An Introduction to Recursive Partitioning for Heterogeneous Causal Effects Estimation Using `causalTree` package

Susan Athey
Guido Imbens
Yanyang Kong
Vikas Ramachandra

August 22, 2016

Contents

1 Introduction **2**

2 Notation **4**

3 Building Causal Trees **5**

3.1 Splitting rules 5

3.1.1 Transformed Outcome Trees (**TOT**) 6

3.1.2 Causal Trees (**CT**) 6

3.1.3 Fit-based Trees (**fit**) 7

3.1.4 Squared T-statistic Trees (**tstats**) 8

3.2 Discrete splitting 8

3.3 Example 8

4 Cross Validation and Pruning **9**

4.1 Cross validation options 9

4.1.1 TOT 10

4.1.2 CT 10

4.1.3 fit 11

4.1.4 matching 11

4.2 Example 12

5 Honest Estimation **14**

5.1 Example 14

6 Causal Forests **15**
6.1 Example 15

7 Propensity Forests **16**
7.1 Example 16

1 Introduction

This document is a brief introduction of `causalTree` package, which includes the `causalTree` function and the `honest.causalTree` function. These implement the methods from *Recursive Partitioning for Heterogeneous Causal Effects* [1]. The package also includes the functions `causalForest` and `propensityForest` which implement versions of the causal forest algorithm from *Estimation and Inference of Heterogeneous Treatment Effects using Random Forests* [3]

The `causalTree` function builds a regression model and returns an `rpart` object, which is the object derived from `rpart` package, implementing many ideas in the CART (Classification and Regression Trees), written by Breiman, Friedman, Olshen and Stone [2]. Like `rpart`, `causalTree` builds a binary regression tree model in two stages, but focuses on estimating heterogeneous treatment effects. The function requires the user to specify a binary treatment variable in addition to the outcome variable and the features that are usually passed to `rpart`.

Following `rpart`, in the first stage, the tree is grown from the root node based on a specified splitting rule. In each node, the data in a leaf will be split into two groups to best minimize the risk function. Next, in the left sub-node and right sub-node, the splitting routine will be applied separately and so on recursively until no improvements can made, or until some limits are reached (e.g. the routine will stop if it cannot make splits that have at least `minsize` of treated observations and `minsize` control observations in each terminal node.) The risk for each node is calculated using a risk function associated with the splitting rule.

Unlike `rpart`, which estimates the average of the outcome variable in each leaf, `causalTree` estimates a treatment effect in each leaf by taking the difference of the sample average of the treated group and the sample average of the control group within the leaf. The user selects one of several splitting rules for determining the optimal split at each step.

The user also specifies a cross-validation method. Cross-validation is implemented similar to `rpart`, where the cross-validation penalty parameter penalizes the number of nodes in the tree. The main difference arises in the cross-validation criterion, which is selected by the user. The `cptable` of the `rpart` object includes the cross-validation error. The user can choose to prune the tree using a specified cross-validation method, where The leaves to be pruned are selected according to the risk function calculated while the tree is built.

The causalTree package incorporates an additional function not included in rpart, which is honest re-estimation honest.causalTree of causal effects. Honest here means that we estimate causal effects in the leaves of a given tree on an independent estimation sample rather than the data used to build and cross-validate the tree. The user first builds the tree with causalTree, specifying the training data for building the tree, and then passes the tree object as well as the estimation sample data into honest.causalTree, which replaces the leaf estimates from the input tree with new estimates in each leaf, calculated on the estimation sample.

To allow the user more control, there is another function, estimate.causalTree, that takes in an rpart object (which could have been estimated using the rpart package) and replaces the leaf estimates with sample average treatment effects within each leaf, using a new dataset passed in by the user. So it is possible to build a tree in either causalTree or rpart using any one of a number of methods, and then use estimate.causalTree to estimate treatment effects in each leaf on an arbitrary dataset.

The package has a few additional features not present in rpart. One is that the minsize parameter requires that there are at least minsize treated and minsize control observations in each leaf, so that a sample average treatment effect can be calculated.

Another feature is that we allow the user the option to restrict the set of potential split points considered, and further, in the splitting process we rescale the covariate values within each leaf and each treatment group in order to ensure that when moving from one potential split point to the next, we move the same number of treatment and control observations from the right leaf to the left leaf.

The rpart algorithm considers every value of every feature as a possible split point. An obvious disadvantage of this approach is that computation time can grow prohibitively large in models with many observations and features. But there are some more subtle disadvantages as well. The first is that there will naturally be sampling variation in calculating the risk function as we vary the possible split points. A problem akin to a multiple hypothesis testing problem arises: since we are looking for the maximum value of an estimated criterion across a large number of possible split points, as the number of split points tested grows, it becomes more and more likely that one of the splits for a given covariate appears to improve the fit criterion even if the true value of the criterion would indicate that it is better not to split. One way to mitigate both the computation time problem and the multiple-testing problem is to consider only a limited number of split points.

A third problem is specific to considering treatment effect heterogeneity. To see the problem, suppose that a covariate strongly affects the mean of outcomes, but not treatment effects. Within a leaf, some observations are treated and some are control. If we consider every level of the covariate in the leaf as a possible split point, then shifting from one split point to the next shifts a single observation from the right leaf to the left leaf. This observation is in the treatment or the control group, but not both; suppose it is in the treatment group. If the covariate has a strong effect on the level of outcomes, the observation that is shifted will be likely have an outcome more extreme than average. It will change the sample average of the treatment group, but not the control group, leading to a large change in the estimated treatment effect difference. We expect the estimated difference in treatment effects across the left and right leaves to fluctuate greatly with the split point in this scenario. This variability around the true mean difference in treatment effects occurs more often when covariates affect mean outcomes, and thus it can lead the estimators to split too much on such covariates, and also to find spurious opportunities to split.

To address this problem, we incorporate the following modifications to the splitting rule. We include a parameter `bucketNum`, the target number of observations per "bucket." For each leaf, before testing possible splits for a particular covariate, we order the observations by the covariate value in the treatment and control group separately. Within each group, we place the observations into buckets with `bucketNum` observations per bucket. If this results in less than `minsize` (another user-set parameter) buckets, then we use fewer observations per bucket (to attain `minsize` buckets). If this results in more than `bucketMax` buckets, we use more observations per bucket to obtain `bucketMax` buckets. We number the buckets, and considering potential splits by bucket number rather than the raw values of the covariates. This guarantees that when we shift from one split point to the next, we add both treatment and control observations, leading to a smoother estimate of the goodness of fit function as a function of the split point. After the best bucket number to split on is selected, we translate that into a split point by averaging the maximum covariate value in the corresponding treatment and control buckets.

The `CausalForest` function code extends the `CausalTree` method, specifically by building an ensemble of trees, based on a user specified number. The data is repeatedly sampled to build each tree (with replacement).This function returns a causalForest object, which is a list of `rpart` objects. Note that this function always performs honest estimation, by using one subset of the data to build the tree ensemble and another subset of the data to estimate the treatment effects at the leaves of the built trees. To predict, call R's predict function with new test data and the causalForest object (estimated on the training data) obtained after calling the causalForest function. During the prediction phase, the average value over all tree predictions is returned as the final prediction.

An alternative method to build an ensemble of trees is also implemented using the function `propensityForest`. In this function, an ensemble of trees are built, and splitting nodes for each tree are determined using the treatment variable (along with the input covariates) as a proxy for the output variable. To evaluate the tree ensemble built, treatment effects are estimated using the true output variable along with the input covariates.

2 Notation

X_i	$i = 1, 2, ..., N$	observed variables or feature matrix for observation i.

Y_i $i = 1, 2, ..., N$ observed outcome of observation i.

W_i $i = 1, 2, ..., N$ binary indicator for the treatment,
with $W_i = 0$ indicating that observation i received the control treatment,
and $W_i = 1$ indicating that observation i received the active treatment.

$\mathcal{S}$ a data sample drawn from data sample population,
$\mathcal{S}^{\text{tr}}$ denotes a training sample,
$\mathcal{S}^{\text{te}}$ denotes a test sample,
$\mathcal{S}^{\text{est}}$ denotes an estimation sample.
$\mathcal{S}_{\text{treat}}$ and $\mathcal{S}_{\text{control}}$ denote the subsamples of treated and control units.

N N^{tr} denotes the number of observations in training sample,
N^{te} denotes the number of observations in testing sample,
N^{est} denotes the number of observations in estimation sample.

Π a partitioning tree $\Pi = \{\ell_1, \ldots, \ell_{\#(\Pi)}\}$ with $\cup_{j=1}^{\#(\Pi)} \ell_j = \mathbb{X}$
corresponds to a partitioning of the feature space the feature space $\mathbb{X}$, with
$\#(\Pi)$ the number of elements in the partition.

$\ell(x; \Pi)$ the leaf $\ell \in \Pi$ such that $x \in \ell$.

$\tau(\ell)$ $l = 1, 2, ..., k$ causal effect or treatment effect in leaf ℓ.

p marginal treatment probability, $p = \text{pr}(W_i = 1)$.

3 Building Causal Trees

3.1 Splitting rules

`causalTree` function offers four different splitting rules for user to choose. Each splitting rule corresponds to a specific risk function, and each split at a node aims to minimize the risk function. For each observation $(Y_i^{\text{obs}}, X_i, W_i)$, given a tree Π, the population average outcome is

$$\mu(w, x; \Pi) \equiv \mathbb{E}\left[Y_i(w) \right| X_i \in \ell(x; \Pi)],$$

and its average causal effect is

$$\tau(x; \Pi) \equiv \mathbb{E}\left[Y_i(1) - Y_i(0) \right| X_i \in \ell(x; \Pi)].$$

the estimated outcome is

$$\hat{\mu}(w, x; \mathcal{S}, \Pi) \equiv \frac{1}{\#(\{i \in \mathcal{S}_w : X_i \in \ell(x;\Pi)\})} \sum_{i \in \mathcal{S}_w : X_i \in \ell(x;\Pi)} Y_i^{\text{obs}},$$

the estimated causal effect is the difference of treated mean and control mean in the leaf l where it belongs,

$$\hat{\tau}(x; \mathcal{S}, \Pi) \equiv \tau(\ell) = \hat{\mu}(1, x; \mathcal{S}, \Pi) - \hat{\mu}(0, x; \mathcal{S}, \Pi).$$

3.1.1 Transformed Outcome Trees (TOT)

We first define the transformed outcome as

$$Y_i^* = Y_i \cdot \frac{W_i - p}{p \cdot (1-p)}$$

where $p = N_{\text{treat}}/N$ is the reatment probability, and

$$Y_i^* = \begin{cases} Y_i/p & W_i = 1 \\ -Y_i/(1-p) & W_i = 0 \end{cases}$$

In **TOT** splitting rule, the risk function is given by

$$\widehat{\text{MSE}}(\mathcal{S}^{\text{tr}}, \mathcal{S}^{\text{tr}}, \Pi) = \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \left\{ (Y_i^* - \hat{\tau}(X_i; \mathcal{S}^{\text{tr}}, \Pi))^2 - Y_i^{*2} \right\}$$

Note that the paper [1] envisions that treatment effects would be estimated by taking the mean of Y_i^* within a leaf, but points out that this is inefficient because the treated fraction in a leaf may differ from the population proportion due to sampling variation. Thus, our package uses $\hat{\tau}$ instead. The `rpart` package can be used off-the-shelf (applied with Y_i^* as the outcome) to implement the method precisely as described in [1].

3.1.2 Causal Trees (CT)

For the causal tree "CT" splitting rule, we have two versions, the adaptive verison, denoted as **CT-A**, and the honest version, **CT-H**. The user can select the honest version by setting `split.Honest = TRUE` in `causalTree` function.

For **CT-A**, we use $\widehat{\text{MSE}}_\tau(\mathcal{S}^{\text{tr}}, \mathcal{S}^{\text{tr}}, \Pi)$ as the criterion (risk) function, and

$$-\widehat{\text{MSE}}_\tau(\mathcal{S}^{\text{tr}}, \mathcal{S}^{\text{tr}}, \Pi) = \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \hat{\tau}^2(X_i; \mathcal{S}^{\text{tr}}, \Pi).$$

For **CT-H**, the honest version, the splitting criterion function is $\widehat{\text{EMSE}}_\tau(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi)$, and

$$-\widehat{\text{EMSE}}_\tau(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi) = \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \hat{\tau}^2(X_i; \mathcal{S}^{\text{tr}}, \Pi) - \left(\frac{1}{N^{\text{tr}}} + \frac{1}{N^{\text{est}}}\right) \cdot \sum_{\ell \in \Pi} \left(\frac{S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell)}{p} + \frac{S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell)}{1-p} \right).$$

where $S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell)$ is the within-leaf variance on outcome Y for $\mathcal{S}^{\text{tr}}_{\text{control}}$ in leaf ℓ, and $S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell)$ is the counter part for $\mathcal{S}^{\text{tr}}_{\text{treat}}$. N^{est} (the number of observations in re-estimation sample) is specified as `HonestSampleSize` in `causalTree` function, and the default value is N^{tr}.
In our package we incorporate an additional parameter `split.alpha` $= \alpha \in (0, 1)$ as a parameter to adjust the proportion of $\widehat{\text{MSE}}$ and the varaince term in $\widehat{\text{EMSE}}$.

$$-\widehat{\text{EMSE}}_\tau(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi, \alpha) = \alpha \cdot \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \hat{\tau}^2(X_i; \mathcal{S}^{\text{tr}}, \Pi) - (1-\alpha) \cdot \left(\frac{1}{N^{\text{tr}}} + \frac{1}{N^{\text{est}}}\right) \cdot \sum_{\ell \in \Pi} \left(\frac{S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell)}{p} + \frac{S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell)}{1-p} \right)$$

3.1.3 Fit-based Trees (fit)

In fit-based splitting rule, we decide at what value of the feature to split based on the goodness-of-fit of the outcome rather than the treatment effect. As **CT**, there are two versions of **fit**, namely adaptive version **fit-A** and honest version **fit-H**.
For **fit-A**, the objective risk function in splitting is

$$\widehat{\text{MSE}}_{\mu,W}(\mathcal{S}^{\text{tr}}, \mathcal{S}^{\text{tr}}, \Pi) = \sum_{i \in \mathcal{S}^{\text{tr}}} \left\{ (Y_i - \hat{\mu}_w(W_i, X_i; \mathcal{S}^{\text{tr}}, \Pi))^2 - Y_i^2 \right\}$$

where $\hat{\mu}_w$ is the mean of outcome in treatment/control group.

For **fit-H**, the honest version, the risk function is $\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi)$,

$$-\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi) = \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \hat{\mu}^2_w(W_i, X_i; \mathcal{S}^{\text{tr}}, \Pi) - \left(\frac{1}{N^{\text{tr}}} + \frac{1}{N^{\text{est}}}\right) \cdot \sum_{\ell \in \Pi} \left(S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell) + S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell) \right),$$

where $S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell)$ is the within-leaf variance on outcome Y for $\mathcal{S}^{\text{tr}}_{\text{control}}$ in leaf ℓ, and $S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell)$ is the counter part for $\mathcal{S}^{\text{tr}}_{\text{treat}}$. N^{est} (the number of observations in re-estimation sample) is specified as `HonestSampleSize` in `causalTree` function, and the default value is N^{tr}. Also like **CT**, we have adjusted honest verison for $\widehat{\text{EMSE}}_{\mu,W}$ using `split.alpha`,

$$-\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr}}, N^{\text{est}}, \Pi, \alpha) = \alpha \cdot \frac{1}{N^{\text{tr}}} \sum_{i \in \mathcal{S}^{\text{tr}}} \hat{\mu}_w^2(W_i, X_i; \mathcal{S}^{\text{tr}}, \Pi) \\ - (1-\alpha) \cdot \left(\frac{1}{N^{\text{tr}}} + \frac{1}{N^{\text{est}}}\right) \cdot \sum_{\ell \in \Pi} \left(S^2_{\mathcal{S}^{\text{tr}}_{\text{treat}}}(\ell) + S^2_{\mathcal{S}^{\text{tr}}_{\text{control}}}(\ell) \right),$$

3.1.4 Squared T-statistic Trees (tstats)

In squared t-statistic ("tstats") trees, we consider the splits with the largest value for square of the t-statistic for testing the null hypothesis that the average treatment effect is the same in the two potential leaves. Denote the left leaf as L and right leaf as R, the square of the t-statistic is

$$T^2 \equiv \frac{((\overline{Y}_{L1} - \overline{Y}_{L0}) - (\overline{Y}_{R1} - \overline{Y}_{R0}))^2}{S^2_{L1}/N_{L1} + S^2_{L0}/N_{L0} + S^2_{R1}/N_{R1} + S^2_{R0}/N_{R0}},$$

where $S^2_{\ell,w}$ is the conditional within treatment group sample variance given the split. Unlike the mean-squared error criteria, the "tstats" criterion does not neatly decompose into a risk for each observation; thus, it is not obvious what criterion to use for determining which leaves to prune first when pruning the tree. Our package uses the "CT-H" criterion function.

3.2 Discrete splitting

As motivated in the introduction, the package includes the option to consider a discrete set of splitting points. To use discrete splitting, the user should set `split.Bucket = TRUE` and specify`bucketNum`, `bucketMax`. The default value of `bucketNum = 5` and `bucketMax = 100`.

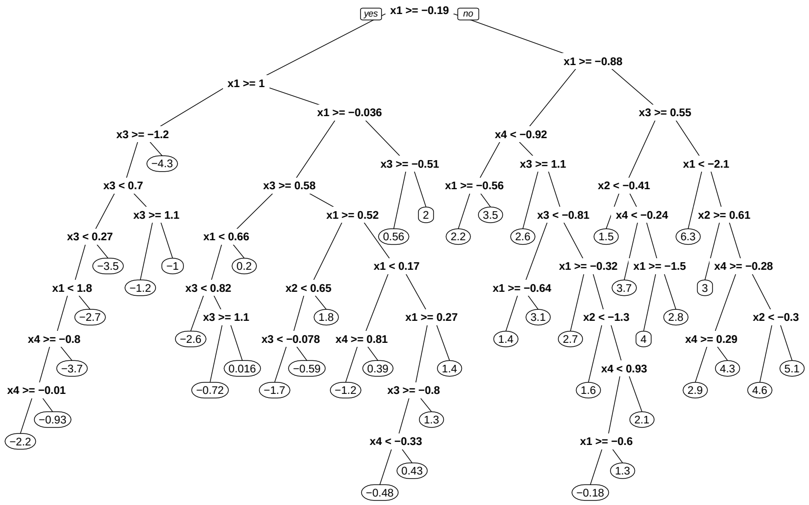
In discrete splitting, the samples in a node will be sorted by value of a feature separately for the treated and control groups, and then observations will be partitioned in to buckets. The number of buckets is determined as min(`bucketMax`, max(n_{min}/`bucketNum`, `minsize`)) observations, where n_{min} is the minimum of the number of treated observations and the number of control observations in the leaf to be split. Then one bucket will be treated as a whole and assigned into left branch or right branch.

3.3 Example

The data we use in this example is a simulated data set called `simulation.1` built in `causalTree` package.

In this model, we choose **TOT** as splitting rule and **fit** as cross validation method by setting `split.Rule = "TOT"` and `cv.option = "fit"`. The propensity score (treatment probability) is set as `propensity = 0.5` for **TOT** splitting rule. We also use the discrete splitting option by setting `split.Bucket = T`.

```
> library(causalTree)
> tree <- causalTree(y ~ x1 + x2 + x3 + x4, data = simulation.1,
+                   treatment = simulation.1$treatment, split.Rule = "TOT",
+                   cv.option = "fit", cv.Honest = F, split.Bucket = T,
+                   xval = 10, cv.alpha = 0.5, propensity = 0.5)
> rpart.plot(tree)
```



From the plot we can see, without pruning, the tree we get is quite large. The following section will describe different cross validation methods to prune the tree.

4 Cross Validation and Pruning

Following `rpart`, we will build cross validation trees to select a complexity parameter corresponding to the minimum cross validaiton error used for pruning. Different from `rpart`, in cross validation, we can choose different evaluation criteria to calculate the error.

4.1 Cross validation options

We offer four criteria for cross validation, **TOT**, **CT**, **fit** and **matching**. Each criterion corresponds to an evaluation function for computing the cross-validation error. When building trees in the cross-validation process, we use the original splitting rule (specified by `split.Rule`), but the cross validation error evaluation function is determined by `cv.option`.

4.1.1 TOT

When `cv.option`=**TOT**, the evaluation function is

$$\widehat{\mathrm{MSE}}(\mathcal{S}^{\mathrm{tr,cv}}, \mathcal{S}^{\mathrm{tr,tr}}, \Pi) = \frac{1}{N^{\mathrm{tr,cv}}} \sum_{i \in \mathcal{S}^{\mathrm{tr,cv}}} \left\{ (Y_i^* - \hat{\tau}(X_i; \mathcal{S}^{\mathrm{tr,tr}}, \Pi))^2 - Y_i^{*2} \right\}$$

where $\mathcal{S}^{\mathrm{tr,tr}}$ is part of training sample used for building cross validaiton trees and $\mathcal{S}^{\mathrm{tr,cv}}$ is the other part of training sample (referred to here as the validation sample) used for predicting and calculating the error, and $N^{\mathrm{tr,cv}}$ is the number of observations in $\mathcal{S}^{\mathrm{tr,cv}}$. There is no “honest” option for this method.

4.1.2 CT

When `cv.option`=**CT**, there are two versions, adaptive and honest. We denote them as **CT-A** and **CT-H**.
For the **CT-A** cross validation method, the evaluation funciton is

$$\begin{aligned} \widehat{\mathrm{MSE}}_\tau(\mathcal{S}^{\mathrm{tr,cv}}, \mathcal{S}^{\mathrm{tr,tr}}, \Pi) = & -\frac{2}{N^{\mathrm{tr,cv}}} \sum_{i \in \mathcal{S}^{\mathrm{tr,cv}}} \hat{\tau}(X_i; \mathcal{S}^{\mathrm{tr,cv}}, \Pi) \hat{\tau}(X_i; \mathcal{S}^{\mathrm{tr,tr}}, \Pi) \\ & + \frac{1}{N^{\mathrm{tr,cv}}} \sum_{i \in \mathcal{S}^{\mathrm{tr,cv}}} \hat{\tau}^2(X_i; \mathcal{S}^{\mathrm{tr,tr}}, \Pi), \end{aligned}$$

where $\hat{\tau}(X_i; \mathcal{S}^{\mathrm{tr,cv}}, \Pi)$ is the treatment effect calculated in the validation sample and $\hat{\tau}(X_i; \mathcal{S}^{\mathrm{tr,tr}}, \Pi)$ is the treatment effect in the training component $\mathcal{S}^{\mathrm{tr,tr}}$.
For **CT-H** cross validation method, the evaluation function is $\widehat{\mathrm{EMSE}}_\tau(\mathcal{S}^{\mathrm{tr,cv}}, N^{\mathrm{est}}, \Pi)$, and

$$\begin{aligned} -\widehat{\mathrm{EMSE}}_\tau(\mathcal{S}^{\mathrm{tr,cv}}, N^{\mathrm{est}}, \Pi) = & \frac{1}{N^{\mathrm{tr,cv}}} \sum_{i \in \mathcal{S}^{\mathrm{tr,cv}}} \hat{\tau}^2(X_i; \mathcal{S}^{\mathrm{tr,cv}}, \Pi) \\ & - \left(\frac{1}{N^{\mathrm{tr,cv}}} + \frac{1}{N^{\mathrm{est}}} \right) \cdot \sum_{\ell \in \Pi} \left(\frac{S^2_{\mathcal{S}^{\mathrm{tr,cv}}_{\mathrm{treat}}}(\ell)}{p} + \frac{S^2_{\mathcal{S}^{\mathrm{tr,cv}}_{\mathrm{control}}}(\ell)}{1-p} \right). \end{aligned}$$

Like its splitting method, we also incorporate an additional factor `cv.alpha` for adjustment of two terms in the formula,

$$-\widehat{\text{EMSE}}_\tau(\mathcal{S}^{\text{tr,cv}}, N^{\text{est}}, \Pi, \alpha) = \alpha \cdot \frac{1}{N^{\text{tr,cv}}} \sum_{i \in \mathcal{S}^{\text{tr,cv}}} \hat{\tau}^2(X_i; \mathcal{S}^{\text{tr,cv}}, \Pi) \\ - (1-\alpha) \cdot \left(\frac{1}{N^{\text{tr,cv}}} + \frac{1}{N^{\text{est}}} \right) \cdot \sum_{\ell \in \Pi} \left(\frac{S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{treat}}}(\ell)}{p} + \frac{S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{control}}}(\ell)}{1-p} \right).$$

4.1.3 fit

Like its splitting counterpart, **fit** cross validation criterion also has adaptive version **fit-A** and honest version **fit-H** to evaluate the model.
For **fit-A** criterion, the evaluation risk function is

$$\widehat{\text{MSE}}_{\mu,W}(\mathcal{S}^{\text{tr,cv}}, \mathcal{S}^{\text{tr,tr}}, \Pi) = \sum_{i \in \mathcal{S}^{\text{tr,cv}}} \left\{ (Y_i - \hat{\mu}_w(W_i, X_i; \mathcal{S}^{\text{tr,tr}}, \Pi))^2 - Y_i^2 \right\}$$

where $\hat{\mu}_w(W_i, X_i; \mathcal{S}^{\text{tr,tr}}, \Pi)$ is the mean of outcome in treatment/control group of the built cross validation tree where validation sample $(X_i, Y_i, W_i) \in \mathcal{S}^{\text{tr,cv}}$ finally be assigned.

For **fit-H** criterion, the evaluation function is $\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr,cv}}, N^{\text{est}}, \Pi)$,

$$-\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr,cv}}, N^{\text{est}}, \Pi) = \frac{1}{N^{\text{tr,cv}}} \sum_{i \in \mathcal{S}^{\text{tr,cv}}} \hat{\mu}_w^2(W_i, X_i; \mathcal{S}^{\text{tr,cv}}, \Pi) \\ - \left(\frac{1}{N^{\text{tr,cv}}} + \frac{1}{N^{\text{est}}} \right) \cdot \sum_{\ell \in \Pi} \left(S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{treat}}}(\ell) + S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{control}}}(\ell) \right),$$

In contrast to the adaptive version, $\hat{\mu}_w(W_i, X_i; \mathcal{S}^{\text{tr,cv}}, \Pi)$ is the mean of outcome derived from the validation group.
Also we incorporate `cv.alpha` for adjustment in **CT-H** evaluation function,

$$-\widehat{\text{EMSE}}_{\mu,W}(\mathcal{S}^{\text{tr,cv}}, N^{\text{est}}, \Pi, \alpha) = \alpha \cdot \frac{1}{N^{\text{tr,cv}}} \sum_{i \in \mathcal{S}^{\text{tr,cv}}} \hat{\mu}_w^2(W_i, X_i; \mathcal{S}^{\text{tr,cv}}, \Pi) \\ - (1-\alpha) \cdot \left(\frac{1}{N^{\text{tr,cv}}} + \frac{1}{N^{\text{est}}} \right) \cdot \sum_{\ell \in \Pi} \left(S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{treat}}}(\ell) + S^2_{\mathcal{S}^{\text{tr,cv}}_{\text{control}}}(\ell) \right).$$

4.1.4 matching

In **matching** method, we first define $n(W_i, X_i; \mathcal{S})$ to be the nearest neighbor of (X_i, Y_i, W_i) in feature space with opposite W.

To be more specific, let $j = n(W_i, X_i; \mathcal{S})$, then

$$\begin{aligned}
&(X_j, Y_j, W_j) \in \mathcal{S},\\
&W_j = 1 - W_i,\\
&d_X((X_j, Y_j, W_j), (X_i, Y_i, W_i)) = \min_{\substack{(X_k, Y_k, W_k) \in \mathcal{S},\\ W_k = 1 - W_i}} d_X((X_k, Y_k, W_k), (X_i, Y_i, W_i)).
\end{aligned}$$

Then we can define the **matching** estimator of treatment effect as

$$\tau^*(X_i, W_i; \mathcal{S}) \equiv (2W_i - 1)(Y_i - Y_{n(W_i, X_i; \mathcal{S})})$$

The evaluation risk function in **matching** method is

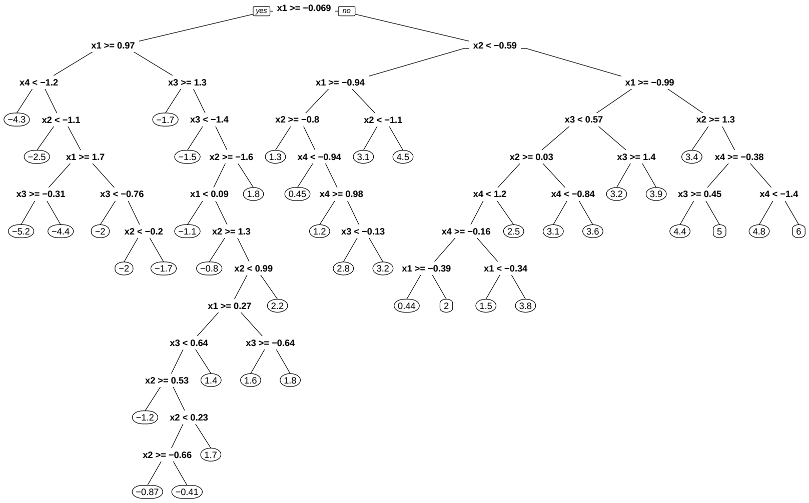
$$\widehat{\text{MSE}}_\tau(\mathcal{S}^{\text{tr,cv}}, \mathcal{S}^{\text{tr,tr}}, \Pi) = \sum_{i \in \mathcal{S}^{\text{tr,cv}}} \left(\tau^*(X_i, W_i; \mathcal{S}) - \frac{\hat{\tau}(X_i; \mathcal{S}^{\text{tr,tr}}, \Pi) + \hat{\tau}(X_{n(W_i, X_i; \mathcal{S}^{\text{tr,cv}})}; \mathcal{S}^{\text{tr,tr}}, \Pi)}{2} \right)^2$$

4.2 Example

In the following example, we choose honest splitting rule as **CT-H** (`split.Rule ="CT"`, `split.Honest = T`), and cross validation method as **matching** (`cv.option = "matching"` and `cv.Honest = F`). We set 10 folds cross validation (`xval = 10`) and print out the `cptable` to check out the complexity parameter (`cp`) and normalized cross validation error (`xerror`).

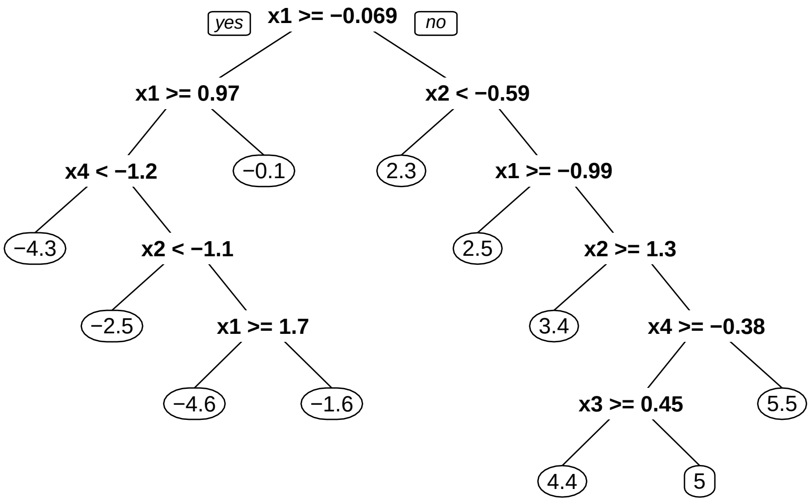
```
> tree <- causalTree(y ~ x1 + x2 + x3 + x4, data = simulation.1,
+                    treatment = simulation.1$treatment, split.Rule = "CT",
+                    split.Honest = T, cv.option = "matching", cv.Honest = F,
+                    split.Bucket = F, xval = 10)
> tree$cptable
             CP nsplit rel error    xerror        xstd
1  1.145837e-02      0 1.0000000 1.0000000 0.002544452
2  1.941622e-03      1 0.9885416 0.5141833 0.001487195
3  1.862875e-03      2 0.9866000 0.5129707 0.001411806
4  1.031403e-03      3 0.9847371 0.4762972 0.001260437
5  7.763028e-04      5 0.9826743 0.4471484 0.001116234
6  5.128562e-04      8 0.9803454 0.4203597 0.001078958
7  4.327552e-04     10 0.9793197 0.4197124 0.001068104
8  3.525900e-04     11 0.9788870 0.4226628 0.001067524
9  3.052253e-04     13 0.9781818 0.4302321 0.001095403
10 2.632508e-04     14 0.9778766 0.4621311 0.001207203
11 2.198657e-04     16 0.9773500 0.4899981 0.001275376
```

```
12 1.945478e-04      17 0.9771302 0.5248392 0.001358469
13 1.887622e-04      18 0.9769356 0.5248630 0.001364536
14 1.792430e-04      19 0.9767469 0.5342639 0.001361384
15 1.677139e-04      20 0.9765676 0.5296446 0.001341866
16 1.366938e-04      25 0.9756676 0.5243033 0.001334980
17 1.111979e-04      28 0.9752575 0.5361940 0.001354982
18 1.096178e-04      32 0.9747981 0.5363380 0.001346708
19 1.002128e-04      36 0.9743597 0.5360979 0.001347304
20 7.376079e-05      37 0.9742595 0.5436176 0.001345552
21 4.858865e-05      38 0.9741857 0.5597567 0.001375235
22 2.870558e-05      39 0.9741371 0.5652205 0.001389057
23 0.000000e+00      40 0.9741084 0.5682949 0.001396325
> rpart.plot(tree)
```



The built tree in the plot is large and deep. Like `rpart`, we choose the complexity paramter `opcp` corresponding to the minimum cross validation error (`xerror`) in `tree$cptable`, and use the function `prune()` to trim the tree:

```
> opcp <- tree$cptable[, 1][which.min(tree$cptable[,4])]
> optree <- prune(tree, cp = opcp)
> rpart.plot(optree)
```

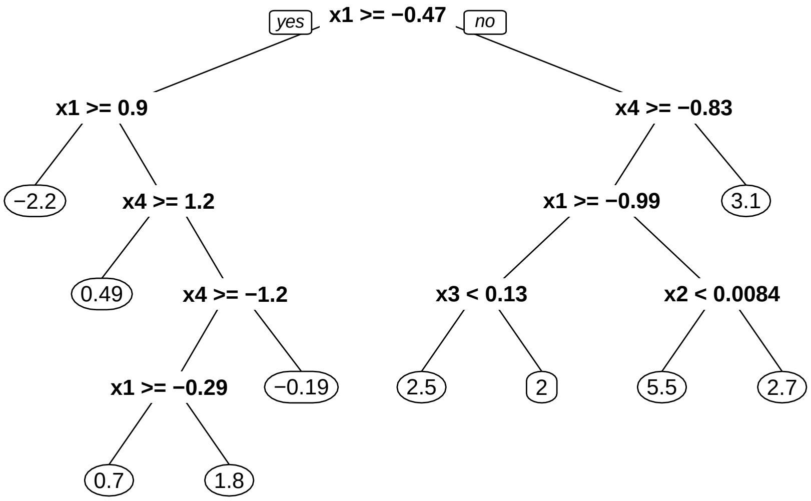


5 Honest Estimation

In addtion to `causalTree`, we also support one-step honest re-estimation in function `honest.causalTree`. It can fit a `causalTree` model and get honest estimation results with tree structre built on training sample (including cross validation) and leaf treatment effect estimates taken from estimation sample.

5.1 Example

```
> n <- nrow(simulation.1)
> trIdx <- which(simulation.1$treatment == 1)
> conIdx <- which(simulation.1$treatment == 0)
> train_idx <- c(sample(trIdx, length(trIdx) / 2),
+                sample(conIdx, length(conIdx) / 2))
> train_data <- simulation.1[train_idx, ]
> est_data <- simulation.1[-train_idx, ]
> honestTree <- honest.causalTree(y ~ x1 + x2 + x3 + x4,
+                       data = train_data,
+                       treatment = train_data$treatment,
+                       est_data = est_data,
+                       est_treatment = est_data$treatment,
+                       split.Rule = "CT", split.Honest = T,
+                         HonestSampleSize = nrow(est_data),
+                         split.Bucket = T, cv.option = "fit",
+                         cv.Honest = F)
> opcp <-  honestTree$cptable[,1][which.min(honestTree$cptable[,4])]
> opTree <- prune(honestTree, opcp)
> rpart.plot(opTree)
```



6 Causal Forests

6.1 Example

In this section, we provide an example which illustrates how a causalForest is built, as well as honest estimation of the treatment effects using the built ensemble.

```
>cf <- causalForest(as.formula(paste("y~",f)), data=dataTrain,
+    treatment=dataTrain$w,
+    split.Rule="CT", split.Honest=T,  split.Bucket=F, bucketNum = 5,
+    bucketMax = 100, cv.option="CT", cv.Honest=T, minsize = 2L,
+    split.alpha = 0.5, cv.alpha = 0.5,
+    sample.size.total = floor(nrow(dataTrain) / 2), sample.size.train.frac = .5,
+    mtry = ceiling(ncol(dataTrain)/3), nodesize = 3, num.trees= 5)
>cfpredtest <- predict(cf, newdata=dataTest, type="vector")
>plot(dataTest$tau_true,cfpredtest)
```

In the above example, the causalForest function is called using the split.Rule ="CT", with 5 trees, and minimum node size of 3. For building each tree, we specify the data sample size for building each tree (sample.size.total), as well as fraction of the data used for training (sample.size.train.frac). Note that (1-sample.size.train.frac) fraction of the sample.size.total data set is used for evaluating the treatment effects of the built trees. To predict the treatment effect for a new set of data (dataTest), R's default predict() function is called, and the built causalForest object is passed in. The variable cfpredtest will be populated with the treatment effects predicted for each data point in the dataTest testing data vector. Internally, the causalForest function in turn calls the function honest.causalTree in a loop num.trees number of times, sampling the data set each time to pick the user specified number of data points to build the tree and for honest estimation, and collects the ensemble of trees as a list, which it returns.

7 Propensity Forests

The propensity forest is an ensemble method similar to a causal forest. As discussed above, causalForest builds an ensemble of causalTrees, by repeated random sampling of the data with replacement. For prediction, the average value over all the tree predictions is used. The propensity forest technique *differs* from a causal forest in the following ways: The tree building phase is done by using the covariates and treatment vector as the dummy output/outcomes variable (instead of the actual outcomes variable). During the tree evaluation phase, however, the reestimation error is calculated on the actual outcomes variable to evaluate the tree performance. Note that this is done using the same dataset used for building the trees, thus there is no honest estimation. Please refer to *Estimation and Inference of Heterogeneous Treatment Effects using Random Forests* [3] for more details.

7.1 Example

An example call to the propensityForest function is shown below.

```
>pf <- propensityForest(as.formula(paste("y~",f)),
+         data=dataTrain, treatment=dataTrain$w,
+         split.Bucket=F,
+         sample.size.total = floor(nrow(dataTrain) / 2),
+         mtry = ceiling(ncol(dataTrain)/3),
+         nodesize = 25, num.trees=num.trees.temp)
>pfpredtest <- predict(pf, newdata=dataTest, type="vector")
```

References

[1] Susan Athey and Guido Imbens. Machine learning methods for estimating heterogeneous causal effects. *arXiv preprint arXiv:1504.01132*, 2015.

[2] L. Breiman, J.H. Friedman, R.A. Olshen, , and C.J Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Ca, 1983.

[3] Stefan Wager and Susan Athey. Estimation and inference of heterogeneous treatment effects using random forests. *arXiv preprint arXiv:1510.04342*, 2015.