ITR Forest: Constructing a Treatment Decision Rule

Kevin Doubleday

June 1, 2017

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 $z_i \in \{0, 1\}$.

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

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.

Example - Growing a Tree

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

$$Y = 1 + 2X_2 + 4X_4 + \beta_1 * sign(X_1 > 0.3 \text{ and } X_3 > 0.1) * T + \beta_2 * sign(X_1 \leq 0.3 \text{ or } X_3 \leq 0.1) * (1 - T) + \epsilon.$$

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(1)
dat <- gdataM(n=1000, beta1=1, beta2=1, depth=2)
head(dat)
```

```
##     X1   X2   X3   X4        y trt  prtx id
## 1 0.28 0.54 0.88 0.82 3.554516   1 0.776  1
## 2 0.38 0.70 0.98 0.48 4.641959   1 0.885  2
## 3 0.58 0.40 0.88 0.18 3.046642   1 0.895  3
## 4 0.92 0.96 0.44 0.40 6.547417   1 0.723  4
## 5 0.22 0.12 0.20 0.82 3.922612   1 0.088  5
## 6 0.90 0.04 0.10 0.96 5.212572   0 0.310  6
```

This dataset has 1000 observations with $\theta = 1$. The argument `depth=2` indicates that there are two subgroup defining variables, X_1 and X_3. Changing this arugment do `depth=1` would change the subgroup definition to $X_1 < 0.5$, having only a single interacting covariate. The tree is constructed using the `grow.ITR()` function as follows.

```r
tre <- grow.ITR(data = dat, split.var = 1:4)
tre
```

```
##    node size n.1 n.0 trt.effect var vname cut.1 cut.2  score
## 1     0 1000 479 521  0.3040730   1    X1     r  0.28 4.9968
## 3    01  277  86 191 -0.8692737   4    X4     l  0.24 5.1101
## 5   011   78  18  60 -0.6519866   2    X2     l  0.66 5.1391
## 10 0111   55  12  43 -0.4069918  NA  <NA>  <NA>  <NA>     NA
## 8  0112   23   6  17 -1.3360101  NA  <NA>  <NA>  <NA>     NA
## 7   012  199  68 131 -1.2316641  NA  <NA>  <NA>  <NA>     NA
## 2    02  723 393 330  0.7079357   3    X3     r   0.1 5.0773
## 6   021   78   9  69 -1.6635469  NA  <NA>  <NA>  <NA>     NA
## 4   022  645 384 261  1.0149507   1    X1     r  0.32 5.1104
## 11 0221   28  10  18 -0.3549501  NA  <NA>  <NA>  <NA>     NA
## 9  0222  617 374 243  1.0703235  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 best splitting variable with a cut point of 0.28 (`cut.2=0.28`). The decision is the send treatment to the right node (`cut.1="r"`). `size`, `n.1`, and `n.0` indicate there are 1000 observations in the root node, with 479 treated and 521 on control. The second row with `node=01` contains information from the left child node with interpretations like those for the root node. The splitting inforation displayed as `NA` indicates a terminal node. For instance, node `0111` is a terminal node.

Note that in the case of this simulated data 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 at $X_3 = 0.1$ 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.

```
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     022        11         6 5.1392 5.1391 4.8891   <NA>    <NA>
## 2       2     011         9         5 5.1101 5.1392 4.9392   <NA>    <NA>
## 3       3      01         7         4 5.0773 5.1101 4.9601   <NA>    <NA>
## 4       4      02         5         3 4.9968 5.0773 4.9773   <NA>    <NA>
## 5       5       0         3         2  4.368 4.9968 4.9468   <NA>    <NA>
## 6       6      NA         1         1   9999 4.6906 4.6906   <NA>    <NA>
```

The output contains information about the size of the tree such as the number of terminal nodes (`n.tmnl`) and the total number of nodes in the tree (`size.tree`). The first row represents the entire tree and summarizes the value, penalized value, and weakest node (node to be removed next) by `V`, `V.a`, and `node.rm`. The value of the penalty λ was selected as 0.05 by a validation technique. We want to select the tree with the highest penalized value, which would correspond to subtree 4 with a penalized value of `V.a=4.9773`. This subtree has 3 terminal nodes corresponding to the correct tree structure.

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 = 1:4, col.y="y", col.trt="trt",
                       col.prtx="prtx", N0=5, n0=2, ntree = 100)
forest$TREES[1:2]
```

```
## [[1]]
##   node size n.1 n.0 trt.effect var vname cut.1 cut.2  score
## 1    0  637 309 328  0.3392574   1    X1     r   0.3 5.0082
## 3   01  201  71 130 -0.6981351  NA  <NA>  <NA>  <NA>     NA
## 2   02  436 238 198  0.7712967  NA  <NA>  <NA>  <NA>     NA
##
## [[2]]
##        node size n.1 n.0 trt.effect var vname cut.1 cut.2  score
## 1         0  648 307 341  0.2218311   4    X4     l   0.8 4.7088
## 2        01  508 247 261  0.3235519   3    X3     r   0.1 4.7554
## 6       011   53   5  48 -1.5019299   4    X4     l  0.42 4.9724
## 12     0111   31   2  29 -1.9865891  NA  <NA>  <NA>  <NA>     NA
## 8      0112   22   3  19 -1.8904778  NA  <NA>  <NA>  <NA>     NA
## 4       012  455 242 213  0.5047220   2    X2     r  0.36 4.9586
## 13     0121  169  98  71  0.2480356   4    X4     l  0.06 5.0618
## 15    01211   20  11   9  0.5083127  NA  <NA>  <NA>  <NA>     NA
## 17    01212  149  87  62  0.1888384   1    X1     r  0.32 5.1822
## 19   012121   49  23  26 -1.3363680  NA  <NA>  <NA>  <NA>     NA
## 20   012122  100  64  36  0.9257233   2    X2     l   0.3 5.1974
## 22 0121221   85  52  33  0.8656012  NA  <NA>  <NA>  <NA>     NA
## 23 0121222   15  12   3  1.3564121  NA  <NA>  <NA>  <NA>     NA
## 9      0122  286 144 142  0.7636885   1    X1     r  0.28 5.0355
## 16    01221   85  24  61 -0.7300302   3    X3     l  0.36 5.0799
## 21   012211   25   2  23  0.6461527  NA  <NA>  <NA>  <NA>     NA
## 18   012212   60  22  38 -0.7897891  NA  <NA>  <NA>  <NA>     NA
## 14    01222  201 120  81  1.4609421  NA  <NA>  <NA>  <NA>     NA
## 3        02  140  60  80  0.1869079   1    X1     r   0.5 4.8737
## 5       021   69  22  47 -0.5557305  NA  <NA>  <NA>  <NA>     NA
## 7       022   71  38  33  1.0088165   1    X1     l  0.94 4.9751
## 11     0221   64  34  30  1.1365307  NA  <NA>  <NA>  <NA>     NA
## 10     0222    7   4   3 -0.2098627  NA  <NA>  <NA>  <NA>     NA
```

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(1,2,1,1)
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
```

```
preds <- predict.ITR(forest, new.obs)
preds$SummaryTreat
```

```
## [1] 0.875
```

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

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

Note that the observation should receive treatment since $X_1 > 0.3$ and $X_3 > 0.1$. This forest has about 87%

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. If we want to avoid having null trees in the ITR forest we can specify `avoid.nul.tree=T`. We will repeat the above analyses while avoiding null trees.

```r
set.seed(2)
forest2 <- Build.RF.ITR(dat, split.var = 1:4, col.y="y", col.trt="trt",
                        col.prtx="prtx", N0=5, n0=2, ntree = 100, avoid.nul.tree = T)
set.seed(10)
new.obs <- gdataM(1,2,1,1)
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
```

```r
preds <- predict.ITR(forest2, new.obs)
preds$SummaryTreat
```

```
## [1] 0.89
```

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

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

Now the treatment probability raises to 0.89 and we have no null tree votes.

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(10)
new.dat <- gdataM(4, 2, 1, 1)
preds <- predict.ITR(tre, new.dat)
new.dat
```

```
##     X1   X2   X3   X4        y trt  prtx id
## 1 0.52 0.10 0.62 0.12 3.781780   1 0.659  1
## 2 0.32 0.24 0.44 0.60 3.969347   0 0.302  2
## 3 0.44 0.28 0.66 0.36 3.761766   1 0.650  3
## 4 0.70 0.28 0.58 0.44 3.124850   0 0.731  4
```

```
preds$SummaryTreat
```

```
## [1] 1 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        X2
## 0.5995756 0.1925664 0.1790094 0.0288486
```

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