Package ‘optinterv’

March 19, 2020

Type Package

Title Optimal Intervention

Version 0.1.0

Description Provides both parametric and non-parametric estimates of the correlates of some desired outcome (e.g. test scores, income) using a new method proposed by Danieli, Devi and Fryer (2019). This method relaxes the assumption that one can alter individual characteristics in any way the data suggest is optimal, and so can be used anytime one wants to use observational data to better optimize social experiments designed to increase some desired outcome.

License GPL-3

Encoding UTF-8

LazyData true

Imports rootSolve, distances, weights, boot, lattice, pbapply, Hmisc, stats, graphics

RoxygenNote 6.1.1

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Elad Guttman [aut, cre], Oren Danieli [aut]

Maintainer Elad Guttman <eladguttman@mail.tau.ac.il>

Repository CRAN

Date/Publication 2020-03-19 16:30:02 UTC

R topics documented:

`add_sign` *Add signs to variable names*

Description

Add signs to variable names

Usage

```
add_sign(names, signs)
```

Arguments

`names`	vector of variable names.
`signs`	vector of signs (the same length as names).

boot_ci *Bootstrap Confidence Intervals*

Description

Calculates bootstrap confidence intervals for matrix of bootstrap replicates

Usage

```
boot_ci(boot.res, alpha = 0.05)
```

Arguments

boot.res	matrix of bootstrap replicates
alpha	significance level

Value

matrix of confidence intervals

boot_default *Bootstrap (default) Bootstrap function for the non-parametric and the nearest neighbor methods*

Description

Bootstrap (default) Bootstrap function for the non-parametric and the nearest neighbor methods

Usage

```
boot_default(func, Y, Y_pos, X, X_std, control, wgt, n.quant, lambda,
  sigma, grp.size, n.boot, quick)
```

Arguments

func	a function for weights calculation (nn / non_parm).
Y	the original outcome.
Y_pos	outcome after exponential transformation (if needed).
X	the original X matrix.
X_std	X matrix after standardization.
control	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
wgt	an optional vector of weights.

n.quant	number of quantiles to use when calculating CDF distance.
lambda	the lagrange multiplier. also known as the shadow price of an intervention.
sigma	distance penalty for the nearest-neighbors method.
grp.size	for the nearest-neighbors method; if the number of examples in each control group is smaller than grp.size, performs weight adjustment using wgt_adjust. else, calculate weights seperatly for each control group.
n.boot	number of bootstrap replications to use for the standard errors / confidence intervals calculation.
quick	logical. if TRUE, returns only $E(X\|I = 1) - E(X\|I = 0)$ as an estimate. this estimate is used by optint_by_group.

Value

a list - the output from the function 'boot()'.

---

dev_moments *Moment deviation*

Description

Finds the moment deviation for a given lagrange multiplier

Usage

```
dev_moments(beta, base, control, wgt)
```

Arguments

beta	a lagrange multiplier
base	basline weights
control	control matrix (with a constant)
wgt	original weights

Value

vector of moment deviations

`kl_dist_cor` *Kullback-Leibler Divergence*

Description

Calculates Kullback-Liebler Divergence for two multivariate normal distributions.

Usage

```
kl_dist_cor(X, wgt, ni)
```

Arguments

`X`	numeric data frame or matrix of factors to be considered.
`wgt`	an optional vector of weights.
`ni`	difference in means (mu1 - mu0)

Value

scalar of kullback-liebler divergence.

`kl_dist_def` *Kullback-Leibler Divergence*

Description

Calculates Kullback-Liebler Divergence for two weight vectors.

Usage

```
kl_dist_def(wgt, wgt1)
```

Arguments

`wgt`	original weights.
`wgt1`	weights under I = 1.

Value

scalar of kullback-liebler divergence.

nn *Nearest-neighbors method*

Description

Calculates adjusted weights under I = 1, using the nearest-neighbors method

Usage

```
nn(Y, X, control = NULL, wgt = rep(1, length(Y)), lambda = 100,
  sigma = 1, grp.size = 30, ...)
```

Arguments

`Y`	outcome vector (must be numeric without NA's).
`X`	numeric data frame or matrix of factors to be considered.
`control`	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
`wgt`	an optional vector of weights.
`lambda`	the lagrange multiplier. also known as the shadow price of an intervention.
`sigma`	distance penalty for the nearest-neighbors method.
`grp.size`	for the nearest-neighbors method; if the number of examples in each control group is smaller than grp.size, performs weight adjustment using `wgt_adjust`. else, calculate weights seperatly for each control group.
`...`	additional arguments.

Value

vector of adjusted weights under I = 1

nn_wgt *Nearest-neighbors weights*

Description

Calculates unadjusted weights under I = 1, using the nearest-neighbors method

Usage

```
nn_wgt(Y, X, control = NULL, wgt = rep(1, length(Y)), lambda = 100,
  sigma = 1, test = F)
```

Arguments

Y	outcome vector (must be numeric without NA's).
X	numeric data frame or matrix of factors to be considered.
control	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
wgt	an optional vector of weights.
lambda	the lagrange multiplier. also known as the shadow price of an intervention.
sigma	distance penalty for the nearest-neighbors method.
test	if TRUE, returns weights matrix (only used for testing).

Value

vector of unadjusted weights under I = 1

---

non_parm *Non-parametric method*

Description

Calculates weights under I = 1, using the non-parametric method

Usage

```
non_parm(Y, X, control = NULL, wgt = rep(1, length(Y)), lambda = 100,
  ...)
```

Arguments

Y	outcome vector (must be numeric without NA's).
X	numeric data frame or matrix of factors to be considered.
control	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
wgt	an optional vector of weights.
lambda	the lagrange multiplier. also known as the shadow price of an intervention.
...	additional arguments.

Value

vector of weights under I = 1

optint *Optimal intervention*

Description

identifies the factors with the greatest potential to increase a pre-specified outcome, using varius methods.

Usage

```
optint(Y, X, control = NULL, wgt = rep(1, length(Y)),
  method = "non-parametric", lambda = 100, sigma = 1,
  grp.size = 30, n.boot = 1000, sign.factor = 2/3, alpha = 0.05,
  n.quant = floor(length(Y)/10), perm.test = T, n.perm = 1000,
  quick = F, plot = T, seed = runif(1, 0, .Machine$integer.max))
```

Arguments

`Y`	outcome vector (must be numeric without NA's).
`X`	numeric data frame or matrix of factors to be considered.
`control`	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
`wgt`	an optional vector of weights.
`method`	the method to be used. either "non-parametric" (default), "correlation" or "nearest-neighbors".
`lambda`	the lagrange multiplier. also known as the shadow price of an intervention.
`sigma`	distance penalty for the nearest-neighbors method.
`grp.size`	for the nearest-neighbors method; if the number of examples in each control group is smaller than grp.size, performs weight adjustment using `wgt_adjust`. else, calculate weights seperatly for each control group.
`n.boot`	number of bootstrap replications to use for the standard errors / confidence intervals calculation.
`sign.factor`	what proportion of quantiles should to be increased (decreased) in order to return a positive (negative) sign? not relevant for the correlation method (there the correlation sign is returned).
`alpha`	significance level for the confidence intervals.
`n.quant`	number of quantiles to use when calculating CDF distance.
`perm.test`	logical. if TRUE (default) performs permutation test and calculates p-values.
`n.perm`	number of permutations for the permutation test.
`quick`	logical. if TRUE, returns only $E(X\|I = 1) - E(X\|I = 0)$ as an estimate. this estimate is used by `optint_by_group`.
`plot`	logical. if TRUE (default), the results are plotted by `plot.optint`.
`seed`	the seed of the random number generator.

Value

an object of class "optint". This object is a list containing the folowing components:

`estimates` standardized point estimates (correlations for the correlation method and cdf distances otherwise).

`estimates_sd` estimates standard deviation.

`details` a list containing further details, such as:

- Y_diff - $E(Y|I = 1) - E(Y|I = 0)$.
- Y_diff_sd - standard deviation for Y_diff.
- method - the method used.
- lambda - the lagrange multiplier used.
- signs - signs (i.e. directions) for the estimates.
- p_value - p-values for the estimates.
- ci - a matrix of confidence intervals for the estimates.
- stand_factor - the standardization factor used to standardize the results.
- kl_distance - the Kullback–Leibler divergence of $P(X|I = 0) from P(X|I = 1)$.
- new_sample - a data frame containing X, control (if provided), wgt (the original weights) and wgt1 (the new weights under $I = 1$.)

In addition, the function `summary` can be used to print a summary of the results.

Examples

```
# generate data
n <- 50
p <- 3
features <- matrix(rnorm(n*p), ncol = p)
men <- matrix(rbinom(n, 1, 0.5), nrow = n)
outcome <- 2*(features[,1] > 1) + men*pmax(features[,2], 0) + rnorm(n)
outcome <- as.vector(outcome)

#find the optimal intervention using the non-parametric method:
imp_feat <- optint(Y = outcome, X = features, control = men,
                   method = "non-parametric", lambda = 10, plot = TRUE,
                   n.boot = 100, n.perm = 100)

#by default, only the significant features are displayed
#(see ?plot.optint for further details).
#for customized variable importance plot, use plot():
plot(imp_feat, plot.vars = 3)

#show summary of the results using summary():
summary(imp_feat)
```

optint_by_group *Optimal intervention, by group*

Description

Similar to optint, identifies the factors with the greatest potential to increase a pre-specified outcome for each group separately, and thus allowing to detect heterogeneity between groups.

Usage

```
optint_by_group(Y, X, group, control = NULL, wgt = rep(1, length(Y)),
  method = "non-parametric", lambda = 100, sigma = 1,
  grp.size = 30, n.boot = 1000, alpha = 0.05, plot = TRUE)
```

Arguments

Y	outcome vector (must be numeric without NA's).
X	numeric data frame or matrix of factors to be considered.
group	vector with group labels (i.e. grouping variable). the function optint implemented for each group separately.
control	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
wgt	an optional vector of weights.
method	the method to be used. either "non-parametric" (default), "correlation" or "nearest-neighbors".
lambda	the lagrange multiplier. also known as the shadow price of an intervention.
sigma	distance penalty for the nearest-neighbors method.
grp.size	for the nearest-neighbors method; if the number of examples in each control group is smaller than grp.size, performs weight adjustment using wgt_adjust. else, calculate weights seperatly for each control group.
n.boot	number of bootstrap replications to use for the standard errors / confidence intervals calculation.
alpha	significance level for the confidence intervals.
plot	logical. if TRUE (default), the results are plotted by plot.optint_by_group.

Value

an object of class "optint_by_group". This object is a list containing two components:

est	a matrix of estimates (in their original units), for each group. here estimates are $E(X|I=1) - E(X|I=0)$, and they are used by plot.optint_by_group.
sd	estimates standard deviation.

Examples

```
# generate data
n <- 50
p <- 3
features <- matrix(rnorm(n*p), ncol = p)
men <- matrix(rbinom(n, 1, 0.5), nrow = n)
outcome <- 2*(features[,1] > 1) + men*pmax(features[,2], 0) + rnorm(n)
outcome <- as.vector(outcome)

#find the optimal intervention using the non-parametric method:
imp_feat <- optint(Y = outcome, X = features, control = men,
                   method = "non-parametric", lambda = 10, plot = TRUE,
                   n.boot = 100, n.perm = 100)

#we can explore how the optimal intervention varies between genders using optint_by_group():
men <- as.vector(men)
imp_feat_by_gender <- optint_by_group(Y = outcome, X = features,
                                      group = men,
                                      method = "non-parametric",
                                      lambda = 10)

#by default, only the significant features are displayed
#(see ?plot.optint_by_group for further details).
#for customized variable importance plot, use plot():
plot(imp_feat_by_gender, plot.vars = 3)
```

`outcome_diff` *Outcome difference*

Description

Calculates the diffrence between E(Y|I=1) and E(Y|I=0)

Usage

```
outcome_diff(Y, wgt1, wgt = rep(1, length(Y)))
```

Arguments

`Y`	outcome vector (must be numeric without NA's).
`wgt1`	weights under I = 1
`wgt`	an optional vector of weights.

Value

outcome difference

par_cor — *Partial Correlation*

Description

Calculates correlation / covariance between Y and X, holding control constant

Usage

```
par_cor(Y, X, control = NULL, wgt = rep(1, length(Y)))
```

Arguments

`Y`	outcome vector (must be numeric without NA's).
`X`	numeric data frame or matrix of factors to be considered.
`control`	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
`wgt`	an optional vector of weights.

Value

data frame with partial correlations, partial covariance & p-values.

perm_test — *Permutation test Test the null hypothesis P(X|I=0) = P(X|I=1), using permutation test.*

Description

Permutation test Test the null hypothesis P(X|I=0) = P(X|I=1), using permutation test.

Usage

```
perm_test(estimates, wgt, wgt1, X, n.quant, n.perm = 1000, Y = NULL,
  control = NULL, func = "non_parm")
```

Arguments

`estimates`	point estimates of the percentile distance between P(X|I=0) & P(X|I=1).
`wgt`	an optional vector of weights.
`wgt1`	weights under I = 1.
`X`	numeric data frame or matrix of factors to be considered.
`n.quant`	number of quantiles to use when calculating CDF distance.

n.perm	number of permutations to permute from wgt1.
Y	outcome vector (must be numeric without NA's).
control	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
func	either "non_parm" or "nn". for "nn", weights are recalculated for each permutation, and thus Y and control are needed. the default is "non_parm", and Y and control aren't needed.

Value

vector of p values.

---

per_distance *Distance Between Distributions*

Description

Calculate distance in RMSE between quantiles of distributions

Usage

```
per_distance(x, n.quant, wgt, wgt1, p = 2/3, plot.sign = F)
```

Arguments

x	variable.
n.quant	number of quantiles.
wgt	original weights.
wgt1	weights under I = 1.
p	proportion of quantiles that need to be increase (decrease) in order to return a positive (negative) sign.
plot.sign	if F returns RMSE, if T returns the sign of effect.

Value

scalar for distance. If sign = TRUE, returns +1 (-1) for increasing (decreasing) p of quantiles. Else returns 0

plot.optint *Plot optint object*

Description

Produce variable importance plot from an optint object.

Usage

```
## S3 method for class 'optint'
plot(x, plot.vars = "sig", plot.ci = T,
  graph.col = 1, alpha = 0.05, ...)
```

Arguments

x	an optint object.
plot.vars	which variables to plot? either a number (n) - indicating to plot the first n variables, "sig" (default) - plot only significant variables, or a vector with names of variables to plot.
plot.ci	logical. if TRUE (default) plot confidence intervals. Otherwise plot only point estimates.
graph.col	graph color/s.
alpha	significance level for the confidence intervals. also used in order to determine which variables are significant.
...	additional arguments.

plot.optint_by_group *Plot optint object, by group*

Description

Produce variables importance plot from an optint_by_group object. This plot has several features:

1. Estimates here are $E(X|I = 1) - E(X|I = 0)$ and not cdf distances.
2. Star is added to variable name if there is a significant difference between at least two groups.
3. Estimates are standardized before they plotted (so different set of variables will have different standardization factor.)

Usage

```
## S3 method for class 'optint_by_group'
plot(x, plot.vars = "sig", graph.col = NULL,
  alpha = 0.05, ...)
```

Arguments

`x`	an optint_by_group object.
`plot.vars`	which variables to plot? either a number (n) - indicating to plot the first n variables, "sig" (default) - plot only significant variables (here significant means that variabe is signifcant for all groups, or that there is significant heterogeneity), or a vector with names of variables to plot.
`graph.col`	graph color/s.
`alpha`	significance level for the confidence intervals. also used in order to determine which variables are significant.
`...`	additional arguments.

`plot_change` *Plot the change in the distribution of X*

Description

Illustrates how the intervention changes the distribution of X by plotting barchart (for categorical variables) or denisty plot (for continouous variables), before and after the intervention.

Usage

```
plot_change(x, plot.vars = "sig", graph.col = c("red", "blue"),
  alpha = 0.05, line.type = c(1, 2), n.val = 10)
```

Arguments

`x`	an optint object.
`plot.vars`	which variables to plot? either a number (n) - indicating to plot the first n variables, "sig" (default) - plot only significant variables, or a vector with names of variables to plot.
`graph.col`	graph color/s.
`alpha`	significance level for the confidence intervals. also used in order to determine which variables are significant.
`line.type`	line type for `densityplot`
`n.val`	variable with more values than 'n.val' will be displayed by density plot, while variable with fewer values will be displayed by barchart.

Examples

```
# generate data
n <- 50
p <- 3
features <- matrix(rnorm(n*p), ncol = p)
men <- matrix(rbinom(n, 1, 0.5), nrow = n)
outcome <- 2*(features[,1] > 1) + men*pmax(features[,2], 0) + rnorm(n)
```

```
outcome <- as.vector(outcome)

#find the optimal intervention using the non-parametric method:
imp_feat <- optint(Y = outcome, X = features, control = men,
                   method = "non-parametric", lambda = 10, plot = TRUE,
                   n.boot = 100, n.perm = 100)

#we can look on the new features distribution more deeply, using plot_change():
plot_change(imp_feat, plot.vars = "sig")
```

---

summary.optint *Summary for optint object*

Description

Report results from an optint object.

Usage

```
## S3 method for class 'optint'
summary(object, r = 4, ...)
```

Arguments

object	an optint object.
r	number of decimal places to use.
...	additional arguments.

---

var_pos *Variable Position*

Description

Find which variables to plot.

Usage

```
var_pos(x, plot.vars = "sig", alpha)
```

Arguments

x	an optint object.
plot.vars	which variables to plot? either a number (n) - indicating to plot the first n variables, "sig" (default) - plot only significant variables, or a vector with names of variables to plot.
alpha	significance level for the confidence intervals. also used in order to determine which variables are significant.

Value

vector of variables incidents

`wgt_adjust` *Weights adjustment*

Description

Adjust new weights so that the distribution of the control variables would not change

Usage

```
wgt_adjust(control, base.wgt1, wgt)
```

Arguments

`control`	numeric data frame or matrix of factors to control for. these are factors that we can't consider while looking for the optimal intervention (e.g. race).
`base.wgt1`	baseline weights under I = 1
`wgt`	an optional vector of weights.

Value

vector of adjusted weights under I = 1

`wtd_bin` *Weighted Bin*

Description

Divide the data into equal size bins and calculate mean

Usage

```
wtd_bin(x, n.quant, wgt)
```

Arguments

`x`	variable.
`n.quant`	number of quantiles.
`wgt`	vector of weigts.

Value

vector of means.

Index