BA HW 6

Devarshi Pancholi

10/31/2019

Problem 1: Student Application Data(Redo)

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

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

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

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

```
#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
```

```
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

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

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

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

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
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,]

fitControl <- trainControl(method = "none")

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

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

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

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.1409             nan     0.1000    0.0004
##      2        0.1401             nan     0.1000    0.0003
##      3        0.1396             nan     0.1000    0.0002
##      4        0.1388             nan     0.1000    0.0003
##      5        0.1383             nan     0.1000    0.0002
##      6        0.1380             nan     0.1000    0.0001
##      7        0.1378             nan     0.1000    0.0000
##      8        0.1373             nan     0.1000    0.0002
##      9        0.1369             nan     0.1000    0.0002
##     10        0.1367             nan     0.1000    0.0000
##     20        0.1349             nan     0.1000    0.0000
##     40        0.1332             nan     0.1000    0.0000
##     50        0.1328             nan     0.1000   -0.0000
```

```r
lrOver <- summary(lr)
lrOver
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2629  -0.1964  -0.1384  -0.0914   3.8151
##
## Coefficients: (1 not defined because of singularities)
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)                1.292e+01  3.956e+03   0.003    0.997
```

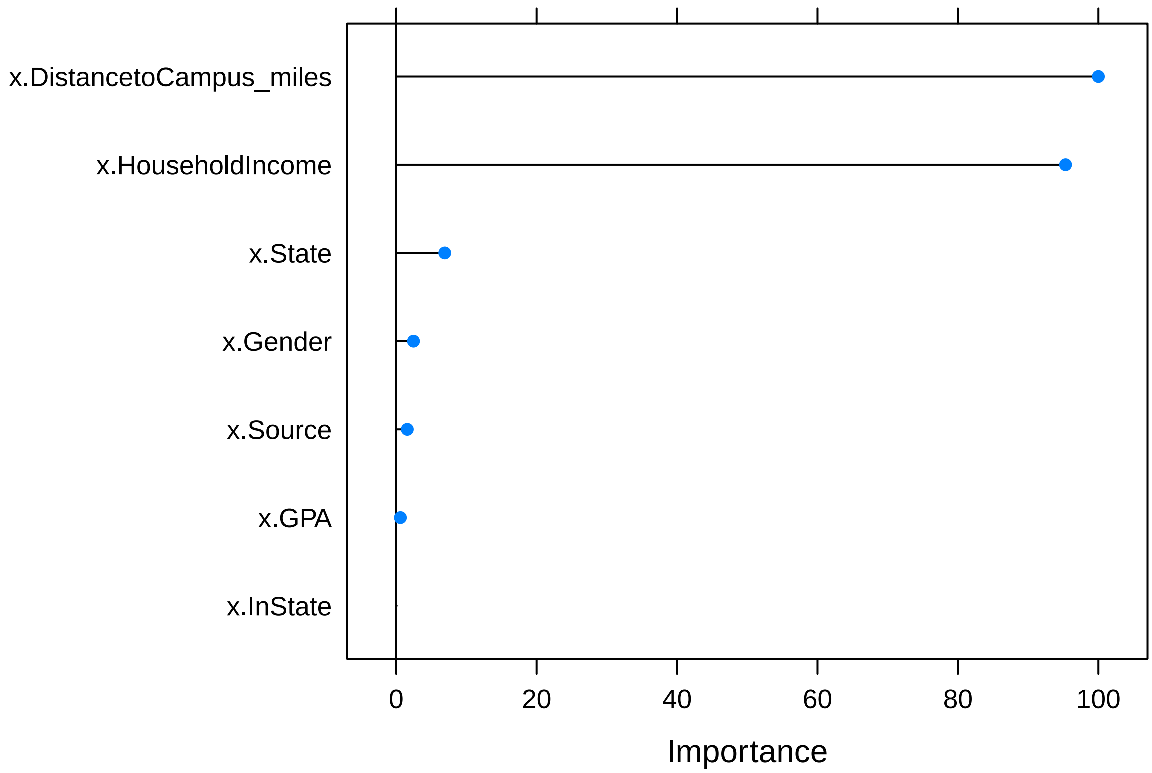
```
## x.StateCA                  5.935e+00  3.983e+03   0.001    0.999
## x.StateCO                 -7.158e+00  5.595e+03  -0.001    0.999
## x.StateCT                 -1.434e+01  3.956e+03  -0.004    0.997
## x.StateDC                 -2.445e+01  4.262e+03  -0.006    0.995
## x.StateFL                 -1.594e+01  4.248e+03  -0.004    0.997
## x.StateGA                 -1.815e+01  4.563e+03  -0.004    0.997
## x.StateIL                 -6.831e+00  3.956e+03  -0.002    0.999
## x.StateIN                 -1.920e+01  5.595e+03  -0.003    0.997
## x.StateLA                 -1.257e+01  4.843e+03  -0.003    0.998
## x.StateMA                 -1.427e+01  3.956e+03  -0.004    0.997
## x.StateMD                 -1.409e+01  3.956e+03  -0.004    0.997
## x.StateME                 -2.566e+01  4.556e+03  -0.006    0.996
## x.StateMN                 -1.350e+01  5.595e+03  -0.002    0.998
## x.StateMO                 -1.413e+01  5.595e+03  -0.003    0.998
## x.StateMT                 -1.518e+00  5.595e+03   0.000    1.000
## x.StateNC                 -2.194e+01  4.553e+03  -0.005    0.996
## x.StateNE                 -1.206e+01  5.595e+03  -0.002    0.998
## x.StateNH                 -1.419e+01  3.956e+03  -0.004    0.997
## x.StateNJ                 -1.472e+01  3.956e+03  -0.004    0.997
## x.StateNV                  1.892e+01  3.956e+03   0.005    0.996
## x.StateNY                 -1.453e+01  3.956e+03  -0.004    0.997
## x.StatePA                 -1.453e+01  3.956e+03  -0.004    0.997
## x.StateRI                 -1.560e+01  3.956e+03  -0.004    0.997
## x.StateTN                 -2.102e+01  5.595e+03  -0.004    0.997
## x.StateTX                 -8.432e+00  4.425e+03  -0.002    0.998
## x.StateVA                 -2.492e+01  3.968e+03  -0.006    0.995
## x.StateVT                 -2.586e+01  3.974e+03  -0.007    0.995
## x.StateWI                 -1.765e+01  5.595e+03  -0.003    0.997
## x.StateWY                 -3.632e+00  5.595e+03  -0.001    0.999
## x.GenderMale              -4.700e-01  1.167e-01  -4.027 5.66e-05 ***
## x.GPA                     -1.468e-01  1.166e+00  -0.126    0.900
## x.DistancetoCampus_miles -1.325e-02  3.098e-03  -4.275 1.91e-05 ***
## x.HouseholdIncome         -1.175e-05  1.654e-06  -7.102 1.23e-12 ***
## x.InStateY                        NA         NA       NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3451.2  on 24342  degrees of freedom
## Residual deviance: 3219.8  on 24309  degrees of freedom
## AIC: 3287.8
##
## Number of Fisher Scoring iterations: 16
```

```
#plot(lrOver)

rfOver <- varImp(rf)
rfOver
```

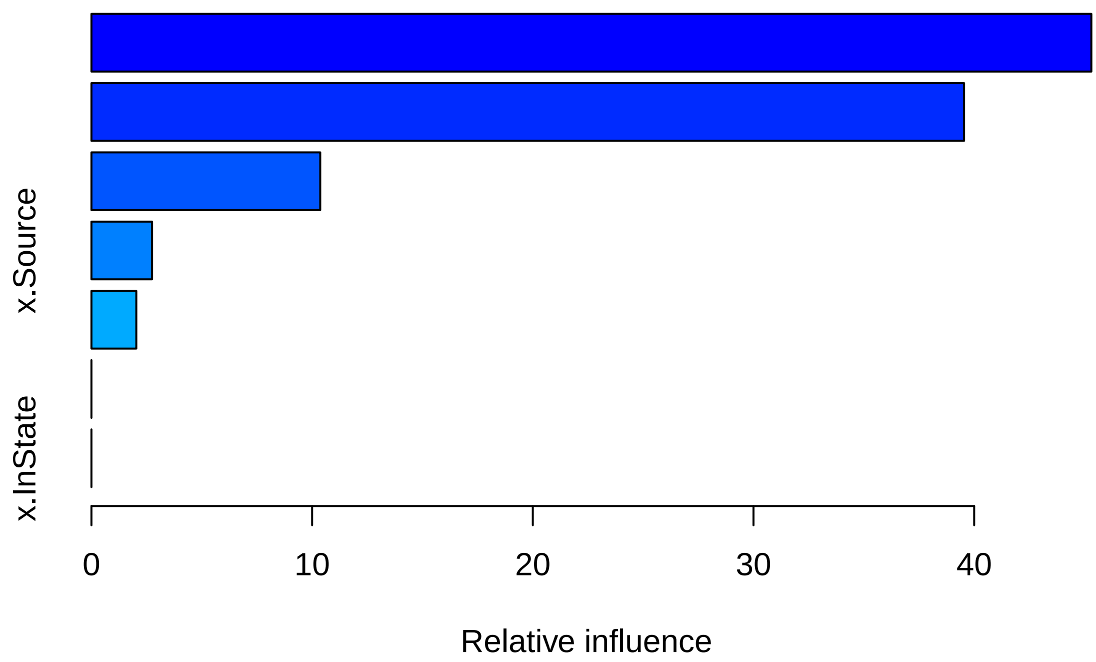
```
## rf variable importance
##
##                          Overall
## x.DistancetoCampus_miles 100.000
```

```
## x.HouseholdIncome          95.327
## x.State                     6.919
## x.Gender                    2.464
## x.Source                    1.583
## x.GPA                       0.601
## x.InState                   0.000
```

```r
plot(rfOver)
```



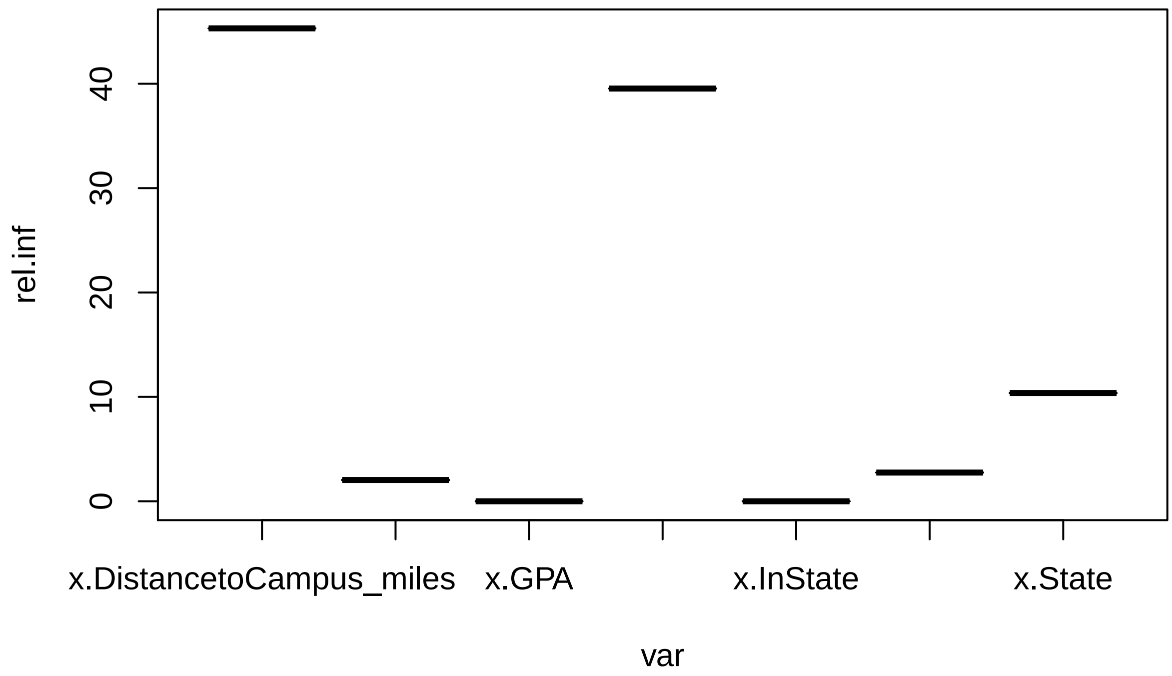

```r
gbmOver <- summary(gbm)
```




```r
gbmOver
```

```
##                                                 var   rel.inf
## x.DistancetoCampus_miles x.DistancetoCampus_miles 45.309491
## x.HouseholdIncome                 x.HouseholdIncome 39.541585
## x.State                                     x.State 10.368049
## x.Source                                   x.Source  2.747476
## x.Gender                                   x.Gender  2.033398
## x.GPA                                         x.GPA  0.000000
## x.InState                                 x.InState  0.000000
```

```r
plot(gbmOver)
```




```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
##
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
## 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
##
```

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

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

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

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

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

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

```r
library(pROC)
```

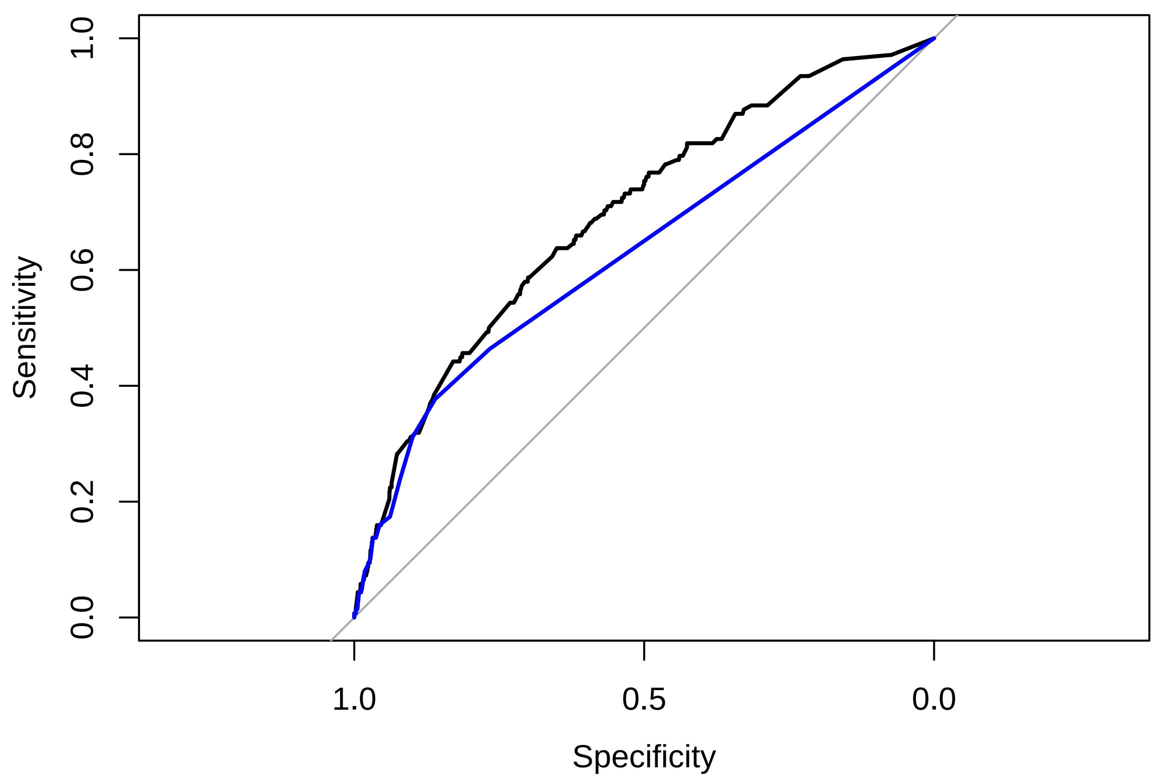
```
## Type 'citation("pROC")' for a citation.

##
## 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,]

fitControl <- trainControl(method = "none")

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

rf2 <- (train(train.df2[,dependent],train.df2[,target], method='rf', trControl=fitControl))

gbm2 <- train(train.df2[,dependent],train.df2[,target], method='gbm', trControl=fitControl)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        0.1411             nan     0.1000    0.0001
##      2        0.1404             nan     0.1000    0.0003
##      3        0.1398             nan     0.1000    0.0002
##      4        0.1391             nan     0.1000    0.0003
##      5        0.1386             nan     0.1000    0.0002
##      6        0.1382             nan     0.1000    0.0001
##      7        0.1378             nan     0.1000    0.0002
##      8        0.1375             nan     0.1000    0.0000
##      9        0.1372             nan     0.1000    0.0001
##     10        0.1368             nan     0.1000    0.0002
##     20        0.1354             nan     0.1000   -0.0000
##     40        0.1339             nan     0.1000    0.0000
##     50        0.1334             nan     0.1000    0.0000
```

```
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
```

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

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

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

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

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

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

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

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

```
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  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
##
```

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

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

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

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

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

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

```r
recall <- (q2[2,2]/(q2[2,2] + q2[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
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  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
##
```

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

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

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

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

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

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

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

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

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

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

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

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

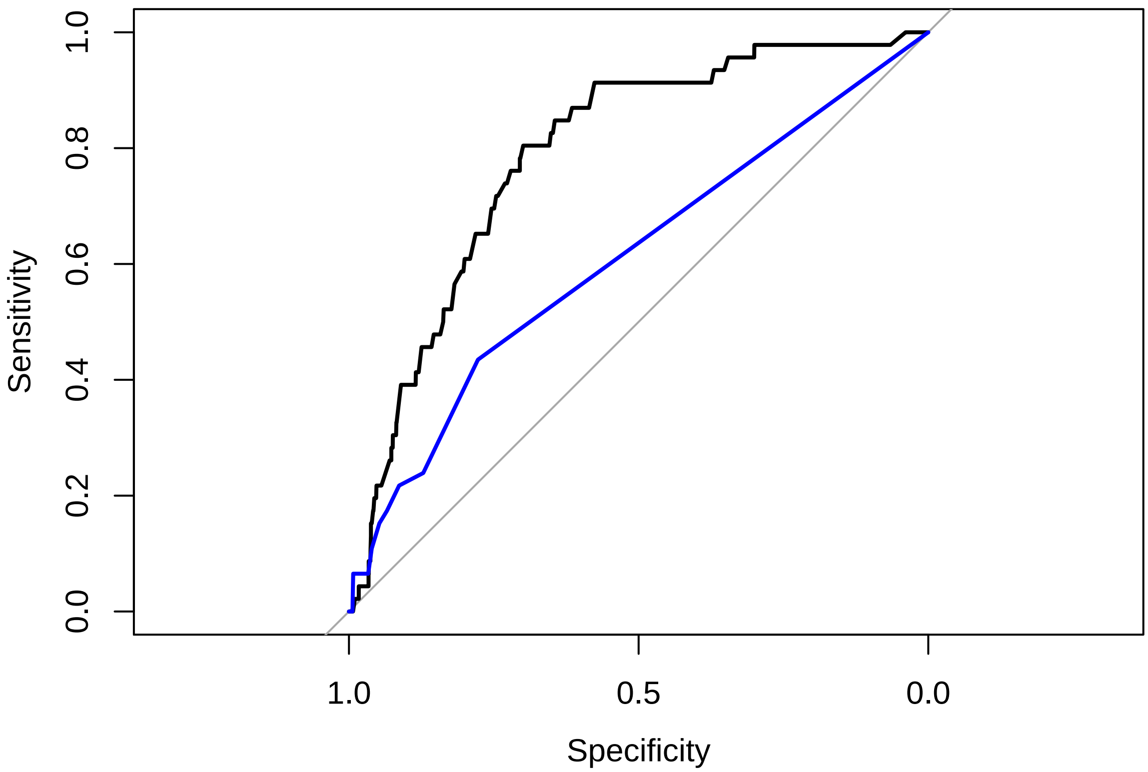
```
gbm.probs2 <- predict(gbm2,test.df2[,dependent],type="prob")
rf.probs2 <- predict(rf2,test.df2[,dependent],type="prob")

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

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

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

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

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



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