An Introduction to the GPLTR package

Cyprien Mbogning
Inserm UNIT 1178, France

October 25, 2024

Contents

1 Introduction

This document is intended to give a short overview of the methods found in the `GPLTR` package. The acronym `GPLTR` is designed for — Generalized Partially Linear Tree-based Regression model —.

The `GPLTR` programs build classification or regression models of a very general structure using a three stage procedure with several additional tools (see (Mbogning et al. 2014)); the resulting model is an hybrid model combining a generalized linear model with an additional tree part on the same scale. The model was first proposed by (Chen et al. 2007) for genetic epidemiology studies in order to assess complex joint gene-gene and gene-environment effects taking into account confounding variables. In practice, the GPLTR models represent a new class of semi-parametric regression models that integrates the advantages of generalized linear regression and tree-structure models. To our best knowledge, there is currently no

implemented package dealing with this kind of model. The available classical tree-based methods do not provide a way for controlling confounding factors outside the tree part (the final tree is generally a mixture of confounders and explanatory variables lacking of clear interpretation and resulting in a distorted joint effect).

We also proposed an ensemble method (see (Mbogning et al. 2015)), mainly the bagging, to address a classical concern of tree-based methods, their instability. the Bagged GPLTR is proposed with several scores of variable importance for prediction and variable selection in the framework of the GPLTR model.

2 GPLTR model

Denote $\mathbf{Y}$ the outcome of interest, $\mathbf{X}$ a set of confounding variables, and $\mathbf{G}$ the explanatory variables. The model fitted inside the `GPLTR` package is specified by:

$$g\left(\mathbb{E}\left(\mathbf{Y}|\mathbf{X},\mathbf{G}\right)\right) = \mathbf{X}'\theta + \beta_T F\left(T\left(\mathbf{G}\right)\right), \tag{1}$$

where $g(\cdot)$ is a known link function (generalized linear model), $F\left(T\left(\mathbf{Z}\right)\right)$ is a vector of indicator variables representing the leaves of the tree $T\left(\mathbf{G}\right)$.

The variables considered in the linear part (confounding variables or variables we wish to control) of the model (1) have a direct impact on the structure of the tree, beginning by the split criterion and ending by the pruning procedure.

3 Fitting methods

The method we used in this package to fit the model (1) can be summarized into three major steps:

Step1 Fit the linear part and build a maximal tree within an iterative procedure by playing on several offsets terms. The nodes of the tree are splitted by maximizing a deviance criterion, while an intercept coefficient is fitted inside the node using the corresponding glm with the linear part considered as offset.

Step2 In order to prune back the maximal tree obtained previously, we use a forward procedure to build a sequence of nested subtrees.

Step3 The optimal tree is selected, using either a BIC criterion, a AIC criterion, a K-fold Cross-validation procedure on the underlying GPLTR models corresponding to the nested trees sequence. The original parametric bootstrap test procedure proposed by Chen et al. is also available.

We further propose a procedure to test the joint effect of the selected tree while adjusting for confounders. The users are encouraged to read the recent paper of (Mbogning et al. 2014) for a more thorough explanation about the model and the methods.

Table (1) represents a brief summary of the main functions available inside the `GPLTR` package. A more complete description is available inside the package documentation.

Table 1: Main function in the `GPLTR` package

Function	Description
pltr.glm	Fit an unprunned pltr model
best.tree.BIC.AIC	Prunned back the unprunned pltr with a BIC or AIC criterion
best.tree.CV	Prunned back the unprunned pltr with a CV procedure
best.tree.bootstrap	prunned back the unprunned pltr with a parametric bootstrap procedure
p.val.tree	Test the joint effect of the selected tree part while adjusting for confounders
bagging.pltr	Aggregate a set of pltr models
predict_bagg.pltr	Predict new features with bagging pltr predictor
VIMPBAG	compute several variable importance score with the bagging pltr predictor

4 Illustration via several examples

In the following, we will present the results obtained on the publicly available "burn" Data Set (Times to Infection for Burn Patients from the book of Klein and Moeschberger (Klein and Moeschberger 2003)). This dataset comes from a study (Ichida et al. 1993) that evaluates a protocol change in disinfectant practices for a cohort of 154 patients. A complete description of the data is also available inside the `GPLTR` package documentation.

In this example, the dependent variable is the administration of prophylactic antibiotic treatment ($D2$: yes/no), the confounding variable is the gender ($Z2$: male/female) and the potential explanatory variables are: ethnicity, severity of the burn as measured by percentage of total surface area of body burned, burn site (head, buttocks, trunk, upper legs, lower legs, respiratory tract), and type of burn (chemical, scald, electric, flame). In this analysis, we included gender as a confounding factor since this factor has already been described as related to infections among burn patients (Wisplinghoff et al. 1999). Such

adjustment for confounders cannot be performed within the classical CART framework.

Results obtained with the classical CART algorithm

First of all, we have fitted a classical tree model on the dependent variable $D2$, using the CART algorithm (Breiman et al. 1984) via the 'rpart' routines (Therneau and Atkinson 2013) of the R software:

```
> data(burn)
> head(burn, n = 10)
   Obs Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8 Z9 Z10 Z11 T1 D1 T2 D2 T3 D3
1    1  0  0  0 15  0  0  1  1  0   0   2 12  0 12  0 12  0
2    2  0  0  1 20  0  0  1  0  0   0   4  9  0  9  0  9  0
3    3  0  0  1 15  0  0  0  1  1   0   2 13  0 13  0  7  1
4    4  0  0  0 20  1  0  1  0  0   0   2 11  1 29  0 29  0
5    5  0  0  1 70  1  1  1  1  0   0   2 28  1 31  0  4  1
6    6  0  0  1 20  1  0  1  0  0   0   4 11  0 11  0  8  1
7    7  0  0  1  5  0  0  0  0  0   1   4 12  0 12  0 11  1
8    8  0  0  1 30  1  0  1  1  0   0   4  8  1 34  0  4  1
9    9  0  0  1 25  0  1  0  1  1   0   4 10  1 53  0  4  1
10  10  0  0  1 20  0  1  0  1  0   0   4  7  0  1  1  7  0
> rpart.burn <- rpart(D2 ~ Z1  + Z2 + Z3 + Z4 + Z5 + Z6 + Z7 + Z8 + Z9
                     + Z10 + Z11, data = burn, method = "class")
> print(rpart.burn)
n= 154

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 154 63 0 (0.5909091 0.4090909)
   2) Z4< 16.5 67 17 0 (0.7462687 0.2537313) *
   3) Z4>=16.5 87 41 1 (0.4712644 0.5287356)
     6) Z10< 0.5 52 21 0 (0.5961538 0.4038462)
      12) Z2< 0.5 41 13 0 (0.6829268 0.3170732)
        24) Z5>=0.5 18  3 0 (0.8333333 0.1666667) *
        25) Z5< 0.5 23 10 0 (0.5652174 0.4347826)
          50) Z6< 0.5 11  3 0 (0.7272727 0.2727273) *
          51) Z6>=0.5 12  5 1 (0.4166667 0.5833333) *
      13) Z2>=0.5 11  3 1 (0.2727273 0.7272727) *
     7) Z10>=0.5 35 10 1 (0.2857143 0.7142857) *
```

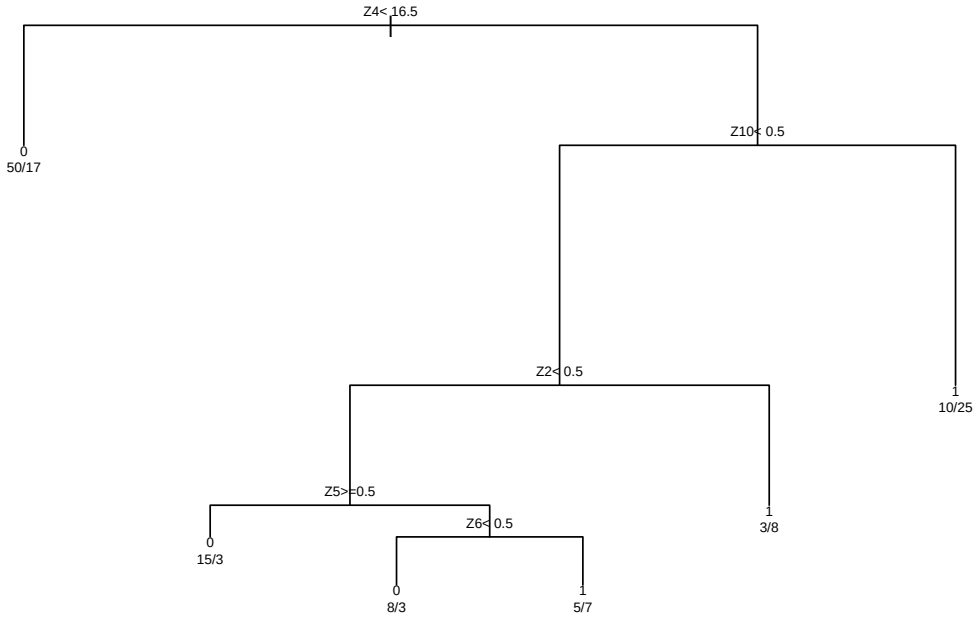


Figure 1: Tree obtained on the burn dataset with rpart.

```
> #par(mar = rep(0.1, 4))
>
> plot(rpart.burn)
> text(rpart.burn, xpd = TRUE, cex = .6, use.n = TRUE)
```

Figure (1) represents the tree obtained using the classical CART algorithm.

The tree is built by the following process: first the single variable is found which best splits the data into two groups (via the Gini index in the example). The data is separated, and then this process is applied separately to each sub-group, and so on recursively until the subgroups either reach a minimum size or until no improvement can be made.

Results obtained with our proposed method within the `GPLTR` package

A logistic partially linear tree-based regression model is fitted by using our proposed method:

```
> ## use example(GPLTR) to have a brief overview of the contain of the package.
>
> ## fit the PLTR model after adjusting on gender (Z2) using the proposed method
> ## ?GPLTR or ?pltr.glm to access the help section
>
> ## setting the parameters
>
> args.rpart <- list(minbucket = 10, maxdepth = 4, cp = 0,
                         maxcompete = 0, maxsurrogate = 0)
> family <- "binomial"
> X.names = "Z2"
> Y.name = "D2"
> G.names = c('Z1','Z3','Z4','Z5','Z6','Z7','Z8','Z9','Z10','Z11')
> ## Build the maximal tree with an adjustment  on gender (Z2)
>
> pltr.burn <- pltr.glm(burn, Y.name, X.names, G.names, args.rpart =
                        args.rpart, family = family,iterMax =8, iterMin = 6,
                          verbose = TRUE)
Iteration process...

Iteration  1 in PLTR; Diff_norm_gamma =  0.3450989
Iteration  2 in PLTR; Diff_norm_gamma =  0.1035102
Iteration  3 in PLTR; Diff_norm_gamma =  0.1555209
Iteration  4 in PLTR; Diff_norm_gamma =  0.04005871
Iteration  5 in PLTR; Diff_norm_gamma =  0.01202298
Iteration  6 in PLTR; Diff_norm_gamma =  0.003612611
Iteration  7 in PLTR; Diff_norm_gamma =  0.001085865
Iteration  8 in PLTR; Diff_norm_gamma =  0.0003264183
End of iteration process
Number of iterations:  8
> ## Prunned back the maximal tree using either the BIC or the AIC criterion
>
> pltr.burn_prun <- best.tree.BIC.AIC(xtree = pltr.burn$tree, burn, Y.name,
                          X.names, family = family, verbose = FALSE)
> ## Summary of the selected tree by a BIC criterion
>
> summary(pltr.burn_prun$tree$BIC)
Call:
rpart(formula = as.formula(paste(Y.name, " ~ ", paste(G.names,
```

```
    collapse = " + "), paste("+ offset(offsetX)"))), data = eval(parse(text = paste("data
    product, "hat_gamma)"))), method = method, control = args.rpart)
  n= 154

            CP nsplit rel error
1 0.0604978419      0 1.0000000
2 0.0408578707      1 0.9395022
3 0.0318009580      2 0.8986443
4 0.0261741219      3 0.8668433
5 0.0147618258      5 0.8144951
6 0.0073443205      6 0.7997333
7 0.0025318226      7 0.7923889
8 0.0002897679      8 0.7898571
9 0.0000000000      9 0.7895673

Variable importance
 Z1  Z4 Z10  Z6  Z5
 36  30  19  11   3

Node number 1: 154 observations,    complexity param=0.06049784
events = 63, coef = -0.6005625, deviance = 202.69720
  left son=2 (64 obs) right son=3 (90 obs)
  Primary splits:
      Z4 < 15.5 to the left,  improve=12.26274, (0 missing)

Node number 2: 64 observations,    complexity param=0.03180096
events = 16, coef = -1.3621820, deviance =  71.21259
  left son=4 (25 obs) right son=5 (39 obs)
  Primary splits:
      Z1 < 0.5  to the left,  improve=6.445964, (0 missing)

Node number 3: 90 observations,    complexity param=0.04085787
events = 47, coef = -0.1251768, deviance = 119.22180
  left son=6 (55 obs) right son=7 (35 obs)
  Primary splits:
      Z10 < 0.5  to the left,  improve=8.281774, (0 missing)

Node number 4: 25 observations
events = 2, coef = -2.6717880, deviance =  14.89458
```

```
Node number 5: 39 observations
events = 14, coef = -0.8618721, deviance =  49.87204

Node number 6: 55 observations,    complexity param=0.02617412
events = 22, coef = -0.6175948, deviance =  68.93242
  left son=12 (37 obs) right son=13 (18 obs)
  Primary splits:
      Z6 < 0.5  to the left,  improve=4.886466, (0 missing)

Node number 7: 35 observations
events = 25, coef =  0.6995323, deviance =  42.00763

Node number 12: 37 observations,    complexity param=0.02617412
events = 11, coef = -1.0794600, deviance =  42.43535
  left son=24 (20 obs) right son=25 (17 obs)
  Primary splits:
      Z1 < 0.5  to the left,  improve=5.724374, (0 missing)

Node number 13: 18 observations
events = 11, coef =  0.2519882, deviance =  21.61060

Node number 24: 20 observations
events = 3, coef = -2.0596390, deviance =  14.90882

Node number 25: 17 observations
events = 8, coef = -0.2338515, deviance =  21.80216
> ## Summary of the final selected pltr model
>
> summary(pltr.burn_prun$fit_glm$BIC)
Call:
glm(formula = xformula, family = family, data = xdata)

Coefficients:
                                                          Estimate
(Intercept)                                                0.6805
Z2                                                         1.1349
as.integer(Z4 < 15.5 & Z1 < 0.5)                          -3.3883
as.integer(Z4 < 15.5 & Z1 >= 0.5)                         -1.5766
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 < 0.5)  -2.7869
```

```
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 >= 0.5)  -0.9263
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 >= 0.5)             -0.4475
                                                          Std. Error
(Intercept)                                                   0.3882
Z2                                                            0.4686
as.integer(Z4 < 15.5 & Z1 < 0.5)                              0.8405
as.integer(Z4 < 15.5 & Z1 >= 0.5)                             0.5175
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 < 0.5)      0.7534
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 >= 0.5)     0.6241
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 >= 0.5)                0.6238
                                                          z value
(Intercept)                                                 1.753
Z2                                                          2.422
as.integer(Z4 < 15.5 & Z1 < 0.5)                           -4.031
as.integer(Z4 < 15.5 & Z1 >= 0.5)                          -3.047
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 < 0.5)   -3.699
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 >= 0.5)  -1.484
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 >= 0.5)             -0.717
                                                          Pr(>|z|)
(Intercept)                                               0.079594
Z2                                                        0.015445
as.integer(Z4 < 15.5 & Z1 < 0.5)                          5.54e-05
as.integer(Z4 < 15.5 & Z1 >= 0.5)                         0.002315
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 < 0.5)  0.000217
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 >= 0.5) 0.137707
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 >= 0.5)            0.473180

(Intercept)                                               .
Z2                                                        *
as.integer(Z4 < 15.5 & Z1 < 0.5)                          ***
as.integer(Z4 < 15.5 & Z1 >= 0.5)                         **
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 < 0.5)  ***
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 < 0.5 & Z1 >= 0.5)
as.integer(Z4 >= 15.5 & Z10 < 0.5 & Z6 >= 0.5)
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)
```

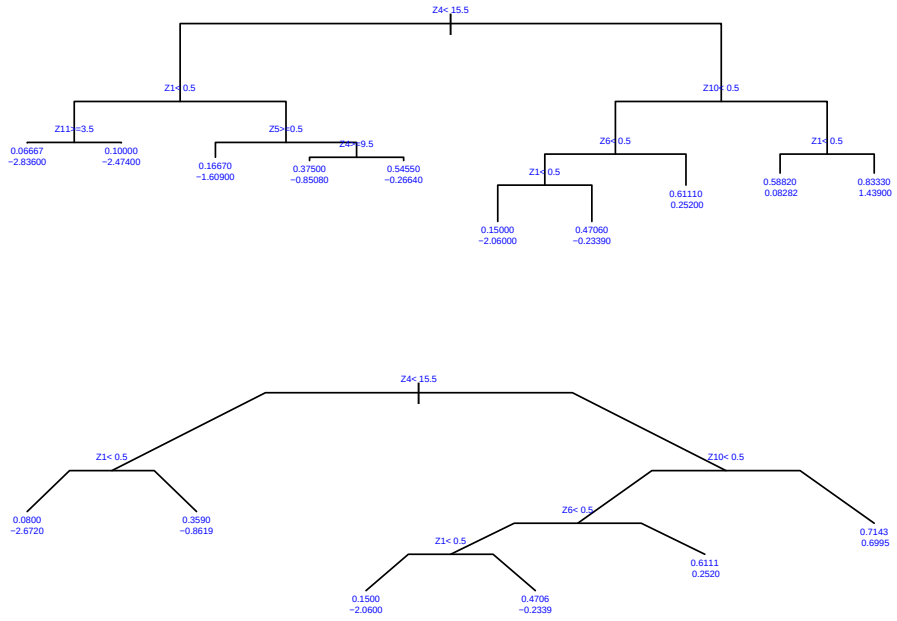


Figure 2: The figure on the top is the maximal tree obtained with pltr.glm; the figure on the bottom is a pruned tree via a BIC criterion

```
    Null deviance: 208.37  on 153  degrees of freedom
Residual deviance: 165.03  on 147  degrees of freedom
AIC: 179.03

Number of Fisher Scoring iterations: 5
> ## Plot the maximal tree and the BIC prunned tree
>
> par(mfrow = c(2,1))
> plot(pltr.burn$tree, main = '', margin = 0.05)
> text(pltr.burn$tree, xpd = TRUE, cex = .4, col = 'blue')
> plot(pltr.burn_prun$tree$BIC, branch = .5, main = '', margin = 0.05)
> text(pltr.burn_prun$tree$BIC, xpd = TRUE, cex = .4, col = 'blue' )
```

- The underlying method behind the 'binomial' family above is a new one, different from those implemented inside the **rpart** package. The splitting criterion is based

on a logistic deviance criterion considering the linear part as offset (Mbogning et al. 2014).

- The child nodes of node x are always $2x$ and $2x+1$, to help in navigating the tree summary (compare the summary to the bottom figure 2).
- They are many Items in the tree summary list:
  - the complexity table
  - the variable importance
  - the node number
  - the number of cases within the node
  - the number of events (number of cases with attribute 1) inside the node
  - the logistic intercept coefficient fitted inside the node, which represents the summary statistic of the node. This coefficient represents the predicted value of the node. that's the main difference with a conventional tree where the predicted value is the modal class of the node.
  - the logistic deviance of the previous model inside the node which is used as the splitting criterion
- * indicates that the node is terminal.
- the first split is on the Percentage of total surface area burned (Z_4). 64 individuals with $Z_4 < 15.5$ go to the left while the remaining 90 go to the right. The split with the maximum number of events is always on the right.
- The improvement listed is the change in deviance for the split, ie., $D(parent) - (D(leftson) + D(rightson))$, where D is the deviance operator. This is similar to a likelihood ratio test statistic.
- the other nodes can be described similarly.

For all the two models (tree with rpart (Fig 1) and the tree with our proposed procedure (Fig 2)), the first split is due to the percentage of total surface area burned ($(< 16.5\%, > 16.5\%)$ for `rpart` and $(< 15.5\%, > 15.5\%)$ for the proposed method). The subsequent splits are different between patients having a high or low percentage of total surface area burned. The classical rpart model shows only one split whereas our proposed PLTR model shows two splits. With the exception of CART model where no split occurs, the subsequent split for the group of patients with a low percentage of area burned is due to the initial treatment (routine bathing/body cleansing). For high percentage of surface area burned, the split for the two models is due to the respiratory tract damage. Our proposed procedure identifies

other splits due to the treatment and the buttock burns. In particular, we observed that the group of patients with a high percentage of surface area of body burned, without tract respiratory damage, buttock injury and without routine bathing shows a lower proportion of prophylactic antibiotics administration. This group is not detected by the original PLTR method. It is worth noting that the confounding factor $Z2$ is significant with a higher proportion of prophylactic antibiotics administration among women.

The particularity of the PLTR model is that in addition with the tree part, the final model is a classical logistic model with new risk factors emerging from the tree part. The summary of the final logistic model is presented within the R code above. We can see for example that the new risk factor constituted by individuals sharing the attributes $Z4 < 15.5$ and $Z1 < 0.5$ is highly significant (although naively due to randomness) w.r.t the reference leaf. Similar interpretation can be made for other factors.

5 Compute the generalization error of the procedure

We can further compute the generalization error of the procedure. This can be computed via the function `best.tree.CV` which can also provide the best tree based on a K-fold cross-validation procedure.

```
> set.seed(150)
> pltr.burn_CV <- best.tree.CV(pltr.burn$tree, burn, Y.name, X.names,
                    G.names, family = family, args.rpart = args.rpart,
                    epsi = 0.001, iterMax = 15, iterMin = 8, ncv = 10,
                    verbose = FALSE)
Max tree size  10 reached
Max tree size  10 reached
Max tree size  10 reached
> pltr.burn_CV$CV_ERROR
[[1]]
[1] 0.3515789

[[2]]
[1] 0.3729825 0.3649123 0.3677193 0.3810526 0.3810526
[6] 0.3624561 0.3663158 0.3515789 0.3649123
> Bic_size <- sum(pltr.burn_prun$tree$BIC$frame$var == '<leaf>')
> ## Bic_size <- tree_select$best_index[[1]]
>
> CV_ERROR_BIC <- pltr.burn_CV$CV_ERROR[[2]][Bic_size]
> CV_ERROR_BIC
```

```
[1] 0.3624561
```

The generalization error of the procedure using the BIC criterion is computed as presented in the previous r code.

6 Test the joint effect of the selected tree while adjusting for confounders.

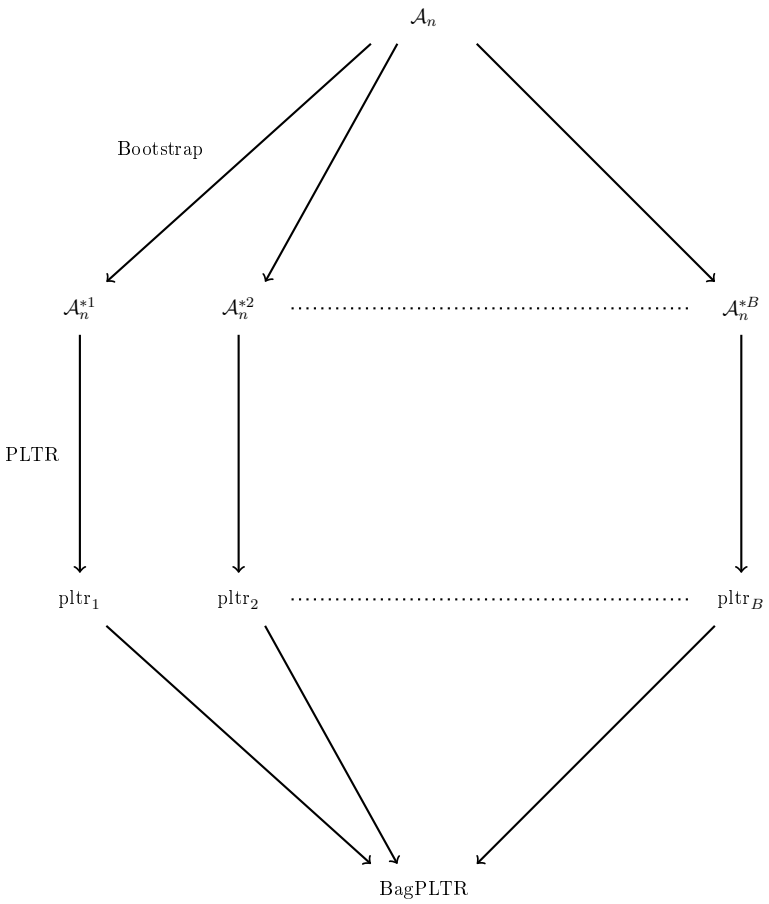
We can also test the joint effect of the selected tree after adjusting for the confounding variable

```
> ## Use only one worker on a window plateform.
>
> args.parallel = list(numWorkers = 10, type = "PSOCK")
> index = Bic_size
> p_value <- p.val.tree(xtree = fit_pltr$tree, data_pltr, Y.name, X.names,
              G.names, B = 1000, args.rpart = args.rpart, epsi = 1e-3,
              iterMax = 15, iterMin = 8, family = family, LB = FALSE,
              args.parallel = args.parallel, index = index, verbose =
              FALSE)
> p_value$P.value
```

7 Bagging a set of PLTR models

The high variability of the tree within the PLTR model can result in an unstable selected model. An ensemble method such as Bagging can stabilize the predictor. The user are encouraged to read the paper of (Mbogning et al. 2015) for a more thorough explanation about the procedure.

A flowchart of the bagging procedure related to the PLTR model is as follow:




```
>  ##  ?bagging.pltr
> set.seed(250)
> Bag.burn <-  bagging.pltr(burn, Y.name, X.names, G.names, family,
                args.rpart,epsi = 0.01, iterMax = 4, iterMin = 3,
                Bag = 20, verbose = FALSE, doprune = FALSE)
 ncores =  1  for bagging trees !
> ## The thresshold values used
>
> Bag.burn$CUT
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> ##The set of PLTR models in the bagging procedure
>
```

```
> PLTR_BAG.burn <- Bag.burn$Glm_BAG
> ##The set of trees in the bagging procedure
>
> TREE_BAG.burn <- Bag.burn$Tree_BAG
> ## Look for the variability of trees in the bagging sequence
>
> par(mfrow = c(3,2))
> plot(TREE_BAG.burn[[1]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[1]], xpd = TRUE, cex = .6 )
> plot(TREE_BAG.burn[[2]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[2]], xpd = TRUE, cex = .6 )
> plot(TREE_BAG.burn[[3]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[3]], xpd = TRUE, cex = .6 )
> plot(TREE_BAG.burn[[4]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[4]], xpd = TRUE, cex = .6 )
> plot(TREE_BAG.burn[[5]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[5]], xpd = TRUE, cex = .6 )
> plot(TREE_BAG.burn[[6]], branch = .5, main = '', margin = 0.05)
> text(TREE_BAG.burn[[6]], xpd = TRUE, cex = .6 )
```

Prediction using the bagged pltr predictor

```
> ## Use the bagging procedure to predict new features
> # ?predict_bagg.pltr
>
> Pred_Bag.burn <- predict_bagg.pltr(Bag.burn, Y.name, newdata = burn,
                  type = "response", thresshold = seq(0, 1, by = 0.1))
> ## The confusion matrix for each thresshold value using the majority vote
>
> Pred_Bag.burn$CONF1
$CUT1
               Observed Class
Predicted Class  0  1
              1 91 63

$CUT2
               Observed Class
Predicted Class  0  1
```

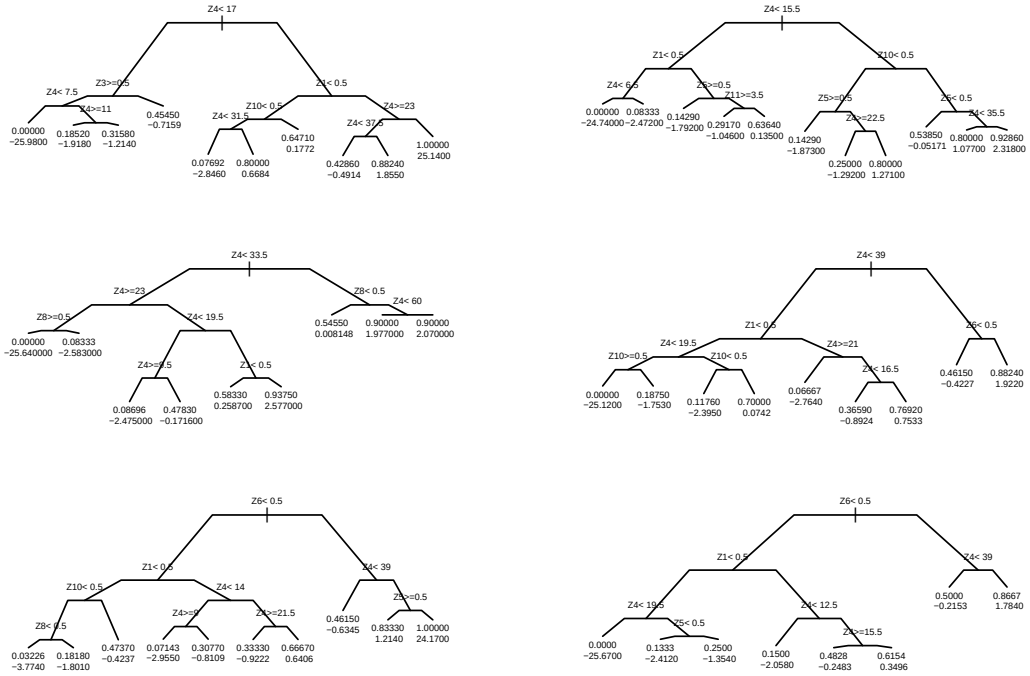


Figure 3: set of the first 6 trees within the bagging predictor: The trees seem to fluctuate with the sample considered during the resampling step.

```
              0 25  2
              1 66 61

$CUT3
               Observed Class
Predicted Class  0  1
              0 44  5
              1 47 58

$CUT4
               Observed Class
Predicted Class  0  1
              0 64 12
              1 27 51

$CUT5
               Observed Class
Predicted Class  0  1
              0 73 17
              1 18 46

$CUT6
               Observed Class
Predicted Class  0  1
              0 80 23
              1 11 40

$CUT7
               Observed Class
Predicted Class  0  1
              0 85 29
              1  6 34

$CUT8
               Observed Class
Predicted Class  0  1
              0 90 42
              1  1 21

$CUT9
```

```
               Observed Class
Predicted Class  0  1
              0 91 50
              1  0 13

$CUT10
               Observed Class
Predicted Class  0  1
              0 91 58
              1  0  5

$CUT11
               Observed Class
Predicted Class  0  1
              0 91 63
> ## The prediction error for each thresshold value
>
> Pred_err.burn <- Pred_Bag.burn$PRED_ERROR1
> Pred_err.burn
     CUT1      CUT2      CUT3      CUT4      CUT5      CUT6
0.5909091 0.4415584 0.3376623 0.2532468 0.2272727 0.2207792
     CUT7      CUT8      CUT9     CUT10     CUT11
0.2272727 0.2792208 0.3246753 0.3766234 0.4090909
```

Compute the variable importances of the bagging procedure

Several scores for variable importance are proposed (Mbogning et al. 2015). Among them,

- the Permutation Importance Score (PIS)
- the Deviance Importance Score (DIS)
- the Depth Deviance Importance Score (DDIS)
- the minimal depth score

```
> Var_Imp_BAG.burn <- VIMPBAG(Bag.burn, burn, Y.name)
> ## Importance score using the permutaion method for each thresshold value
>
> Var_Imp_BAG.burn$PIS
```

```
$CUT1
 Z1 Z10  Z3  Z4 Z11  Z5  Z8  Z6  Z7  Z9
  0   0   0   0   0   0   0   0   0   0

$CUT2
           Z4            Z1            Z5            Z8
 0.0289866790  0.0256372334  0.0103759725  0.0086566377
           Z6            Z9            Z7           Z11
 0.0043566597  0.0027272727  0.0024074074  0.0009259259
           Z3           Z10
-0.0008771930 -0.0011768972

$CUT3
           Z4            Z1           Z10            Z8
 0.0434068073  0.0193202113  0.0152575713  0.0078091801
           Z6            Z5           Z11            Z7
 0.0061916765  0.0054450125  0.0018518519  0.0001851852
           Z9            Z3
-0.0009090909 -0.0019410228

$CUT4
           Z4           Z10            Z1            Z8
 0.0552722586  0.0267374003  0.0138682895  0.0043691299
          Z11            Z6            Z7            Z9
 0.0009259259  0.0007809783  0.0001851852 -0.0008474576
           Z3            Z5
-0.0010638298 -0.0025064318

$CUT5
           Z4           Z10            Z1            Z8
 0.0526079447  0.0293728528  0.0132722723  0.0034600390
           Z6           Z11            Z7            Z3
 0.0033852941  0.0009259259  0.0001851852  0.0000000000
           Z9            Z5
-0.0008474576 -0.0060830953

$CUT6
           Z4           Z10            Z1            Z8
 0.0415287942  0.0319301668  0.0067039630  0.0043372320
           Z6            Z3           Z11            Z7
```

```
 0.0033852941  0.0000000000  0.0000000000  0.0000000000
           Z9            Z5
-0.0008474576 -0.0033540505

$CUT7
           Z4           Z10            Z1            Z6
 0.0477827955  0.0201663489  0.0137082407  0.0043353910
           Z8           Z11            Z3            Z7
 0.0017056530  0.0009259259  0.0000000000  0.0000000000
           Z9            Z5
 0.0000000000 -0.0005780448

$CUT8
          Z4           Z10           Z1           Z6
0.0424875527 0.0244430674 0.0128987887 0.0078501462
          Z5           Z8           Z3          Z11
0.0009259259 0.0007987247 0.0000000000 0.0000000000
          Z7           Z9
0.0000000000 0.0000000000

$CUT9
          Z4           Z10           Z1           Z6
 0.031460167  0.021185506  0.010787404  0.006237243
          Z8           Z3          Z11           Z7
 0.001724651  0.000000000  0.000000000  0.000000000
          Z9           Z5
 0.000000000 -0.003638136

$CUT10
          Z4           Z10           Z6           Z1
 0.021132390  0.016617768  0.007164059  0.004402077
          Z8           Z3          Z11           Z7
 0.003479037  0.000000000  0.000000000  0.000000000
          Z9           Z5
 0.000000000 -0.000860358

$CUT11
 Z1 Z10  Z3  Z4 Z11  Z5  Z8  Z6  Z7  Z9
  0   0   0   0   0   0   0   0   0   0
```

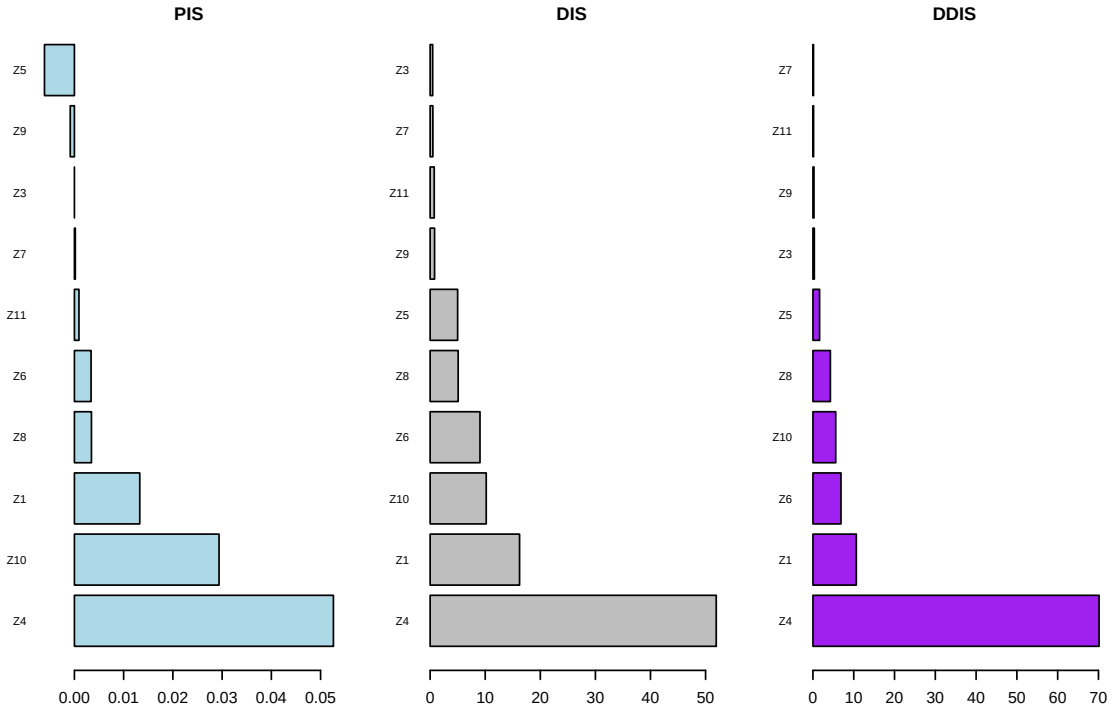


Figure 4: Variable importances using the bagging procedure on the burn dataset: The permutation importance score (PIS on the left), the deviance importance score (DIS on the middle) and the depth deviance importance score (DDIS on the right).

```
> par(mfrow=c(1,3))
> barplot(Var_Imp_BAG.burn$PIS$CUT5, main = 'PIS', horiz = TRUE, las = 1,
          cex.names = .8, col = 'lightblue')
> barplot(Var_Imp_BAG.burn$DIS, main = 'DIS', horiz = TRUE, las = 1,
          cex.names = .8, col = 'grey')
> barplot(Var_Imp_BAG.burn$DDIS, main = 'DDIS', horiz = TRUE, las = 1,
          cex.names = .8, col = 'purple')
```

compute the AUC of the Bagged predictor based on OOB samples

```
> auc_BAG_oob <- bag.aucoob(Bag.burn, burn, Y.name)
> ## AUC of the predictor on OOB samples
>
> auc_BAG_oob$AUCOOB
[1] 0.6759114
> ## Plot the ROC curve of the predictor based on OOB samples
>
> par(mfrow=c(1, 1))
> plot(auc_BAG_oob$FPR, auc_BAG_oob$TPR, type = 'b', lty = 3, col = 'blue',
       xlab = 'false positive rate', ylab = 'true positive rate')
> legend(0.7, 0.3, sprintf('%3.3f', auc_BAG_oob$AUCOOB), lty = c(1, 1),
         lwd = c(2.5, 2.5), col = 'blue', title = 'AUC')
```

References

Breiman, L., J. H. Olshen, and C. J. Stone (1984). *Classification and Regression Trees.* Belmont, California: Wadsworth International Group.

Chen, J., K. Yu, A. Hsing, and T. M. Therneau (2007). A partially linear tree-based regression model for assessing complex joint gene-gene and gene-environment effects. *Genetic Epidemiology 31*, 238–251.

Ichida, J. M., J. T. Wassell, M. D. Keller, and L. W. Ayers (1993). Evaluation of Protocol Change in Burn-Care Management Using the Cox Proportional Hazards Model with Time-Dependent Covariates. *Statistics in Medicine 12*, 301–310.

Klein, J. P. and M. L. Moeschberger (2003). *SURVIVAL ANALYSIS Techniques for Censored and Truncated Data* (second ed.). New York: Springer.

Mbogning, C., H. Perdry, and P. Broët (2015). A bagged partially linear tree-based regression procedure for prediction and variable selection. *Human Heredity 79*(1), 82–93.

Mbogning, C., H. Perdry, W. Toussile, and P. Broët (2014). A novel tree-based procedure for deciphering the genomic spectrum of clinical disease entities. *Journal of Clinical Bioinformatics 4*(6).

Therneau, T. M. and E. J. Atkinson (2013). An introduction to recursive partitioning using the RPART routines. *Mayo Foundation*.

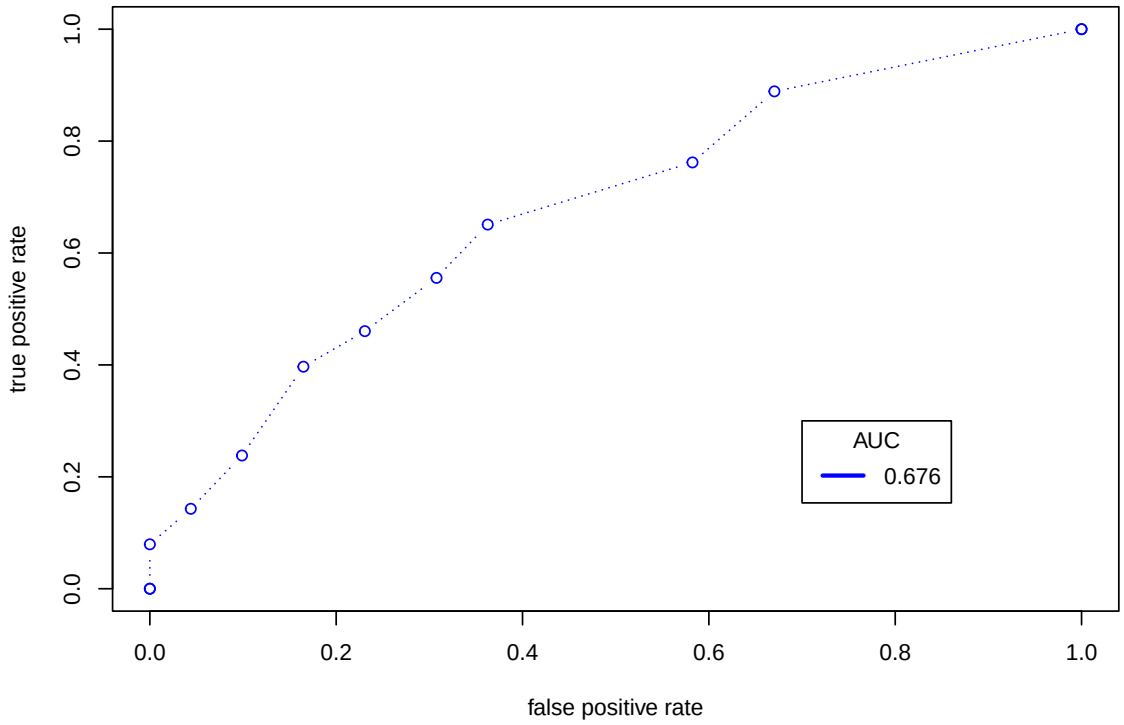


Figure 5: Roc curve with AUC of the Bagged predictor using OOB samples.

Wisplinghoff, H., W. Perbix, and H. Seifert (1999). Risk Factors for Nosocomial Bloodstream Infections Due to Acinetobacter baumannii: A Case-Control Study of Adult Burn Patients. *Clin. Infect. Dis. 28*(1), 59–66.