Package 'policytree'

March 10, 2021

Title Policy Learning via Doubly Robust Empirical Welfare Maximization over Trees

Version 1.0.4

Description Learn optimal policies via doubly robust empirical welfare maximization over trees. This package implements the multi-action doubly robust approach of Zhou, Athey and Wager (2018) <arXiv:1810.04778> in the case where we want to learn policies that belong to the class of depth k decision trees.

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

Suggests testthat (>= 2.1.0), DiagrammeR

RoxygenNote 7.1.1

LinkingTo Rcpp, BH

Imports Rcpp, grf (>= 1.1.0)

URL https://github.com/grf-labs/policytree

BugReports https://github.com/grf-labs/policytree/issues

NeedsCompilation yes

Author Erik Sverdrup [aut, cre],
Ayush Kanodia [aut],
Zhengyuan Zhou [aut],
Susan Athey [aut],
Stefan Wager [aut]

Maintainer Erik Sverdrup <erikcs@stanford.edu>

Repository CRAN

Date/Publication 2021-03-10 10:30:02 UTC

R topics documented:

`conditional_means.causal_forest`

Estimate mean rewards μ for each treatment a

Description

$\mu_a = m(x) + (1 - e_a(x))\tau_a(x)$

Usage

```
## S3 method for class 'causal_forest'
conditional_means(object, ...)

## S3 method for class 'instrumental_forest'
conditional_means(object, ...)

## S3 method for class 'multi_causal_forest'
conditional_means(object, ...)

conditional_means(object, ...)
```

Arguments

`object`	An appropriate causal forest type object
`...`	Additional arguments

Value

A matrix of estimated mean rewards

Methods (by class)

- causal_forest: Mean rewards μ for control/treated
- instrumental_forest: Mean rewards μ for control/treated
- multi_causal_forest: Mean rewards μ for each treatment a

Examples

```
# Compute conditional means for a multi_causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- sample(c("A", "B", "C"), n, replace = TRUE)
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
forests <- multi_causal_forest(X = X, Y = Y, W = W)
mu.hats <- conditional_means(forests)
head(mu.hats)

# Compute conditional means for a causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
Y <- pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
mu.hats <- conditional_means(c.forest)
```

---

double_robust_scores.causal_forest

Matrix Γ of scores for each treatment a

Description

Computes a matrix of double robust scores $\Gamma_{ia} = \mu_a(x) + \frac{1}{e_a(x)}(Y_i - \mu_a(x))1(A_i = a)$

Usage

```
## S3 method for class 'causal_forest'
double_robust_scores(object, ...)

## S3 method for class 'instrumental_forest'
double_robust_scores(object, compliance.score = NULL, ...)

## S3 method for class 'multi_causal_forest'
double_robust_scores(object, ...)

double_robust_scores(object, ...)
```

Arguments

`object`	An appropriate causal forest type object
`...`	Additional arguments
`compliance.score`	An estimate of the causal effect of Z on W. i.e., Delta(X) = E(W | X, Z = 1) - E(W | X, Z = 0), for each sample i = 1, ..., n. If NULL (default) then this is estimated with a causal forest.

Details

This is the matrix used for CAIPWL (Cross-fitted Augmented Inverse Propensity Weighted Learning)

Value

A matrix of scores for each treatment

Methods (by class)

- `causal_forest`: Scores (Γ_0, Γ_1)
- `instrumental_forest`: Scores $(-\Gamma, \Gamma)$
- `multi_causal_forest`: Matrix Γ of scores for each treatment a

Note

For instrumental_forest this method returns $(-\Gamma_i, \Gamma_i)$ where Γ_i is the double robust estimator of the treatment effect as in eqn. (44) in Athey and Wager (2021).

References

Athey, Susan, and Stefan Wager. "Policy Learning With Observational Data." Econometrica 89.1 (2021): 133-161.

Examples

```
# Compute double robust scores for a multi_causal forest
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- sample(c("A", "B", "C"), n, replace = TRUE)
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
forests <- multi_causal_forest(X = X, Y = Y, W = W)
scores <- double_robust_scores(forests)
head(scores)

# Compute double robust scores for a causal forest
n <- 500
p <- 10
```

```
X <- matrix(rnorm(n * p), n, p)
W <- rbinom(n, 1, 0.5)
Y <- pmax(X[, 1], 0) * W + X[, 2] + pmin(X[, 3], 0) + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
scores <- double_robust_scores(c.forest)
```

`gen_data_epl` *Example data generating process from Policy Learning With Observational Data*

Description

The DGP from section 5.2 in Athey and Wager (2021)

Usage

```
gen_data_epl(n, type = c("continuous", "jump"))
```

Arguments

`n`	Number of observations
`type`	tau is "continuous" (default - equation 46) or exhibits "jumps" (equation 47)

Value

A list

References

Athey, Susan, and Stefan Wager. "Policy Learning With Observational Data." Econometrica 89.1 (2021): 133-161.

`gen_data_mapl` *Example data generating process from Offline Multi-Action Policy Learning: Generalization and Optimization*

Description

The DGP from section 6.4.1 in Zhou, Athey, and Wager (2018): There are $d = 3$ actions (a_0, a_1, a_2) which depend on 3 regions the covariates $X \sim U[0,1]^p$ reside in. Observed outcomes: $Y \sim N(\mu_{a_i}(X_i), 4)$

Usage

```
gen_data_mapl(n, p = 10, sigma2 = 4)
```

Arguments

n	Number of observations X.
p	Number of features (minimum 7). Default is 10.
sigma2	Noise variance. Default is 4.

Value

A list with realized action a_i, region r_i, conditional mean μ, outcome Y and covariates X

References

Zhou, Zhengyuan, Susan Athey, and Stefan Wager. "Offline multi-action policy learning: Generalization and optimization." arXiv preprint arXiv:1810.04778 (2018).

---

multi_causal_forest *One vs. all causal forest for multiple treatment effect estimation*

Description

For K treatments this "naive" multivariate-grf proceeeds by fitting K separate causal forests where in forest k the treatment assignment vector is one-hot encoded for treament k (i.e. treatment vector w_k entry i is one where individual i receives treatment k, else zero). The steps are:

1. Estimate propensities $e_k(x)$ for each action 1..K: This is done with k separate regression forests with propensities normalized to sum to 1 at the final step.
2. Estimate the expected response m(x) = E(Y | X) marginalizing over treatment. This is done with one regression forest.
3. Estimate the treatment effect $\tau_k(x) = \frac{\mu_k(x) - m(x)}{1 - e_k(x)}$ with a causal forest (where $\mu_k(x) = E[Y|X, W = W_k]$)

Usage

```
multi_causal_forest(
  X,
  Y,
  W,
  Y.hat = NULL,
  W.hat = NULL,
  num.trees = 2000,
  sample.weights = NULL,
  clusters = NULL,
  equalize.cluster.weights = FALSE,
  sample.fraction = 0.5,
  mtry = min(ceiling(sqrt(ncol(X)) + 20), ncol(X)),
  min.node.size = 5,
  honesty = TRUE,
```

```
  honesty.fraction = 0.5,
  honesty.prune.leaves = TRUE,
  alpha = 0.05,
  imbalance.penalty = 0,
  stabilize.splits = TRUE,
  ci.group.size = 2,
  tune.parameters = "none",
  tune.num.trees = 200,
  tune.num.reps = 50,
  tune.num.draws = 1000,
  compute.oob.predictions = TRUE,
  orthog.boosting = FALSE,
  num.threads = NULL,
  seed = runif(1, 0, .Machine$integer.max)
)
```

Arguments

X	The covariates used in the causal regression.
Y	The outcome (must be a numeric vector with no NAs).
W	The treatment assignment (must be a categorical vector with no NAs).
Y.hat	Estimates of the expected responses E[Y | Xi], marginalizing over treatment. If Y.hat = NULL, these are estimated using a separate regression forest. See section 6.1.1 of the GRF paper for further discussion of this quantity. Default is NULL.
W.hat	Matrix with estimates of the treatment propensities E[Wk | Xi]. If W.hat = NULL, these are estimated using a k separate regression forests. Default is NULL.
num.trees	Number of trees grown in the forest. Note: Getting accurate confidence intervals generally requires more trees than getting accurate predictions. Default is 2000.
sample.weights	(experimental) Weights given to each sample in estimation. If NULL, each observation receives the same weight. Note: To avoid introducing confounding, weights should be independent of the potential outcomes given X. Default is NULL.
clusters	Vector of integers or factors specifying which cluster each observation corresponds to. Default is NULL (ignored).
equalize.cluster.weights	If FALSE, each unit is given the same weight (so that bigger clusters get more weight). If TRUE, each cluster is given equal weight in the forest. In this case, during training, each tree uses the same number of observations from each drawn cluster: If the smallest cluster has K units, then when we sample a cluster during training, we only give a random K elements of the cluster to the tree-growing procedure. When estimating average treatment effects, each observation is given weight 1/cluster size, so that the total weight of each cluster is the same. Note that, if this argument is FALSE, sample weights may also be directly adjusted via the sample.weights argument. If this argument is TRUE, sample.weights must be set to NULL. Default is FALSE.

`sample.fraction`
: Fraction of the data used to build each tree. Note: If honesty = TRUE, these subsamples will further be cut by a factor of honesty.fraction. Default is 0.5.

`mtry`
: Number of variables tried for each split. Default is $\sqrt{p} + 20$ where p is the number of variables.

`min.node.size`
: A target for the minimum number of observations in each tree leaf. Note that nodes with size smaller than min.node.size can occur, as in the original randomForest package. Default is 5.

`honesty`
: Whether to use honest splitting (i.e., sub-sample splitting). Default is TRUE. For a detailed description of honesty, honesty.fraction, honesty.prune.leaves, and recommendations for parameter tuning, see the grf algorithm reference.

`honesty.fraction`
: The fraction of data that will be used for determining splits if honesty = TRUE. Corresponds to set J1 in the notation of the paper. Default is 0.5 (i.e. half of the data is used for determining splits).

`honesty.prune.leaves`
: If true, prunes the estimation sample tree such that no leaves are empty. If false, keep the same tree as determined in the splits sample (if an empty leave is encountered, that tree is skipped and does not contribute to the estimate). Setting this to false may improve performance on small/marginally powered data, but requires more trees (note: tuning does not adjust the number of trees). Only applies if honesty is enabled. Default is TRUE.

`alpha`
: A tuning parameter that controls the maximum imbalance of a split. Default is 0.05.

`imbalance.penalty`
: A tuning parameter that controls how harshly imbalanced splits are penalized. Default is 0.

`stabilize.splits`
: Whether or not the treatment should be taken into account when determining the imbalance of a split. Default is TRUE.

`ci.group.size`
: The forest will grow ci.group.size trees on each subsample. In order to provide confidence intervals, ci.group.size must be at least 2. Default is 2.

`tune.parameters`
: A vector of parameter names to tune. If "all": all tunable parameters are tuned by cross-validation. The following parameters are tunable: ("sample.fraction", "mtry", "min.node.size", "honesty.fraction", "honesty.prune.leaves", "alpha", "imbalance.penalty"). If honesty is false these parameters are not tuned. Default is "none" (no parameters are tuned).

`tune.num.trees`
: The number of trees in each 'mini forest' used to fit the tuning model. Default is 200.

`tune.num.reps`
: The number of forests used to fit the tuning model. Default is 50.

`tune.num.draws`
: The number of random parameter values considered when using the model to select the optimal parameters. Default is 1000.

`compute.oob.predictions`
: Whether OOB predictions on training set should be precomputed. Default is TRUE.

orthog.boosting
: (experimental) If TRUE, then when Y.hat = NULL or W.hat is NULL, the missing quantities are estimated using boosted regression forests. The number of boosting steps is selected automatically. Default is FALSE.

num.threads
: Number of threads used in training. By default, the number of threads is set to the maximum hardware concurrency.

seed
: The seed of the C++ random number generator.

Value

A trained multi causal forest object (collection of causal forests). If tune.parameters is enabled, then tuning information will be included through the `tuning.output` attribute of each forest.

Examples

```
# Train a multi causal forest.
n <- 250
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- sample(c("A", "B", "C"), n, replace = TRUE)
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
multi.forest <- multi_causal_forest(X = X, Y = Y, W = W)

# Predict using the forest.
multi.forest.pred <- predict(multi.forest)
head(multi.forest.pred$predictions)
```

---

plot.policy_tree *Plot a policy_tree tree object.*

Description

Plot a policy_tree tree object.

Usage

```
## S3 method for class 'policy_tree'
plot(x, leaf.labels = NULL, ...)
```

Arguments

x
: The tree to plot.

leaf.labels
: An optional character vector of leaf labels for each treatment.

...
: Additional arguments (currently ignored).

Examples

```
# Plot a policy_tree object
## Not run:
n <- 250
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- sample(c("A", "B", "C"), n, replace = TRUE)
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
multi.forest <- multi_causal_forest(X = X, Y = Y, W = W)
Gamma.matrix <- double_robust_scores(multi.forest)
tree <- policy_tree(X, Gamma.matrix, depth = 2)
plot(tree)

# Provide optional names for the treatment names in each leaf node
# `action.names` is by default the column names of the reward matrix
plot(tree, leaf.labels = tree$action.names)
# Providing a custom character vector
plot(tree, leaf.labels = c("treatment A", "treatment B", "placebo C"))

# Saving a plot in a vectorized SVG format can be done with the `DiagrammeRsvg` package.
install.packages("DiagrammeRsvg")
tree.plot = plot(tree)
cat(DiagrammeRsvg::export_svg(tree.plot), file = 'plot.svg')

## End(Not run)
```

`policy_tree` *Fit a policy with exact tree search*

Description

Finds the optimal (maximizing the sum of rewards) depth L tree by exhaustive search. If the optimal action is the same in both the left and right leaf of a node, the node is pruned.

Usage

```
policy_tree(X, Gamma, depth = 2, split.step = 1, min.node.size = 1)
```

Arguments

`X`	The covariates used. Dimension Np where p is the number of features.
`Gamma`	The rewards for each action. Dimension Nd where d is the number of actions.
`depth`	The depth of the fitted tree. Default is 2.
`split.step`	An optional approximation parameter (integer above zero), the number of possible splits to consider when performing tree search. split.step = 1 (default) considers every possible split, split.step = 10 considers splitting at every 10'th sample and may yield a substantial speedup for dense features. Manually rounding or re-encoding continuous covariates with very high cardinality in a problem

specific manner allows for finer-grained control of the accuracy/runtime tradeoff and may in some cases be the preferred approach over this option.

`min.node.size` An integer indicating the smallest terminal node size permitted. Default is 1.

Details

The amortized runtime of the exact tree search is $O(p^k n^k (logn + d) + pnlogn)$ where p is the number of features, d the number of treatments, n the number of observations, and $k \geq 1$ the tree depth.

For a depth two tree this is $O(p^2 n^2 (logn + d))$ (ignoring the last term which is a global sort done at the beginning) meaning that it scales quadratically with the number of observations, i.e. if you double the number of observations, the search will take at least four times as long.

For a depth three tree it is $O(p^3 n^3 (logn + d))$. If a depth two tree with 1000 observations, 4 features and 3 actions took around t seconds, you can expect the level three tree to take approximately $1000 \cdot 4$ times as long ($\approx \frac{p^3 n^2}{p^2 n^2} = pn$)

The runtime above is with continuous features. There are considerable time savings when the features are discrete. In the extreme case with all binary observations, the runtime will be practically linear in n.

The optional approximation parameter `split.step` emulates rounding the data and is recommended to experiment with in order to reduce the runtime.

Value

A policy_tree object.

References

Sverdrup, Erik, Ayush Kanodia, Zhengyuan Zhou, Susan Athey, and Stefan Wager. "policytree: Policy learning via doubly robust empirical welfare maximization over trees." Journal of Open Source Software 5, no. 50 (2020): 2232.

Zhou, Zhengyuan, Susan Athey, and Stefan Wager. "Offline multi-action policy learning: Generalization and optimization." arXiv preprint arXiv:1810.04778 (2018).

Examples

```
# Fit a depth two tree on doubly robust treatment effect estimates
# from a causal forest.
n <- 10000
p <- 5
X <- round(matrix(rnorm(n * p), n, p), 2)
W <- rbinom(n, 1, 1 / (1 + exp(X[, 3])))
tau <- 1 / (1 + exp((X[, 1] + X[, 2]) / 2)) - 0.5
Y <- X[, 3] + W * tau + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
dr.scores <- double_robust_scores(c.forest)

tree <- policy_tree(X, dr.scores, 2)
```

```
tree

# Predict treatment assignment.
predicted <- predict(tree, X)

plot(X[, 1], X[, 2], col = predicted)
legend("topright", c("control", "treat"), col = c(1, 2), pch = 19)
abline(0, -1, lty = 2)

# Predict the leaf assigned to each sample.
node.id <- predict(tree, X, type = "node.id")
# Can be reshaped to a list of samples per leaf node with `split`.
samples.per.leaf <- split(1:n, node.id)

# The value of all arms (along with SEs) by each leaf node.
values <- aggregate(dr.scores, by = list(leaf.node = node.id),
                    FUN = function(x) c(mean = mean(x), se = sd(x) / sqrt(length(x))))
print(values, digits = 2)
```

`predict.multi_causal_forest` *Predict with multi_causal_forest*

Description

Computes estimates of $\tau_a(x)$

Usage

```
## S3 method for class 'multi_causal_forest'
predict(object, newdata = NULL, ...)
```

Arguments

`object`	The trained forest.
`newdata`	Points at which predictions should be made. If NULL, makes out-of-bag predictions on the training set instead (i.e., provides predictions at Xi using only trees that did not use the i-th training example). Note that this matrix should have the number of columns as the training matrix, and that the columns must appear in the same order.
`...`	Additional arguments passed to grf::predict.causal_forest.

Value

List containing matrix of predictions and other estimates (debiased error, etc.) for each treatment.

Examples

```
# Train a multi causal forest.
n <- 250
p <- 10
X <- matrix(rnorm(n * p), n, p)
W <- sample(c("A", "B", "C"), n, replace = TRUE)
Y <- X[, 1] + X[, 2] * (W == "B") + X[, 3] * (W == "C") + runif(n)
multi.forest <- multi_causal_forest(X = X, Y = Y, W = W)

# Predict using the forest.
multi.forest.pred <- predict(multi.forest)
head(multi.forest.pred$predictions)
```

---

predict.policy_tree *Predict method for policy_tree*

Description

Predict values based on fitted policy_tree object.

Usage

```
## S3 method for class 'policy_tree'
predict(object, newdata, type = c("action.id", "node.id"), ...)
```

Arguments

object	policy_tree object
newdata	A data frame with features
type	The type of prediction required, "action.id" is the action id and "node.id" is the integer id of the leaf node the sample falls into. Default is "action.id".
...	Additional arguments (currently ignored).

Value

A vector of predictions. For type = "action.id" each element is an integer from 1 to d where d is the number of columns in the reward matrix. For type = "node.id" each element is an integer corresponding to the node the sample falls into (level-ordered).

Examples

```
# Fit a depth two tree on doubly robust treatment effect estimates
# from a causal forest.
n <- 10000
p <- 5
```

```
X <- round(matrix(rnorm(n * p), n, p), 2)
W <- rbinom(n, 1, 1 / (1 + exp(X[, 3])))
tau <- 1 / (1 + exp((X[, 1] + X[, 2]) / 2)) - 0.5
Y <- X[, 3] + W * tau + rnorm(n)
c.forest <- grf::causal_forest(X, Y, W)
dr.scores <- double_robust_scores(c.forest)

tree <- policy_tree(X, dr.scores, 2)
tree

# Predict treatment assignment.
predicted <- predict(tree, X)

plot(X[, 1], X[, 2], col = predicted)
legend("topright", c("control", "treat"), col = c(1, 2), pch = 19)
abline(0, -1, lty = 2)

# Predict the leaf assigned to each sample.
node.id <- predict(tree, X, type = "node.id")
# Can be reshaped to a list of samples per leaf node with `split`.
samples.per.leaf <- split(1:n, node.id)

# The value of all arms (along with SEs) by each leaf node.
values <- aggregate(dr.scores, by = list(leaf.node = node.id),
                    FUN = function(x) c(mean = mean(x), se = sd(x) / sqrt(length(x))))
print(values, digits = 2)
```

`print.multi_causal_forest` *Print a multi_causal_forest object.*

Description

Print a multi_causal_forest object.

Usage

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

Arguments

`x`	The object to print.
`...`	Additional arguments (currently ignored).

print.policy_tree *Print a policy_tree object.*

Description

Print a policy_tree object.

Usage

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

Arguments

x	The tree to print.
...	Additional arguments (currently ignored).

Index