Personalized Medicine - Sepsis Treatement Recommender Using Virtual Twins Model

Suma Venugopal

Oct 2019

Contents

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.In this SEPSIS dataset (http://biopharmnet.com/subgroup-analysis-software/) the variables present are:

- `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. **Not used 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. In thi simple implementation of this 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 here is to select tuning parameters for random forest such that it will suggest the best treatment for a patient.

```r
# split the data into 75% for training and 25% for testing

# Read in data
library(tidyverse)
sepsis <- read_csv(".//data//Sepsis.csv")
attach(sepsis)


set.seed(16)

sepsis.dat <- sepsis[-c(1)]  #excluding index

# Random sample indexes for test and train
train.index <- sample(1:nrow(sepsis.dat), 0.75 * nrow(sepsis.dat))
test.index <- setdiff(1:nrow(sepsis.dat), train.index)

# Build train and test dataframes
sepsis.train <- sepsis.dat[train.index, ]

# split train data into two sets based on THERAPY variable value = 1 or 0
sepsis.train_0 <- sepsis.train[sepsis.train$THERAPY == "0", ]
sepsis.train_1 <- sepsis.train[sepsis.train$THERAPY == "1", ]

sepsis.X.train_0 <- sepsis.train_0[, -c(1, 14)]  #removing variables 'Health' and BEST
sepsis.Y.train_0 <- sepsis.train_0[, 1]  #assigning Health as outcome variable

sepsis.X.train_1 <- sepsis.train_1[, -c(1, 14)]  #removing variables 'Health' and BEST
sepsis.Y.train_1 <- sepsis.train_1[, 1]  #assigning Health as outcome variable

sepsis.X.train_0_formodel <- sepsis.X.train_0[, -c(1)]  #removing variable THERAPY from train set 0

sepsis.X.train_1_formodel <- sepsis.X.train_1[, -c(1)]  #removing variable THERAPY from train set 1


x.test <- sepsis.dat[test.index, -c(1, 14)]  #removing variables 'Health' and BEST from test data
y.test <- sepsis.dat[test.index, "Health"]  #assigning Health as outcome variable

x.test_formodel <- x.test[, -c(1)]  #removing variable THERAPY from x.test
```

```r
# fit the virtual twins model and then use the testing data to suggest the best treatment.  variable `B
# when fitting the models

# Virtual Twins Method - Creating random forest models

library(randomForest)

# Create the random forest model 1
f_1hatx.model <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formod

# Create the random forest model 0
f_0hatx.model <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formod

# function that taken in virtual twin models, train data , test data and does predictions
```

```r
vtm.function <- function(model0, model1, X.train_0, X.train_1, x.test, y.test, X.dat) {

    f_1hatx.model.ypred <- predict(model1, newdata = x.test)  #model 1 prediction on test data
    f_1hatx.model.ypred


    f_0hatx.model.ypred <- predict(model0, newdata = x.test)  #model 0 prediction on test data
    f_0hatx.model.ypred

    # which model predicts better 'Health' score?
    p <- apply(as.matrix(f_0hatx.model.ypred), 2, `<`, as.matrix(f_1hatx.model.ypred))

    # which model's data should be used to predict THERAPY?
    pred.actual.lst <- c(rep(0, nrow(y.test)))
    for (i in 1:nrow(y.test)) {
        if (p[i] == "TRUE")
            pred.actual.lst[i] = 1
    }


    # Accuracy %- comparison of chosen 'THERAPY' with 'BEST' therapy associated with each test record
    acc <- (sum(pred.actual.lst == as.matrix(X.dat[test.index, "BEST"]))/nrow(y.test)) * 100

    return(acc)
}

# calling the vtm function above to determine test data prediction accuracy
acc1 <- vtm.function(f_0hatx.model, f_1hatx.model, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel,

# predicted accuracy on test data in precentage (single run)
acc1
```

```
## [1] 73.72881
```

```r
# Pick three different `mtry` values and three different `nodesize` to create multiple models, leave al
# parameters as default
mtry.lst = list(3, 3, 3, 5, 5, 5, 7, 7, 7)
n.size.lst = list(200, 150, 80, 200, 150, 80, 200, 150, 80)
acc.lst = c(rep(0, 9))

set.seed(16)
f_1hatx.model2 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model2 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc2 <- vtm.function(f_0hatx.model2, f_1hatx.model2, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc2
```

```
## [1] 70.33898
```

```r
acc.lst[1] <- acc2

set.seed(16)
f_1hatx.model3 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
```

```
f_0hatx.model3 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc3 <- vtm.function(f_0hatx.model3, f_1hatx.model3, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc3
```

```
## [1] 68.64407
```

```
acc.lst[2] <- acc3

set.seed(16)
f_1hatx.model4 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model4 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc4 <- vtm.function(f_0hatx.model4, f_1hatx.model4, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc4
```

```
## [1] 75.42373
```

```
acc.lst[3] <- acc4

set.seed(16)
f_1hatx.model5 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model5 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc5 <- vtm.function(f_0hatx.model5, f_1hatx.model5, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc5
```

```
## [1] 74.57627
```

```
acc.lst[4] <- acc5

set.seed(16)
f_1hatx.model6 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model6 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc6 <- vtm.function(f_0hatx.model6, f_1hatx.model6, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc6
```

```
## [1] 74.57627
```

```
acc.lst[5] <- acc6

set.seed(16)
f_1hatx.model7 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model7 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc7 <- vtm.function(f_0hatx.model7, f_1hatx.model7, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc7
```

```
## [1] 74.57627
```

```
acc.lst[6] <- acc7

set.seed(16)
f_1hatx.model8 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model8 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc8 <- vtm.function(f_0hatx.model8, f_1hatx.model8, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc8
```

```
## [1] 76.27119
```

```
acc.lst[7] <- acc8

set.seed(16)
f_1hatx.model9 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_formo
f_0hatx.model9 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_formo
acc9 <- vtm.function(f_0hatx.model9, f_1hatx.model9, sepsis.X.train_0, sepsis.X.train_1, x.test_formodel
acc9
```

```
## [1] 76.27119
```

```
acc.lst[8] <- acc9

set.seed(16)
f_1hatx.model10 <- randomForest(as.matrix(sepsis.Y.train_1) ~ ., data = as.matrix(sepsis.X.train_1_forme
f_0hatx.model10 <- randomForest(as.matrix(sepsis.Y.train_0) ~ ., data = as.matrix(sepsis.X.train_0_forme
acc10 <- vtm.function(f_0hatx.model10, f_1hatx.model10, sepsis.X.train_0, sepsis.X.train_1, x.test_forme
acc10
```

```
## [1] 76.27119
```

```
acc.lst[9] <- acc10


# Parameters that seems to work the best; mtry = 7, nodesize = 200.  Defining the models using these pa
# best model below

f_1hatx.bestmodel <- f_1hatx.model8
f_0hatx.bestmodel <- f_0hatx.model8
```

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

```
#Repeat this entire process 100 times and average the prediction errors

prederr.lst <- c(rep(0, 100)) #list to store prediction errors for each run
X.dat <- sepsis.dat
for (i in 1:100) #prediction loop
{
# Data split
train.index <- sample(1:nrow(X.dat), .75 * nrow(X.dat))
test.index <- setdiff(1:nrow(X.dat), train.index)


X.train <- X.dat[train.index,]

X.train_0 <- X.train[X.train$THERAPY == '0',]
X.train_1 <- X.train[X.train$THERAPY == '1',]

dat.X.train_0 <- X.train_0[,-c(1,14)]
dat.Y.train_0 <- X.train_0[,1]
```

```
dat.X.train_1 <- X.train_1[,-c(1,14)]
dat.Y.train_1 <- X.train_1[,1]


x.test <- X.dat[test.index, -c(1,14)]
y.test <- X.dat[test.index, "Health"]


#Virtual Twins Method
#Using the best random forest models identified above and
#reusing the vtm function defined above to determine prediction accuracy on sepsis data
acc.this <- vtm.function(f_0hatx.bestmodel,f_1hatx.bestmodel,
                         dat.X.train_0,dat.X.train_1,x.test,y.test,  X.dat)

prederr.lst[i] <- 100-acc.this

}

#average prediction error across 100 cycles
avg.prederr.100runs <- mean(prederr.lst)
avg.prederr.100runs
```

```
## [1] 22.61864
```

```
#average prediction accuracy across 100 cycles
avg.accuracy.100runs <- 100-avg.prederr.100runs
avg.accuracy.100runs
```

```
## [1] 77.38136
```

- Model performance and the effect of tuning parameters - Conclusion

Parameters that seems to work the best for random forest models are - mtry = 7, nodesize = 200 and hence these parameters were chosen to define the best models above, which were then used to do the 100 iterations to determine average accuracy and average error.

Nodesize parameter in the 'randomForest' package defines the minimum size of terminal nodes and thus decides the tree depth. Mtry parameter corresponds to the number of variables available for splitting at each tree node. Default values for the package seemed to be mtry = 4 and nodesize = 400. Based on the 18 individual models created above for sepsis data and comparing the accuracies returned by these models, it is evident that use of mtry = 7 increases the accuracy for multiple nodesizes considered. This indicated that around 7 variables considered for spitting after excluding index, Health , THERAPY and BEST contributed to the accuracy of the model. Hence mtry = 7 was chosen. Considering the size of the dataset, decided to choose a lesser nodesizes of which 200 seemed to work better in this context and was hence chosen.

The effect of chosen parameters on the accuracy is demonstated below

```
cbind(mtry.lst, n.size.lst, acc.lst)
```

```
##       mtry.lst n.size.lst acc.lst
##  [1,] 3        200        70.33898
##  [2,] 3        150        68.64407
```

```
##  [3,] 3        80         75.42373
##  [4,] 5        200        74.57627
##  [5,] 5        150        74.57627
##  [6,] 5        80         74.57627
##  [7,] 7        200        76.27119
##  [8,] 7        150        76.27119
##  [9,] 7        80         76.27119
```

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, fitted the Virtual Twins model from above. `BEST` variable is again not used.

```
# Based on optimal tuning parameter, fitting the Virtual Twins model

rm(list = ls())  #clear all global variables
# Read in data and do test rrain split(same as above)
library(tidyverse)
sepsis <- read_csv(".//data//Sepsis.csv")
attach(sepsis)

set.seed(16)

sepsis.dat <- sepsis[-c(1)]  #excluding index
sepsis.X <- sepsis.dat[, -c(1, 2, 14)]
sepsis.Y <- sepsis.dat[, "BEST"]

sepsis.dat0 <- sepsis.dat[sepsis.dat$THERAPY == "0", ]
sepsis.dat1 <- sepsis.dat[sepsis.dat$THERAPY == "1", ]

sepsis.dat0.X <- sepsis.dat0[-c(1, 2, 14)]
sepsis.dat0.Y <- sepsis.dat0["Health"]

sepsis.dat1.X <- sepsis.dat1[-c(1, 2, 14)]
sepsis.dat1.Y <- sepsis.dat1["Health"]

# Virtual Twins Methods - Creating models

# Create the random forest models 1 & 0 using best parameters identified above (fitting on the whole da

library(randomForest)

set.seed(16)
f_1hatx.bestmodel1 <- randomForest(as.matrix(sepsis.dat1.Y) ~ ., data = as.matrix(sepsis.dat1.X), mtry =
f_0hatx.bestmodel1 <- randomForest(as.matrix(sepsis.dat0.Y) ~ ., data = as.matrix(sepsis.dat0.X), mtry =
```

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

```
# Obtaining the predicted best treatment using the best virtual twin models defined above

# predict
```

```
f_1hatx.model.ypred <- predict(f_1hatx.bestmodel1, newdata = sepsis.X)


# predict
f_0hatx.model.ypred <- predict(f_0hatx.bestmodel1, newdata = sepsis.X)


p <- apply(as.matrix(f_0hatx.model.ypred), 2, `<`, as.matrix(f_1hatx.model.ypred))  #which model predic


# virtual twin model predictions on sepsis data prediction result stored in 'pred.actual.lst ', 'pred.r

# which model's data should be used to predict THERAPY?
pred.actual.lst <- c(rep(0, nrow(sepsis.Y)))
for (i in 1:nrow(sepsis.Y)) {
    if (p[i] == "TRUE")
        pred.actual.lst[i] = 1
}

pred.rslt <- Reduce(c, Reduce(c, pred.actual.lst))
```

- Treating the label of best treatment as the outcome,fitting a single tree model to predict it. Also does variable selection to decide which variables need to be removed from this model fitting.

```
# Treating the label of best treatment as the outcome, fitting a single tree model to predict it


# using R library for the CART algorithm - RPART (Recursive Partitioning And Regression Trees)
library(rpart)

# fitting a single tree model treating the label of best treatment from the previous step as the outcom
temp <- rpart.control(xval = 10, minbucket = 1, minsplit = 1)
rpart.tree <- rpart(pred.rslt ~ ., data = sepsis.X, method = "class", control = temp)
```

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

```
# Pruning the rpart single tree model defined above using the cost-complexity tuning.

# using R library for the CART algorithm - RPART

library(rpart)
library(rpart.plot)


# finding the cp value that corresponds to least cross validation error
cp.1se = rpart.tree$cptable[which.min(rpart.tree$cptable[, "xerror"]), "CP"]

# pruning the single tree model based on the cp value identified above
rpart.tree1 <- prune(rpart.tree, cp = cp.1se)


# predictions using the new tree
```

```r
y.pred1 <- predict(rpart.tree1, sepsis.X, type = "class")

# Accuracy of predictions post pruning
mean(data.frame(y.pred1) == sepsis.Y) * 100
```

```
## [1] 79.57447
```