BA HW 6

Devarshi Pancholi

10/31/2019

Problem 1: Student Application Data(Redo)

```r
table(stu$x.Status.1)
```

```
##
## APPLICANT  PROSPECT   SUSPECT
##       463       698     33613
```

```r
prop.table(table(stu$x.Status.1))
```

```
##
##  APPLICANT   PROSPECT    SUSPECT
## 0.01331455 0.02007247 0.96661299
```

```r
#barplot(table(stu$x.Status.1))

stu$APPLICANT <- as.logical(0)
stu$PROSPECT <- as.logical(0)
stu$SUSPECT <- as.logical(0)

for(i in 1:nrow(stu)) {
  if (stu$x.Status.1[i]=="APPLICANT")
    stu$APPLICANT[i] <- as.logical(1)
  else if(stu$x.Status.1[i]=="PROSPECT")
    stu$PROSPECT[i] <- as.logical(1)
  else
    stu$SUSPECT[i] <- as.logical(1)
}

#barplot(table(stu$APPLICANT))
View(stu)

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
```

```r
admit <- select(stu, x.State, x.Gender, x.Source, x.GPA, x.DistancetoCampus_miles, x.HouseholdIncome, x

admitlr <- select(stu, x.State, x.Gender, x.GPA, x.DistancetoCampus_miles, x.HouseholdIncome, x.InState
str(admit)
```

```
## 'data.frame':    34774 obs. of  8 variables:
##  $ x.State               : Factor w/ 51 levels "AK","AL","AR",..: 35 35 35 35 35 35 35 35 35 35 ..
##  $ x.Gender              : Factor w/ 2 levels "Female","Male": 1 2 2 1 1 2 2 2 1 2 ...
##  $ x.Source              : Factor w/ 29 levels "ACT-Juniors_Search",..: 10 10 10 10 10 10 10 10 10
##  $ x.GPA                 : num  2 2 2 2 2.3 2.3 2.3 2.3 2.3 2.3 ...
##  $ x.DistancetoCampus_miles: num  49.3 43.3 53.2 46.7 38.9 ...
##  $ x.HouseholdIncome     : num  23022 24838 37150 30499 56764 ...
##  $ x.InState             : Factor w/ 2 levels "N","Y": 1 1 1 1 1 1 1 1 1 1 ...
##  $ APPLICANT             : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  - attr(*, "na.action")= 'omit' Named int  1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr  "1" "2" "3" "4" ...
```

1. Run 3 model to predict if a student will apply to university or not.
2. Create 90:10 split and validate those models using ratio of correct predictions vs total predictions.
3. Create 70:30 split and validate those models using ratio of correct predictions vs total predictions.
4. Asses those 3 model using performance metrics such as accuracy, precision, recall, F-score and G-score.

First, I will build 3 models with 70:30 split. Then I will evaluate them according to the Question 4 and I also will be plotting AUC curve for me to decide up on a particular model

```r
target <- ('APPLICANT')
dependent <- (names(admit)[names(admit) != target])
dependentlr <- (names(admitlr)[names(admitlr) != target])

admit$APPLICANT<-as.factor(admit$APPLICANT)

library(ROSE)
```

```
## Loaded ROSE 0.0-3
```

```r
set.seed(1234)
split <- (.70)
library (caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
index <- createDataPartition(admit$APPLICANT, p=split, list=FALSE)

train.df <- admit[ index,]
test.df <- admit[ -index,]

train.under<-ovun.sample(APPLICANT ~., data = train.df, method = "under", N= 1000)$data
prop.table(table(train.under$APPLICANT))
```

```
##
## FALSE  TRUE
## 0.675 0.325
```

```r
fitControl <- trainControl(method = "none")

lr <- train(train.df[,dependentlr],train.df[,target], method='glm', trControl=fitControl)

rf <- (train(train.under[,dependent],train.under[,target], method='rf', trControl=fitControl))

gbm <- train(train.under[,dependent],train.under[,target], method='gbm', trControl=fitControl)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2422             nan     0.1000    0.0089
##      2        1.2294             nan     0.1000    0.0045
##      3        1.2150             nan     0.1000    0.0070
##      4        1.2011             nan     0.1000    0.0050
##      5        1.1895             nan     0.1000    0.0048
##      6        1.1795             nan     0.1000    0.0046
##      7        1.1705             nan     0.1000    0.0035
##      8        1.1626             nan     0.1000    0.0011
##      9        1.1571             nan     0.1000    0.0023
##     10        1.1497             nan     0.1000    0.0026
##     20        1.1122             nan     0.1000    0.0003
##     40        1.0708             nan     0.1000    0.0006
##     50        1.0591             nan     0.1000   -0.0018
```

```r
lr.predict <- predict(lr,test.df[,dependent],type="raw")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```r
confusionMatrix(lr.predict,test.df[,target], positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE  TRUE
##      FALSE 10293   138
##      TRUE      0     0
##
##                Accuracy : 0.9868
##                  95% CI : (0.9844, 0.9889)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 0.5226
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.98677
##              Prevalence : 0.01323
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : TRUE
##
```

```r
p <- data.frame(Actual = test.df$APPLICANT , Prediction = lr.predict)
p <- table(p)
p
```

```
##        Prediction
## Actual  FALSE  TRUE
##   FALSE 10293     0
##   TRUE    138     0
```

```r
accuracy <- (p[1,1] + p[2,2])/sum(p)
accuracy
```

```
## [1] 0.9867702
```

```r
precision <- (p[2,2]/(p[2,2] + p[1,2]))
precision
```

```
## [1] NaN
```

```r
recall <- (p[2,2]/(p[2,2] + p[2,1]))
recall
```

```
## [1] 0
```

```r
f_score <- 2*((precision*recall)/(precision+recall))
f_score
```

```
## [1] NaN
```

```r
g_score <- sqrt(precision*recall)
g_score
```

```
## [1] NaN
```

```r
rf.predict<-predict(rf,test.df[,dependent],type="raw")
confusionMatrix(rf.predict,test.df[,target], positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  9151   89
##      TRUE   1142   49
##
##                Accuracy : 0.882
##                  95% CI : (0.8756, 0.8881)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0512
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.355072
##             Specificity : 0.889051
##          Pos Pred Value : 0.041142
##          Neg Pred Value : 0.990368
##              Prevalence : 0.013230
##          Detection Rate : 0.004698
##    Detection Prevalence : 0.114179
##       Balanced Accuracy : 0.622062
##
##        'Positive' Class : TRUE
##
```

```r
q <- data.frame(Actual = test.df$APPLICANT , Prediction = rf.predict)
q <- table(q)
q
```

```
##        Prediction
## Actual  FALSE TRUE
##   FALSE  9151 1142
##   TRUE     89   49
```

```
accuracy <- (q[1,1] + q[2,2])/sum(q)
accuracy
```

```
## [1] 0.8819864
```

```
precision <- (q[2,2]/(q[2,2] + q[1,2]))
precision
```

```
## [1] 0.0411419
```

```
recall <- (q[2,2]/(q[2,2] + q[2,1]))
recall
```

```
## [1] 0.3550725
```

```
f_score <- 2*((precision*recall)/(precision+recall))
f_score
```

```
## [1] 0.07373965
```

```
g_score <- sqrt(precision*recall)
g_score
```

```
## [1] 0.120865
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  9067   95
##      TRUE   1226   43
##
##                Accuracy : 0.8734
##                  95% CI : (0.8668, 0.8797)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0382
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.311594
##             Specificity : 0.880890
##          Pos Pred Value : 0.033885
##          Neg Pred Value : 0.989631
```

```
##               Prevalence : 0.013230
##           Detection Rate : 0.004122
##     Detection Prevalence : 0.121657
##        Balanced Accuracy : 0.596242
##
##         'Positive' Class : TRUE
##
```

```
##         Prediction
## Actual  FALSE TRUE
##   FALSE  9067 1226
##   TRUE     95   43
```

```
## [1] 0.8733583
```

```
## [1] 0.03388495
```

```
## [1] 0.3115942
```

```
## [1] 0.06112296
```

```
## [1] 0.1027538
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

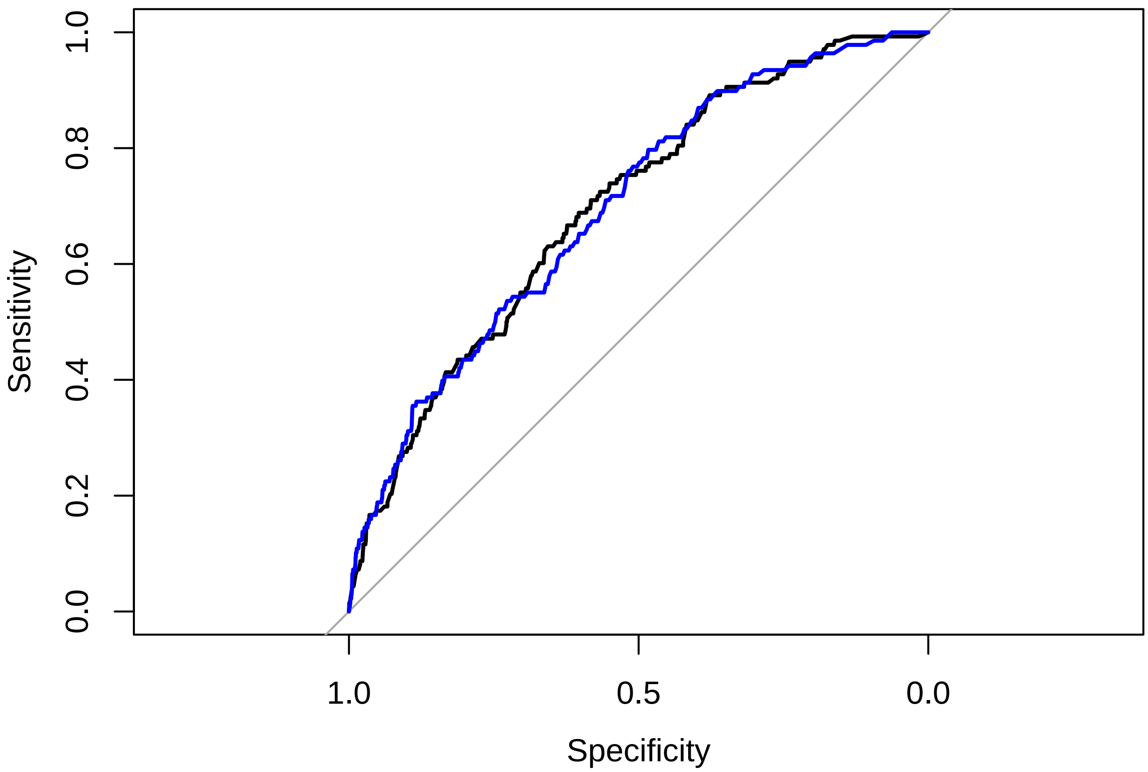
```r
gbm.probs <- predict(gbm,test.df[,dependent],type="prob")
rf.probs <- predict(rf,test.df[,dependent],type="prob")

gbm.plot<-plot(roc(test.df$APPLICANT,gbm.probs[,2]))
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```r
rf.plot<-lines(roc(test.df$APPLICANT,rf.probs[,2]), col="blue")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```



Now, we will plot the same models but with 90:10 split and evaluate those models based on the criteria mentioned in Question 4. ROC curve will also be plotted.

```
set.seed(1234)
split2 <- (.90)
index2 <- createDataPartition(admit$APPLICANT, p=split2, list=FALSE)

train.df2 <- admit[ index2,]
test.df2 <- admit[ -index2,]

train.under2<-ovun.sample(APPLICANT ~., data = train.df2, method = "under", N= 1000)$data
prop.table(table(train.under2$APPLICANT))
```

```
##
## FALSE  TRUE
## 0.583 0.417
```

```
fitControl <- trainControl(method = "none")

lr2 <- train(train.df2[,dependentlr],train.df2[,target], method='glm', trControl=fitControl)

rf2 <- (train(train.under2[,dependent],train.under2[,target], method='rf', trControl=fitControl))

gbm2 <- train(train.under2[,dependent],train.under2[,target], method='gbm', trControl=fitControl)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
```

```
##      1        1.3466             nan     0.1000    0.0050
##      2        1.3362             nan     0.1000    0.0043
##      3        1.3269             nan     0.1000    0.0036
##      4        1.3186             nan     0.1000    0.0031
##      5        1.3120             nan     0.1000   -0.0003
##      6        1.3067             nan     0.1000    0.0021
##      7        1.3007             nan     0.1000    0.0026
##      8        1.2938             nan     0.1000    0.0019
##      9        1.2889             nan     0.1000    0.0020
##     10        1.2836             nan     0.1000    0.0020
##     20        1.2471             nan     0.1000    0.0011
##     40        1.2167             nan     0.1000    0.0002
##     50        1.2090             nan     0.1000   -0.0016
```

```
lr.predict2 <- predict(lr2,test.df2[,dependent],type="raw")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
## == : prediction from a rank-deficient fit may be misleading
```

```
confusionMatrix(lr.predict2,test.df2[,target], positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  3431   46
##      TRUE      0    0
##
##                Accuracy : 0.9868
##                  95% CI : (0.9824, 0.9903)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 0.5391
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 3.247e-11
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.98677
##              Prevalence : 0.01323
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : TRUE
##
```

```
p2 <- data.frame(Actual = test.df2$APPLICANT , Prediction = lr.predict2)
p2 <- table(p2)
p2
```

```
##        Prediction
## Actual  FALSE TRUE
##   FALSE  3431    0
##   TRUE     46    0
```

```r
accuracy <- (p2[1,1] + p2[2,2])/sum(p2)
accuracy
```

```
## [1] 0.9867702
```

```r
precision <- (p2[2,2]/(p2[2,2] + p2[1,2]))
precision
```

```
## [1] NaN
```

```r
recall <- (p2[2,2]/(p2[2,2] + p2[2,1]))
recall
```

```
## [1] 0
```

```r
f_score <- 2*((precision*recall)/(precision+recall))
f_score
```

```
## [1] NaN
```

```r
g_score <- sqrt(precision*recall)
g_score
```

```
## [1] NaN
```

```r
rf.predict2 <- predict(rf2,test.df2[,dependent],type="raw")
confusionMatrix(rf.predict2,test.df2[,target], positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE  2688   21
##      TRUE    743   25
##
##                Accuracy : 0.7803
##                  95% CI : (0.7661, 0.7939)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0374
##
##  Mcnemar's Test P-Value : <2e-16
##
```

```
##             Sensitivity : 0.54348
##             Specificity : 0.78345
##          Pos Pred Value : 0.03255
##          Neg Pred Value : 0.99225
##              Prevalence : 0.01323
##          Detection Rate : 0.00719
##    Detection Prevalence : 0.22088
##       Balanced Accuracy : 0.66346
##
##        'Positive' Class : TRUE
##
```

```
q2 <- data.frame(Actual = test.df2$APPLICANT , Prediction = rf.predict2)
q2 <- table(q2)
q2
```

```
##        Prediction
## Actual  FALSE TRUE
##   FALSE  2688  743
##   TRUE     21   25
```

```
accuracy <- (q2[1,1] + q2[2,2])/sum(q2)
accuracy
```

```
## [1] 0.7802703
```

```
precision <- (q2[2,2]/(q2[2,2] + q2[1,2]))
precision
```

```
## [1] 0.03255208
```

```
recall <- (q2[2,2]/(q2[2,2] + q2[2,1]))
recall
```

```
## [1] 0.5434783
```

```
f_score <- 2*((precision*recall)/(precision+recall))
f_score
```

```
## [1] 0.06142506
```

```
g_score <- sqrt(precision*recall)
g_score
```

```
## [1] 0.1330088
```

```
gbm.predict2 <- predict(gbm2,test.df2[,dependent],type="raw")
confusionMatrix(gbm.predict2,test.df2[,target], positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction FALSE TRUE
##      FALSE  2746   22
##      TRUE    685   24
##
##                Accuracy : 0.7967
##                  95% CI : (0.7829, 0.8099)
##     No Information Rate : 0.9868
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0397
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.521739
##             Specificity : 0.800350
##          Pos Pred Value : 0.033850
##          Neg Pred Value : 0.992052
##              Prevalence : 0.013230
##          Detection Rate : 0.006903
##    Detection Prevalence : 0.203911
##       Balanced Accuracy : 0.661044
##
##        'Positive' Class : TRUE
##
```

```
r2 <- data.frame(Actual = test.df2$APPLICANT , Prediction = gbm.predict2)
r2 <- table(r2)
r2
```

```
##        Prediction
## Actual  FALSE TRUE
##   FALSE  2746  685
##   TRUE     22   24
```

```
accuracy <- (r2[1,1] + r2[2,2])/sum(r2)
accuracy
```

```
## [1] 0.7966638
```

```
precision <- (r2[2,2]/(r2[2,2] + r2[1,2]))
precision
```

```
## [1] 0.03385049
```

```
recall <- (r2[2,2]/(r2[2,2] + r2[2,1]))
recall
```

```
## [1] 0.5217391
```

```r
f_score <- 2*((precision*recall)/(precision+recall))
f_score
```

```
## [1] 0.06357616
```

```r
g_score <- sqrt(precision*recall)
g_score
```

```
## [1] 0.1328952
```

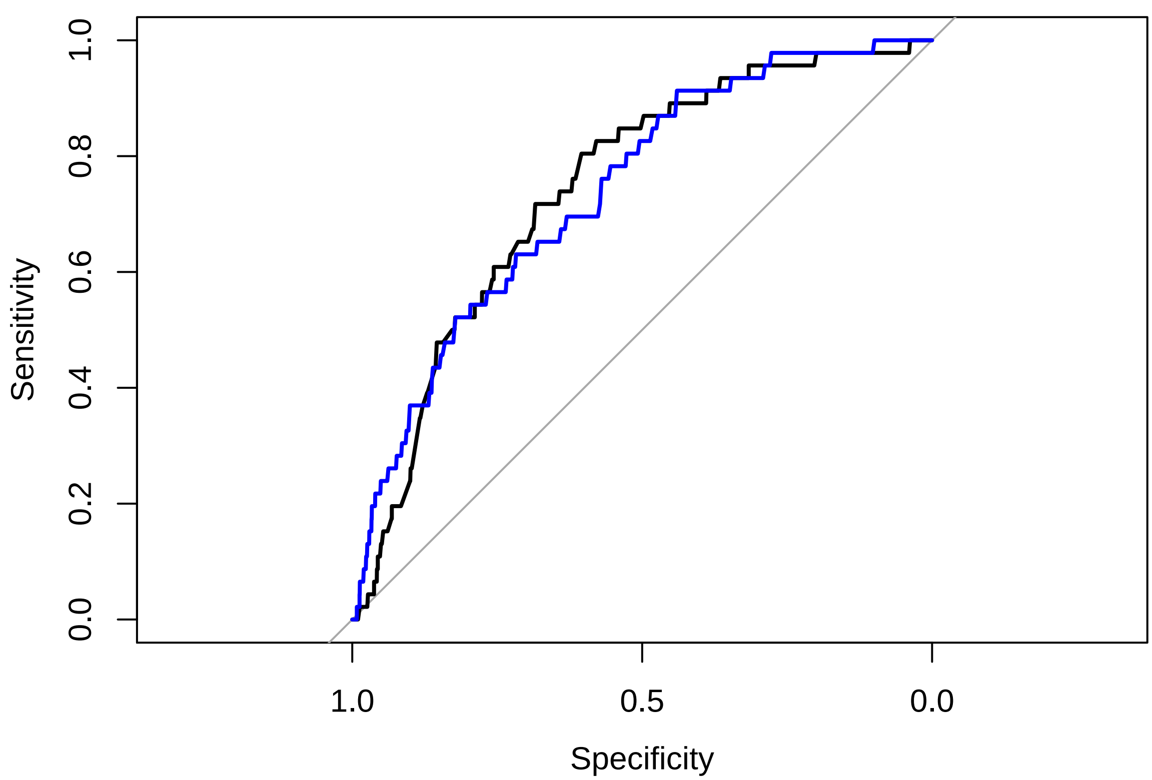
```r
gbm.probs2 <- predict(gbm2,test.df2[,dependent],type="prob")
rf.probs2 <- predict(rf2,test.df2[,dependent],type="prob")

gbm.plot2 <- plot(roc(test.df2$APPLICANT,gbm.probs2[,2]))
```

```
## Setting levels: control = FALSE, case = TRUE
```

```
## Setting direction: controls < cases
```

```r
rf.plot2 <- lines(roc(test.df2$APPLICANT,rf.probs2[,2]), col="blue")
```

```
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
```



```
rfOver <- varImp(rf)
rfOver
```

```
## rf variable importance
##
##                          Overall
## x.HouseholdIncome        100.000
## x.DistancetoCampus_miles  98.874
## x.State                   39.452
## x.Source                  10.655
## x.Gender                   9.045
## x.InState                  5.245
## x.GPA                      0.000
```

5. Select the best model and give actionable recommendations to the marketing department.

Based on the experimentation above, the results are better for the split 90:10. And in that split gradient boosting model performs better. So that can be considered as the best model.

Based on the variable importance in that model we can provide marketing strategies.

1). As distance from campus to school is an important factor, we could target specific students who are closer to the campus. Students from the same state as the campus are most likely to come to school.

2). Students whose parents have high household income are more interested in coming to this school. Hence such students shlould be targetted.