BA\_FINAL

phani varshitha

2023-12-19

# Loading the required packages

library("ISLR")  
library("caret")

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.2

## Loading required package: lattice

library("class")  
library("e1071")

## Warning: package 'e1071' was built under R version 4.3.2

library("dplyr")

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library("tidyverse")

## Warning: package 'tidyverse' was built under R version 4.3.2

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ lubridate 1.9.3 ✔ tibble 3.2.1  
## ✔ purrr 1.0.2 ✔ tidyr 1.3.0  
## ✔ readr 2.1.4

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ✖ purrr::lift() masks caret::lift()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library("ggplot2")  
library("gmodels")  
library("MASS")

##   
## Attaching package: 'MASS'  
##   
## The following object is masked from 'package:dplyr':  
##   
## select

library("broom")  
library("modelr")

##   
## Attaching package: 'modelr'  
##   
## The following object is masked from 'package:broom':  
##   
## bootstrap

library("Hmisc")

## Warning: package 'Hmisc' was built under R version 4.3.2

##   
## Attaching package: 'Hmisc'  
##   
## The following objects are masked from 'package:dplyr':  
##   
## src, summarize  
##   
## The following object is masked from 'package:e1071':  
##   
## impute  
##   
## The following objects are masked from 'package:base':  
##   
## format.pval, units

library("missForest")

## Warning: package 'missForest' was built under R version 4.3.2

library("rpart")  
library("rattle")

## Warning: package 'rattle' was built under R version 4.3.2

## Loading required package: bitops  
## Rattle: A free graphical interface for data science with R.  
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library("pROC")

## Type 'citation("pROC")' for a citation.  
##   
## Attaching package: 'pROC'  
##   
## The following object is masked from 'package:gmodels':  
##   
## ci  
##   
## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library("ROCR")

## Warning: package 'ROCR' was built under R version 4.3.2

library("cutpointr")

## Warning: package 'cutpointr' was built under R version 4.3.2

##   
## Attaching package: 'cutpointr'  
##   
## The following objects are masked from 'package:pROC':  
##   
## auc, roc  
##   
## The following objects are masked from 'package:caret':  
##   
## precision, recall, sensitivity, specificity

library("ROSE")

## Warning: package 'ROSE' was built under R version 4.3.2

## Loaded ROSE 0.0-4

#Loading the data sets

#Loading the train dataset

raw\_data <- read.csv("C:/Users/varshitha/Downloads/Churn\_Train.csv")

# Loading the test dataset

load("C:/Users/varshitha/Downloads/Customers\_To\_Predict.RData")

#Cleaning and transforming the data

#Removing Unnecessary Columns  
Train\_churn <- raw\_data[,-c(1:3)]  
  
#Re-coding few variables  
Train\_churn $churn <- ifelse(Train\_churn $churn =="yes",1,0)  
Train\_churn $international\_plan <- ifelse(Train\_churn$international\_plan =="yes",1,0)  
Train\_churn$voice\_mail\_plan <- ifelse(Train\_churn$voice\_mail\_plan =="yes",1,0)  
  
#Inputing NA Values  
all\_column\_median <- apply(Train\_churn,2,median, na.rm=T)  
   
for(i in colnames(Train\_churn))  
Train\_churn[,i][is.na(Train\_churn[,i])] <- all\_column\_median[i]  
  
#Converting integer to factor  
Train\_churn$churn <- as.factor(Train\_churn$churn)  
  
#Changing the order of the factor levels  
Train\_churn$churn <- factor(Train\_churn$churn,levels(Train\_churn$churn)[c(2,1)])

#Partitioning the data

partition\_data <- createDataPartition(Train\_churn$churn,p=.75,list=F)  
  
Train\_Data <- Train\_churn[partition\_data,]  
Validation\_Data <- Train\_churn[-partition\_data,]

#Logistic Regression Model

set.seed(125)  
train\_control <- trainControl(method = "repeatedcv",number=10,repeats = 3,savePredictions = 'final',classProbs = F)  
   
lr.model <- train(churn~., data = Train\_Data, method = "glm", family="binomial", metric="Accuracy", trControl = train\_control)

#Decision Tree Model

set.seed(765)  
Dec\_Tree.model <- rpart(churn~.,data=Train\_Data,method="class")

#Testing the models over validation set

#Predicting the logistic regression model built over the validation data to check the accuracy  
lr\_validate <- predict(lr.model,Validation\_Data,type ="prob")  
churn.lr.validate <- cbind(Validation\_Data,lr\_validate)  
  
#Predicting the decision tree model built over the validation data to check the accuracy  
dec\_validate <- predict(Dec\_Tree.model,Validation\_Data,type ="prob")  
churn.dec.validate <- cbind(Validation\_Data,dec\_validate)

#Optimal Threshold - Cut Off Point for Logistic Regression

#Logistic Regression  
ROC\_pred\_lr\_test <- prediction(lr\_validate[,1],churn.lr.validate$churn)  
  
ROCR\_perf\_lr\_test <- performance(ROC\_pred\_lr\_test,'tpr','fpr')  
  
acc\_lr\_perf <- performance(ROC\_pred\_lr\_test,"acc")  
  
optimal\_cutoff\_lr <-ROC\_pred\_lr\_test@cutoffs[[1]][which.max(acc\_lr\_perf@y.values[[1]])]  
  
#AUC Value  
roc.curve(churn.lr.validate$churn, lr\_validate[,1], plotit = F)

## Area under the curve (AUC): 0.822

#Decision Tree  
ROC\_pred\_dec\_test <- prediction(dec\_validate[,1],churn.dec.validate$churn)  
  
ROCR\_perf\_dec\_test <- performance(ROC\_pred\_dec\_test,'tpr','fpr')  
  
acc\_dec\_perf <- performance(ROC\_pred\_dec\_test,"acc")  
  
ROC\_pred\_dec\_test@cutoffs[[1]][which.max(acc\_dec\_perf@y.values[[1]])]

## 3304   
## 0.3076923

#AUC Value  
roc.curve(churn.dec.validate$churn,dec\_validate[,1], plotit = F)

## Area under the curve (AUC): 0.899

#Re-Coding Variables - To run the CrossTable()

#Setting the optimal cutoffs for all the models  
#Logistic Regression Model  
churn.lr.validate$prob <- as.factor(ifelse(churn.lr.validate$`1`>0.6705911,"yes","no"))  
#Decision Tree Model  
churn.dec.validate$prob <- as.factor(ifelse(churn.dec.validate$`1`>0.3076923,"yes","no"))  
  
#Converting the churn column back to yes and no  
churn.lr.validate$churn <- as.factor(ifelse(churn.lr.validate$churn==1,"yes","no"))  
churn.dec.validate$churn <- as.factor(ifelse(churn.dec.validate$churn==1,"yes","no"))

#Using CrossTable() to look at the performance metrics and miscalculations for all the models

#Logistic Regression Model  
CrossTable(x=churn.lr.validate$churn,y=churn.lr.validate$prob,prop.chisq = F)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 832   
##   
##   
## | churn.lr.validate$prob   
## churn.lr.validate$churn | no | yes | Row Total |   
## ------------------------|-----------|-----------|-----------|  
## no | 712 | 0 | 712 |   
## | 1.000 | 0.000 | 0.856 |   
## | 0.869 | 0.000 | |   
## | 0.856 | 0.000 | |   
## ------------------------|-----------|-----------|-----------|  
## yes | 107 | 13 | 120 |   
## | 0.892 | 0.108 | 0.144 |   
## | 0.131 | 1.000 | |   
## | 0.129 | 0.016 | |   
## ------------------------|-----------|-----------|-----------|  
## Column Total | 819 | 13 | 832 |   
## | 0.984 | 0.016 | |   
## ------------------------|-----------|-----------|-----------|  
##   
##

###Performance Metrics - Logistic Regression Model### #True Positive (TP) - 13# #True Negative (TN) - 712# #False Positive (FP) - 0# #False Negative (FN) - 107# #Miscalculations - 107# #Accuracy = TP+TN/TP+TN+FP+FN = 13+712/832 = 87.13 %# #Specificity (TNR) = TN/TN+FP = 712/712+0 = 100 %# #Sensitivity (TPR) = TP/TP+FN = 13/13+107 = 10.83 %#

#Decision Tree Model  
CrossTable(x=churn.dec.validate$churn,y=churn.dec.validate$prob,prop.chisq = F)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 832   
##   
##   
## | churn.dec.validate$prob   
## churn.dec.validate$churn | no | yes | Row Total |   
## -------------------------|-----------|-----------|-----------|  
## no | 690 | 22 | 712 |   
## | 0.969 | 0.031 | 0.856 |   
## | 0.961 | 0.193 | |   
## | 0.829 | 0.026 | |   
## -------------------------|-----------|-----------|-----------|  
## yes | 28 | 92 | 120 |   
## | 0.233 | 0.767 | 0.144 |   
## | 0.039 | 0.807 | |   
## | 0.034 | 0.111 | |   
## -------------------------|-----------|-----------|-----------|  
## Column Total | 718 | 114 | 832 |   
## | 0.863 | 0.137 | |   
## -------------------------|-----------|-----------|-----------|  
##   
##

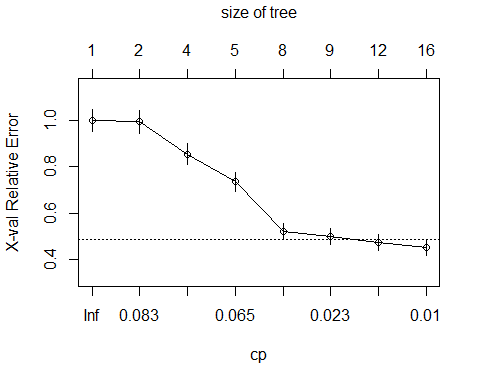
###Performance Metrics - Decision Tree### #True Positive (TP) - 92# #True Negative (TN) - 690# #False Positive (FP) - 22# #False Negative (FN) - 28# #Miscalculations - 50# #Accuracy = TP+TN/TP+TN+FP+FN = 92+690/832 = 93.99 %# #Specificity (TNR) = TN/TN+FP = 690/690+22 = 96.91 %# #Sensitivity (TPR) = TP/TP+FN = 92/92+28 = 76.66 %# ###Eventually, we can see that the decision tree model is working quite good on the validation set when compared to that with the other models. Accuracy, Sensitivity and Specificity is comparatively high so we are proceeding with the decision tree model to be implemented on the “test set”.### #In order to use an effective model on the test set we did try to use pruning as well to check if there’s any rise in the accuracy#

#Pruning the decision tree model

#Base Model  
#The Dec\_Tree.model is the base model which was already built at the beginning  
printcp(Dec\_Tree.model)

##   
## Classification tree:  
## rpart(formula = churn ~ ., data = Train\_Data, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] international\_plan number\_customer\_service\_calls  
## [3] total\_day\_charge total\_eve\_charge   
## [5] total\_intl\_calls total\_intl\_minutes   
## [7] total\_night\_minutes voice\_mail\_plan   
##   
## Root node error: 363/2501 = 0.14514  
##   
## n= 2501   
##   
## CP nsplit rel error xerror xstd  
## 1 0.085399 0 1.00000 1.00000 0.048528  
## 2 0.081267 1 0.91460 0.99449 0.048417  
## 3 0.074380 3 0.75207 0.85399 0.045398  
## 4 0.056474 4 0.67769 0.73554 0.042544  
## 5 0.027548 7 0.49587 0.52066 0.036413  
## 6 0.019284 8 0.46832 0.49862 0.035696  
## 7 0.011019 11 0.41047 0.47107 0.034771  
## 8 0.010000 15 0.36639 0.45179 0.034103

plotcp(Dec\_Tree.model)



#The base model accuracy as seen above is 93.99% (94% approx)  
  
# Pre-Pruning  
# Growing a tree with minsplit of 50 and maxdepth of 6  
Dec\_Tree.model\_preprun <- rpart(churn ~ ., data = Train\_Data, method = "class", control = rpart.control(cp=0,minsplit = 50,maxdepth = 6))  
  
# predicting the above pre-pruned tree on the validation set  
churn.dec.validate.preprun <- predict(Dec\_Tree.model\_preprun, Validation\_Data, type = "prob")  
churn.dec.validate.preprun.df <- cbind(Validation\_Data,churn.dec.validate.preprun)  
  
#Optimal K  
ROC\_pred\_dec.pre\_test <- prediction(churn.dec.validate.preprun[,1],churn.dec.validate.preprun.df$churn)  
  
ROCR\_perf\_dec.pre\_test <- performance(ROC\_pred\_dec.pre\_test,'tpr','fpr')  
  
acc\_dec.pre\_perf <- performance(ROC\_pred\_dec.pre\_test,"acc")  
  
ROC\_pred\_dec.pre\_test@cutoffs[[1]][which.max(acc\_dec.pre\_perf@y.values[[1]])]

## 3169   
## 0.7857143

#AUC Value  
roc.curve(churn.dec.validate.preprun.df$churn,churn.dec.validate.preprun[,1], plotit = F)

## Area under the curve (AUC): 0.890

#Calculating Accuracy  
churn.dec.validate.preprun.df$prob <- as.factor(ifelse(churn.dec.validate.preprun.df$`1`>0.7857143,1,0))  
  
accuracy\_preprun <- mean(churn.dec.validate.preprun.df$churn==churn.dec.validate.preprun.df$prob)  
accuracy\_preprun

## [1] 0.921875

#Post- Pruning  
# Pruning the Dec\_Tree.model based on the optimal cp value  
Dec\_tree.model\_pruned <- prune(Dec\_Tree.model, cp = 0.0100)  
  
#predicting the above pruned tree on the validation set  
churn.dec.validate.pruned <- predict(Dec\_tree.model\_pruned, Validation\_Data, type = "prob")  
churn.dec.validate.pruned.df <- cbind(Validation\_Data,churn.dec.validate.pruned)  
  
#Optimal K  
ROC\_pred\_dec.pos\_test <- prediction(churn.dec.validate.pruned[,1],churn.dec.validate.pruned.df$churn)  
  
ROCR\_perf\_dec.pos\_test <- performance(ROC\_pred\_dec.pos\_test,'tpr','fpr')  
  
acc\_dec.pos\_perf <- performance(ROC\_pred\_dec.pos\_test,"acc")  
  
ROC\_pred\_dec.pos\_test@cutoffs[[1]][which.max(acc\_dec.pos\_perf@y.values[[1]])]

## 3304   
## 0.3076923

#AUC Value  
roc.curve(churn.dec.validate.pruned.df$churn,churn.dec.validate.pruned[,1], plotit = F)

## Area under the curve (AUC): 0.899

#Calculating Accuracy  
churn.dec.validate.pruned.df$prob <- as.factor(ifelse(churn.dec.validate.pruned.df$`1`>0.3076923,1,0))  
  
accuracy\_postprun <- mean(churn.dec.validate.pruned.df$churn==churn.dec.validate.pruned.df$prob)  
accuracy\_postprun

## [1] 0.9399038

#Comparing the base mode, pre-pruning model and post pruning model's accuracy  
#Base model accuracy = 0.9399038  
data.frame(accuracy\_preprun, accuracy\_postprun)

## accuracy\_preprun accuracy\_postprun  
## 1 0.921875 0.9399038

#Pruning can not have significant impact when the data is imbalanced and this can be a possible reason to not see any change in the accuracy in “post - pruning model”. We are thereby affirming to the base model and using the base model (Dec\_Tree\_Model) to predict the test set.#

#Prediction on the test set

#Re-coding the variables as being used in the train set  
Customers\_To\_Predict$international\_plan <- ifelse(Customers\_To\_Predict$international\_plan =="yes",1,0)  
Customers\_To\_Predict$voice\_mail\_plan <- ifelse(Customers\_To\_Predict$voice\_mail\_plan =="yes",1,0)  
  
#Predicting the decision tree model built over the unseen data  
dec.test <- predict(Dec\_Tree.model,Customers\_To\_Predict,type="prob")  
churn.dec.test <- cbind(Customers\_To\_Predict,dec.test)  
  
#Setting the baseline model cutoff point i.e. 0.3076923 on the test set  
churn.dec.test$prob <- as.factor(ifelse(churn.dec.test$`1`>0.3076923,"yes","no"))  
  
#Deleting the probability columns 1 and 0  
churn.dec.test <- churn.dec.test[,-c(20:21)]

*The final file to look for the churns and no churns is the churn.dec.test*