Package ‘multiPIM’

February 25, 2015

Version 1.4-3

Date 2015-02-24

Title Variable Importance Analysis with Population Intervention Models

Author Stephan Ritter <sritter@berkeley.edu>, Alan Hubbard <hubbard@berkeley.edu>, Nicholas Jewell <jewell@berkeley.edu>

Maintainer Stephan Ritter <stephanritterRpacks@gmail.com>

Depends lars (>= 0.9-8), penalized, polspline, rpart

Suggests parallel

LazyLoad yes

Description Performs variable importance analysis using a causal inference approach. This is done by fitting Population Intervention Models. The default is to use a Targeted Maximum Likelihood Estimator (TMLE). The other available estimators are Inverse Probability of Censoring Weighted (IPCW), Double-Robust IPCW (DR-IPCW), and Graphical Computation (G-COMP) estimators. Inference can be obtained from the influence curve (plug-in) or by bootstrapping.

License GPL (>= 2)

URL http://www.jstatsoft.org/v57/i08/

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-25 08:12:42

R topics documented:

`Candidates` *Super learner candidates (regression methods) available for use with the multiPIM and multiPIMboot functions*

Description

When the **multiPIM** package is loaded, four character vectors are made available to the user. They are defined as follows:

```
all.bin.cands <- c("polyclass", "penalized.bin", "main.terms.logistic",
                   "rpart.bin")

default.bin.cands <- c("polyclass", "penalized.bin", "main.terms.logistic")

all.cont.cands <- c("polymars", "lars", "main.terms.linear", "penalized.cont",
                    "rpart.cont")

default.cont.cands <- c("polymars", "lars", "main.terms.linear")
```

These vectors (or subsets thereof) can be supplied as arguments to the `multiPIM` and the `multiPIMboot` functions, in order to specify which regression methods should be used to estimate the nuisance parameters g(0, W) and Q(0|W). The user may also supply custom written regression methods or super learner candidates. The mechanism for this is described in a section below.

Candidates

polyclass and polymars:

These candidate use the functions `polyclass` and `polymars` from the package **polspline**.

penalized.bin and penalized.cont:

These candidates perform L1 penalized logistic (penalized.bin) or linear (penalized.cont) regression using the function `penalized` from the package **penalized**. The value of the L1 penalty is selected by cross validation (using the `profL1` function).

lars:

This candidate uses the function `lars` from the package **lars**. Cross validation is performed using the function `cv.lars`.

main.terms.logistic and main.terms.linear:

These candidates perform standard main terms logistic or linear regression, using the functions `glm` and `lm`.

rpart.bin and rpart.cont:

These candidates use the function `rpart` from the package **rpart**. They are not included as default candidates since methods such as this, which are based on an individual tree, have many drawbacks, see e.g. Hastie, Tibshirani and Friedman (2009, section 9.2).

Forcing of Variables into Q Models

Since some of the available candidates (such as polyclass/polymars, penalized) will sometimes completely drop an input variable from the model, it is necessary to have some mechanism to make sure that the relevant exposure variable stays in the model. How this is done for each candidate is described in greater detail in the technical report referenced below (Ritter, Jewell and Hubbard, 2011).

User-Defined Regression Methods and Super Learner Candidates

Below is the code which defines the `main.terms.logistic` candidate function. This is an example of the form that functions which are passed as elements of the `extra.cands` argument should have. (See code for the multiPIM function in the file multiPIM/R/functions.R to see other examples of how candidates are defined.)

```
candidate.functions$main.terms.logistic <- function(X, Y, newX, force,
                                                    standardize) {

  result <- vector("list", length = 4)
  names(result) <- c("preds", "fitted", "main.model", "force.model")
  class(result) <- "main.terms.logistic.result"

  formula.string <- paste(names(Y), "~", paste(names(X), collapse = "+"),
                          sep = "")

  result$main.model <- glm(formula.string, data = cbind(Y, X),
                           family = binomial, model = FALSE, y = FALSE)

  result$preds <- predict(result$main.model, newdata = newX,
                          type = "response")
  result$fitted <- predict(result$main.model, type = "response")
  return(result)
}
```

The functions muse take these four arguments: `X` will be a data frame of predictors, `Y` will be a single-column data frame containing the outcome, and `newX` will be a data frame with columns corresponding to the columns of `X`, containing data on which to predict the outcome based on the model fit by the function. `force` will be an integer specifying the column number (of `X` and `newX`) corresponding to the variable which should be forced into the model in case the function is being used to fit a Q model, and standardize will just be a logical value indicating whether or not to standardize the input variables before running the fitting algorithm (this is meant for algorithms like penalized, where the scale of the predictors will make a difference).

For g models, the force argument will be missing when the function is called, so if the function must do something differently in order to force in a variable (unlike the `main.terms.logistic` function above), then one can use a conditional such as:

`if(missing(force)) ...`

in order to differentiate between g and Q models.

The list returned by the functions must have a slot named `preds` containing the predictions (or predicted probabilities for the binary outcome case), based on `newX`. Also, for the TMLE estimator it is necessary to have the fitted values, i.e. the predictions on `X`. This should be returned in the `fitted` slot. Note that for binary outcomes, these predictions and fitted values should be predicted probabilities that the outcome is equal to 1. Thus, if the candidate is being used for a g model, where the outcome is an exposure variable, the returned values will be estimated probabilities that the exposure variable is equal to 1. The probabilities will be converted as necessary elsewhere in the multiPIM function.

The other slots (main.model and force.model), and setting the class of the object returned, are not necessary for the multiPIM function to work correctly, but may be useful if one would like to inspect the final g and Q models after running the function (see the `return.final.models` argument).

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard and Nicholas Jewell.

References

multiPIM:

Ritter, Stephan J., Jewell, Nicholas P. and Hubbard, Alan E. (2014) “R Package multiPIM: A Causal Inference Approach to Variable Importance Analysis” *Journal of Statistical Software* **57**, 8: 1–29. `http://www.jstatsoft.org/v57/i08/.`

General Machine Learning Reference:

Hastie, T, Tibshirani, R and Friedman, J (2009). *The Elements of Statistical Learning*. Springer, 2nd edition. ISBN: 0387848576

lars:

Efron, B et al. (2004). “Least angle regression”. *The Annals of statistics*, **32**(2):407-499.

penalized:

Goeman, J. J. (2010). “L1 penalized estimation in the cox proportional hazards model”. *Biometrical Journal*, **52**(1):70-84.

polyclass and polymars:

Friedman, J. H. (1991). “Multivariate adaptive regression splines (with discussion)”. *The Annals of Statistics*, **19**:1-141.

Kooperberg, C. et al. (1997). “Polychotomous regression”. *Journal of the American Statistical Association*, **92**(437):117-127.

Stone, C. J. et al. (1997). “The use of polynomial splines and their tensor products in extended linear modeling (with discussion)” . *Annals of Statistics*, **25**:1371-1470.

rpart:

Breiman, L. et al. (1984). *Classification and regression trees*. Wadsworth International Group, Belmont, CA. ISBN: 0534980538.

See Also

`multiPIM`, `multiPIMboot`

multiPIM *Estimate Variable Importances for Multiple Exposures and Outcomes*

Description

The parameter of interest is a type of causal attributable risk. One effect measure (and a corresponding plug-in standard error) will be calculated for each exposure-outcome pair. The default is to use a Targeted Maximum Likelihood Estimator (TMLE). The other available estimators are Inverse Probability of Censoring Weighted (IPCW), Double-Robust IPCW (DR-IPCW), and Graphical Computation (G-COMP) estimators. PIM stands for Population Intervention Model.

Usage

```
multiPIM(Y, A, W = NULL,
         estimator = c("TMLE", "DR-IPCW", "IPCW", "G-COMP"),
         g.method = "main.terms.logistic", g.sl.cands = NULL,
         g.num.folds = NULL, g.num.splits = NULL,
         Q.method = "sl", Q.sl.cands = "default",
         Q.num.folds = 5, Q.num.splits = 1,
         Q.type = NULL,
         adjust.for.other.As = TRUE,
         truncate = 0.05,
         return.final.models = TRUE,
         na.action,
         check.input = TRUE,
         verbose = FALSE,
         extra.cands = NULL,
         standardize = TRUE,
         ...)
```

Arguments

Y	a data frame of outcomes containing only numeric (integer or double) values. See details for the default method of determining, based on the values in `Y`, which regression types to allow for modelling Q. Must have unique names.
A	a data frame containing binary exposure variables. *Binary* means that all values must be either 0 (indicating unexposed, or part of target group) or 1 (indicating exposed or not part of target group). Must have unique names.
W	an optional data frame containing possible confounders of the effects of the variables in `A` on the variables in `Y`. No effect measures will be calculated for these variables. May contain numeric (integer or double), or factor values. Must be left as `NULL` if not required. See details.
estimator	the estimator to be used. The default is `"TMLE"`, for the targeted maximum likelihood estimator. Alternatively, one may specify `"DR-IPCW"`, for the Double-Robust Inverse Probability of Censoring-Weighted estimator, or `"IPCW"`, for the regular IPCW estimator, or `"G-COMP"` for the Graphical Computation estimator.

If the regular IPCW estimator is selected, all arguments which begin with the letter Q are ignored, since only g (the regression of each exposure on possible confounders) needs to be modeled in this case. Similarly, if the G-COMP estimator is selected, all arguments which begin with the letter g, as well as the truncate argument, will be ignored, since only Q needs to be modeled in this case. Note: an additional characteristic of the G-COMP estimator is that there are no plug-in standard errors available. If you want to use G-COMP and you need standard errors, the `multiPIMboot` function is available and will provide bootstrap standard errors.

`g.method`	a length one character vector indicating the regression method to use in modelling g. The default value, `"main.terms.logistic"`, is meant to be used with the default TMLE estimator. If a different estimator is used, it is recommended to use super learning by specifying `"sl"`. In this case, the arguments `g.sl.cands`, `g.num.folds` and `g.num.splits` must also be specified. Other possible values for the `g.method` argument are: one of the elements of the vector `all.bin.cands`, or, if `extra.cands` is supplied, one of the names of the `extra.cands` list of functions. Ignored if `estimator` is `"G-COMP"`.
`g.sl.cands`	character vector of length ≥ 2 indicating the candidate algorithms that the super learner fits for g should use. The possible values may be taken from the vector `all.bin.cands`, or from the names of the `extra.cands` list of functions, if it is supplied. Ignored if `estimator` is `"G-COMP"`. or if `g.method` is not `"sl"`. NOTE: The TMLE estimator is recommended, but if one is using either of the IPCW estimators, a reasonable choice is to specify `g.method = "sl"` and `g.sl.cands = default.bin.cands`.
`g.num.folds`	the number of folds to use in cross-validating the super learner fit for g (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.
`g.num.splits`	the number of times to randomly split the data into `g.num.folds` folds in cross-validating the super learner fit for g. Cross-validation results will be averaged over all splits. Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.
`Q.method`	character vector of length 1. The regression method to use in modelling Q. See details to find out which values are allowed. The default value, `"sl"`, indicates that super learning should be used for modelling Q. Ignored if `estimator` is `"IPCW"`.
`Q.sl.cands`	either of the length 1 character values `"default"` or `"all"` or a character vector of length ≥ 2 containing elements of either `all.bin.cands` or of `all.cont.cands`, or of the names of the `extra.cands` list of functions, if it is supplied. See details. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.folds`	the number of folds to use in cross-validating the super learner fit for Q (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.splits`	the number of times to randomly split the data into `Q.num.folds` folds in cross-validating the super learner fit for Q. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.type`	either `NULL` or a length 1 character vector (which must be either `"binary.outcome"` or `"continuous.outcome"`). This provides a way to override the default mecha-

	nism for deciding which candidates will be allowed for modeling Q (see details). Ignored if `estimator` is `"IPCW"`.
`adjust.for.other.As`	a single logical value indicating whether the other columns of `A` should be included (for `TRUE`) or not (for `FALSE`) in the g and Q models used to calculate the effect of each column of `A` on each column of `Y`. See details. Ignored if `A` has only one column.
`truncate`	either `FALSE`, or a single number greater than 0 and less than 0.5 at which the values of g(0, W) should be truncated in order to avoid instability of the estimator. Ignored if `estimator` is `"G-COMP"`.
`return.final.models`	single logical value indicating whether final g and Q models should be returned by the function (in the slots `g.final.models` and `Q.final.models`). Default is `TRUE`. If memory is a concern, you will probably want to set this to FALSE.
`na.action`	currently ignored. If any of `Y`, `A` or (a non-null) `W` has missing values, `multiPIM` will throw an error.
`check.input`	a single logical value indicating whether all of the input to the function should be subjected to strict error checking. `FALSE` is not recommended.
`verbose`	a single logical value indicating whether messages about the progress of the evaluation should be printed out. Some of the candidate algorithms may print messages even when `verbose` is set to `FALSE`.
`extra.cands`	a named list of functions. This argument provides a way for the user to specify his or her own functions to use either as stand-alone regression methods, or as candidates for a super learner. See details.
`standardize`	should all predictor variables be standardized before certain regression methods are run. Passed to all candidates, but only used by some (at this point, lars, penalized.bin and penalized.cont)
`...`	currently ignored.

Details

The parameter of interest is a type of attributable risk. This means that it is a measure (adjusted for known confounders) of the *difference* between the mean value of `Y` for the units in the target (or unexposed) group and the *overall* mean value of `Y`. Units which are in the target (or unexposed) group with respect to one of the variables in `A` are characterized as such by having the value 0 in the respective column of `A`. Members of the the non-target (or exposed) group should have a 1 in that column of `A`. *Assuming all causal assumptions hold* (see the paper), each parameter estimate can be thought of as estimating the hypothetical effect on the respective outcome of totally eliminating the respective exposure from the population (i.e. setting everyone to 0 for that exposure). For example, in the case of a binary outcome, a parameter estimate for exposure x and outcome y of -0.03 could be interpreted as follows: the effect of an intervention in which the entire population was set to exposure x = 0 would be to reduce the level of outcome y by 3 percentage points.

If `check.input` is `TRUE` (which is the default and is highly recommended), all of the arguments will be checked to make sure they have permissible values. Many of the arguments, especially those for which a single logical value (`TRUE` or `FALSE`) or a single character value (such as, for example, `"all"`) is expected, are checked using the `identical` function, which means that if any of these arguments has any extraneous attributes (such as names), this may cause `multiPIM` to throw an error.

On the other hand, the arguments `Y` and `A` (and `W` if it is non-null) *must* have valid names attributes. `multiPIM` will throw an error if there is any overlap between the names of the columns of these data frames, or if any of the names cannot be used in a `formula` (for example, because it begins with a number and not a letter).

By default, the regression methods which will be allowed for fitting models for Q will be determined from the contents of `Y` as follows: if all values in Y are either 0 or 1 (i.e. all outcomes are binary), then “logistic”-type regression methods will be used (and only these methods will be allowed in the arguments `Q.method` and `Q.sl.cands`); however, if there are any values in `Y` which are not equal to 0 or 1 then it will be assumed that all outcomes are continuous, “linear”-type regression will be used, and the values allowed for `Q.method` and `Q.sl.cands` will change accordingly. This behavior can be overriden by specifying `Q.type` as either `"binary.outcome"` (for logistic-type regression), or as `"continuous.outcome"` (for linear-type regression). If `Q.type` is specified, `Y` will not be checked for binaryness.

The values allowed for `Q.method` (which should have length 1) are: either `"sl"` if one would like to use super learning, or one of the elements of the vector `all.bin.cands` (for the binary outcome case), or of `all.cont.cands` (for the continuous outcome case), if one would like to use only a particular regression method for all modelling of Q. If `Q.method` is given as `"sl"`, then the candidates used by the super learner will be determined from the value of `Q.sl.cands`. If the value of `Q.sl.cands` is `"default"`, then the candidates listed in either `default.bin.cands` or `default.cont.cands` will be used. If the value of `Q.sl.cands` is `"all"`, then the candidates listed in either `all.bin.cands` or `all.cont.cands` will be used. The function will automatically choose the candidates which correspond to the correct outcome type (binary or continuous). Alternatively, one may specify `Q.sl.cands` explicitly as a vector of names of the candidates to be used.

If `A` has more than one column, the `adjust.for.other.As` argument can be used to specify whether the other columns of `A` should possibly be included in the g and Q models which will be used in calculating the effect of a certain column of `A` on each column of `Y`.

With the argument `extra.cands`, one may supply alternative R functions to be used as stand-alone regression methods, or as super learner candidates, within the `multiPIM` function. `extra.cands` should be given as a named list of functions. See Candidates for the form (e.g. arguments) that the functions in this list should have. In order to supply your own stand alone regression method for g or Q, simply specify `g.method` or `Q.method` as the name of the function you want to use (i.e. the corresponding element of the names attribute of `extra.cands`). To add candidates to a super learner, simply use the corresponding names of your functions (from the names attribute of `extra.cands`) when you supply the `g.sl.cands` or `Q.sl.cands` arguments. Note that you may mix and match between your own extra candidates and the built-in candidates given in the `all.bin.cands` and `all.cont.cands` vectors. Note also that extra candidates must be explicitly specified as `g.method`, `Q.method`, or as elements of `g.sl.cands` or `Q.sl.cands` – Specifying `Q.sl.cands` as `"all"` will not cause any extra candidates to be used.

Value

Returns an object of class `"multiPIM"` with the following elements:

`param.estimates`
a matrix of dimensions `ncol(A)` by `ncol(Y)` with `rownames` equal to `names(A)` and `colnames` equal to `names(Y)`, with each element being the estimated causal

attributable risk for the exposure given by its row name vs. the outcome given by its column name.

`plug.in.stand.errs`	a matrix with the same dimensions as `param.estimates` containing the corresponding plug-in standard errors of the parameter estimates. These are obtained from the influence curve. Note: plug-in standard errors are not available for `estimator = "G-COMP"`. This field will be set to `NA` in this case.
`call`	a copy of the call to `multiPIM` which generated this object.
`num.exposures`	this will be set to `ncol(A)`.
`num.outcomes`	this will be set to `ncol(Y)`.
`W.names`	the names attribute of the `W` data frame, if one was supplied. If no `W` was supplied, this will be `NA`.
`estimator`	the estimator used.
`g.method`	the method used for modelling g.
`g.sl.cands`	in case super learning was used for g, the candidates used in the super learner. Will be `NA` if `g.method` was not `"sl"`.
`g.winning.cands`	if super learning was used for g, this will be a named character vector with `ncol(A)` elements. The ith element will be the name of the candidate which "won" the cross validation in the g model for the ith column of `A`.
`g.cv.risk.array`	array with dim attribute `c(ncol(A), g.num.splits, length(g.sl.cands))` containing cross-validated risks from super learner modeling for g for each exposure-split-candidate triple. Has informative dimnames attribute. Note: the values are technically not risks, but log likelihoods (i.e. winning candidate is the one for which this is a *max*, not a min).
`g.final.models`	a list of length `nrow(A)` containing the objects returned by the candidate functions used in the final g models (see Candidates).
`g.num.folds`	the number of folds used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.
`g.num.splits`	the number of splits used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.
`Q.method`	the method used for modeling Q. Will be `NA` if `double.robust` was `FALSE`.
`Q.sl.cands`	in case super learning was used for Q, the candidates used in the super learner. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.winning.cands`	if super learning was used for Q, this will be a named character vector with `ncol(Y)` elements. The ith element is the name of the candidate which "won" the cross validation in the super learner for the Q model for the ith column of `Y`.
`Q.cv.risk.array`	array with dim attribute `c(ncol(A), ncol(Y), Q.num.splits, length(Q.sl.cands))` containing cross-validated risks from super learner modeling for Q. Has informative dimnames attribute. Note: the values will be log likelihoods when `Q.type` is `"binary.outcome"` (see note above for `g.cv.risk.array`), and they will be mean squared errors when `Q.type` is `"continuous.outcome"`.

`Q.final.models`	a list of length `ncol(A)`, each element of which is another list of length `ncol(Y)` containing the objects returned by the candidate functions used for the Q models. I.e. `Q.final.models[[i]][[j]]` contains the Q model information for exposure i and outcome j.
`Q.num.folds`	the number of folds used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.num.splits`	the number of splits used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.type`	either `"continuous.outcome"` or `"binary.outcome"`, depending on the contents of `Y` or on the value of the `Q.type` argument, if supplied.
`adjust.for.other.As`	logical value indicating whether the other columns of `A` were included in models used to calculate the effect of each column of `A` on each column of `Y`. Will be set to `NA` when `A` has only one column.
`truncate`	the value of the `truncate` argument. Will be set to NA if estimator was `"G-COMP"`.
`truncation.occured`	logical value indicating whether it was necessary to trunctate. `FALSE` when `truncate` is `FALSE`. Will be set to NA if estimator was `"G-COMP"`.
`standardize`	the value of the `standardize` argument.
`boot.param.array`	this slot will be `NULL` for objects returned by the `multiPIM` function. See `multiPIMboot` for details on what this slot is actually used for.
`main.time`	total time (in seconds) taken to generate this multiPIM result.
`g.time`	time in seconds taken for running g models.
`Q.time`	time in seconds taken for running Q models.
`g.sl.time`	if g.method is "sl", time in seconds taken for running cross-validation of g models.
`Q.sl.time`	if Q.method is "sl", time in seconds taken for running cross-validation of Q models.
`g.sl.cand.times`	if g.method is "sl", named vector containing time taken, with each element corresponding to a super learner candidate for g.
`Q.sl.cand.times`	if Q.method is "sl", named vector containing time taken, with each element corresponding to a super learner candidate for Q.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard and Nicholas Jewell.

References

Ritter, Stephan J., Jewell, Nicholas P. and Hubbard, Alan E. (2014) “R Package multiPIM: A Causal Inference Approach to Variable Importance Analysis” *Journal of Statistical Software* **57**, 8: 1–29. `http://www.jstatsoft.org/v57/i08/.`

Hubbard, Alan E. and van der Laan, Mark J. (2008) “Population Intervention Models in Causal Inference.” *Biometrika* **95**, 1: 35–47.

Young, Jessica G., Hubbard, Alan E., Eskenazi, Brenda, and Jewell, Nicholas P. (2009) “A Machine-Learning Algorithm for Estimating and Ranking the Impact of Environmental Risk Factors in Exploratory Epidemiological Studies.” *U.C. Berkeley Division of Biostatistics Working Paper Series*, Working Paper 250. `http://www.bepress.com/ucbbiostat/paper250`

van der Laan, Mark J. and Rose, Sherri (2011) *Targeted Learning*, Springer, New York. ISBN: 978-1441997814

Sinisi, Sandra E., Polley, Eric C., Petersen, Maya L, Rhee, Soo-Yon and van der Laan, Mark J. (2007) “Super learning: An Application to the Prediction of HIV-1 Drug Resistance.” *Statistical Applications in Genetics and Molecular Biology* **6**, 1: article 7. `http://www.bepress.com/sagmb/vol6/iss1/art7`

van der Laan, Mark J., Polley, Eric C. and Hubbard, Alan E. (2007) “Super learner.” *Statistical applications in genetics and molecular biology* **6**, 1: article 25. `http://www.bepress.com/sagmb/vol6/iss1/art25`

See Also

`multiPIMboot` for running `multiPIM` with automatic bootstrapping to get standard errors.

`summary.multiPIM` for printing summaries of the results.

`Candidates` to see which candidates are currently available, and for information on writing user-defined super learner candidates and regression methods.

Examples

```
num.columns <- 3
num.obs <- 250

set.seed(23)

## use rbinom with size = 1 to make a data frame of binary data

A <- as.data.frame(matrix(rbinom(num.columns*num.obs, 1, .5),
                          nrow = num.obs, ncol = num.columns))

## let Y[,i] depend only on A[,i] plus some noise
## (start with the noise then add a multiple of A[,i] to Y[,i])

Y <- as.data.frame(matrix(rnorm(num.columns*num.obs),
                          nrow = num.obs, ncol = num.columns))
for(i in 1:num.columns)
  Y[,i] <- Y[,i] + i * A[,i]

## make sure the names are unique

names(A) <- paste("A", 1:num.columns, sep = "")
names(Y) <- paste("Y", 1:num.columns, sep = "")

result <- multiPIM(Y, A)
```

```
summary(result)
```

---

multiPIMboot *Bootstrap the multiPIM Function*

Description

This function will run multiPIM once on the actual data, then sample with replacement from the rows of the data and run multiPIM again (with the same options) the desired number of times.

Usage

```
multiPIMboot(Y, A, W = NULL,
             times = 5000,
             id = 1:nrow(Y),
             multicore = FALSE,
             mc.num.jobs,
             mc.seed = 123,
             estimator = c("TMLE", "DR-IPCW", "IPCW", "G-COMP"),
             g.method = "main.terms.logistic", g.sl.cands = NULL,
             g.num.folds = NULL, g.num.splits = NULL,
             Q.method = "sl", Q.sl.cands = "default",
             Q.num.folds = 5, Q.num.splits = 1,
             Q.type = NULL,
             adjust.for.other.As = TRUE,
             truncate = 0.05,
             return.final.models = TRUE,
             na.action,
             verbose = FALSE,
             extra.cands = NULL,
             standardize = TRUE,
             ...)
```

Arguments

Y	a data frame of outcomes containing only numeric (integer or double) values. See details section of multiPIM for the default method of determining, based on the values in Y, which regression types to allow for modelling Q. Must have unique names.
A	a data frame containing binary exposure variables. *Binary* means that all values must be either 0 (indicating unexposed, or part of target group) or 1 (indicating exposed or not part of target group). Must have unique names.
W	an optional data frame containing possible confounders of the effects of the variables in A on the variables in Y. No effect measures will be calculated for these variables. May contain numeric (integer or double), or factor values. Must be left as NULL if not required. If not NULL, must have unique names.

`times`	single integer greater than or equal to 2. The number of bootstrap replicates of `Y`, `A` and `W` to generate and pass to `multiPIM`.
`id`	vector which identifies clusters. If obervations i and j are in the same cluster, then `id[i]` should be equal to `id[j]`. Bootstrapping will be carried out by sampling with replacement from the clusters. Keeping the default value will result in sampling with replacement from the observations (i.e. no clustering).
`multicore`	logical value indicting whether bootstrapping should be done using multiple simultaneous jobs (as of multiPIM version 1.3-1 this requires the *parallel* package, which is distributed with R version 2.14.0 or later. For earlier versions of multiPIM, this feature relied on CRAN packages *multicore* and *rlecuyer*.
`mc.num.jobs`	number of simultaneous multicore jobs, e.g. if you want to use a quad core processor with hyperthreading, use `mc.num.jobs = 8`. This must be specified whenever `multicore` is true. Automatic detection of the number of cores is no longer available.
`mc.seed`	integer value with which to seed the RNG when using parallel processing (internally, `RNGkind` will be called to set the RNG to `"L'Ecuyer-CMRG"`). Will be ignored if `multicore` is `FALSE`. If `mulicore` is `FALSE`, one "should" (depending on the candidates used) be able to get reprodicible results by setting the seed normally (with `set.seed`) prior to running multiPIMboot.
`estimator`	the estimator to be used. The default is `"TMLE"`, for the targeted maximum likelihood estimator. Alternatively, one may specify `"DR-IPCW"`, for the Double-Robust Inverse Probability of Censoring-Weighted estimator, or `"IPCW"`, for the regular IPCW estimator. If the regular IPCW estimator is selected, all arguments which begin with the letter Q are ignored, since only g (the regression of each exposure on possible confounders) needs to be modeled in this case.
`g.method`	a length one character vector indicating the regression method to use in modelling g. The default value, `"main.terms.logistic"`, is meant to be used with the default TMLE estimator. If a different estimator is used, it is recommended to use super learning by specifying `"sl"`. In this case, the arguments `g.sl.cands`, `g.num.folds` and `g.num.splits` must also be specified. Other possible values for the `g.method` argument are: one of the elements of the vector `all.bin.cands`, or, if `extra.cands` is supplied, one of the names of the `extra.cands` list of functions. Ignored if `estimator` is `"G-COMP"`.
`g.sl.cands`	character vector of length ≥ 2 indicating the candidate algorithms that the super learner fits for g should use. The possible values may be taken from the vector `all.bin.cands`, or from the names of the `extra.cands` list of functions, if it is supplied. Ignored if `estimator` is `"G-COMP"`. or if `g.method` is not `"sl"`. NOTE: The TMLE estimator is recommended, but if one is using either of the IPCW estimators, a reasonable choice is to specify `g.method = "sl"` and `g.sl.cands = default.bin.cands`.
`g.num.folds`	the number of folds to use in cross-validating the super learner fit for g (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.
`g.num.splits`	the number of times to randomly split the data into `g.num.folds` folds in cross-validating the super learner fit for g. Cross-validation results will be averaged over all splits. Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.

`Q.method`	character vector of length 1. The regression method to use in modelling Q. See details to find out which values are allowed. The default value, `"sl"`, indicates that super learning should be used for modelling Q. Ignored if `estimator` is `"IPCW"`.
`Q.sl.cands`	either of the length 1 character values `"default"` or `"all"` or a character vector of length ≥ 2 containing elements of either `all.bin.cands` or of `all.cont.cands`, or of the names of the `extra.cands` list of functions, if it is supplied. See details. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.folds`	the number of folds to use in cross-validating the super learner fit for Q (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.splits`	the number of times to randomly split the data into `Q.num.folds` folds in cross-validating the super learner fit for Q. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.type`	either `NULL` or a length 1 character vector (which must be either `"binary.outcome"` or `"continuous.outcome"`). This provides a way to override the default mechanism for deciding which candidates will be allowed for modeling Q (see details). Ignored if `estimator` is `"IPCW"`.
`adjust.for.other.As`	a single logical value indicating whether the other columns of `A` should be included (for `TRUE`) or not (for `FALSE`) in the g and Q models used to calculate the effect of each column of `A` on each column of `Y`. See details. Ignored if `A` has only one column.
`truncate`	either `FALSE`, or a single number greater than 0 and less than 0.5 at which the values of g(0, W) should be truncated in order to avoid instability of the estimator. Ignored if `estimator` is `"G-COMP"`.
`return.final.models`	single logical value indicating whether final g and Q models should be returned by the function (in the slots `g.final.models` and `Q.final.models`). Default is `TRUE`. If memory is a concern, you will probably want to set this to FALSE. Note that only g and Q models for the main multiPIM run will be returned, not for each of the bootstrap runs.
`na.action`	currently ignored. If any of `Y`, `A` or (a non-null) `W` has missing values, `multiPIMboot` will throw an error.
`verbose`	single logical value. Should messages about the progress of the evaluation be printed out. Some of the candidate algorithms may print messages even when `verbose` is set to `FALSE`.
`extra.cands`	a named list of functions. This argument provides a way for the user to specify his or her own functions to use either as stand-alone regression methods, or as candidates for a super learner. See details section of `multiPIM`.
`standardize`	should all predictor variables be standardized before certain regression methods are run. Passed to all candidates, but only used by some (at this point, lars, penalized.bin and penalized.cont)
`...`	currently ignored.

Details

Bootstrap standard errors can be calculated by running the `summary` function on the `multiPIMboot` result (see `link{summary.multiPIM}`).

As of *multiPIM* version 1.3-1, support for multicore processing is through R's *parallel* package (distributed with R as of version 2.14.0).

For more details on how to use the arguments, see the details section for `multiPIM`.

Value

Returns an object of class `"multiPIM"` which is identical to the object resulting from running the `multiPIM` function in the original data, except for two slots which are slightly different: the `call` slot contains a copy of the original call to `multiPIMboot`, and the `boot.param.array` slot now contains the bootstrap distribution of the parameter estimates gotten by running `multiPIM` on the bootstrap replicates of the original data. Thus the object returned has the following slots:

`param.estimates`	a matrix of dimensions `ncol(A)` by `ncol(Y)` with `rownames` equal to `names(A)` and `colnames` equal to `names(Y)`, with each element being the estimated causal attributable risk for the exposure given by its row name vs. the outcome given by its column name.
`plug.in.stand.errs`	a matrix with the same dimensions as `param.estimates` containing the corresponding plug-in standard errors of the parameter estimates. These are obtained from the influence curve. Note: plug-in standard errors are not available for `estimator = "G-COMP"`. This field will be set to `NA` in this case.
`call`	a copy of the call to `multiPIMboot` which generated this object.
`num.exposures`	this will be set to `ncol(A)`.
`num.outcomes`	this will be set to `ncol(Y)`.
`W.names`	the names attribute of the `W` data frame, if one was supplied. If no `W` was supplied, this will be `NA`.
`estimator`	the estimator used.
`g.method`	the method used for modelling g.
`g.sl.cands`	in case super learning was used for g, the candidates used in the super learner. Will be `NA` if `g.method` was not `"sl"`.
`g.winning.cands`	if super learning was used for g, this will be a named character vector with `ncol(A)` elements. The ith element will be the name of the candidate which "won" the cross validation in the g model for the ith column of `A`.
`g.cv.risk.array`	array with dim attribute `c(ncol(A), g.num.splits, length(g.sl.cands))` containing cross-validated risks from super learner modeling for g for each exposure-split-candidate triple. Has informative dimnames attribute. Note: the values are technically not risks, but log likelihoods (i.e. winning candidate is the one for which this is a *max*, not a min).

`g.final.models`	a list of length `nrow(A)` containing the objects returned by the candidate functions used in the final g models (see Candidates).
`g.num.folds`	the number of folds used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.
`g.num.splits`	the number of splits used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.
`Q.method`	the method used for modeling Q. Will be `NA` if `double.robust` was `FALSE`.
`Q.sl.cands`	in case super learning was used for Q, the candidates used in the super learner. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.winning.cands`	if super learning was used for Q, this will be a named character vector with `ncol(Y)` elements. The ith element is the name of the candidate which "won" the cross validation in the super learner for the Q model for the ith column of `Y`.
`Q.cv.risk.array`	array with dim attribute `c(ncol(A), ncol(Y), Q.num.splits, length(Q.sl.cands))` containing cross-validated risks from super learner modeling for Q. Has informative dimnames attribute. Note: the values will be log likelihoods when `Q.type` is `"binary.outcome"` (see note above for `g.cv.risk.array`), and they will be mean squared errors when `Q.type` is `"continuous.outcome"`.
`Q.final.models`	a list of length `ncol(A)`, each element of which is another list of length `ncol(Y)` containing the objects returned by the candidate functions used for the Q models. I.e. `Q.final.models[[i]][[j]]` contains the Q model information for exposure i and outcome j.
`Q.num.folds`	the number of folds used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.num.splits`	the number of splits used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.
`Q.type`	either `"continuous.outcome"` or `"binary.outcome"`, depending on the contents of `Y` or on the value of the `Q.type` argument, if supplied.
`adjust.for.other.As`	logical value indicating whether the other columns of `A` were included in models used to calculate the effect of each column of `A` on each column of `Y`. Will be set to `NA` when `A` has only one column.
`truncate`	the value of the `truncate` argument. Will be set to NA if estimator was `"G-COMP"`.
`truncation.occured`	logical value indicating whether it was necessary to trunctate. `FALSE` when `truncate` is `FALSE`. Will be set to NA if estimator was `"G-COMP"`.
`standardize`	the value of the `standardize` argument.
`boot.param.array`	a three dimensional array with `dim` attribute equal to `c(times, ncol(A), ncol(Y))` containing the corresponding parameter estimate for each bootstrap replicatate-exposure-outcome trio. Also has an informative `dimnames` attribute for easy printing.
`main.time`	time (in seconds) taken for main run of multiPIM on the original data.

`g.time`	time in seconds taken for running g models.
`Q.time`	time in seconds taken for running Q models.
`g.sl.time`	if g.method is "sl", time in seconds taken for running cross-validation of g models.
`Q.sl.time`	if Q.method is "sl", time in seconds taken for running cross-validation of Q models.
`g.sl.cand.times`	if g.method is "sl", named vector containing time taken, with each element corresponding to a super learner candidate for g.
`Q.sl.cand.times`	if Q.method is "sl", named vector containing time taken, with each element corresponding to a super learner candidate for Q.

Note that all timing results apply only to the first run of codelinkmultiPIM on the original data, not the subsequent bootstrap runs.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard and Nicholas Jewell.

References

Ritter, Stephan J., Jewell, Nicholas P. and Hubbard, Alan E. (2014) “R Package multiPIM: A Causal Inference Approach to Variable Importance Analysis” *Journal of Statistical Software* **57**, 8: 1–29. `http://www.jstatsoft.org/v57/i08/.`

Hubbard, Alan E. and van der Laan, Mark J. (2008) “Population Intervention Models in Causal Inference.” *Biometrika* **95**, 1: 35–47.

Young, Jessica G., Hubbard, Alan E., Eskenazi, Brenda, and Jewell, Nicholas P. (2009) “A Machine-Learning Algorithm for Estimating and Ranking the Impact of Environmental Risk Factors in Exploratory Epidemiological Studies.” *U.C. Berkeley Division of Biostatistics Working Paper Series*, Working Paper 250. `http://www.bepress.com/ucbbiostat/paper250`

van der Laan, Mark J. and Rose, Sherri (2011) *Targeted Learning*, Springer, New York. ISBN: 978-1441997814

Sinisi, Sandra E., Polley, Eric C., Petersen, Maya L, Rhee, Soo-Yon and van der Laan, Mark J. (2007) “Super learning: An Application to the Prediction of HIV-1 Drug Resistance.” *Statistical Applications in Genetics and Molecular Biology* **6**, 1: article 7. `http://www.bepress.com/sagmb/vol6/iss1/art7`

van der Laan, Mark J., Polley, Eric C. and Hubbard, Alan E. (2007) “Super learner.” *Statistical applications in genetics and molecular biology* **6**, 1: article 25. `http://www.bepress.com/sagmb/vol6/iss1/art25`

See Also

`multiPIM` for the main function which is called by `multiPIMboot`.

`summary.multiPIM` for printing summaries of the results.

`Candidates` to see which candidates are currently available, and for information on writing user-defined super learner candidates and regression methods.

Examples

```
## Warning: This would take a very long time to run!
## Not run:
## load example from multiPIM help file

example(multiPIM)

## this would run 5000 bootstrap replicates:

boot.result <- multiPIMboot(Y, A)

summary(boot.result)
## End(Not run)
```

---

schisto *Schistosomiasis Data Set*

Description

Data on types of water exposure and schistosomiasis infection.

Usage

```
data(schisto)
```

Format

A data frame containing the outcome variable (stoolpos – 1 = positive stool sample (infected), 0 = negative stool sample), 7 water contact exposure variables (total time subject spent doing each activity: laundry, tool washing, bathing, swimming, ditch digging, rice planting, fishing), and two categorical covariates (village id number and age category).

Note: Village label does not correspond to the id numbers from the paper.

Source

Spear RC, Seto E, Liang S, Birkner M, Hubbard A, Qiu D, Yang C, Zhong B, Xu F, Gu X, Davis GM (2004). "Factors Influencing the Transmission of Schistosoma Japonicum in the Mountains of Sichuan Province of China." The American Journal of Tropical Medicine and Hygiene, 70(1), 48-56.

`summary.multiPIM` *Summary methods for class multiPIM*

Description

Generate and print summaries of `"multiPIM"` objects (which result from calling either the `multiPIM` or the `multiPIMboot` function). Summaries may be of type "statistical", "time" or "both" (default). Statistical summaries contain, for each exposure-outcome pair, the parameter estimate, the standard error, the test statistic, the unadjusted p-value, and the Bonferroni-adjusted p-value. Time summaries contain a breakdown by g vs. Q modeling, and (if super learning was used to generate the `"multiPIM"` object) by super learner candidate, of the time taken to run `multiPIM`.

Usage

```
## S3 method for class 'multiPIM'
summary(object,
                type = c("both", "statistical", "time"),
                use.plug.in.se = is.null(object$boot.param.array),
                alternative.se.matrix = NULL,
                two.sided.p.vals = TRUE,
                bf.multiplier = object$num.exp * object$num.out,
                by.exposure = TRUE,
                digits = 4,
                ...)

## S3 method for class 'summary.multiPIM'
print(x, by.exposure, digits, ...)
```

Arguments

`object`	an object of class `"multiPIM"` (the result of a call to `multiPIM` or `multiPIMboot`) to be summarized.
`type`	The type of summary required. Default is to include both statistical and timing information in the summary.
`use.plug.in.se`	logical value indicating whether the plug-in standard errors are to be used. Defaults to `TRUE` when `object` is the result of a call to `multiPIM`, and to `FALSE` when `object` is the result of a call to `multiPIMboot`, in which case the bootstrap standard errors are used to calculate test statistics and p-values. This argument is ignored when `alternative.se.matrix` is non-null.
`alternative.se.matrix`	matrix of standard errors which were obtained by the user through some method other than the normal plug-in (from `multiPIM`) or bootstrap (from `multiPIMboot`). Must have `dim` attribute equal to that of `object$param.estimates` and corresponding elements will be assumed to match up.
`two.sided.p.vals`	logical value. Should p-values be two-sided (for `TRUE`), or one-sided (for `FALSE`).

`bf.multiplier`	what number should p-values be multiplied by in doing the Bonferroni Correction. Defaults to the number of exposure-outcome pairs.
`by.exposure`	logical value. If the summary is printed, and if there is more than one exposure AND more than one outcome, should each table correspond to the exposure-outcome pairs involving each exposure (for `TRUE`, the default), or each outcome (for `FALSE`). The `by.exposure` argument to the print method, if given, will override the one specified when the summary was generated with the summary method.
`digits`	passed to `print.default` when and if the summary object is printed. The `digits` argument to the print method, if given, will override the one specified when the summary was generated with the summary method.
`x`	an object of class `"summary.multiPIM"` (the result of a call to the summary method) to be printed.
`...`	for the summary method: currently ignored. For the print method: passed to `print.default`.

Value

For the summary method:

An object of class `"summary.multiPIM"`, which will have different slots depending on the value of the `type` argument.

The object will always have the following slots (regardless of the value of `type`):

`type`	the value of the type argument to summary.
`digits`	integer value which will be the default value passed to `print.default` when this object is printed.
`call`	a copy of the call which was used to generate the `"multiPIM"` object on which the summary method was called.

If `type` is `"statistical"` or `"both"`, the result of summary will in addition have the following slots:

`summary.array`	a three-dimensional array containing the information which can be used to build summary tables (see below).
`two.sided.p.vals`	logical value indicating whether p-values used are two-sided (for `TRUE`) or one-sided (for `FALSE`).
`stand.err.type`	the type of standard error which has been used to generate this summary object: either `"plug.in"`, `"bootstrap"`, or `"alternative"`.
`bf.multiplier`	the value of the bf.multiplier argument
`by.exposure`	logical value which will be used by default when this object is printed to decide whether the tables should be arranged by exposure (for `TRUE`) or by outcome (for `FALSE`).

Details for the `summary.array` slot: the first dimension corresponds to the exposures (columns of A from the `"multiPIM"` object for which the summary is being generated), the second dimension to the outcomes (columns of Y) and the third dimension has length 5 and corresponds to the 5 relevant attributes for each exposure-outcome pair (i.e. the parameter estimate, the standard error of that estimate, the test statistic, the unadjusted p-value and the Bonferroni-adjusted p-value, in that order). Thus, `summary.array[1,2,3]` would be the test statistic for the pair consisting of the first exposure (first column of A) and the 2nd outcome (2nd column of Y), while `summary.array[3,2,1]` would be the parameter estimate for the pair consisting of the 3rd exposure and the 2nd outcome. To access the matrix containing all unadjusted p-values, use `summary.array[,,4]`, to access the matrix consisting of everything that involves the fourth outcome use `summary.array[,4,]`.

If `type` is `"time"` or `"both"`, the result of summary will in addition have the following slots:

`main.time`	total time (in seconds) taken to generate the multiPIM result which is being summarized.
`g.time`	time in seconds taken for running g models.
`Q.time`	time in seconds taken for running Q models.
`g.Q.time.frame`	data frame containing breakdown of total time by g vs. Q modeling, with seconds and percentages
`g.sl.time`	if g.method is "sl", time in seconds taken for running cross-validation of g models.
`Q.sl.time`	if Q.method is "sl", time in seconds taken for running cross-validation of Q models.
`g.sl.xval.time.mat`	if super learning was used for g, a matrix containing a breakdown by super learner candidate of the time taken for cross-validation of g models, with seconds and percentages.
`Q.sl.xval.time.mat`	if super learning was used for Q, a matrix containing a breakdown by super learner candidate of the time taken for cross-validation of Q models, with seconds and percentages.

Print method:

The print method returns its first argument (`x`, which should be an object of class `"summary.multiPIM"`) invisibly.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard and Nicholas Jewell.

See Also

`multiPIM` and `multiPIMboot`

Examples

```
## load example from multiPIM help file

example(multiPIM)

## The results can also be displayed by outcome instead of by exposure:

summary(result, by.exposure = FALSE)
## now each table corresponds to all the pairs involving a single outcome

## may be best to store the summary object

sum.obj <- summary(result, by.exposure = FALSE)
sum.obj

## now the print method can be used to overide the values for
## by.exposure and digits (but not the other arguments):

print(sum.obj, by.exposure = TRUE, digits =  3)

## also can hand pick the info that we want from the summary.array slot
## e.g. let's say we are interested in all of the standard errors:

sum.obj$summary.array[,,2]

## or we are only interested in the exposure1-outcome2 pair:

sum.obj$summary.array[1,2,]

## or by name

sum.obj$summary.array["A1","Y2",]
```

---

wcgs *Subset of Data from Western Collaborative Group Study*

Description

A subset of the data from the Western Collaborative Group Study (described in Rosenman et al. 1975)

Usage

```
data(wcgs)
```

Format

A data frame containing the following variables (all are type integer except for bmi, which is double):

- chd69: Outcome variable – did subject get CHD by end of follow up (1969)?
- sbp0: Systolic blood pressure at baseline.
- dbp0: Diastolic blood pressure at baseline.
- chol0: Fasting serum cholesterol at baseline (contains 12 missing values – NA).
- ncigs0: Cigarettes per day.
- dibpat0: Behavior type – 1 = type A, 2 = type B.
- age0: Age in years at baseline.
- height0: Height in inches.
- weight0: Weight in lbs.
- bmi0: BMI based on height0 and weight0.

Source

Rosenman RH, Brand RJ, Jenkins CD, Friedman M, Straus R, Wurm M (1975). "Coronary Heart Disease in the Western Collaborative Group Study. Final Follow-up Experience of 8 1/2 Years." JAMA, 233(8), 872-7.

Index