ITR Forest: Constructing a Treatment Decision Rule

Kevin Doubleday

04 February, 2018

Introduction

The package ITR.forest creates an individualized treatment decision rule using a recursive partitioning algorithm to grow a decision tree. The algorithm utilizes either an inverse probability weighted estimator (IPWE) or augmented inverse probability weighted estimator (AIPWE) as the reward function, maximizing the selected function at each split. *The current version of this package can be applied to data from completed randomized controlled trials (RCTs) or an electronic medical record (EMR) database where there are two treatment options, a continuous outcome, and continous or categorical predictors.* Additionlly, each patient needs to have an estimated propensity score, or probability of being assigned to treatment, $\hat{p}(t_i|\mathbf{x}_i)$. For an RCT with 1:1 treatment allocation, for example, $\hat{p}(t_i|\mathbf{x}_i) = 0.5$. For EMR data, $\hat{p}(t_i|\mathbf{x}_i)$ can be found using logistic regression. Without loss of generality, we assume that larger values of the outcome are better.

The IPWE is estimated from the data as

$$\hat{V}_{IPWE}(r) = \left(\sum_{i=1}^{N} \frac{\mathbf{I}_{t_i = r(\mathbf{x}_i)}}{\hat{p}(t_i|\mathbf{x}_i)} y_i \right) / \left(\sum_{i=1}^{N} \frac{\mathbf{I}_{t_i = r(\mathbf{x}_i)}}{\hat{p}(t_i|\mathbf{x}_i)} \right).$$

The AIPWE is estimated from the data as

$$\hat{V}_{AIPWE}(r) = \hat{V}_{IPWE}(r) - \left(\sum_{i=1}^{N} \frac{\mathbf{I}_{t_i = r(\mathbf{x}_i)} - \hat{p}(t_i|\mathbf{x}_i)}{\hat{p}(t_i|\mathbf{x}_i)} m(X_i) \right),$$

where $m(X_i) = \mu(t_i = 1, X_i) \cdot z_i + \mu(t_i = 0, X_i) \cdot (1 - z_i)$ is the estimated mean in a given node and is assigned based on the patient's original treatment assignment $t_i \in \{0, 1\}$ and the new treatment assignment under consideration $z_i \in \{0, 1\}$.

Further details can be found in "A Novel Algorithm for Generating Individualized Treatment Decision Trees and Random Forests" by Doubleday and Zhou (*In Review*).

The input dataset needs to have the following: (1) propensity score column labeled `prtx`, (2) binary treatment column labeled `trt`, (3) ID column labeled `id`, (4) continuous outcome column labeled `y`, and (5) columns of predictors.

Growing a Tree with Continuous Predictors

A single ITR tree can be constructed using the function `grow.ITR` which requires an input dataset and a set of splitting variable columns. By default the number of observations allowed in a terminal node is 20 (`min.ndsz = 20`), there must be at least 5 observations from each treatment group in a terminal node (`n0 = 5`), and the maximum tree depth is set at 15 (`max.depth = 15`). The initial value of the root node is the maximum of $\hat{V}(r)$ with all subjects given treatment or all patients given control. A initial split of the root node is only made if there is a split for which the value in the root node increases. The same is true for additional splits so that the tree cannot grow larger unless there is an increase in overall value of the tree given by the split.

Growing a Tree

We will use the following example dataset generated from the function `gdataM()` which simulates EMR data from the model

$$Y = 1 + 2X_2 + 4X_4 + \beta_1 * T * (\mathbf{X} \in A) + \beta_2 * (1 - T) * (\mathbf{X} \notin A) + \epsilon.$$

where $A = \{X_1 > 0.3 \cap X_3 > 0.1\}$ when `depth=2` is specified. Covariates $X_1 - X_4 \sim \text{Unif}(0,1)$, errors follow $N(0,1)$, treatment assignments are $T \in \{0,1\}$, `prtx` is the propensity score, and the signal to noise ratio θ is defined as β_1/β_2.

```r
set.seed(123)
dat <- gdataM(n = 2500, beta1 = 3, beta2 = 1, depth = 2)
head(dat)
```

```
##     X1   X2   X3   X4        y trt  prtx id
## 1 0.30 0.86 0.26 0.08 4.215369   0 0.142  1
## 2 0.80 0.68 1.00 0.72 7.964857   1 0.968  2
## 3 0.42 0.74 0.72 0.86 4.827130   0 0.703  3
## 4 0.90 0.32 0.66 0.68 8.038266   1 0.881  4
## 5 0.96 0.68 0.26 0.76 6.911391   1 0.545  5
## 6 0.06 0.48 0.10 0.78 8.211139   0 0.035  6
```

```r
set.seed(10)
covExtra<-matrix(sample(1:25, size = 10*nrow(dat), replace = T), nrow=nrow(dat))/25
colnames(covExtra)<-paste("E", seq(1,10), sep = "")

dat <- cbind(dat, covExtra)
```

This dataset has 2500 observations with $\theta = 3$. The argument `depth=2` indicates that there are two subgroup defining variables, X_1 and X_3. Changing this argument to `depth=1` would change the subgroup definition to $X_1 < 0.5$, having only a single interacting covariate. Additionally, there are 10 noise variables added which are not shown above and which users can generate on their own if they would like. The tree is constructed using the `grow.ITR()` function as follows with the AIPWE since we are analyzing an EMR dataset.

```r
tre <- grow.ITR(data = dat, split.var = c(1:4, 9:18), AIPWE = TRUE)
tre
```

```
##       node size  n.1  n.0 trt.effect var vname cut.1 cut.2  score
## 1        0 2500 1294 1206  1.2540902   1    X1     r   0.3 0.7426
## 3       01  758  248  510 -0.9064623  12    E4     l  0.04 0.8245
## 6      011   34    9   25 -0.9289875  NA  <NA>  <NA>  <NA>     NA
## 5      012  724  239  485 -0.9053264  11    E3     l  0.04 0.8260
## 8     0121   25    9   16 -0.3540211  NA  <NA>  <NA>  <NA>     NA
## 9     0122  699  230  469 -0.9250415  18   E10     r  0.96 0.8279
## 11   01221  678  221  457 -0.9461866   1    X1     l  0.02 0.8292
## 13  012211   45    7   38 -0.4767850  NA  <NA>  <NA>  <NA>     NA
## 12  012212  633  214  419 -0.9734814  NA  <NA>  <NA>  <NA>     NA
## 10   01222   21    9   12 -0.3644231  NA  <NA>  <NA>  <NA>     NA
## 2       02 1742 1046  696  2.1685733   3    X3     r   0.1 0.8225
## 4      021  168   28  140 -1.4782590  NA  <NA>  <NA>  <NA>     NA
## 7      022 1574 1018  556  2.5396112  NA  <NA>  <NA>  <NA>     NA
```

The output contains a summary of the tree structure. The `node` column begins with the root node `0` and each subsequent number indicates the direction of the split, with `1` indicating the left (less than or equal to) node and `2` indicating the right (greater than) node. The first row indicates that the covariate X_1 is selected as the first splitting variable with a cut point of `cut.2 = 0.3`. The decision is to send treatment to the right node (`cut.1 = "r"`). `size`, `n.1`, and `n.0` indicate there are 2500 observations in the root node, with 1294 treated and 1206 on control. The second row with `node = 01` contains information from the left child node with interpretations similar to those described for the root node. The splitting information denoted by `NA` indicates a terminal node, `011` for instance.

Note that in the case of this simulated data (depth 2) the correct tree structure splits the root node at $X_1 = 0.3$ and sends treatment to node `02`. Next, node `02` should be split a t $X_3 = 0.1$ and sends treatment to the right. We see that this tree structure represents this well, but has some additional splits which are not necessary.

Pruning a Tree

To avoid overfitting, a pruning procedure is introduced to penalize additional splits in the tree growing process. The function `prune(tre, a, train, test)` accomplishes this with penalty $\lambda = a$ applied using the weakest link criteria. The penalty is applied to subtree Γ.

$$V_\lambda(\Gamma) = V(\Gamma) - \lambda \cdot \left|\Gamma - \tilde{\Gamma}\right|$$

where $\left|\Gamma - \tilde{\Gamma}\right|$ is the number of internal nodes of subtree Γ, $V(\Gamma)$ is the value of the entire subtree, and $V_\lambda(\Gamma)$ is the penalized value. We trim the weakest branches first which are those with (1) the greatest number of parent nodes, and (2) contributes the smallest additional value to the tree. We can prune a tree as follows. The following example uses the tree above (`tre`) with a penalty of 0.05.

```
pruned <- prune(tre, a = 0.05, train = dat)
pruned
```

```
##   subtree node.rm size.tree size.tmnl  alpha      V    V.a V.test Va.test
## 1       1      01        13         7 6.2855 6.3001 6.0001   <NA>    <NA>
## 2       2      02         5         3 6.1903 6.2855 6.1855   <NA>    <NA>
## 3       3       0         3         2 5.9411 6.1903 6.1403   <NA>    <NA>
## 4       4      NA         1         1   9999 5.9411 5.9411   <NA>    <NA>
```

The first row represents the entire tree (`subtree 1`) and summarizes the value, penalized value, and weakest node (node to be removed next) by `V`, `V.a`, and `node.rm`. The `size.tree` and `size.tmnl` columns give the total number of nodes and terminal nodes in a given subtree. We want to select the tree with the highest penalized value, which would correspond to subtree 2 with a penalized value of `V.a = 6.1855`. This subtree has 3 terminal nodes.

Cross Validation for Model Selection

One issue with the approach above for model selection is the risk of overfitting through using the training data alone for model selection. The function `treeCV()` will perform n-fold cross validation to select the optimal tuning parameter (λ). The function returns the optimal model, selected penalty, and several summary measures.

```
cv.model.select <- treeCV(tre = tre, dat = dat, nfolds = 5, sp.var = c(1:4, 9:18),
                          param = seq(0, 1, 0.01), sort = FALSE, AIPWE = TRUE)
```

```
## [1] 0.04
##   node size  n.1  n.0 trt.effect var vname cut.1 cut.2  score
## 1    0 2500 1294 1206  1.2540902   1    X1     r   0.3 0.7426
## 3   01  758  248  510 -0.9064623  NA  <NA>  <NA>  <NA>     NA
## 2   02 1742 1046  696  2.1685733   3    X3     r   0.1 0.8225
## 4  021  168   28  140 -1.4782590  NA  <NA>  <NA>  <NA>     NA
## 7  022 1574 1018  556  2.5396112  NA  <NA>  <NA>  <NA>     NA
```

The optimal lambda selected is 0.04 and the optimal tree has 3 terminal nodes. This tree structure corresponds to the correct tree structure for this particular simulation. Figure 1 shows the results of the cross validation as λ increases.

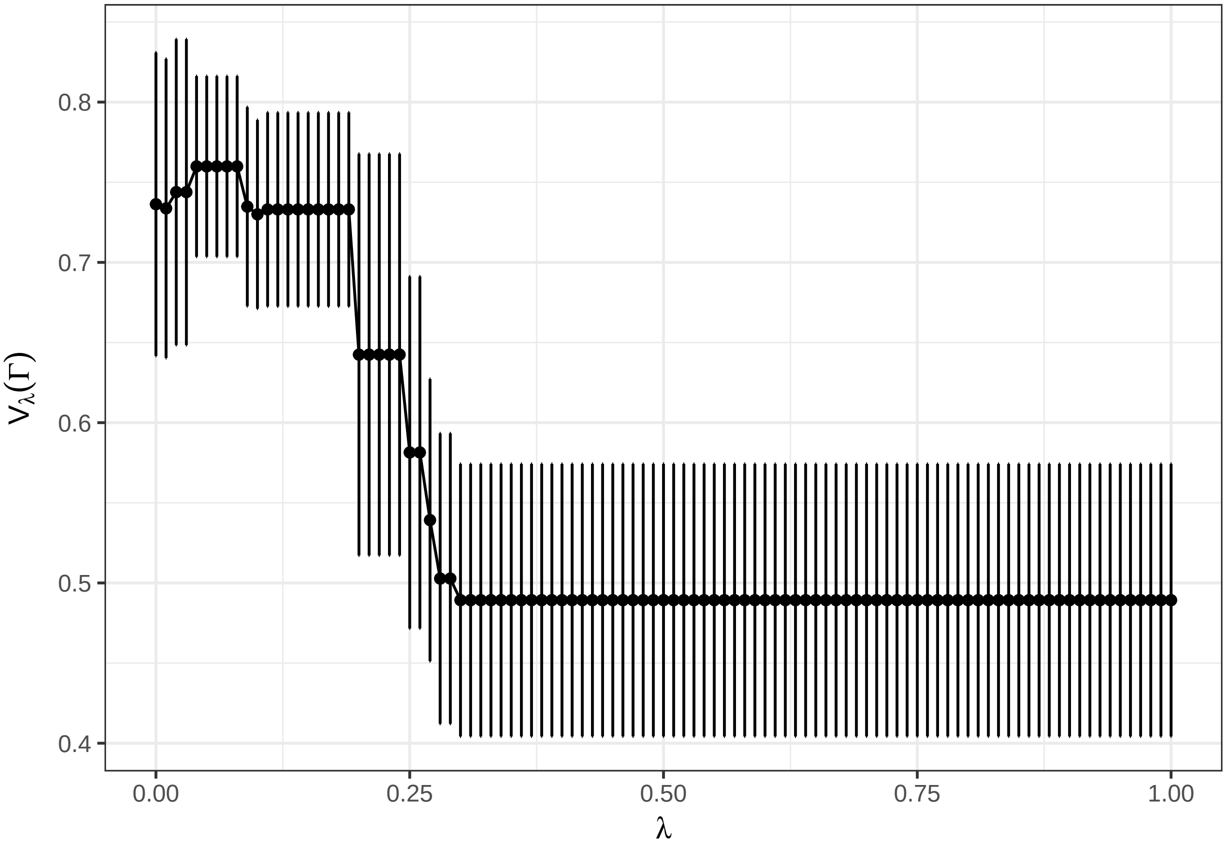

Figure 1. Cross validated value versus λ with 95% error bars.

Constructing an ITR Forest to Give Decision Rule

A single tree which is trained using all available data may be overfitted and not extendable to subsequent observations. Hence, we make a decision rule using a forest of ITR trees in which each tree is more variable, but the aggregation of the trees in the forest mitigates this variance. The ITR forest is contructed using the function `Build.RF.ITR()` and requires the entry of a dataset, columns for the outcome, treatment, propensity score, and splitting variables. To randomized the growth of trees in the forest a subset of predictors, `mtry`, is selected as potential splitting variables at each split which defaults to the maximum of 1/3 the number of splitting variables and 1. The number of observations and the number of treated and control subjects allowed in a terminal node is given by `N0` and `n0`. By default the number of trees contructed, `ntree`, is 500. Each tree is grown using a bootstrap sample taken from the input dataset. The function returns the bootstrap samples used in tree construction, the trees, and the model parameters. The forest is contructed as follows and the first two trees are displayed as examples.

```
set.seed(2)
forest <- Build.RF.ITR(dat, split.var = c(1:4, 9:18), col.y="y", col.trt="trt",
                        col.prtx="prtx", ntree = 500)
forest$TREES[1:2]
```

```
## [[1]]
##   node size n.1 n.0 trt.effect var vname cut.1 cut.2  score
## 1    0 1583 813 770  1.2594453   1    X1     r   0.3 0.7173
## 2   01  483 160 323 -0.9498811  NA  <NA>  <NA>  <NA>     NA
## 3   02 1100 653 447  2.1869441   3    X3     r   0.1 0.8139
## 5  021  105  17  88 -1.8317697  NA  <NA>  <NA>  <NA>     NA
## 4  022  995 636 359  2.5747458  NA  <NA>  <NA>  <NA>     NA
##
## [[2]]
##   node size n.1 n.0 trt.effect var vname cut.1 cut.2 score
## 1    0 1576 831 745   1.309622  NA    NA    NA    NA    NA
```

We see that the first tree corresponds to the expected result and the second tree is a null tree. If the user wants, the generation of null trees can be avoided using the additional argument `avoid.nul.tree = TRUE`. We can now run a new observation down each of the trees and obtain a vote from each tree as to what the treatment assignment should be. The majority vote from the forest will be the ITR forest decision rule. This can be obtained using the `predict.ITR()` function. First we generate a new observation and second we make a treatment predition for this new observation. Shown is the treatment summary as the proportion of trees voting for treatment and the voting record for the first 10 trees.

```
set.seed(10)
new.obs <- gdataM(n = 1,depth = 2,beta1 = 3, beta2 = 1)
new.obs
```

```
##     X1   X2   X3  X4        y trt prtx id
## 1 0.52 0.32 0.44 0.7 3.834144   0 0.44  1
```

```
new.obs <- cbind(new.obs, matrix(sample(1:25, size = 10, replace = TRUE)/25, ncol = 10))
colnames(new.obs) <- colnames(dat)
preds <- predict.ITR(forest, new.obs)
preds$SummaryTreat
```

```
## [1] 1
```

```
preds$tree.votes[1:20]
```

```
##  [1]  1 NA  1  1 NA  1 NA  1  1  1 NA  1  1  1  1  1 NA  1 NA NA
```

Note that the observation should receive treatment since $X_1 > 0.3$ and $X_3 > 0.1$. This forest has 100% of the trees voting that the patient should receive treatment which would be a good decision for this patient since they are in the subgroup which benefits from treatment. Note that several of the trees return votes of `NA`. This means that the tree did not make an initial split, or was a null tree.

Predictions for outside data

Use the `predict.ITR()` function to make treatment predictions for a forest object created using `Build.RF.ITR()` or a single tree object created using `grow.ITR()`. For a forest, the output includes the proportion of trees voting for the treatment (`trt = 1`), votes from all the trees, the number of null trees in the forest, and a summary of the data and trees. An example for a single tree is shown below using the tree `tre` from above and 4 newly generated observations.

```
set.seed(1)
new.dat <- gdataM(4, 2, 1, 1)
new.dat <- cbind(new.dat, matrix(sample(1:25, size = 10, replace = TRUE)/25, ncol = 10))
colnames(new.dat) <- colnames(dat)
preds <- predict.ITR(tre, new.dat)
new.dat
```

```
##     X1   X2   X3   X4        y trt  prtx id  E1   E2   E3   E4   E5   E6
## 1 0.28 0.22 0.64 0.70 6.364931   0 0.510  1 0.8 0.12 0.76 0.44 0.84 0.68
## 2 0.38 0.90 0.08 0.40 4.789843   1 0.079  2 0.8 0.12 0.76 0.44 0.84 0.68
## 3 0.58 0.96 0.22 0.78 6.023810   0 0.239  3 0.8 0.12 0.76 0.44 0.84 0.68
## 4 0.92 0.68 0.18 0.50 3.145300   1 0.416  4 0.8 0.12 0.76 0.44 0.84 0.68
##    E7   E8   E9 E10
## 1 0.8 0.56 0.56 0.8
## 2 0.8 0.56 0.56 0.8
## 3 0.8 0.56 0.56 0.8
## 4 0.8 0.56 0.56 0.8
```

```
preds$SummaryTreat
```

```
## [1] 0 0 1 1
```

Each element of `SummaryTreat` is the treatment decision for one of the new observations.

Variable Importance

Last, we include a function to calculate the importance of a predictor in making the treatment assignment for an ITR forest. The variable importance is calculated by determining the out of bag (OOB) value $V_{OOB}(r)$ for the sample not used in tree construction (OOB sample), permuting the variable values for any predictor used in the tree construction, and re-running the OOB sample down the tree to obtain $V_{OOBpermuted}(r)$. The larger the difference between $V_{OOB}(r)$ and $V_{OOBpermuted}(r)$ the more important the predictor. This is done for each tree in the forest and importance measures for each variable are summed. We scale the measure to be out of 1 for easy interpretibility. This is done using the function `Variable.Importance.ITR()`.

```
VI <- Variable.Importance.ITR(forest)
VI
```

```
##           X1           X3           X4           E5           E8 
## 0.7064653941 0.2561928760 0.0209976418 0.0044585622 0.0027144093 
##           X2           E3           E9           E7           E1 
## 0.0022568726 0.0016150340 0.0015710675 0.0012761554 0.0010778375 
##           E2          E10           E4           E6 
## 0.0005656072 0.0003416131 0.0002766305 0.0001902988
```

We see that the variable forming the interaction subgroup, X_1 and X_3, are returned as the most important predictors.

A Note on Numerical Consistency

References

[In Review] Doubleday, K., Zhou, J., Fu, H. (2017), “A Novel Algorithm for Generating Individualized Treatment Decision Trees and Random Forests,” *Journal of Computational and Graphical Statistics.*