HW7

Justin Markey

11/29/2021

# Import the data

defaultdata <- read.csv("data/HW7 Data.csv")  
defaultdata$y <- ifelse(defaultdata$y == "yes", 1, 0)   
  
set.seed(12345)  
row\_index<-sample(1:nrow(defaultdata),30000)  
train<-defaultdata[row\_index,]  
valid<-defaultdata[-row\_index,]

# Question 1

## Set proportions

trainyes <-sum(ifelse(train[,"y"] == 1, 1, 0))  
traintotal <- length(train[,"y"])  
trainproportion = trainyes/traintotal  
  
validyes <-sum(ifelse(valid[,"y"] == 1, 1, 0))  
validtotal <- length(valid[,"y"])  
validproportion = validyes/validtotal  
  
  
trainproportion

## [1] 0.1123667

validproportion

## [1] 0.1134251

### Conclusion

The proportion between yes and no in our y variable datasets are very similar. This is a good indication that our train and valid sets have adequate yes and no data for the decision tree fitting.

# Question 2

## Plot with Rattle

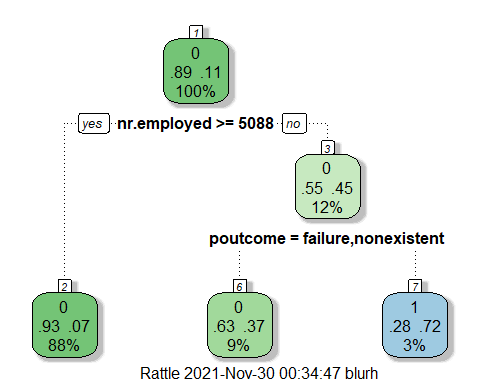
library(rpart)  
library(rattle)

## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.  
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

default.tree<-rpart(data=train, as.factor(y)~., method="class")  
fancyRpartPlot(default.tree)



## Variable importance

default.tree$variable.importance

## nr.employed euribor3m emp.var.rate cons.conf.idx cons.price.idx   
## 908.2393015 786.8523173 531.3543462 482.2396964 401.0605602   
## month poutcome previous marital   
## 236.4119673 153.1229335 12.3486237 0.1899788

## CP Table

default.tree$cptable

## CP nsplit rel error xerror xstd  
## 1 0.05221003 0 1.0000000 1.0000000 0.01622697  
## 2 0.01000000 2 0.8955799 0.8955799 0.01545757

### Conclusion

We can see that the tree isn’t very complicated and because of that, its rigidness is quite low. The algo doesn’t find a lot of the other variables contributing to the performance.

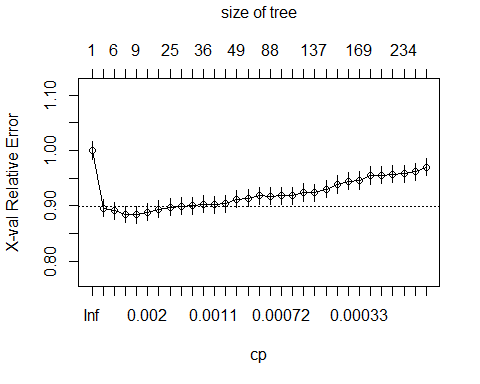
# Question 3

## We need to set the CP value for a larger tree

complicated.tree<-rpart(  
 data=train,  
 as.factor(y)~.,  
 method="class",  
 control=rpart.control(cp=0.0001))

## Identify if the model is overfitting by looking at the CP

plotcp(complicated.tree)



### Conclusion

We can see a minimum in the line plot. This min value is the ideal cp control value. Any higher CP values means the model is starting to overfit to the training data which isn’t good for out of sample predictability.

## Minimize cross validation error by getting best CP

optimal.cp<-complicated.tree$cptable[which.min(complicated.tree$cptable[,"xerror"]),"CP"]  
optimal.cp

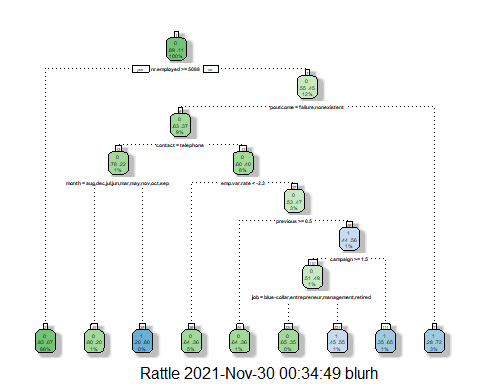
## [1] 0.002175418

### Conclusion

We can now prune the tree to this optimal CP to have a model that’s both predictive and not over fitted.

## Create optimal tree with new CP

optimal.tree<-prune(complicated.tree, cp=optimal.cp)  
fancyRpartPlot(optimal.tree)



# Question 4

## Create random forest

set.seed(67890)  
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.1.2

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':  
##   
## importance

forestmodel<- randomForest(data=train, as.factor(y)~., ntree=500, mtry=5)

## Variable importance for random forest

forestmodel$importance

## MeanDecreaseGini  
## age 855.37613  
## job 417.79021  
## marital 209.79606  
## education 371.89018  
## default 77.30650  
## housing 182.38879  
## loan 134.80719  
## contact 81.56422  
## month 118.82983  
## day\_of\_week 313.15015  
## campaign 391.65577  
## previous 111.54646  
## poutcome 274.10058  
## emp.var.rate 115.02319  
## cons.price.idx 129.75868  
## cons.conf.idx 161.73435  
## euribor3m 825.68199  
## nr.employed 387.41641

# Question 5 AUC

## Predictions and actuals for models

in.sample.actual <- as.factor(train$y)  
in.sample.pred.tree<-predict(optimal.tree)[,2]  
in.sample.pred.forest<-predict(forestmodel, type = "prob")[,2]

## AUC

treeauc <- ROSE::roc.curve(in.sample.actual, in.sample.pred.tree, plotit = FALSE)  
forestauc <- ROSE::roc.curve(in.sample.actual, in.sample.pred.forest, plotit = FALSE)  
  
treeauc

## Area under the curve (AUC): 0.706

forestauc

## Area under the curve (AUC): 0.784

### Conclusion

The forest model has a higher AUC meaning overall it has better in sample performance.

# Question 6 AUC out of sample

out.sample.actual <- as.factor(valid$y)  
out.sample.pred.tree<-predict(optimal.tree, newdata=valid)[,2]  
out.sample.pred.forest<-predict(forestmodel, newdata=valid, type="prob")[,2]

## AUC

OS.treeauc <- ROSE::roc.curve(out.sample.actual, out.sample.pred.tree, plotit = FALSE)  
OS.forestauc <- ROSE::roc.curve(out.sample.actual, out.sample.pred.forest, plotit = FALSE)  
  
OS.treeauc

## Area under the curve (AUC): 0.709

OS.forestauc

## Area under the curve (AUC): 0.781

### Conclusion

The forest model continues to have a higher AUC. This indicates it is better at predicting and distinguishing than the optimal tree structure.