Random Forest

Rishabh Vaish

Tuning Random Forests in Virtual Twins

Personalized medicine draws a lot of attention in medical research. The goal of personalized medicine is to make a tailored decision for each patient, such that his/her clinical outcome can be optimized. Let's consider data modified from the SIDES method. In this dataset, 470 patients and 13 variables are observed. You can download the data from our website. The variables are listed below.

- `Health`: health outcome (larger the better)
- `THERAPY`: 1 for active treatment, 0 for the control treatment
- `TIMFIRST`: Time from first sepsis-organ fail to start drug
- `AGE`: Patient age in years
- `BLLPLAT`: Baseline local platelets
- `blSOFA`: Sum of baseline sofa score (cardiovascular, hematology, hepatorenal, and respiration scores)
- `BLLCREAT`: Base creatinine
- `ORGANNUM`: Number of baseline organ failures
- `PRAPACHE`: Pre-infusion apache-ii score
- `BLGCS`: Base GLASGOW coma scale score
- `BLIL6`: Baseline serum IL-6 concentration
- `BLADL`: Baseline activity of daily living score
- `BLLBILI`: Baseline local bilirubin
- `BEST`: The true best treatment suggested by Doctors. **You should not use this variable when fitting the model**!

For each patient, sepsis was observed during their hospital stay. Hence, they need to choose one of the two treatments (indicated by variable `THERAPY`) to prevent further adverse events. After the treatment, their health outcome (`health`) were measured, with a larger value being the better outcome. However, since treatments were assigned randomly, we are not able to suggest better treatment for a new patient. A strategy called Virtual Twins was proposed by Foster et al. (2011) to tackle this problem. We consider a simpler version of the method. We fit two random forests to model the outcome `health`: one model uses all patients who received treatment 1, and another model for all patients who received treatment 0. Denote these two models as $\widehat{f}_1(x)$ and $\widehat{f}_0(x)$, respectively. When a new patient arrives, we use both models to predict the outcomes and see which model gives a better health status. We will suggest the treatment label associated with the model that gives a larger prediction value. In other words, for a new x^*, we compare $\widehat{f}_1(x^*)$ and $\widehat{f}_0(x^*)$ and suggest the better lable. The goal for this question is to select tuning parameters for random forest such that it will suggest the best treatment for a patient. Perform the following:

```r
# Data import
library(readr)
sepsis <- read_csv("data/Sepsis.csv")

#Removing row ID
sepsis$X1 <- NULL
```

- Randomly split the data into 75% for training and 25% for testing.

```r
#generate data
sample <-
  sample.int(n = nrow(sepsis),
             size = floor(.75 * nrow(sepsis)),
             replace = F)
#split test and train
train_best <- sepsis[sample,]
test_best  <- sepsis[-sample,]

#create dataset without BEST column
train <- train_best[, 1:13]
test <- test_best[, 1:13]

# Seperate for train data for cases
train_1 <- train[which(train$THERAPY == 1), ]
train_0 <- train[which(train$THERAPY == 0), ]

#remove therapy for training and testing
train_1$THERAPY <- NULL
train_0$THERAPY <- NULL
test$THERAPY <- NULL
```

- For the training data, fit the virtual twins model and then use the testing data to suggest the best treatment.
  - You should not use the variable `BEST` when fitting the models
  - Pick three different `mtry` values and three different `nodesize`, leave all other tuning parameters as default
  - After predicting the best treatment in the testing data, compare it to the truth `BEST`

```r
#Setting tuning parameters
library(randomForest)
mtry = c(1, 3, 9)
nodesize = c(2, 5, 8)
tunegrid = expand.grid(mtry = mtry, nodesize = nodesize)
accuracy <- rep(0, nrow(tunegrid))

#Checking RF fit on 9 mtry nodesize combinations
for (i in 1:nrow(tunegrid)) {
  #predicting using zero treatment
  rf_zero <-
    randomForest(Health ~ . ,
                 data = train_0,
                 mtry = tunegrid[i, 1],
                 nodesize = tunegrid[i, 2])
  health_0_hat <- predict(rf_zero, newdata = test)
  #predicting using one treatment
  rf_one <-
    randomForest(Health ~ . ,
                 data = train_1,
                 mtry = tunegrid[i, 1],
                 nodesize = tunegrid[i, 2])
  health_1_hat <- predict(rf_one, newdata = test)
  #Adding predicted value of BEST to test_best
  test_best$best_hat <- 0
  test_best$best_hat[health_1_hat > health_0_hat] <- 1
  #Checking accuracy
  correct = nrow(test_best[which(test_best$BEST == test_best$best_hat),])
  accuracy[i] = floor((correct / nrow(test_best)) * 100)
}

#Results
cbind(tunegrid, accuracy)
```

```
##   mtry nodesize accuracy
## 1    1        2       72
## 2    3        2       75
## 3    9        2       79
## 4    1        5       74
## 5    3        5       83
## 6    9        5       80
## 7    1        8       73
## 8    3        8       78
## 9    9        8       82
```

- Repeat this entire process 100 times and average the prediction errors

```
start.time <- Sys.time()

#initiating variables
iter = 100
mtry = c(1, 3, 9)
nodesize = c(2, 5, 8)
tunegrid = expand.grid(mtry = mtry, nodesize = nodesize)
accuracy <- matrix(0, nrow = nrow(tunegrid), ncol = iter)

#Checking RF fit on 9 mtry nodesize combinations
for (j in 1:iter) {
  sample <-
    sample.int(n = nrow(sepsis),
               size = floor(.75 * nrow(sepsis)),
               replace = F)
  #split test and train
  train_best <- sepsis[sample, ]
  test_best  <- sepsis[-sample, ]

  #create dataset without BEST column
  train <- train_best[, 1:13]
  test <- test_best[, 1:13]

  # Seperate for train data for cases
  train_1 <- train[which(train$THERAPY == 1),]
  train_0 <- train[which(train$THERAPY == 0),]

  #removing therapy
  train_1$THERAPY <- NULL
  train_0$THERAPY <- NULL
  test$THERAPY <- NULL

  for (i in 1:nrow(tunegrid)) {
    #predicting using zero treatment
    rf_zero <-
      randomForest(Health ~ . ,
                   data = train_0,
                   mtry = tunegrid[i, 1],
                   nodesize = tunegrid[i, 2])
    health_0_hat <- predict(rf_zero, newdata = test)
    #predicting using one treatment
    rf_one <-
      randomForest(Health ~ . ,
                   data = train_1,
                   mtry = tunegrid[i, 1],
                   nodesize = tunegrid[i, 2])
    health_1_hat <- predict(rf_one, newdata = test)
    #Adding predicted value of BEST to test_best
    test_best$best_hat <- 0
    test_best$best_hat[health_1_hat > health_0_hat] <- 1
    #calculating accuracy
    correct = nrow(test_best[which(test_best$BEST == test_best$best_hat),])
    accuracy[i, j] = floor((correct / nrow(test_best)) * 100)
  }
}

#storing resulats
results <- cbind(tunegrid, rowMeans(accuracy))
colnames(results) <- c("mtry", "nodesize", "accuracy")
results$error <- 100 - results$accuracy
results
```

```
##   mtry nodesize accuracy error
## 1    1        2    73.57 26.43
## 2    3        2    76.40 23.60
## 3    9        2    77.15 22.85
## 4    1        5    74.09 25.91
## 5    3        5    76.91 23.09
## 6    9        5    77.36 22.64
## 7    1        8    74.59 25.41
## 8    3        8    77.26 22.74
## 9    9        8    77.79 22.21
```

```
#best parameters
tunegrid[which.max(rowMeans(accuracy)), ]
```

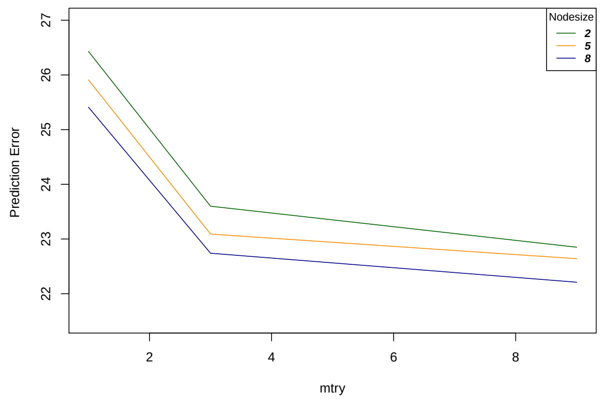
```
##   mtry nodesize
## 9    9        8
```

```
#time taken
end_time <- Sys.time()
start.time - end_time
```

```
## Time difference of -4.034463 mins
```

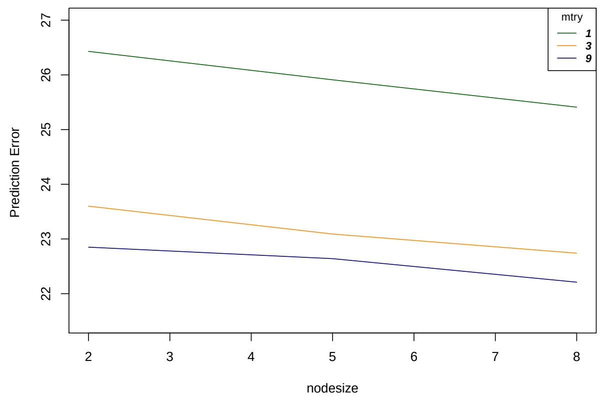
- Summarize your results, including the model performance and the effect of tuning parameters. Intuitively demonstrate them.

```r
#plotting error vs mtry for constant nodesize
plot(
  mtry,
  results$error[which(results$nodesize == 2)],
  type = "n",
  ylim = c(21.5, 27),
  xlab = "mtry",
  ylab = "Prediction Error "
)
lines(mtry, results$error[which(results$nodesize == 2)], col = "darkgreen")
lines(mtry, results$error[which(results$nodesize == 5)], col = "darkorange")
lines(mtry, results$error[which(results$nodesize == 8)], col = "darkblue")
legend(
  "topright",
  legend = c("2", "5", "8"),
  col = c("darkgreen", "darkorange", "darkblue"),
  lty = 1,
  cex = 0.8,
  title = "Nodesize",
  text.font = 4
)
```



For a constant nodesize, the error in prediction reduces with increasing mtry. But after certain mtry the error doesn't decrease further because all the columns don't contribute towards the outcome and decreases the randomness in random forest thus not utilising the full power. This generates an elbow type graph.

```r
#plotting error vs nodesize for constant mtry
plot(
  nodesize,
  results$error[which(results$mtry == 1)],
  type = "n",
  ylim = c(21.5, 27),
  xlab = "nodesize",
  ylab = "Prediction Error "
)
lines(nodesize, results$error[which(results$mtry == 1)], col = "darkgreen")
lines(nodesize, results$error[which(results$mtry == 3)], col = "darkorange")
lines(nodesize, results$error[which(results$mtry == 9)], col = "darkblue")
legend(
  "topright",
  legend = c("1", "3", "9"),
  col = c("darkgreen", "darkorange", "darkblue"),
  lty = 1,
  cex = 0.8,
  title = "mtry",
  text.font = 4
)
```



For a constant mtry, the error in prediction reduces with increasing nodesize. As the nodesize is increased the variance decreases thus error decreases. After certain point the increase in bias overcomes decrease in variance and the error increases.

Second Step in Virtual Twins

The second step in a virtual twins model is to use a single tree model (CART) to describe the choice of the best treatment. Perform the following: * Based on your optimal tuning parameter, fit the Virtual Twins model described in Question 1. Again, you should not use the `BEST` variable.

```r
# Optimal parameters of 100 iterations
best_param <- tunegrid[which.max(rowMeans(accuracy)), ]

#rebuilding data
test_best$best_hat <- NULL
data_best <- rbind(train_best, test_best)
data <- data_best[, 1:13]
data_0 <- data[data$THERAPY == 0, ]
data_1 <- data[data$THERAPY == 1, ]
data_0$THERAPY <- NULL
```

```r
data_1$THERAPY <- NULL
data$THERAPY <- NULL

#treatment 1 model
f_1_hat <-
  randomForest(Health ~ . ,
               data = data_1,
               mtry = best_param[1, 1],
               nodesize = best_param[1, 2])
#treatment 0 model
f_0_hat <-
  randomForest(Health ~ . ,
               data = data_0,
               mtry = best_param[1, 1],
               nodesize = best_param[1, 2])
```

- For each subject, obtain the predicted best treatment of the training data itself

```r
#Making prediction for full test set using both f_1_hat and f_0_hat
y_1_hat <- predict(f_1_hat, newdata = data)
y_0_hat <- predict(f_0_hat, newdata = data)

#Adding predicted value of BEST to test_best
data_best$best_hat <- 0
data_best$best_hat[y_1_hat > y_0_hat] <- 1

#correct predictions
correct = nrow(data_best[which(data_best$BEST == data_best$best_hat),])
accuracy = (correct / nrow(data_best)) * 100
accuracy
```

```
## [1] 76.17021
```

- Treating the label of best treatment as the outcome, and fit a single tree model to predict it. Be careful which variables should be removed from this model fitting.

```r
#removing Health, Therapy and BEST for single tree model
library(rpart)
data <- data_best
data_best$Health <- NULL
data_best$BEST <- NULL
data_best$THERAPY <- NULL
data_best$best_hat_hat <- NULL

#predicting a single tree model
cart_tree = rpart(as.factor(best_hat) ~ ., data = data_best)
best_hat_hat <- data.frame(predict(cart_tree, newdata = data_best))

#storing predictions
best_hat_hat$prediction <- ifelse(best_hat_hat$X0 >= 0.5, 0, 1)
data$best_hat_hat <- best_hat_hat$prediction
```

```
#accuracy with BEST variable
correct <- nrow(data[which(data$best_hat_hat == data$BEST), ])
accuracy_final <- (correct / nrow(data)) * 100
accuracy_final
```

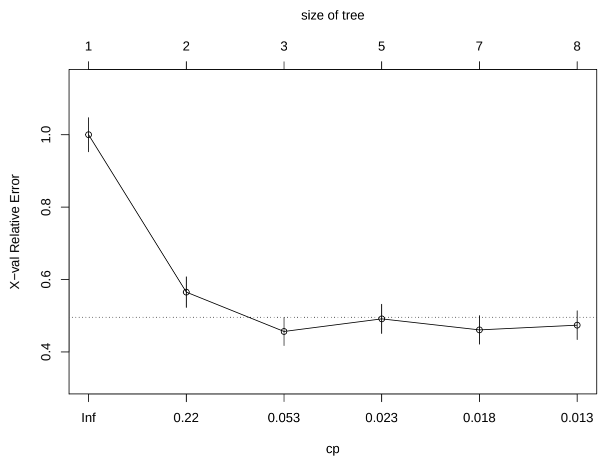
```
## [1] 85.53191
```

- Consider tuning the tree model using the cost-complexity tuning.

```
#pruning tree based on value
printcp(cart_tree)
```

```
##
## Classification tree:
## rpart(formula = as.factor(best_hat) ~ ., data = data_best)
##
## Variables actually used in tree construction:
## [1] AGE      BLADL    BLLCREAT PRAPACHE TIMFIRST
##
## Root node error: 230/470 = 0.48936
##
## n= 470
##
##         CP nsplit rel error  xerror     xstd
## 1 0.434783      0   1.00000 1.00000 0.047119
## 2 0.108696      1   0.56522 0.56522 0.042163
## 3 0.026087      2   0.45652 0.45652 0.039261
## 4 0.019565      4   0.40435 0.49130 0.040281
## 5 0.017391      6   0.36522 0.46087 0.039394
## 6 0.010000      7   0.34783 0.47391 0.039782
```

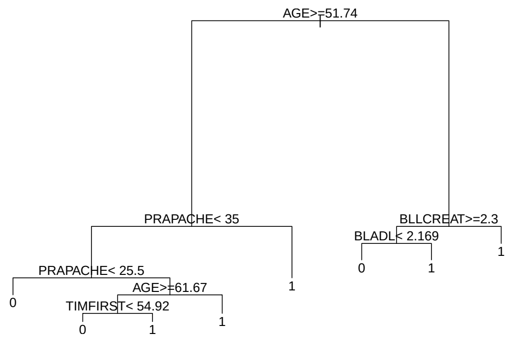
```
plotcp(cart_tree)
```



```
prune.rpart(cart_tree, cp = 0.046)
```

```
## n= 470
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 470 230 0 (0.5106383 0.4893617)
##   2) AGE>=51.7395 310 100 0 (0.6774194 0.3225806)
##     4) PRAPACHE< 35 269  67 0 (0.7509294 0.2490706) *
##     5) PRAPACHE>=35 41   8 1 (0.1951220 0.8048780) *
##   3) AGE< 51.7395 160  30 1 (0.1875000 0.8125000) *
```

```r
#plot the final tree
plot(cart_tree)
text(cart_tree)
```


AGE>=51.74
PRAPACHE< 35
BLLCREAT>=2.3
BLADL< 2.169
0
1
1
PRAPACHE< 25.5
AGE>=61.67
1
0
TIMFIRST< 54.92
1
0
1