stage randomized treatments were consistent with the decision rules.

Value

value	estimated plug-in value of the regime
fixedReg	estimated plug-in value of all possible fixed regimes

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Laber, E.B., Linn, K.A., and Stefanski, L.A. (2014). Interactive Q-learning. Biometrika, 101, 831-847.

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

  # generate examples of optimal treatments
  optTx1 <- 2*rbinom(210, 1, 0.4)-1
  optTx2 <- 2*rbinom(210, 1, 0.5)-1

# Plug-in Values
  tx1 <- numeric(nrow(bmiData)) + 1.0
  tx1[bmiData$A1 == "CD"] <- -1.0

  tx2 <- numeric(nrow(bmiData)) + 1.0
  tx2[bmiData$A2 == "CD"] <- -1.0

  plugInValue(optTx1 = optTx1,
              optTx2 = optTx2,
              response = bmiData$y,
              tx1 = tx1,
              tx2 = tx2)
```

---

propen *Retrieve Regression Objects for Propensity Score*

Description

Retrieve the value object(s) returned by the propensity score regression method(s).

Usage

```
propen(object,...)
```

Arguments

object	An object of class "DynTxRegime".
...	Ignored.

Value

The exact structure of the returned list will depend on the statistical method.

If multiple decision points, the list will have an element for each decision point named with 'dp=x' where x takes the value of the decision point. For example, [['dp=1']], [['dp=2']], etc.

Subset modeling will also result in sub-lists, the names of which are 'Subset=x' where x takes the values of the subset names as defined by the user in the input.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

`qLearn` *Q-learning*

Description

As a single call, regresses the response on covariates and treatment histories to estimate the optimal treatment decision rule. If called repeatedly, once for each treatment decision point, each call is a step of the Q-Learning algorithm.

Usage

```
qLearn(..., moMain, moCont, data, response, txName, fSet, iter = 0L, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moMain`	A single object of class `"modelObj"` or an object of class `"list"` containing objects of class `"modelObjSubset."` This object specifies the model(s) of the main effects component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable.
`moCont`	A single object of class `"modelObj"` or an object of class `"list"` containing objects of class `"modelObjSubset."` This object specifies the models of the contrasts component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class `"data.frame."` The covariates and treatment histories.
`response`	An object of class `"vector"` or `"QLearn."` If performing a single decision point analysis or the first step of the Q-Learning algorithm, i.e., the final-stage regression, response is a vector of the outcome of interest. For multiple decision point analysis, this is the value object returned by the previous call to `qLearn()`.
`txName`	An object of class `"character."` The column header of the stage treatment variable as given in input `data`.
`fSet`	An object of class `"function"` or NULL. See fSet for further information.
`iter`	An object of class `"numeric."` If >0, the iterative method will be used to obtain parameter estimates in the outcome regression step. See iter for further information.
`verbose`	An object of class `"logical."` If FALSE, screen prints will be suppressed.

Value

Returns an object of class `"QLearn"` that inherits directly from class `"DynTxRegime."`

Methods

coef `signature(object = "QLearn")`: Retrieve parameter estimates for all regression steps.

DTRstep `signature(object = "QLearn")`: Retrieve description of method used to create object.

estimator `signature(x = "QLearn")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "QLearn")`: Retrieve value object returned by regression methods.

optTx `signature(x = "QLearn", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "QLearn", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "QLearn")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "QLearn")`: Generate plots for regression analyses.

print `signature(object = "QLearn")`: Print main results of analysis.

show `signature(object = "QLearn")`: Show main results of analysis.

summary `signature(object = "QLearn")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData[,6] - bmiData[,4])/bmiData[,4]

# Second-stage regression
  # Create modeling object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method='lm')

  # Create modeling object for contrast component
  moCont <- buildModelObj(model = ~ parentBMI + month4BMI,
                          solver.method='lm')

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

```
##Available methods

  # Coefficients of the propensity score regression
  coef(fitQ2)

  # Description of method used to obtain object
  DTRstep(fitQ2)

  # Estimated value of the optimal treatment regime for training set
  estimator(fitQ2)

  # Value object returned by propensity score regression method
  fitObject(fitQ2)

  # Estimated optimal treatment for training data
  optTx(fitQ2)

  # Estimated optimal treatment for new data
  optTx(fitQ2, bmiData)

  # Value object returned by outcome regression method
  outcome(fitQ2)

  # Plots if defined by propensity regression method
  dev.new()
  par(mfrow = c(2,4))

  plot(fitQ2)
  plot(fitQ2, suppress = TRUE)

  # Show main results of method
  show(fitQ2)

  # Show summary results of method
  summary(fitQ2)


# First-stage regression

  # Create modeling object for main effect component
  moMain <- buildModelObj(model = ~ gender + race + parentBMI + baselineBMI,
                          solver.method='lm')

  # Create modeling object for contrast component
  moCont <- buildModelObj(model = ~ gender + parentBMI,
                          solver.method='lm')

  fitQ1 <- qLearn(moMain = moMain,
                  moCont = moCont,
                  response = fitQ2,
                  data = bmiData,
                  txName = "A1",
```

```
                    iter = 100L)
```

---

qqplot *IQ-Learning: Generate QQ-Plots for Variance Modeling .*

Description

Generate QQ plots for Variance modeling in IQ-Learning.

Usage

```
qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
                  ylab = deparse(substitute(y)),...)
## S4 method for signature 'IQLearnFS_VHet'
qqplot(x, y, plot.it = TRUE, xlab = deparse(substitute(x)),
                  ylab = deparse(substitute(y)),...)
```

Arguments

x	An object of class "IQLearnFS_VHet".
y	Ignored.
plot.it	logical. Should the result be plotted?
xlab	Plot labels.
ylab	Plot labels.
...	Graphical parameters.

Details

Generates a qq-plot for the variance modeling step of the IQ-Learning algorithm. This plot is used to determine if the normal or empirical density estimator is appropriate. If the observations deviate from the line, den='nonpar' should be used in the final IQ-Learning step.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

regimeCoef *Retrieve Regime Parameter Estimates*

Description

Retrieves treatment regime parameter estimates.

Usage

```
regimeCoef(object,...)
```

Arguments

object	An object of class "DynTxRegime".
...	Ignored.

Value

Defined only for objects returned by owl(), bowl(), rwl(), earl(), and optimalSeq(). For single-decision-point scenarios, a vector of the estimated treatment regime parameters. For multiple-decision-point scenarioes, a list; each element containing the estimated treatment regime parameters for the decision point. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

residuals *Extract Model Residuals*

Description

For methods that employ residuals, retrieve residuals.

Usage

```
residuals(object,...)
```

Arguments

object	An object of class "DynTxRegime".
...	Ignored.

Value

For objects returned by iqLearnFSC(), the residuals of the contrast component regression. For iqLearnFSV(), the standardized fitted residuals. For rwl(), a vector of the residuals used by the method to obtain the optimal treatment. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

`rwl` *Residual Weighted Learning to Estimate Optimal Treatment Regimes*

Description

Estimation of an optimal dynamic treatment regime using residual weighted learning (RWL). The method is limited to single-decision-point scenarios with binary treatment options.

Usage

```
rwl(..., moPropen, moMain, data, reward, txName, regime,
    lambdas = 2.0, cvFolds = 0L, kernel = "linear",
    kparam = NULL, responseType = "continuous",
    guess = NULL, verbose = TRUE)
```

Arguments

`...`	ignored. Included to require named input.
`moPropen`	An object of class "modelObj." This object specifies the model of the propensity for treatment regression and the R methods to be used to obtain parameter estimates and predictions. The method specified to obtain predictions must return the prediction on the scale of the probability, i.e., returned values must be in the interval (0,1). See moPropen for further information.
`moMain`	A single object of class "modelObj." This object specifies the model of the main effects component of the outcome regression and the R methods to be used to obtain parameter estimates and predictions. The method chosen to obtain predictions must return the prediction on the scale of the response variable.
`data`	An object of class "data.frame." The covariates and treatment histories.
`reward`	An object of class "vector." A vector of the outcome of interest.
`txName`	An object of class "character." The column header of the stage treatment variable as given in input data. Treatment must be binary and will be recoded as -1/+1 if not provided as such.
`regime`	An object of class "formula." The formula defines the decision rule, i.e., the covariates to be included in the kernel.

`lambdas` An object of class "numeric." Tuning parameter to avoid overfitting. If more than one is given and cvFolds > 0, cross-validation will be used to select an optimal from among those provided.

`cvFolds` An object of class "integer." If cross-validation is to be used to find an optimal lambda and/or kernel parameter, the number of folds to use in the CV.

`kernel` An object of class "character." In conjunction with input `kparam`, this input specifies the kernel function to be used. Must be one of {'linear', 'poly', or 'radial'}. If 'linear,' the linear kernel; `kparam` is ignored. If 'poly,' the polynomial kernel; `kparam` must be specified. If 'radial,' the Gaussian radial basis function kernel; `kparam` must be specified.

`kparam` An object of class "numeric." If input `kernel` = 'linear', this input is ignored. If input `kernel` = 'poly', this input is the order of the polynomial. If input `kernel` = 'radial', this input is sigma; i.e.,

$$K(x, y) = exp(||x - y||^2/(2 * sigma^2)).$$

For `kernel` = 'radial', a vector of kernel parameters can be provided, and cross-validation will be used to determine the optimal of those provided. Note that input `cvFolds` must be > 0.

`responseType` An object of class "character." One of continuous, binary, count indicating the type of response variable.

`guess` An object of class "numeric" or NULL. Starting parameter values for optimization method.

`verbose` An object of class "logical." If FALSE, screen prints will be suppressed.

Value

Returns an object of class "RWL" that inherits directly from class "DynTxRegime."

Methods

coef `signature(object = "RWL")`: Retrieve parameter estimates for all regression steps.

cvInfo `signature(object = "RWL")`: Retrieve cross-validation results.

DTRstep `signature(object = "RWL")`: Retrieve description of method used to create object.

estimator `signature(x = "RWL")`: Retrieve the estimated value of the estimated optimal regime for the training data set.

fitObject `signature(object = "RWL")`: Retrieve value object returned by regression methods.

optimObj `signature(object = "RWL")`: Retrieve value object returned by optimization routine.

optTx `signature(x = "RWL", newdata = "missing")`: Retrieve the estimated optimal treatment regime for training data set.

optTx `signature(x = "RWL", newdata = "data.frame")`: Estimate the optimal treatment regime for newdata.

outcome `signature(x = "RWL")`: Retrieve value object returned by outcome regression methods.

plot `signature(x = "RWL")`: Generate plots for regression analyses.

print `signature(object = "RWL")`: Print main results of analysis.

propen `signature(x = "RWL")`: Retrieve value object returned by propensity score regression methods.

regimeCoef `signature(object = "RWL")`: Retrieve the estimated decision function parameter estimates.

residuals `signature(object = "RWL")`: Retrieve residuals of outcome regression.

show `signature(object = "RWL")`: Show main results of analysis.

summary `signature(object = "RWL")`: Retrieve summary information from regression analyses.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

References

Zhou, X., Mayer-Hamblett, N., Kham, U., and Kosorok, M. R. (2016+). Residual Weighted Learning for Estimating Individualized Treatment Rules. Journal of the American Statistical Association, in press.

Examples

```
# Load and process data set
  data(bmiData)

  # define response y to be the negative 12 month
  # change in BMI from baseline
  bmiData$y <- -100*(bmiData$month12BMI - bmiData$baselineBMI) /
                    bmiData$baselineBMI


# Constant propensity model
  moPropen <- buildModelObj(model = ~1,
                            solver.method = 'glm',
                            solver.args = list('family'='binomial'),
                            predict.method = 'predict.glm',
                            predict.args = list(type='response'))

# Create modelObj object for main effect component
  moMain <- buildModelObj(model = ~ gender + parentBMI + month4BMI,
                          solver.method = 'lm')
  ## Not run:
    rwlRes <- rwl(moPropen = moPropen, moMain = moMain,
                  data = bmiData, reward = bmiData$y, txName = "A2",
                  regime = ~ parentBMI + month4BMI)

##Available methods

  # Coefficients of the propensity score regression
  coef(rwlRes)
```

```
# Description of method used to obtain object
DTRstep(rwlRes)

# Estimated value of the optimal treatment regime for training set
estimator(rwlRes)

# Value object returned by propensity score regression method
fitObject(rwlRes)

# Summary of optimization routine
optimObj(rwlRes)

# Estimated optimal treatment for training data
optTx(rwlRes)

# Estimated optimal treatment for new data
optTx(rwlRes, bmiData)

# Value object returned by outcome regression method
outcome(rwlRes)

# Plots if defined by propensity regression method
dev.new()
par(mfrow = c(2,4))
plot(rwlRes)

dev.new()
par(mfrow = c(2,4))
plot(rwlRes, suppress = TRUE)

# Value object returned by propensity score regression method
propen(rwlRes)

# Parameter estimates for decision function
regimeCoef(rwlRes)

# Residuals used on method
residuals(rwlRes)

# Show main results of method
show(rwlRes)

# Show summary results of method
summary(rwlRes)


## End(Not run)
```

---

sd *Standard Deviation of IQ-Learning Variance Step*

Description

Returns the standard deviation of the the residuals of the IQ-Learning contrast regression (IQ3).

Usage

```
sd(x, na.rm = FALSE)
## S4 method for signature 'IQLearnFS_C'
sd(x, na.rm = FALSE)
```

Arguments

x	An object of class "IQLearnFS_C".
na.rm	TRUE if NA values are to be removed.

Value

For objects returned by iqLearnFSC(), a numeric object. For all other "DynTxRegime" objects, NA.

Author(s)

Shannon T. Holloway <sthollow@ncsu.edu>

---

show *Show an Object*

Description

Display key results of an object.

Usage

```
show(object)
```

Arguments

object	An object of class "DynTxRegime".

summary *Summary Results*

Description

Retrieve summary results for a DynTxRegime method.

Usage

```
summary(object,...)
```

Arguments

object	An object of class "DynTxRegime".
...	Additional arguments affecting the summary produced.q

Index