Package: PTE (via r-universe)

October 13, 2026

Type Package

Title Personalized Treatment Evaluator

Version 2.0

Date 2026-07-26

Description We provide inference for personalized medicine models. Namely, we answer the questions: (1) how much better does a purported personalized recommendation engine for treatments do over a business-as-usual approach and (2) is that difference statistically significant?

License GPL-3

URL https://github.com/kapelner/PTE

BugReports https://github.com/kapelner/PTE/issues

Depends R (>= 3.0)

Imports foreach, parallel, doParallel, ggplot2, stats, survival, Rcpp, checkmate

LinkingTo Rcpp, RcppEigen, RcppNumerical

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), mirai

Config/testthat/edition 3

NeedsCompilation yes

Author Adam Kapelner [aut, cre], Alina Levine [aut], Justin Bleich [aut]

Maintainer Adam Kapelner <kapelner@qc.cuny.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-27 08:50:31 UTC

RemoteUrl https://github.com/cran/PTE

RemoteRef HEAD

RemoteSha df28b99683020cd69089d6198f03f824408be06f

Contents

continuous_example 2
plot.PTE_bootstrap_results 2
print.PTE_bootstrap_results 3
PTE 4
PTE_bootstrap_inference 4
select_optimal_m_prop 9
summary.PTE_bootstrap_results 12
survival_example 13

Index **14**

`continuous_example` *Mock RCT data with a continuous endpoint*

Description

A list with two objects (a) `X`, a dataframe with n rows representing clinical subjects and columns: treatment, x1, x2, x3, x4 and x5 where treatment is binary indicating the two arms of the clinical trial and x1, ..., x5 are covariates that were collected about each subject and (b) `y`, a length n vector storing the continuous response values where, in this mock dataset, larger values indicate "better" outcomes for the subjects.

Author(s)

My Name <kapelner@qc.cuny.edu>

`plot.PTE_bootstrap_results`
Plots a summary of the bootstrap samples

Description

Plots a summary of the bootstrap samples

Usage

```
## S3 method for class 'PTE_bootstrap_results'
plot(x, ...)
```

Arguments

`x`	A `PTE_bootstrap_results` model object built via running the `PTE_bootstrap_inference` function.
`...`	Other methods passed to plot

Author(s)

Adam Kapelner

Examples

```
library(PTE)
data(continuous_example)
pte_results = PTE_bootstrap_inference(continuous_example$X, continuous_example$y,
    regression_type = "continuous", B = 5, num_cores = 1)
plot(pte_results)
```

---

print.PTE_bootstrap_results

Prints a summary of the model to the console

Description

Prints a summary of the model to the console

Usage

```
## S3 method for class 'PTE_bootstrap_results'
print(x, ...)
```

Arguments

x	A PTE_bootstrap_results model object built via running the PTE_bootstrap_inference function.
...	Other methods passed to print

Author(s)

Adam Kapelner

Examples

```
library(PTE)
data(continuous_example)
pte_results = PTE_bootstrap_inference(continuous_example$X, continuous_example$y,
    regression_type = "continuous", B = 5, num_cores = 1)
print(pte_results)
```

PTE *Personalized Medicine Inference*

Description

Personalized Medicine...

Author(s)

Adam Kapelner <kapelner@qc.cuny.edu>, Alina Levine and Justin Bleich

References

Kapelner, A, Bleich, J, Cohen, ZD, DeRubeis, RJ and Berk, R (2014) Inference for Treatment Regime Models in Personalized Medicine, arXiv

See Also

Useful links:

- https://github.com/kapelner/PTE
- Report bugs at https://github.com/kapelner/PTE/issues

PTE_bootstrap_inference

Bootstrap inference for a prespecified personalization / recommendation model

Description

Runs B bootstrap samples using a prespecified model then computes the two I estimates based on cross validation. p values of the two I estimates are computed for a given $H_0 : \mu_{I_0} = \mu_0$ and confidence intervals are provided using the basic, percentile methods by default and the BCa method as well if desired.

Usage

```
PTE_bootstrap_inference(
  X,
  y,
  regression_type = "continuous",
  incidence_metric = "odds_ratio",
  personalized_model_build_function = NULL,
  censored = NULL,
  predict_function = NULL,
  difference_function = NULL,
```

```
  cleanup_mod_function = NULL,
  y_higher_is_better = TRUE,
  verbose = FALSE,
  full_verbose = FALSE,
  H_0_mu_equals = NULL,
  pct_leave_out = 0.1,
  m_pow_of_n = 0.75,
  B = 3000,
  alpha = 0.05,
  run_bca_bootstrap = FALSE,
  display_adversarial_score = FALSE,
  num_cores = NULL
)
```

Arguments

X	A $n \times p$ dataframe of covariates where one column is labeled "treatment" and it is a binary vector of treatment allocations in the study.
y	An n-length numeric vector which is the response
regression_type	A string indicating the regression problem. Legal values are "continous" (the response y is a real number with no missing data, the default), "incidence" (the reponse y is either 0 or 1) and "survival". If the type is "survival", the user must also supply additional data via the parameter censored.
incidence_metric	Ignored unless the regression_type is "incidence" and difference_function is set to NULL (in the latter case, you have specified a more custom metric). Then, this parameter allows the user to select which of the three standard metrics to use for comparison: "probability_difference", "risk_ratio", "odds_ratio" where the default is "odds_ratio".
personalized_model_build_function	An R function that will be evaluated to construct the personalized medicine / recommendation model. In the formula for the model, the response is "y", the treatment vector is "treatment" and the data is "Xytrain". This function must return some type of object that can be used for prediction later via predict_function. Here are the defaults for each regression_type. They are linear models with first order interactions:

```
personalized_model_build_function = switch(regression_type,
  continuous = function(Xytrain) { #defalt is OLS regression
                 lm(y ~ . * treatment,
                      data = Xytrain)
               },
  incidence = function(Xytrain) { #default is logistic regress
                glm(y ~ . * treatment,
                      data = Xytrain,
                   family = "binomial")
               },
```

```
survival = function(Xytrain) { #default is Weibull regressio
  survreg(Surv(Xytrain$y, Xytrain$censored) ~ (. - censore
                  data = Xytrain,
                 dist = "weibull")
      }
  )
```

`censored` Only required if the `regression_type` is "survival". In this case, this vector is of length n and is binary where 0 indicates censored and 1 indicates uncensored. In a clinical trial, someone who is still alive at the end of the study or was lost to follow up will receive a censor value of 0, while someone who died during the study will receive a censor value of 1. n and is binary where 0 indicates censorship (e.g. the patient died).

`predict_function`

An R function that will be evaluated on left out data after the model is built with the training data. This function uses the object "mod" that is the result of the `personalized_model_build_function` and it must make use of "Xyleftout", a subset of observations from `X`. This function must return a scalar numeric quantity for comparison. The default function is `predict(mod, obs_left_out)` e.g. the default looks like:

```
function(mod, Xyleftout) {
    predict(mod, Xyleftout)
}
```

`difference_function`

A function which takes the result of one out of sample experiment (boostrap or not) of all n samples and converts it into a difference that will be used as a score in a score distribution to determine if the personalization model is statistically significantly able to distinguish subjects. The function looks as follows:

```
function(results, indices_1_1, indices_0_0, indices_0_1, indi
                  ...
  c(rec_vs_non_rec_diff_score, rec_vs_all_score, rec_vs_best
          }
```

where `results` is a matrix consisting of columns of the estimated response of the treatment administered, the estimated response of the counterfactual treatment, the administered treatment, the recommended treatment based on the personalization model, the real response, and if this subject was censored (0 if so). Here are a couple of example entries:

est_true est_counterfactual given_tx rec_tx real_y censored 166.8 152.2 1 1 324 1 1679.1 2072.0 1 0 160 0

The arguments `indices_1_1`, `indices_0_0`, `indices_0_1`, `indices_1_0` give the indices of the subjects whose treatment was administered as 1 and whose optimal was 1, whose treatment was administered 0 and whose optimal was 0, etc.

This function should return three numeric scores: the recommend vs. the non-recommended (adversarial), the recommended vs. all (all) and the recommended

vs. the best average treatment (best) as a 3-dimensional vector as illustrated above.

By default, this parameter is NULL which means for continuous and incidence the average difference is used and for survival, the median Kaplan-Meier survival is used.

`cleanup_mod_function`
: A function that is called at the end of a cross validation iteration to cleanup the model in some way. This is used for instance if you would like to release the memory your model is using but generally does not apply. The default is NA for "no function."

`y_higher_is_better`
: True if a response value being higher is clinically "better" than one that is lower (e.g. cognitive ability in a drug trial for the mentally ill). False if the response value being lower is clinically "better" than one that is higher (e.g. amount of weight lost in a weight-loss trial). Default is TRUE.

`verbose`
: Prints out a dot for each bootstrap sample. This only works on some platforms.

`full_verbose`
: Prints out full information for each cross validation model for each bootstrap sample. This only works on some platforms.

`H_0_mu_equals`
: The μ_{I_0} value in H_0. Default is NULL which specifies 0 for regression types continuous, survival and incidence (with incidence metric "probability_difference") or 1 if the regression type is incidence and the incidence metric is "risk_ratio" or "odds_ratio". These defaults essentially answer the question: does my allocation procedure do better than the business-as-usual / naive allocation procedure?

`pct_leave_out`
: In the cross-validation, the proportion of the original dataset left out to estimate out-of-sample metrics. The default is 0.1 which corresponds to 10-fold cross validation.

`m_pow_of_n`
: Within each bootstrap sample, the exponent γ used to set the resample size (sampled with replacement) to $m = \lfloor n^\gamma \rfloor$, implementing the "m-out-of-n bootstrap." Politis, Romano & Wolf (*Subsampling*, 1999) show that bootstrap consistency for non-smooth (non-regular) functionals – as arises here because the treatment recommendation is a hard threshold on the estimated treatment effect – requires $m \to \infty$ and $m/n \to 0$, i.e. $\gamma < 1$; $\gamma = 1$ recovers the classical (and here potentially inconsistent) n-out-of-n bootstrap. The default $\gamma = 0.75$ follows the Politis-Romano-Wolf recommended rate as a practical compromise between bias (γ too close to 1, reintroducing the non-regularity) and excess bootstrap Monte Carlo variance (γ too small); see their book for the data-driven "minimum volatility" method for calibrating γ to a specific dataset. When $\gamma < 1$, the reported confidence intervals are rescaled by $\sqrt{m/n}$ as required by m-out-of-n bootstrap theory; this rescaling is a no-op when $\gamma = 1$ (i.e. $m = n$).

`B`
: The number of bootstrap samples to take. We recommend making this as high as you can tolerate given speed considerations. The default is 3000.

`alpha`
: Defines the confidence interval size (1 - alpha). Defaults to 0.05.

`run_bca_bootstrap`
: Do the BCA bootstrap as well. This takes double the time. It defaults to FALSE.

`display_adversarial_score`
: The adversarial score records the personalization metric versus the deliberate opposite of the personalization. This does not correspond to any practical situation but it is useful for debugging. Default is `FALSE`.

`num_cores`
: The number of cores to use in parallel to run the bootstrap samples more rapidly. Defaults to `NULL` which uses `max(cores - 1, 1)` where `cores` is detected once when the PTE package is loaded and cached in `options(mc.cores = ...)` (see `?options`). Pass an explicit value here to override the cached default for a single call.

Value

A results object of type "PTE_bootstrap_results" that contains much information about the observed results and the bootstrap runs, including hypothesis testing and confidence intervals.

Author(s)

Adam Kapelner

Examples

```
library(PTE)

##B and num_cores are kept small here so the examples run quickly; in practice make B as
##large as you can tolerate and omit num_cores to use all-but-one core.

##response: continuous
data(continuous_example)
X = continuous_example$X
y = continuous_example$y
pte_results = PTE_bootstrap_inference(X, y, regression_type = "continuous", B = 5, num_cores = 1)
pte_results

##response: incidence. incidence_metric also accepts "risk_ratio" and "probability_difference"
##(see the incidence_metric parameter above); odds_ratio is illustrated here for speed.
##force incidence and pretend y came to you this way
y_incidence = ifelse(y > quantile(y, 0.75), 1, 0)
pte_results = PTE_bootstrap_inference(X, y_incidence,
    regression_type = "incidence",
    B = 5, num_cores = 1)
pte_results

##response: survival
data(survival_example)
pte_results = PTE_bootstrap_inference(survival_example$X, survival_example$y,
    censored = survival_example$censored,
    regression_type = "survival",
    B = 5, num_cores = 1)
pte_results
```

`select_optimal_m_prop` *Select an optimal m-out-of-n bootstrap resample size via the minimum volatility method*

Description

`PTE_bootstrap_inference`'s `m_pow_of_n` argument sets each bootstrap replicate's resample size to $m = \lfloor n^{\gamma} \rfloor$, which is required to be less than n (i.e. $\gamma < 1$) for the bootstrap to be consistent for our non-smooth policy-value estimands (see `PTE_bootstrap_inference`). There is, however, no single population-optimal γ – asymptotic theory only requires $m \to \infty$ and $m/n \to 0$. Politis, Romano & Wolf (*Subsampling*, 1999, Ch. 9) and Bickel & Sakov (2008) address this by proposing the "minimum volatility" (MV) method: compute the bootstrap answer (here, the width of the rate-corrected percentile confidence interval for one of the three PTE estimands) over a grid of candidate γ (equivalently m) values, then, for each grid point, measure how much that answer varies over a small centered neighborhood of adjacent grid points ("volatility"). The chosen γ is the one whose neighborhood is most stable, i.e. the region of the grid where the bootstrap's answer has stopped changing in response to further tuning of m – this is the practical, data-driven proxy for "m large enough for the CLT-type approximation underlying the bootstrap to have kicked in, but still small enough relative to n for consistency."

Usage

```
select_optimal_m_prop(
  X,
  y,
  regression_type = "continuous",
  incidence_metric = "odds_ratio",
  personalized_model_build_function = NULL,
  censored = NULL,
  predict_function = NULL,
  difference_function = NULL,
  cleanup_mod_function = NULL,
  y_higher_is_better = TRUE,
  pct_leave_out = 0.1,
  m_pow_of_n_grid = seq(0.5, 1, by = 0.05),
  estimand = "average",
  B = 250,
  alpha = 0.05,
  volatility_window = 3,
  num_cores = NULL
)
```

Arguments

`X`	A $n \times p$ dataframe of covariates where one column is labeled "treatment" and it is a binary vector of treatment allocations in the study. See `PTE_bootstrap_inference`.
`y`	An n-length numeric vector which is the response. See `PTE_bootstrap_inference`.

`regression_type`
: See `PTE_bootstrap_inference`. Default `"continuous"`.

`incidence_metric`
: See `PTE_bootstrap_inference`. Default `"odds_ratio"`.

`personalized_model_build_function`
: See `PTE_bootstrap_inference`. Default `NULL` (use the built-in default for `regression_type`).

`censored`
: See `PTE_bootstrap_inference`. Required if `regression_type` is `"survival"`.

`predict_function`
: See `PTE_bootstrap_inference`. Default `NULL` (use the built-in default).

`difference_function`
: See `PTE_bootstrap_inference`. Default `NULL` (use the built-in default for `regression_type`).

`cleanup_mod_function`
: See `PTE_bootstrap_inference`. Default `NULL` (no cleanup).

`y_higher_is_better`
: See `PTE_bootstrap_inference`. Default `TRUE`.

`pct_leave_out`
: See `PTE_bootstrap_inference`. Default `0.10`.

`m_pow_of_n_grid`
: The grid of candidate γ exponents to evaluate (each entry is a candidate value for `PTE_bootstrap_inference`'s `m_pow_of_n` argument). Must have at least `volatility_window` entries so every interior grid point has a complete neighborhood. The default, `seq(0.5, 1, by = 0.05)`, brackets the Politis-Romano-Wolf-recommended $\gamma = 0.75$ on both sides, up to (but not below) $\gamma = 0.5$ ($m = \sqrt{n}$, a common lower-rate benchmark) and $\gamma = 1$ (the classical, here potentially inconsistent, n-out-of-n bootstrap, included only as a reference point – see `volatility_window`).

`estimand`
: Which of the three PTE estimands' confidence interval width to stabilize: `"adversarial"`, `"average"` (the default) or `"best"`; see `PTE_bootstrap_inference`.

`B`
: The number of bootstrap samples to take *at each grid point*. Since the total cost is `length(m_pow_of_n_grid) * B` bootstrap replicates, this defaults to a much smaller `250` than `PTE_bootstrap_inference`'s own default of `3000` – this function is for calibrating `m_pow_of_n`, not for the final reported inference.

`alpha`
: Confidence interval size (1 - alpha) used to measure CI width at each grid point. Defaults to 0.05.

`volatility_window`
: The number of adjacent grid points (must be odd and ≥ 3) averaged into the volatility measure at each interior grid point. Only grid points with a complete, centered window are eligible to be selected (this excludes the `(volatility_window - 1) / 2` grid points at each end of `m_pow_of_n_grid` from selection, since their neighborhoods would otherwise be one-sided and their apparent volatility artificially low). Default `3`.

`num_cores`
: See `PTE_bootstrap_inference`. Passed through unchanged to each grid point's call (grid points are evaluated sequentially, not in additional parallel, to avoid nested parallelism on top of `PTE_bootstrap_inference`'s own).

Details

This function is a calibration step: it runs PTE_bootstrap_inference once per grid point (so its total cost is roughly length(m_pow_of_n_grid) times that of a single call) and does not itself return final inference – rerun PTE_bootstrap_inference at the selected m_pow_of_n_optimal (with a larger B) to get the confidence intervals and p-values you report.

Value

A list with class "PTE_optimal_m_selection" containing:

m_pow_of_n_optimal The selected γ, ready to pass as m_pow_of_n to PTE_bootstrap_inference.

m_optimal The corresponding resample size, round(n^m_pow_of_n_optimal).

grid_table A data.frame with one row per grid point: m_pow_of_n, m, the point estimate and ci_width for estimand, and its volatility (NA for the non-eligible boundary points described above).

estimand, B, alpha, volatility_window Echoed back for reference.

Author(s)

Adam Kapelner

References

Politis, D.N., Romano, J.P. and Wolf, M. (1999) *Subsampling*. Springer Series in Statistics.

Bickel, P.J. and Sakov, A. (2008) On the choice of m in the m out of n bootstrap and confidence bounds for extrema. *Statistica Sinica*, 18(3), 967-985.

Examples

```
## Not run:
 library(PTE)
 data(continuous_example)
 X = continuous_example$X
 y = continuous_example$y

 m_selection = select_optimal_m_prop(
  X, y,
  regression_type = "continuous",
  m_pow_of_n_grid = seq(0.5, 1, by = 0.05),
  B = 250,
  num_cores = 1
 )
 m_selection$grid_table
 m_selection$m_pow_of_n_optimal

 #now rerun inference at the selected m_pow_of_n with a production-sized B
 pte_results = PTE_bootstrap_inference(
  X, y,
  regression_type = "continuous",
  m_pow_of_n = m_selection$m_pow_of_n_optimal,
```

```
 B = 3000
)
pte_results

## End(Not run)
```

---

summary.PTE_bootstrap_results *Prints a summary of the model to the console*

Description

Prints a summary of the model to the console

Usage

```
## S3 method for class 'PTE_bootstrap_results'
summary(object, ...)
```

Arguments

object	A PTE_bootstrap_results model object built via running the PTE_bootstrap_inference function.
...	Other methods passed to summary

Author(s)

Adam Kapelner

Examples

```
library(PTE)
data(continuous_example)
pte_results = PTE_bootstrap_inference(continuous_example$X, continuous_example$y,
    regression_type = "continuous", B = 5, num_cores = 1)
summary(pte_results)
```

`survival_example` *Mock RCT data with a survival endpoint*

Description

A list with three objects (a) `X`, a dataframe with n rows representing clinical subjects and columns: treatment, x1, x2, x3 and x4 where treatment is binary indicating the two arms of the clinical trial and x1, ..., x4 are covariates that were collected about each subject (b) `y`, a length n vector storing the survival response values (a time measurement) where, in this mock dataset, smaller values indicate "better" survival outcomes for the subjects and (c) `censored`, a length n vector storing the censor dummies where c_16 = 1 means the response y_16 was censored and thus the truth value of y_16 is unknown and y_16 only represents the moment it was censored (and c_16 = 0 means it was uncensored and y_16 is the true response value).

Author(s)

My Name <kapelner@qc.cuny.edu>

Index

∗ **bootstrap**
PTE, 4
∗ **medicine**
PTE, 4
∗ **personalized**
PTE, 4

continuous_example, 2

plot.PTE_bootstrap_results, 2
print.PTE_bootstrap_results, 3
PTE, 4
PTE-package *(*PTE*)*, 4
PTE_bootstrap_inference, 4, *9–11*

select_optimal_m_prop, 9
summary.PTE_bootstrap_results, 12
survival_example, 13