STAT 542 / CS 598: Homework 4

Pushpit Saxena (netid: pushpit2)

Contents

Question 1 Tuning Random Forests in Virtual Twins

Data Loading:

```
# Reading the data file
df <- read.csv(file="./Sepsis.csv", header = TRUE, sep = ",", row.names = 1)
```

Randomly split the data into training and testing set:

- Using the sample function to split the data into test and train set. Kept the default **splitRatio = 0.75**. Also, this method generate training data for **THERAPY == 1** and **THERAPY == 0**

```
# Method to split the data into test and train set
test_train_split <- function(df, splitRatio=0.75) {
  train <- sample(nrow(df), splitRatio*nrow(df), replace = FALSE)
  trainSet <- df[train,]
  testSet <- df[-train, ]
  trainData_0 <- trainSet[trainSet$THERAPY == 0,]
  trainData_1 <- trainSet[trainSet$THERAPY == 1,]
  return(list("all_train" = trainSet, "train_0" = trainData_0,
              "train_1"=trainData_1, "test" = testSet))
}
```

Function to train/test a virtual twin randomForest model

- In this function I have used **randomForest** library to build two RandomForest regression models (one for training data with THERAPY == 1 and other for THERAPY == 0) and then use the health prediction done by these two models for test data to predict BEST outcome for each of the record.

```
runTwinRF <- function(train_0, train_1, test, mtryVal, nodeSizeVal) {
  set.seed(17)
  rf_0 <- randomForest(Health ~ ., data=train_0[-which(names(df) %in% c("THERAPY", "BEST"))],
                       mtry=mtryVal, nodesize=nodeSizeVal)
  pred_0 <- predict(rf_0, newdata = test[-which(names(df) %in% c("THERAPY", "BEST"))])
  mse_0 = mean((pred_0 - test$Health)^2)

  set.seed(17)
  rf_1 <- randomForest(Health ~ ., data=train_1[-which(names(df) %in% c("THERAPY", "BEST"))],
                       mtry=mtryVal, nodesize=nodeSizeVal)
  pred_1 <- predict(rf_1, newdata = test[-which(names(df) %in% c("THERAPY", "BEST"))])
  mse_1 <- mean((pred_1 - test$Health)^2)

  predFinal = rep(1, nrow(test))
  predFinal[which(pred_0 >= pred_1)] = 0
  accuracy = length(which(predFinal == test$BEST))/nrow(test)
```

```r
  error = length(which(predFinal != test$BEST))/nrow(test)
  return (list("accuracy" = accuracy, "error" = error, "mse_0" = mse_0,
               "mse_1" = mse_1, "finalPred" = predFinal))
}
```

Grid Search for mtry and nodesize : (*I have removed the code from the report pdf, it can be seen in the rmd file*)

- This is a search to figure out the 3 mtry and 3 nodesize values. I have tried mtry values in *[1:(p=11)]* and nodeSize = *[1:50]*.
  - Generated 10 different split of the data, train and test virtual twin model and record the accuracy (correct prediction of BEST/no. of records in testSet)
  - Picked the best **3 mtry (7, 6, 9)** and **3 nodeSize (43, 48, 50)** for 100 iterations. The procedure (*actual code ommitted because it is similar to 100 iteration code later in the report, only difference is 6 mtry and 6 nodeSize values instead of 3*) I followed is to pick the top 6 for each of mtry and nodesize from this grid search and ran 100 iterations and pick the top 3 from that to demonstrate in the report.
  - I have not included the output of this code also in the PDF report (it takes sometime to run, however this code can be ran using the RMD code file)

Running the 100 iterations:

- Following steps are done here:
  - Selected the 3 mtry and 3 nodesize values in grid-search section.
  - Train the twin random forest model with each of the combination of selected mtry and nodesize.
  - Collected the accuracy for each of the 9 models for each of the 100 iterations.
  - Picked the best mtry (9) & nodesize(48,50) combination based on the model giving the best average accuracy over 100 iterations
  - Please note the accuracy is defined as the (correct prediction of BEST) / (total number of record in testSet)

```r
set.seed(13)
mtry <- c(7, 6, 9)
nodeSize <- c(43, 48, 50)
predictionError <- data.frame("Iteration" = numeric(0), "Mtry" = numeric(0),
                              "NodeSize" = numeric(0), "Error" = numeric(0),
                              "Accuracy" = numeric(0),
                              "MSE_0" = numeric(0), "MSE_1" = numeric(0))
set.seed(13)
for (i in 1:100) {
  allData <- test_train_split(df)
  for (m in mtry) {
    for (n in nodeSize) {
      res <- runTwinRF(train_0=allData$train_0, train_1=allData$train_1, allData$test, m, n)
      predictionError <- rbind(predictionError,
                               setNames(as.list(c(i,m,n, res$error, res$accuracy, res$mse_0, res$mse_1))
                                        names(predictionError)))
    }
  }
  # cat("Done: ", i, "\n")
}

dd <- as.data.frame(predictionError %>%
  group_by(Mtry, NodeSize) %>%
  summarize(AccuracyAvgOver100Iteration = mean(Accuracy),
            medianAccuracy = median(Accuracy), MSE_0_Avg = mean(MSE_0), MSE_1_Avg=mean(MSE_1)))
```

Result summary:

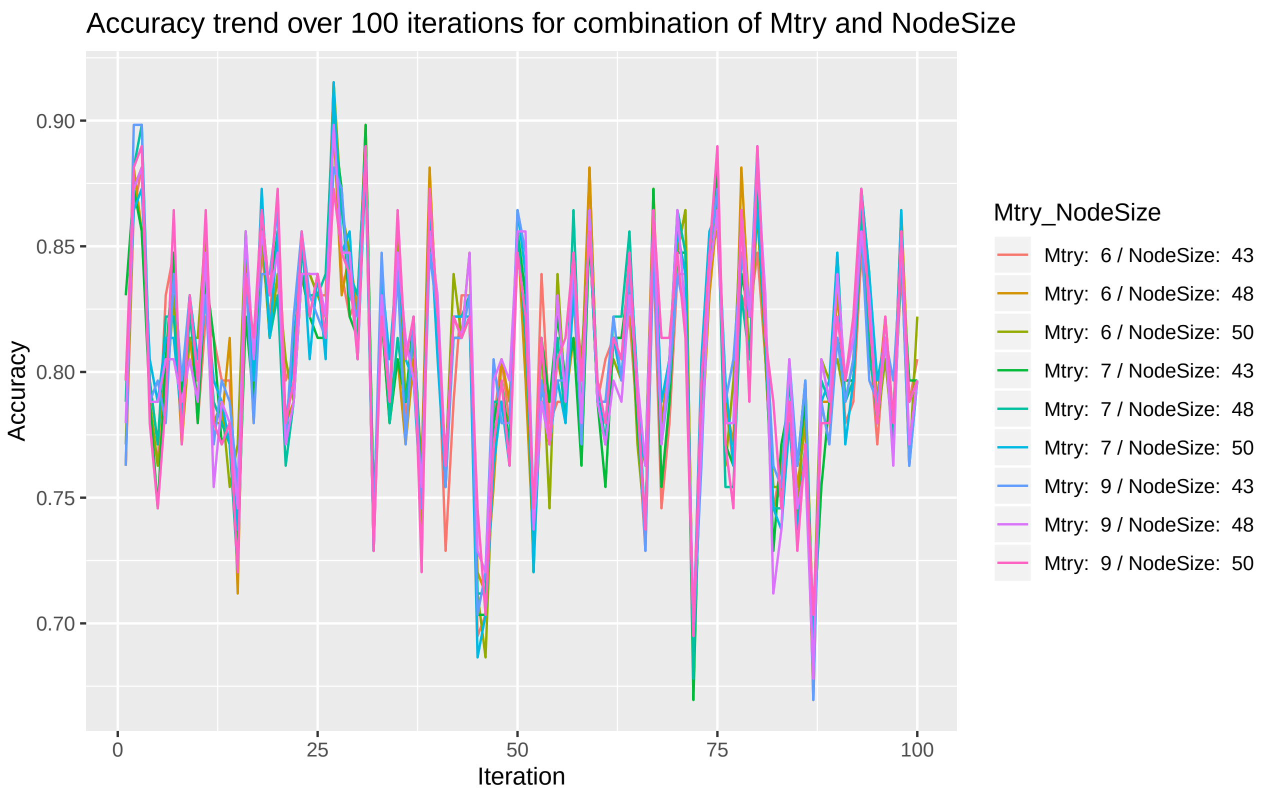
- Average accuracy for all the 9 models (ordered in decreasing order of avg accuracy. Omitted the columns for MSEs for each of the individual (THERAPY == 1 and THERAPY==0) RF, can be included by removing the code *-c(5,6)* below). The accuracy for each combination is very close to each between 0.80 - 0.81 depending on the seed used before RF, this can also be seen from the plots later.

```
dd[with(dd, order(-AccuracyAvgOver100Iteration)), -c(5,6)]
```

```
##   Mtry NodeSize AccuracyAvgOver100Iteration medianAccuracy
## 9    9       50                   0.8066949      0.8093220
## 8    9       48                   0.8060169      0.8050847
## 7    9       43                   0.8056780      0.8008475
## 5    7       48                   0.8054237      0.8008475
## 3    6       50                   0.8052542      0.8050847
## 6    7       50                   0.8051695      0.8050847
## 2    6       48                   0.8040678      0.8050847
## 1    6       43                   0.8031356      0.8050847
## 4    7       43                   0.8030508      0.8135593
```

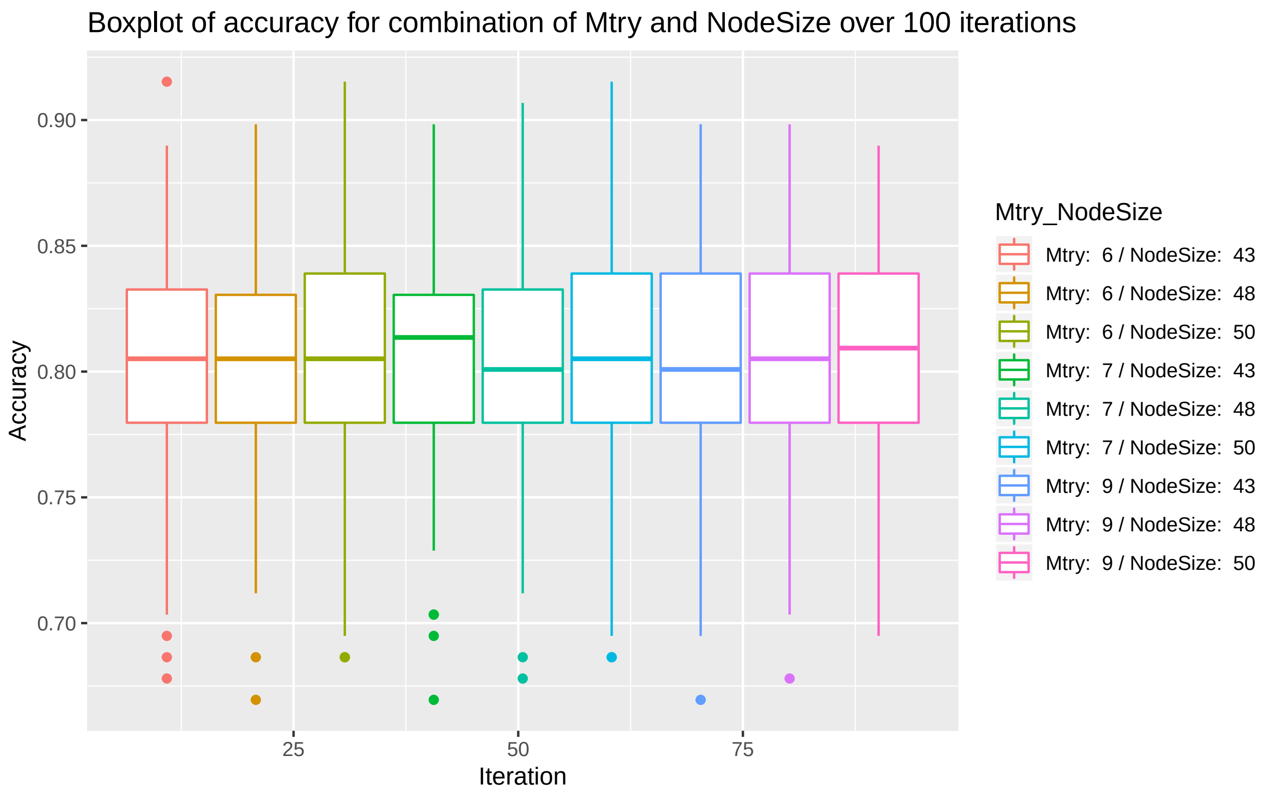
- Accuracy for each of the 9 models (mtry-nodesize combination) over 100 iterations:

```
ggplot(predictionError, aes(x=Iteration, y = Accuracy, color=Mtry_NodeSize)) +
  geom_line() +
  ggtitle("Accuracy trend over 100 iterations for combination of Mtry and NodeSize")
```



- Box plot of the acurracy for each of the 9 Models:

```r
ggplot(predictionError, aes(x=Iteration, y = Accuracy, color=Mtry_NodeSize)) +
  geom_boxplot() +
  ggtitle("Boxplot of accuracy for combination of Mtry and NodeSize over 100 iterations")
```



Question 2 [30 Points] Second Step in Virtual Twins

Optimal tuning parameter based virtual twin model:

- Optimal parameter (Based on Best Average Accuracy):
  - There are two choices which are very close to each other (same average accuracy upto 3 decimal points) **Mtry:9, NodeSize:48** and **Mtry:9, NodeSize:50**. So I tried with both of these combinations and best result I got is with **Mtry:9,NodeSize:48** which is demonstrated in the report below. *nodeSize = 50 can be uncommented below to see the performance with that hyperparam value*

```r
set.seed(17)
mtry = 9
nodeSize = 48
# nodeSize = 50
train_0 = df[df$THERAPY == 0,]
train_1 = df[df$THERAPY == 1,]
allRf <- runTwinRF(train_0 = train_0 , train_1=train_1, df, mtry, nodeSize)
```

Accuracy(AllData) of Virtual Twin model (using optimal tuning parameters):

```r
length(which(allRf$finalPred == df$BEST))/nrow(df)
```

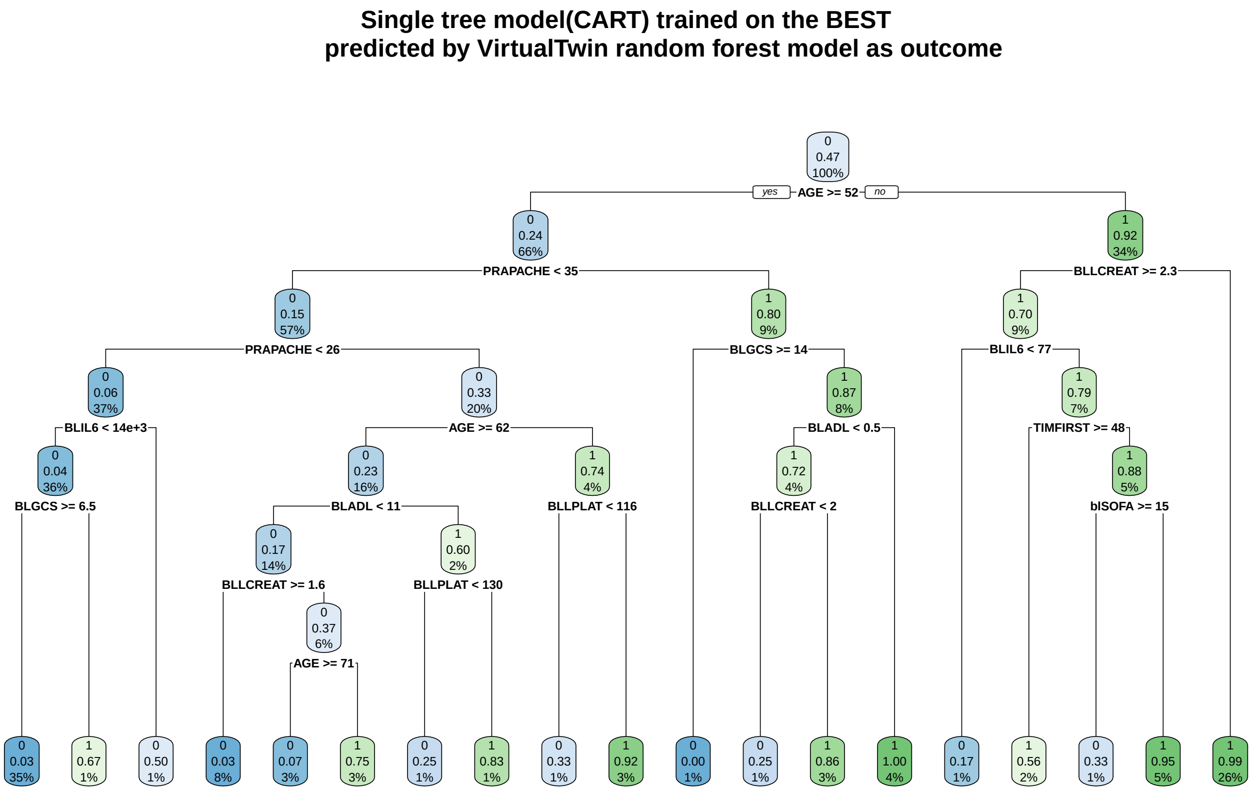
```
## [1] 0.8404255
```

Fiting a single tree model (CART) on the BEST treatment (predicted by Tuned Hyperparameter VirtualTwin randomforest model) as outcome:

- I have used the **rpart** and **rpart.plot** library here to fit/train a single tree model and prune it.

```
set.seed(1512)
new_df <- df
new_df['outcome'] = allRf$finalPred
library(rpart.plot)

# Building a single tree model
sTree <- rpart(outcome ~ ., data=
                 new_df[-which(names(new_df) %in% c("THERAPY", "BEST", "Health"))],
               method = "class", control=rpart.control(minsplit=10, cp=0.001))
tree_pred <- predict(sTree, newdata =
                       new_df[-which(names(df) %in% c("THERAPY", "BEST", "Health"))])
final_pred_tree <- argmax(tree_pred) - 1
rpart.plot(sTree, main="Single tree model(CART) trained on the BEST
           predicted by VirtualTwin random forest model as outcome")
```



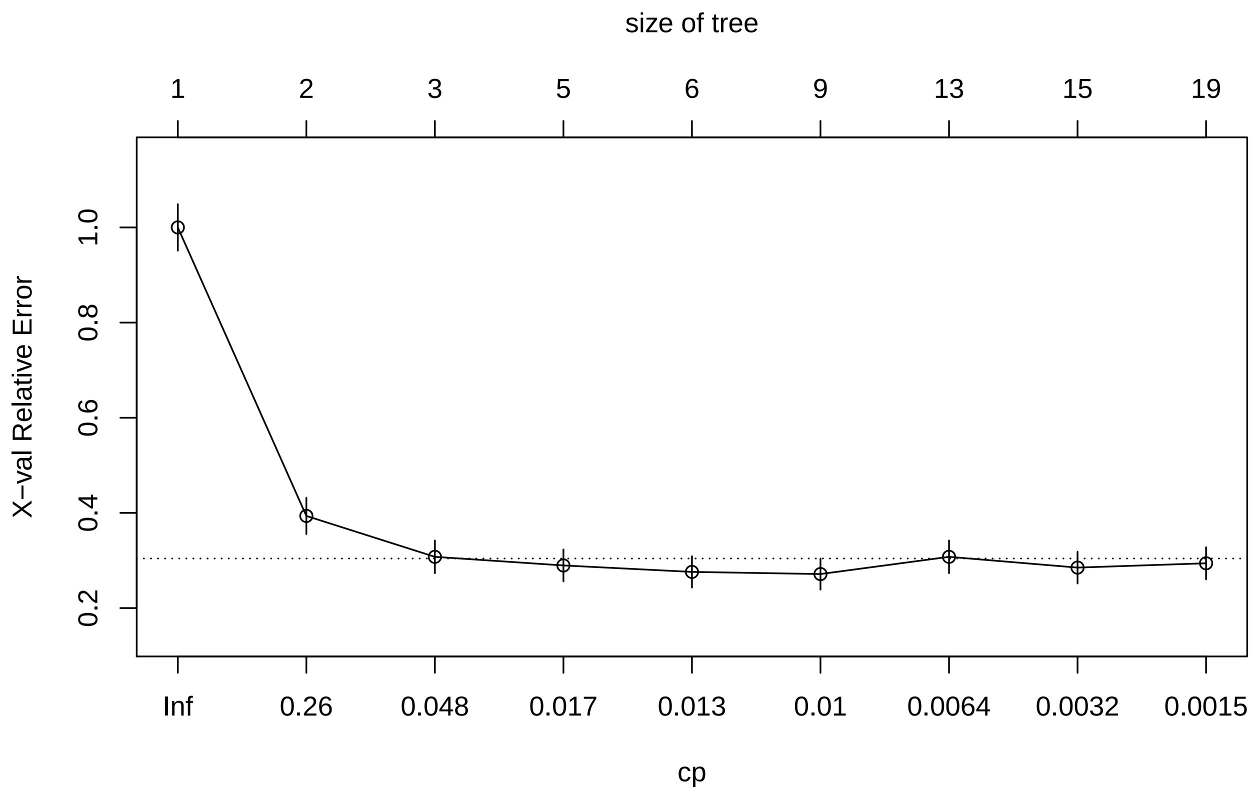
- Single Tree (Not pruned) accuracy:

```
length(which(final_pred_tree == df$BEST))/nrow(df)
```

```
## [1] 0.8489362
```

- Cost Complexity Pruning/Tuning:
  - For pruning the tree I am using **prune** function.
  - Plot of **cp** cost-complexity parameter

```
plotcp(sTree)
```



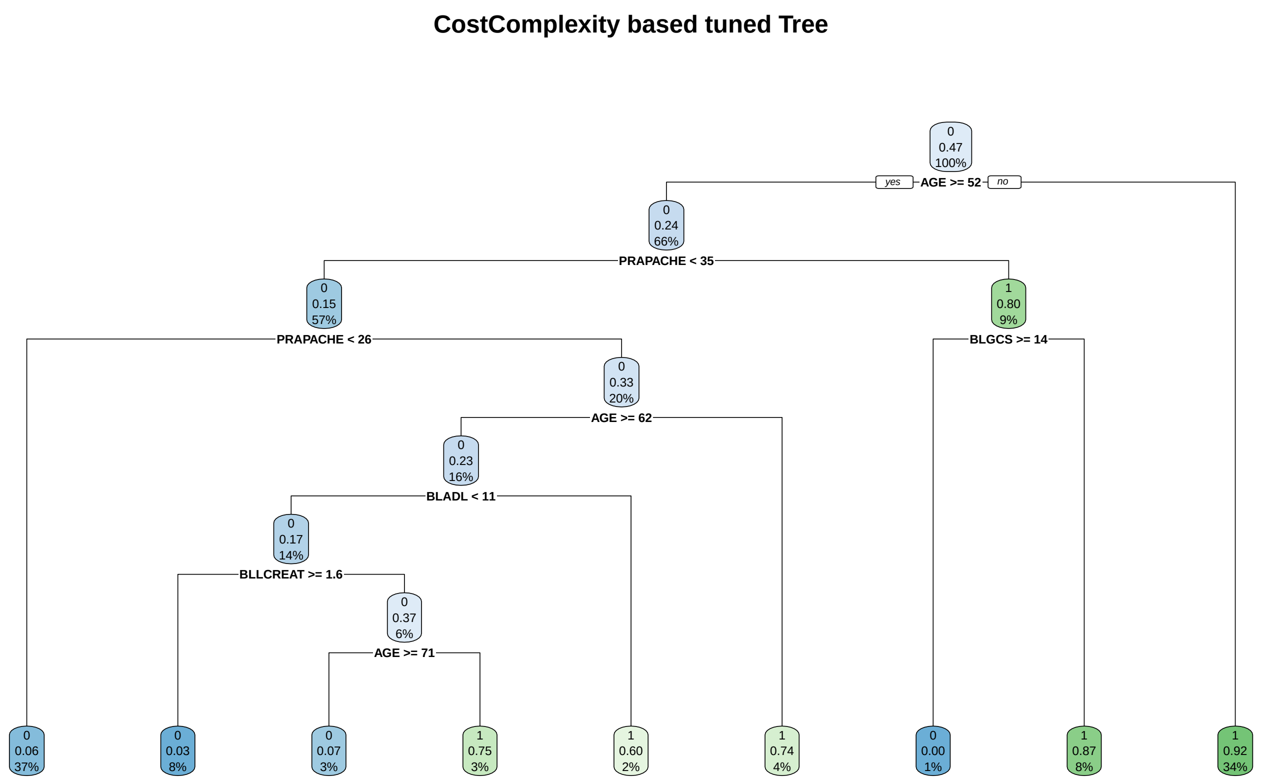
- Values of **cp** as obtained from single tree model fitted before:

```
sTree$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1 0.606334842      0 1.0000000 1.0000000 0.04896149
## 2 0.113122172      1 0.3936652 0.3936652 0.03809937
## 3 0.020361991      2 0.2805430 0.3076923 0.03450852
## 4 0.013574661      4 0.2398190 0.2895928 0.03364432
## 5 0.012066365      5 0.2262443 0.2760181 0.03296745
## 6 0.009049774      8 0.1900452 0.2714932 0.03273606
## 7 0.004524887     12 0.1538462 0.3076923 0.03450852
## 8 0.002262443     14 0.1447964 0.2850679 0.03342153
## 9 0.001000000     18 0.1357466 0.2941176 0.03386437
```

- Using the value of **cp** which minimizes error to prune the single tree model:

```
pfit<- prune(sTree, cp=sTree$cptable[which.min(sTree$cptable[,"xerror"]),"CP"])
tree_pred <- predict(pfit, newdata =
                       new_df[-which(names(df) %in% c("THERAPY", "BEST", "Health"))])
final_pred_tree <- argmax(tree_pred) - 1
rpart.plot(pfit, main="CostComplexity based tuned Tree")
```



- Accuracy of predictions done on Pruned tree (As can be seen here the tuning/pruning based on CP parameter improved the accuracy of the decision model):

```
length(which(final_pred_tree == df$BEST))/nrow(df)
```

```
## [1] 0.887234
```