Package: CRE (via r-universe)

October 20, 2024

Type Package

Title Interpretable Discovery and Inference of Heterogeneous Treatment Effects

Version 0.2.7

Maintainer Falco Joannes Bargagli Stoffi <fbargaglistoffi@hsph.harvard.edu>

Description Provides a new method for interpretable heterogeneous treatment effects characterization in terms of decision rules via an extensive exploration of heterogeneity patterns by an ensemble-of-trees approach, enforcing high stability in the discovery. It relies on a two-stage pseudo-outcome regression, and it is supported by theoretical convergence guarantees. Bargagli-Stoffi, F. J., Cadei, R., Lee, K., & Dominici, F. (2023) Causal rule ensemble: Interpretable Discovery and Inference of Heterogeneous Treatment Effects. arXiv preprint <doi:10.48550/arXiv.2009.09036>.

License GPL-3

URL `https://github.com/NSAPH-Software/CRE`

BugReports `https://github.com/NSAPH-Software/CRE/issues`

Depends R (>= 3.5.0)

Imports MASS, stats, logger, gbm, randomForest, methods, xgboost, RRF, data.table, xtable, glmnet, bartCause, stabs, stringr, SuperLearner, magrittr, ggplot2, arules

Suggests grf, BART, gnm, covr, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Copyright Harvard University

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Repository https://nsaph-software.r-universe.dev

RemoteUrl https://github.com/nsaph-software/cre

RemoteRef HEAD

RemoteSha ee2821c89f4896c672f8e528cc0c6d2b67563859

Contents

`CRE-package` *The CRE package*

Description

In health and social sciences, it is critically important to identify subgroups of the study population where a treatment has notable heterogeneity in the causal effects with respect to the average treatment effect. Data-driven discovery of heterogeneous treatment effects (HTE) via decision tree methods has been proposed for this task. Despite its high interpretability, the single-tree discovery of HTE tends to be highly unstable and to find an oversimplified representation of treatment heterogeneity. To accommodate these shortcomings, we propose Causal Rule Ensemble (CRE), a new method to discover heterogeneous subgroups through an ensemble-of-trees approach. CRE has the following features:

1. provides an interpretable representation of the HTE; 2) allows extensive exploration of complex heterogeneity patterns; and 3) guarantees high stability in the discovery. The discovered subgroups are defined in terms of interpretable decision rules, and we develop a general two-stage approach for subgroup-specific conditional causal effects estimation, providing theoretical guarantees.

Author(s)

Naeem Khoshnevis

Daniela Maria Garcia

Riccardo Cadei

Kwonsang Lee

Falco Joannes Bargagli Stoffi

References

Bargagli-Stoffi, F. J., Cadei, R., Lee, K. and Dominici, F. (2023). Causal rule ensemble: Interpretable Discovery and Inference of Heterogeneous Treatment Effects,arXiv preprint arXiv:2009.09036

See Also

Useful links:

- `https://github.com/NSAPH-Software/CRE`
- Report bugs at `https://github.com/NSAPH-Software/CRE/issues`

`cre` *Causal rule ensemble*

Description

Performs the Causal Rule Ensemble on a data set with a response variable, a treatment variable, and various features.

Usage

```
cre(y, z, X, method_params = NULL, hyper_params = NULL, ite = NULL)
```

Arguments

`y`	An observed response vector.
`z`	A treatment vector.
`X`	A covariate matrix (or a data frame). Should be provided as numerical values.
`method_params`	The list of parameters to define the models used, including:

- *Parameters for Honest Splitting*
  - *ratio_dis*: The ratio of data delegated to rules discovery (default: 0.5).
- *Parameters for Discovery and Inference*
  - *ite_method*: The method for ITE (pseudo-outcome) estimation (default: `"aipw"`, options: `"aipw"` for Augmented Inverse Probability Weighting, `"cf"` for Causal Forest, `"bart"` for Causal Bayesian Additive Regression Trees, `"slearner"` for S-Learner, `"tlearner"` for T-Learner, `"xlearner"` for X-Learner, `"tpoisson"` for T-Learner with Poisson regression).
  - *learner_ps*: The model for the propensity score estimation (default: `"SL.xgboost"`, options: any SuperLearner prediction model i.e., `"SL.lm"`, `"SL.svm"`, used only for `"aipw"`, `"bart"`, `"cf"` ITE estimators).
  - *learner_y*: The model for the outcome estimation (default: `"SL.xgboost"`, options: any SuperLearner prediction model i.e., `"SL.lm"`, `"SL.svm"`, used only for `"aipw"`, `"slearner"`, `"tlearner"` and `"xlearner"` ITE estimators).

`hyper_params`	The list of hyper parameters to fine-tune the method, including:

- *General hyper parameters*
  - *intervention_vars*: Array with intervention-able covariates names used for Rules Generation. Empty or null array means that all the covariates are considered as intervention-able (default: `NULL`).
  - *ntrees*: The number of decision trees for random forest (default: 20).
  - *node_size*: Minimum size of the trees' terminal nodes (default: 20).
  - *max_rules*: Maximum number of generated candidates rules (default: 50).
  - *max_depth*: Maximum rules length (default: 3).
  - *t_decay*: The decay threshold for rules pruning. Higher values will carry out an aggressive pruning (default: 0.025).
  - *t_ext*: The threshold to truncate too generic or too specific (extreme) rules (default: 0.01, range: [0, 0.5)).
  - *t_corr*: The threshold to define correlated rules (default: 1, range: `[0,+inf)`).
  - *stability_selection*: Method for stability selection for selecting the rules. `"vanilla"` for stability selection, `"error_control"` for stability selection with error control and `"no"` for no stability selection (default: `"vanilla"`).
  - *B*: Number of bootstrap samples for stability selection in rules selection and uncertainty quantification in estimation (default: 20).
  - *subsample*: Bootstrap ratio subsample for stability selection in rules selection and uncertainty quantification in estimation (default: 0.5).
- *Method specific hyper parameters*
  - *offset*: Name of the covariate to use as offset (i.e., `"x1"`) for T-Poisson ITE estimation. Use `NULL` if offset is not used (default: `NULL`).
  - *cutoff*: Threshold (percentage) defining the minimum cutoff value for the stability scores for Stability Selection (default: 0.9).
  - *pfer*: Upper bound for the per-family error rate (tolerated amount of falsely selected rules) for Error Control Stability Selection (default: 1).

`ite`	The estimated ITE vector. If given both the ITE estimation steps in Discovery and Inference are skipped (default: `NULL`).

Value

An S3 object composed by:

`M`	the number of Decision Rules extracted at each step,
`CATE`	the data.frame of Conditional Average Treatment Effect decomposition estimates with corresponding uncertainty quantification,
`method_params`	the list of method parameters,
`hyper_params`	the list of hyper parameters,
`rules`	the list of rules (implicit form) decomposing the CATE.

Note

- If `intervention_vars` are provided, it is important to note that the individual treatment effect will still be computed using all covariates.

Examples

```
set.seed(123)
dataset <- generate_cre_dataset(n = 400,
                                rho = 0,
                                n_rules = 2,
                                p = 10,
                                effect_size = 2,
                                binary_covariates = TRUE,
                                binary_outcome = FALSE,
                                confounding = "no")
y <- dataset[["y"]]
z <- dataset[["z"]]
X <- dataset[["X"]]

method_params <- list(ratio_dis = 0.5,
                      ite_method ="aipw",
                      learner_ps = "SL.xgboost",
                      learner_y = "SL.xgboost")

hyper_params <- list(intervention_vars = NULL,
                     offset = NULL,
                     ntrees = 20,
                     node_size = 20,
                     max_rules = 50,
                     max_depth = 3,
                     t_decay = 0.025,
                     t_ext = 0.025,
                     t_corr = 1,
                     stability_selection = "vanilla",
                     cutoff = 0.6,
                     pfer = 1,
                     B = 20,
                     subsample = 0.5)

cre_results <- cre(y, z, X, method_params, hyper_params)
```

`generate_cre_dataset` *Generate CRE synthetic data*

Description

Generates synthetic data sets to run simulation for causal inference experiments composed by an outcome vector (`y`), a treatment vector (`z`), a covariates matrix (`X`), and an unobserved individual treatment effects vector (`ite`). The arguments specify the data set characteristic, including the number of individuals (`n`), the number of covariates (`p`), the correlation within the covariates (`rho`), the number of decision rules (`n_rules`) decomposing the Conditional Average Treatment Effect (CATE), the treatment effect magnitude (`effect_size`), the confounding mechanism (`confounding`), and whether the covariates and outcomes are binary or continuous (`binary_covariates`, `binary_outcome`).

Usage

```
generate_cre_dataset(
  n = 1000,
  rho = 0,
  n_rules = 2,
  p = 10,
  effect_size = 2,
  binary_covariates = TRUE,
  binary_outcome = TRUE,
  confounding = "no"
)
```

Arguments

`n`	An integer number that represents the number of observations. Non-integer values will be converted into an integer number.
`rho`	A positive double number that represents the correlation within the covariates (default: 0, range: [0,1)).
`n_rules`	The number of causal rules (default: 2, range: {1,2,3,4}).
`p`	The number of covariates (default: 10).
`effect_size`	The treatment effect size magnitude (default: 2, range: $\geq$0).
`binary_covariates`	Whether to use binary or continuous covariates (default: `TRUE`).
`binary_outcome`	Whether to use binary or continuous outcomes (default: `TRUE`).
`confounding`	Only for continuous outcome, add confounding variables: • `"lin"` for linear confounding, • `"nonlin"` for non-linear confounding, • `"no"` for no confounding (default).

Details

The covariates matrix is generated with the specified correlation among individuals, and each covariate is sampled either from a `Bernoulli(0.5)` if binary, or a `Gaussian(0,1)` if continuous. The treatment vector is sampled from a `Bernoulli(`$\frac{1}{1+\exp(1-x_1+x_2-x_3)}$`)`, enforcing the treatment assignment probabilities to be a function of observed covariates. The potential outcomes ($y(0)$ and

$y(1)$) are then sampled from a Bernoulli if binary, or a Gaussian (with standard deviation equal to 1) if continuous. Their mean is equal to a confounding term (null, linear or non-linear and always null for binary outcome) plus 1-4 decision rules weighted by the treatment effect magnitude. The two potential outcomes characterizes the CATE (and then the unobserved individual treatment effects vector) as the sum of different additive contributions for each decision rules considered (plus an intercept). The final expression of the CATE depends on the treatment effect magnitude and the number of decision rules considered.

The 4 decision rules are:

- Rule 1: $1\{x_1 > 0.5; x_2 \leq 0.5\}(\mathbf{x})$
- Rule 2: $1\{x_5 > 0.5; x_6 \leq 0.5\}(\mathbf{x})$
- Rule 3: $1\{x_4 \leq 0.5\}(\mathbf{x})$
- Rule 4: $1\{x_5 \leq 0.5; x_7 > 0.5; x_8 \leq 0.5\}(\mathbf{x})$ with corresponding additive average treatment effect (AATE) equal to:
- Rule 1: $-$ `effect_size`,
- Rule 2: $+$ `effect_size`,
- Rule 3: $-0.5\cdot$ `effect_size`,
- Rule 4: $+2\cdot$ `effect_size`.

In example, setting `effect_size`=4 and `n_rules`=2:

$$\text{CATE}(\mathbf{x}) = -4 \cdot 1\{x_1 > 0.5; x_2 \leq 0.5\}(\mathbf{x}) + 4 \cdot 1\{x_5 > 0.5; x_6 \leq 0.5\}(\mathbf{x})$$

The final outcome vector `y` is finally computed by combining the potential outcomes according to the treatment assignment.

Value

A list, representing the generated synthetic data set, containing:

`y`	an outcome vector,
`z`	a treatment vector,
`X`	a covariates matrix,
`ite`	an individual treatment vector.

Note

Set the covariates domain (`binary_covariates`) and outcome domain (`binary_outcome`) according to the experiment of interest. Increase complexity in heterogeneity discovery:

- decreasing the sample size (`n`),
- adding correlation among covariates (`rho`),
- increasing the number of rules (`n_rules`),
- increasing the number of covariates (`p`),
- decreasing the absolute value of the causal effect (`effect_size`),
- adding linear or not-linear confounders (`confounding`).

Examples

```
set.seed(123)
dataset <- generate_cre_dataset(n = 1000, rho = 0, n_rules = 2, p = 10,
                                effect_size = 2, binary_covariates = TRUE,
                                binary_outcome = TRUE, confounding = "no")
```

`get_logger` *Get Logger settings*

Description

Returns current logger settings.

Usage

```
get_logger()
```

Value

Returns a list that includes **logger_file_path** and **logger_level**.

See Also

`set_logger` for information on setting the log level and file path.

Examples

```
set_logger("mylogger.log", "INFO")
log_meta <- get_logger()
```

`plot.cre` *Extend generic plot functions for cre class*

Description

A wrapper function to extend generic plot functions for cre class.

Usage

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

Arguments

`x`	A `cre` object.
`...`	Additional arguments passed to customize the plot.

Value

Returns a ggplot2 object, invisibly. This function is called for side effects.

`predict.cre` *Predict individual treatment effect via causal rule ensemble*

Description

Predicts individual treatment effect via causal rule ensemble algorithm.

Usage

```
## S3 method for class 'cre'
predict(object, X, ...)
```

Arguments

`object`	A `cre` object from running the CRE function.
`X`	A covariate matrix (or data.frame)
`...`	Additional arguments passed to customize the prediction.

Value

An array with the estimated Individual Treatment Effects

`print.cre` *Extend print function for the CRE object*

Description

Prints a brief summary of the CRE object

Usage

```
## S3 method for class 'cre'
print(x, verbose = 2, ...)
```

Arguments

`x`	A cre object from running the CRE function.
`verbose`	Set level of results description details: 0 for only results summary, 1 for results and parameters summary, 2 for results and parameters and rules summary (default 2).
`...`	Additional arguments passed to customize the results description.

Value

No return value. This function is called for side effects.

`set_logger` *Set Logger settings*

Description

Updates logger settings, including log level and location of the file.

Usage

```
set_logger(logger_file_path = "CRE.log", logger_level = "INFO")
```

Arguments

`logger_file_path`
A path (including file name) to log the messages. (Default: CRE.log)

`logger_level`
The log level. When a log level is set, all log levels below it are also activated (if implemented). Available levels include:

- TRACE: Provides verbose detailed logging, including the steps taken to achieve a result, often used for debugging. Activating TRACE will also enable DEBUG, INFO, SUCCESS, WARN, ERROR, and FATAL logs.
- DEBUG: Provides detailed logging about the flow of the application, used mostly by developers to understand potential issues. Activating DEBUG will also enable INFO, SUCCESS, WARN, ERROR, and FATAL logs.
- INFO (Default): Standard messages that inform the user about the normal operation of the system. Activating INFO will also enable SUCCESS, WARN, ERROR, and FATAL logs.
- SUCCESS: Messages indicating successful completion of a particular operation or task. Activating SUCCESS will also enable WARN, ERROR, and FATAL logs.
- WARN: Warning messages about events that might cause problems in the future, but are not yet errors. Activating WARN will also enable ERROR and FATAL logs.
- ERROR: Reports an error due to which the system may not be able to achieve its functionality, but the application won't halt. Activating ERROR will also enable FATAL logs.

- FATAL: Reports very severe error events that will presumably lead the application to abort.

Value

No return value. This function is called for side effects.

Note

Log levels are specified by developers during the initial implementation. Future developers or contributors can leverage these log levels to better capture and document the application's processes and events.

Examples

```
set_logger("Debug")
```

`summary.cre` *Print summary of CRE object*

Description

Prints a brief summary of the CRE object

Usage

```
## S3 method for class 'cre'
summary(object, verbose = 2, ...)
```

Arguments

`object`	A cre object from running the CRE function.
`verbose`	Set level of results description details: only results summary 0, results+parameters summary 1, results+parameters+rules summary (default 2).
`...`	Additional arguments passed to customize the results description.

Value

A summary of the CRE object

Index