INFS 692 - Final Project Model 1

Agata

2022-12-14

Helper Packages

These are all the packages necessary for running the Model 1 code.

```
library(readr)
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

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

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

```
library(ggplot2)
library(ggpubr)
```

```
##
## Attaching package: 'ggpubr'

## The following object is masked from 'package:plyr':
##
##     mutate
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(COUNT)
```

```
## Loading required package: msme

## Loading required package: MASS

##
## Attaching package: 'MASS'

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

## Loading required package: lattice

## Loading required package: sandwich
```

```
library(caret)
library(rstatix)
```

```
##
## Attaching package: 'rstatix'

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

## The following objects are masked from 'package:plyr':
##
##     desc, mutate

## The following object is masked from 'package:stats':
##
##     filter
```

```
library(modeldata)
library(rsample)    # for creating validation splits
library(recipes)    # for feature engineering
```

```
##
## Attaching package: 'recipes'

## The following object is masked from 'package:stats':
##
##     step
```

```r
library(purrr)      #for mapping
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:caret':
##
##     lift

## The following object is masked from 'package:plyr':
##
##     compact
```

```r
library(tidyverse)  # for filtering
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --

## v tibble  3.1.8     v stringr 1.5.0
## v tidyr   1.2.1     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::arrange()     masks plyr::arrange()
## x gridExtra::combine() masks dplyr::combine()
## x purrr::compact()     masks plyr::compact()
## x dplyr::count()       masks plyr::count()
## x dplyr::failwith()    masks plyr::failwith()
## x rstatix::filter()    masks dplyr::filter(), stats::filter()
## x stringr::fixed()     masks recipes::fixed()
## x dplyr::id()          masks plyr::id()
## x dplyr::lag()         masks stats::lag()
## x purrr::lift()        masks caret::lift()
## x rstatix::mutate()    masks ggpubr::mutate(), dplyr::mutate(), plyr::mutate()
## x dplyr::rename()      masks plyr::rename()
## x rstatix::select()    masks MASS::select(), dplyr::select()
## x dplyr::summarise()   masks plyr::summarise()
## x dplyr::summarize()   masks plyr::summarize()
```

```r
library(ROCR)      # ROC Curves
library(pROC)      # ROC Curves
```

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

```r
library(rpart)      # decision tree application
library(rpart.plot)  # plotting decision trees
library(vip)         # for feature importance
```

```
##
## Attaching package: 'vip'
##
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(pdp)
```

```
##
## Attaching package: 'pdp'
##
## The following object is masked from 'package:purrr':
##
##     partial
```

Loading Data

Before pre=processing the data, we need to load it into a data1 variable for easier manipulation. The str() function helps identify which variables in the dataset are categorical, factors, etc.

```
data1 <- read.csv("radiomics_completedata.csv", sep = ",")
```

Preprocessing Data

First, we must check the data for null and missing values.

```
#Check for null and missing values

which(is.null(data1))
```

```
## integer(0)
```

```
which(is.na(data1))
```

```
## integer(0)
```

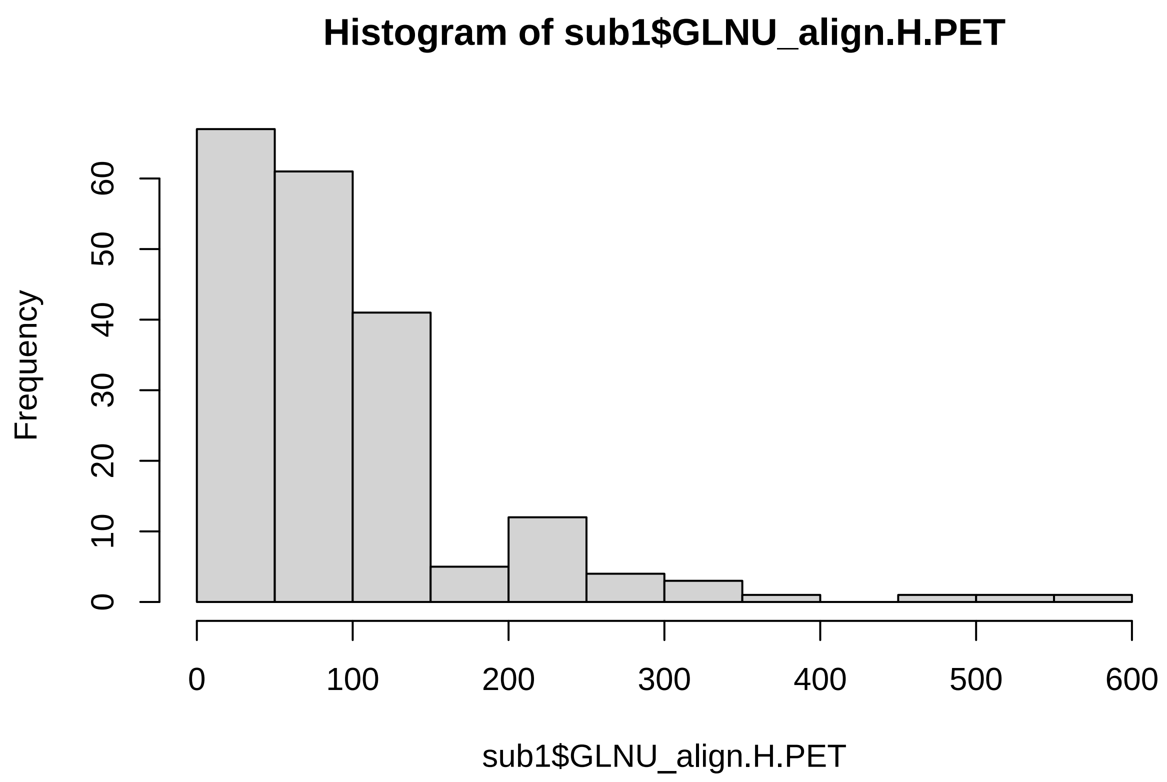
In this case, we have neither missing nor null values. We can proceed to splitting the data.

We want to split the data so that it doesn't include any categorical variables, or the Failure column.

```
#Data split

sub1 <- subset(data1, select= -c(Institution, Failure))
```

Next, we must check if the data has a normal distribution. We can do this using two methods: using a histogram and determining visually if it has a bell curve (meaning data is normalized), or using the Shapiro test in which if the p-value is < than 0.5, that means that the data is not normally distributed.

```r
#Check for normality

hist(sub1$GLNU_align.H.PET)
```



```r
sub1shapiro <- shapiro.test(sub1$GLNU_align.H.PET)
sub1shapiro
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sub1$GLNU_align.H.PET
## W = 0.76271, p-value < 2.2e-16
```

In this case, the histogram doesn't show us a bell curve, and the p-value from the Shapiro test is < 0.5, meaning data is not distributed normally.

We must perform data normalization using the scale() function. Once done, to check if data is normalized, we can use the summary() function to see if the mean is $= 0$ and use the sd() function to see if the standard deviation is $= 1$.

```r
#Normalize Data

scale_data <-  as.data.frame(scale(sub1, center = TRUE, scale = TRUE))

summary(scale_data$GLNU_align.H.PET)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.9982 -0.6721 -0.1783  0.0000  0.1947  5.3894
```

```
sd(scale_data$GLNU_align.H.PET)
```

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

Now the data has a mean of 0 and a standard deviation of 1, meaning the data is normalized.

We then check correlation of the full dataset without the categorical variables:

```
cor1 <- cor(select(scale_data, -c(Failure.binary)))
#cor1 this has been commented out or else there would be 700 pages in the pdf
```

Dataset Training and Testing Split

With preprocessing done, we can split the training and testing dataset. For this, we begin by factoring the Failure.binary column, and transforming the levels so that they represent Failure and Success. This is important for KNN and Decision Trees.

For memory purposes and faster processing, we split the dataframe to take only a "sample" of the full dataset, or just the 50 first columns. The training is split at 80%, using Failure.binary as the output.

```
scale_data$Failure.binary <- factor(data1$Failure.binary)


#data1 <- select(scale_data, -c("Institution", "Failure"))

data1 <- select(scale_data, 1:50)

levels(data1$Failure.binary) <- c("Failure", "Success")


df <- data1

str(df)
```

```
## 'data.frame':    197 obs. of  50 variables:
##  $ Failure.binary             : Factor w/ 2 levels "Failure","Success": 1 2 1 2 1 2 1 1 2 2 ...
##  $ Entropy_cooc.W.ADC         : num  0.5529 -0.0649 0.4599 1.1432 0.345 ...
##  $ GLNU_align.H.PET           : num  -0.5706 -0.789 -0.0602 2.6747 -0.0674 ...
##  $ Min_hist.PET               : num  -0.454 0.5 -1.15 -0.445 -0.989 ...
##  $ Max_hist.PET               : num  -0.436 0.149 -1.177 -0.152 -1.106 ...
##  $ Mean_hist.PET              : num  -0.42 0.315 -1.136 -0.349 -1.116 ...
##  $ Variance_hist.PET          : num  -0.263 0.395 -0.896 -0.28 -0.934 ...
##  $ Standard_Deviation_hist.PET: num  -0.236 0.297 -1.129 -0.253 -1.24 ...
##  $ Skewness_hist.PET          : num  -0.323 -0.177 -0.959 -0.116 0.958 ...
##  $ Kurtosis_hist.PET          : num  -0.273 -0.266 -0.472 0.12 0.907 ...
##  $ Energy_hist.PET            : num  0.0502 0.0919 0.0474 -0.0124 0.1533 ...
##  $ Entropy_hist.PET           : num  -0.38 -0.747 -0.37 -0.157 -0.853 ...
##  $ AUC_hist.PET               : num  -0.568 -0.563 -0.581 -0.407 -0.408 ...
##  $ H_suv.PET                  : num  -0.121 0.95 -1.072 -0.393 -1.211 ...
##  $ Volume.PET                 : num  -0.7713 -0.8698 -0.4849 0.0587 -0.4229 ...
##  $ X3D_surface.PET            : num  -0.52 -0.431 -0.155 0.244 -0.45 ...
##  $ ratio_3ds_vol.PET          : num  -0.228 0.422 -0.248 -0.701 0.409 ...
##  $ ratio_3ds_vol_norm.PET     : num  -0.37675 0.00118 -0.11356 -0.06927 -0.00444 ...
##  $ irregularity.PET           : num  -0.404 -0.259 -0.501 -0.779 -0.396 ...
##  $ tumor_length.PET           : num  -0.499 -0.625 -0.314 0.368 -0.691 ...
##  $ Compactness_v1.PET         : num  -0.072 -0.0845 -0.0816 -0.0828 -0.0844 ...
##  $ Compactness_v2.PET         : num  -0.425 -0.427 -0.426 -0.426 -0.427 ...
##  $ Spherical_disproportion.PET: num  -0.37675 0.00118 -0.11356 -0.06927 -0.00444 ...
##  $ Sphericity.PET             : num  -0.443 -0.505 -0.49 -0.496 -0.504 ...
##  $ Asphericity.PET            : num  -0.3646 0.0201 -0.0967 -0.0517 0.0143 ...
##  $ Center_of_mass.PET         : num  -0.0305 -0.3264 -0.5841 0.0433 -0.4082 ...
##  $ Max_3D_diam.PET            : num  -0.6641 -0.7524 -0.5337 -0.0528 -0.7991 ...
##  $ Major_axis_length.PET      : num  -0.7799 -0.7671 -0.4524 -0.0649 -0.7462 ...
##  $ Minor_axis_length.PET      : num  -0.81 -0.749 -0.616 0.43 -0.899 ...
##  $ Least_axis_length.PET      : num  -0.553 -0.74 -0.43 0.74 -0.728 ...
##  $ Elongation.PET             : num  -0.377 -0.3 -0.683 -0.111 -0.601 ...
##  $ Flatness.PET               : num  0.0389 -0.3472 -0.4444 0.3031 -0.3724 ...
##  $ Max_cooc.L.PET             : num  0.0191 0.1307 0.0195 0.0526 0.1083 ...
##  $ Average_cooc.L.PET         : num  -0.3868 -0.4758 0.0139 -0.8511 -1.0757 ...
##  $ Variance_cooc.L.PET        : num  -0.1075 0.0906 -0.0764 -1.0807 -0.7069 ...
##  $ Entropy_cooc.L.PET         : num  -0.498 -0.586 -0.456 -0.598 -0.688 ...
##  $ DAVE_cooc.L.PET            : num  -0.3221 0.0172 -0.2548 -1.0184 -0.5794 ...
##  $ DVAR_cooc.L.PET            : num  -0.438 0.284 -0.42 -1.081 -0.515 ...
##  $ DENT_cooc.L.PET            : num  -0.489 -0.392 -0.485 -0.774 -0.58 ...
##  $ SAVE_cooc.L.PET            : num  -0.3871 -0.4761 0.0138 -0.8516 -1.0763 ...
##  $ SVAR_cooc.L.PET            : num  -0.0267 -0.0503 0.0164 -1.0376 -0.7682 ...
##  $ SENT_cooc.L.PET            : num  -0.437 -0.452 -0.416 -0.592 -0.614 ...
##  $ ASM_cooc.L.PET             : num  0.0857 0.0965 0.0819 0.0996 0.1113 ...
##  $ Contrast_cooc.L.PET        : num  -0.221 0.302 -0.214 -1.004 -0.515 ...
##  $ Dissimilarity_cooc.L.PET   : num  -0.3221 0.0172 -0.2548 -1.0184 -0.5794 ...
##  $ Inv_diff_cooc.L.PET        : num  -0.5668 -0.6568 -0.673 0.0153 -0.3554 ...
##  $ Inv_diff_norm_cooc.L.PET   : num  -0.576 -0.626 -0.591 -0.458 -0.533 ...
##  $ IDM_cooc.L.PET             : num  -0.53 -0.577 -0.66 0.158 -0.277 ...
##  $ IDM_norm_cooc.L.PET        : num  -0.567 -0.605 -0.57 -0.506 -0.544 ...
##  $ Inv_var_cooc.L.PET         : num  -0.533 -0.581 -0.618 0.213 -0.239 ...
```

```r
# Create training (80%) and test (20%) sets for the
set.seed(123)  # for reproducibility
churn_split <- initial_split(df, prop = 0.8, strata = "Failure.binary")
churn_train <- training(churn_split)
churn_test  <- testing(churn_split)
```

Model 1: Linear Regression

The first model we will look at is Linear Regression.

First, we train 3 different models.

```r
#Model training
 set.seed(123)
 cv_model1 <- train(
   Failure.binary ~ H_suv.PET,
   data = churn_train,
   method = "glm",
   family = "binomial",
   trControl = trainControl(method = "cv", number = 5)
   )

set.seed(123)
cv_model2 <- train(
   Failure.binary ~ Entropy_cooc.W.ADC + GLNU_align.H.PET,
   data = churn_train,
   method = "glm",
   family = "binomial",
   trControl = trainControl(method = "cv", number = 5)

 )

 set.seed(123)
 cv_model3 <- train(
   Failure.binary ~ .,  #overall datasets
   data = churn_train,
   method = "glm",
   family = "binomial",
   trControl = trainControl(method = "cv", number = 5)
 )
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

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

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

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

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

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

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

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

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Then we extract the sample performance measures:

```
# extract out of sample performance measures
summary(
  resamples(
    list(
      model1 = cv_model1,
      model2 = cv_model2,
      model3 = cv_model3
    )
  )
)$statistics$Accuracy
```

```
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## model1 0.6451613 0.6562500 0.6562500 0.6625000 0.6774194 0.6774194    0
## model2 0.7741935 0.8064516 0.8437500 0.8530242 0.9032258 0.9375000    0
## model3 0.6250000 0.6875000 0.7096774 0.7141129 0.7741935 0.7741935    0
```

As seen above, model 2 has the best results. As a whole, model 1 is the weakest (Final.binary on its own).

Next, we create the prediction classes for each model and their confusion matrices.

Because we changed the levels for Faliure.binary to "Failure" and "Success" we need to change the reference parameters to that.

```
# predict class
pred_class_1 <- predict(cv_model1, churn_train)


#balanced accuracy is most important

# create confusion matrix
confusionMatrix(
  data = relevel(pred_class_1, ref = "Success"),
  reference = relevel(churn_train$Failure.binary, ref = "Success")
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Success Failure
##    Success       0       0
##    Failure      53     104
##
##                Accuracy : 0.6624
##                  95% CI : (0.5827, 0.7359)
##     No Information Rate : 0.6624
##     P-Value [Acc > NIR] : 0.5372
```

```
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 9.148e-13
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.6624
##              Prevalence : 0.3376
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : Success
##
```

```
pred_class_2 <- predict(cv_model2, churn_train)

# create confusion matrix
confusionMatrix(
  data = relevel(pred_class_2, ref = "Success"),
  reference = relevel(churn_train$Failure.binary, ref = "Success")
)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction Success Failure
##    Success      41      10
##    Failure      12      94
##
##                Accuracy : 0.8599
##                  95% CI : (0.7956, 0.9101)
##     No Information Rate : 0.6624
##     P-Value [Acc > NIR] : 1.688e-08
##
##                   Kappa : 0.6838
##
##  Mcnemar's Test P-Value : 0.8312
##
##             Sensitivity : 0.7736
##             Specificity : 0.9038
##          Pos Pred Value : 0.8039
##          Neg Pred Value : 0.8868
##              Prevalence : 0.3376
##          Detection Rate : 0.2611
##    Detection Prevalence : 0.3248
##       Balanced Accuracy : 0.8387
##
##        'Positive' Class : Success
##
```

```
pred_class_3 <- predict(cv_model3, churn_train)
```

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

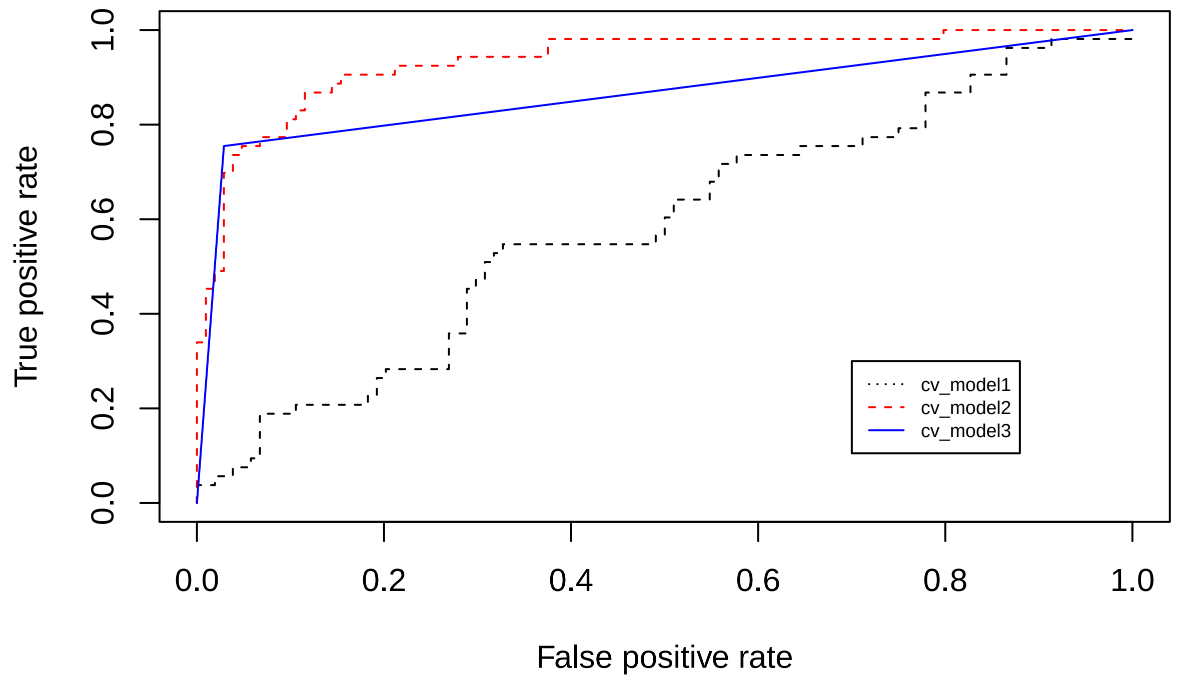
```
# create confusion matrix
confusionMatrix(
  data = relevel(pred_class_3, ref = "Success"),
  reference = relevel(churn_train$Failure.binary, ref = "Success")
)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction Success Failure
##    Success      40       3
##    Failure      13     101
##
##                Accuracy : 0.8981
##                  95% CI : (0.8398, 0.9406)
##     No Information Rate : 0.6624
##     P-Value [Acc > NIR] : 6.463e-12
##
##                   Kappa : 0.7611
##
##  Mcnemar's Test P-Value : 0.02445
##
##             Sensitivity : 0.7547
##             Specificity : 0.9712
##          Pos Pred Value : 0.9302
##          Neg Pred Value : 0.8860
##              Prevalence : 0.3376
##          Detection Rate : 0.2548
##    Detection Prevalence : 0.2739
##       Balanced Accuracy : 0.8629
##
##        'Positive' Class : Success
##
```

```
# Compute predicted probabilities on training data
m1_prob <- predict(cv_model1, churn_train, type = "prob")$Success
m2_prob <- predict(cv_model2, churn_train, type = "prob")$Success
m3_prob <- predict(cv_model3, churn_train, type = "prob")$Success
```

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

```
# Compute AUC metrics for cv_model1,2 and 3
perf1 <- prediction(m1_prob, churn_train$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
perf2 <- prediction(m2_prob, churn_train$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
perf3 <- prediction(m3_prob, churn_train$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")

# Plot ROC curves for cv_model1,2 and 3
plot(perf1, col = "black", lty = 2)
plot(perf2,  add = TRUE, col = "red", lty = 2)
plot(perf3, add = TRUE, col = "blue")
legend(0.7, 0.3, legend = c("cv_model1", "cv_model2", "cv_model3"),
       col = c("black","red", "blue"), lty = 3:1, cex = 0.6)
```



```
# ROC plot for training data
roc(churn_train$Failure.binary ~ m1_prob, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = Failure, case = Success

## Setting direction: controls < cases

##
## Call:
## roc.formula(formula = churn_train$Failure.binary ~ m1_prob, plot = TRUE,     legacy.axes = FALSE, pe
##
## Data: m1_prob in 104 controls (churn_train$Failure.binary Failure) < 53 cases (churn_train$Failure.b
## Area under the curve: 58.33%
```

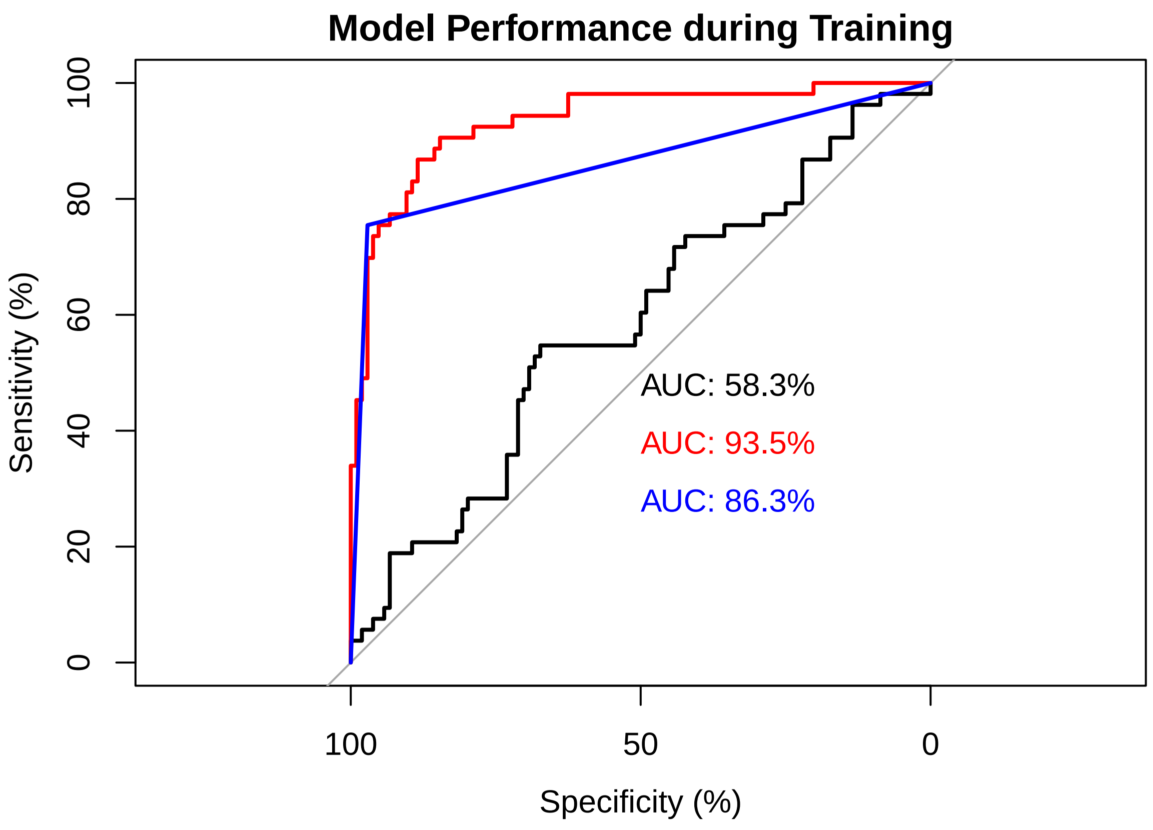
```
plot.roc(churn_train$Failure.binary ~ m2_prob,  percent=TRUE, col="red",
         lwd=2, print.auc=TRUE, add=TRUE, print.auc.y=40)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```

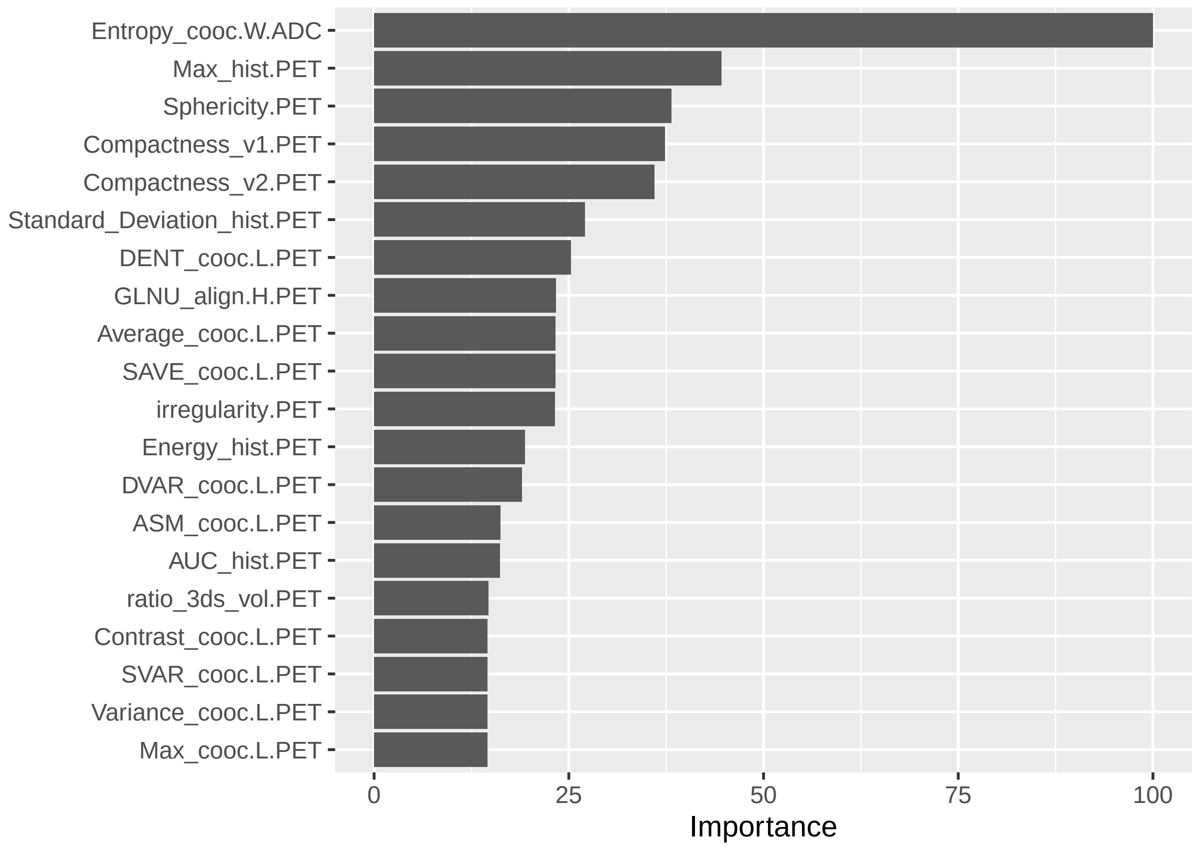
```
plot.roc(churn_train$Failure.binary ~ m3_prob,  percent=TRUE, col="blue",
         lwd=2, print.auc=TRUE, add=TRUE, print.auc.y=30)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```

```
title(main = "Model Performance during Training", line = 2.5)
```



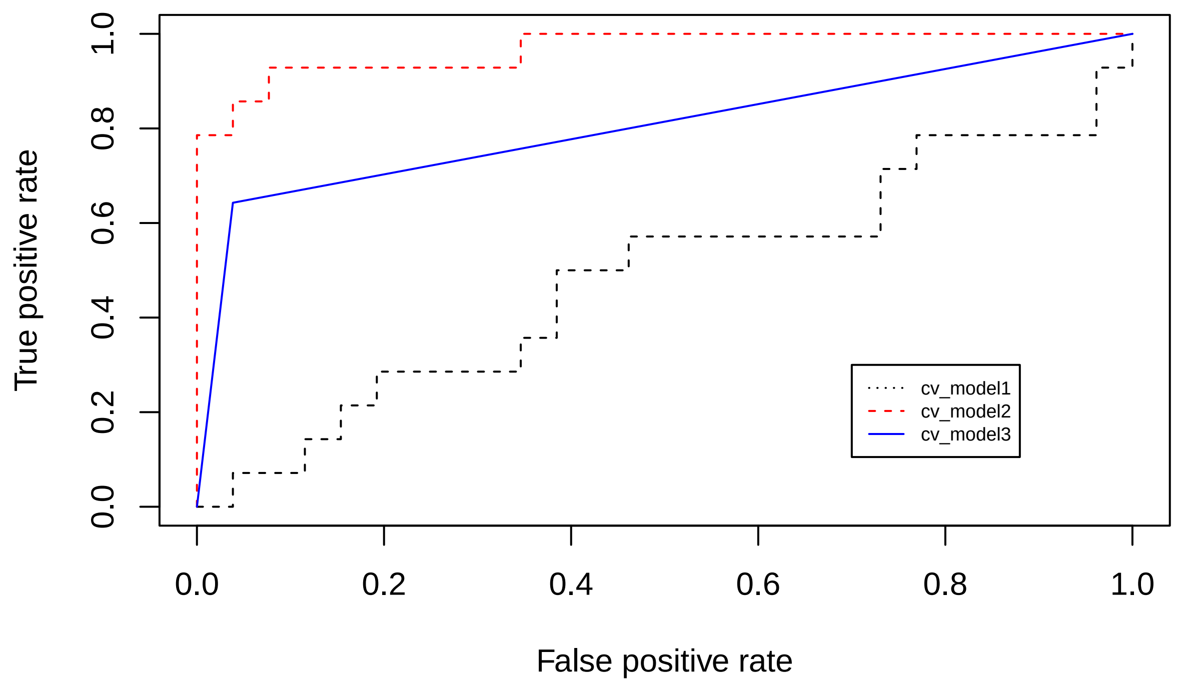
```
#Feature Interpretation
vip(cv_model3, num_features = 20)
```



```r
# Compute predicted probabilities on test data
m1_prob <- predict(cv_model1, churn_test, type = "prob")$Success
m2_prob <- predict(cv_model2, churn_test, type = "prob")$Success
m3_prob <- predict(cv_model3, churn_test, type = "prob")$Success
```

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

```r
# Compute AUC metrics for cv_model1,2 and 3
perf1 <- prediction(m1_prob, churn_test$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
perf2 <- prediction(m2_prob, churn_test$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
perf3 <- prediction(m3_prob, churn_test$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")

# Plot ROC curves for cv_model1,2 and 3
plot(perf1, col = "black", print.auc=TRUE, lty = 2)
plot(perf2,  add = TRUE, col = "red",  print.auc=TRUE, lty = 2)
plot(perf3, add = TRUE, col = "blue", print.auc=TRUE)
legend(0.7, 0.3, legend = c("cv_model1", "cv_model2", "cv_model3"),
       col = c("black","red", "blue"), lty = 3:1, cex = 0.6)
```



```r
# ROC plot for testing data
roc(churn_test$Failure.binary ~ m1_prob, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases

##
## Call:
## roc.formula(formula = churn_test$Failure.binary ~ m1_prob, plot = TRUE,     legacy.axes = FALSE, perc
##
## Data: m1_prob in 26 controls (churn_test$Failure.binary Failure) < 14 cases (churn_test$Failure.bina
## Area under the curve: 48.35%
```

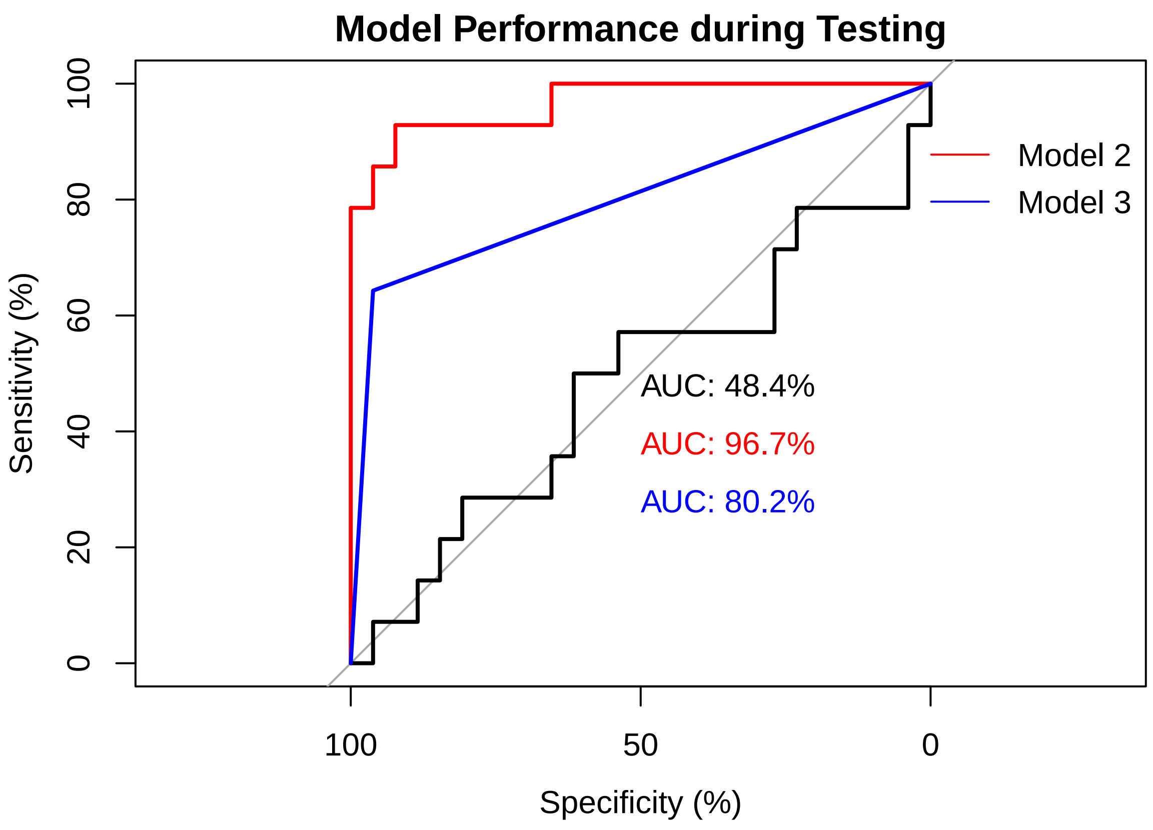
```r
plot.roc(churn_test$Failure.binary ~ m2_prob,  percent=TRUE, col="red",
         lwd=2, print.auc=TRUE, add=TRUE, print.auc.y=40)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```

```r
plot.roc(churn_test$Failure.binary ~ m3_prob,  percent=TRUE, col="blue",
         lwd=2, print.auc=TRUE, add=TRUE, print.auc.y=30)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```

```r
title(main = "Model Performance during Testing", line = 2.5)


legend("topright", c("Model 1", "Model 2", "Model 3"), lty=1,
    col = c("black", "red", "blue"), bty="n")
```



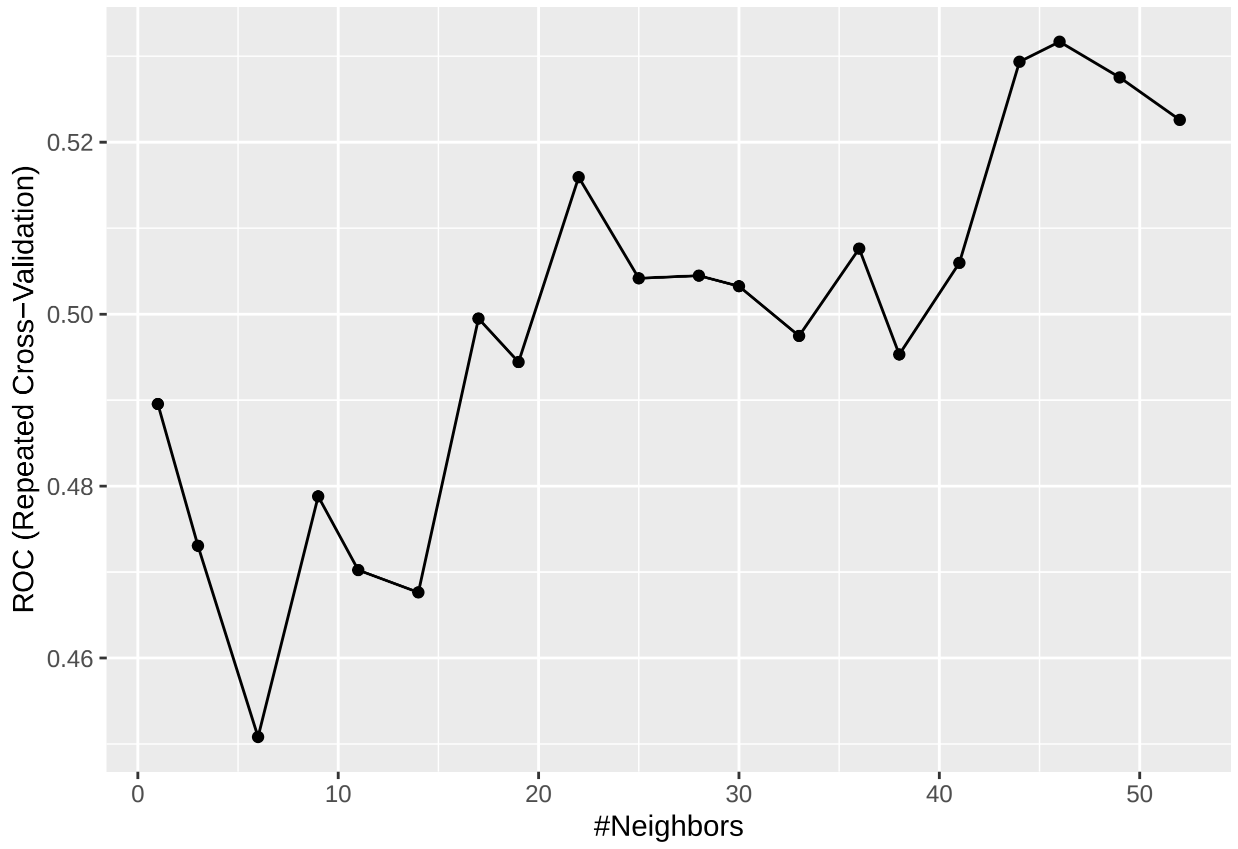
As seen in the ROC graph, Model 2 performs in an outstancing manner of distinguishing failures and successes, whereas model 3, in which Failure.binary as a whole performs on the entire dataset, still performs at an excellent level. Model 1, technically, shouldn't really be considered.

Model 2: KNN

Here we are creating a second model using KNN. The grid search on my computer takes about 5-7 minutes with a sample size of 50 variables. Again, reference points for prediction is "Success".

```r
#------Blueprint-------------------------------------------#

blueprint_attr <- recipe(Failure.binary ~ ., data = churn_train) %>%
  step_nzv(all_nominal()) %>%
  step_integer(contains("Entropy")) %>%
  step_integer(contains("GLNU")) %>%
  step_dummy(all_nominal(), -all_outcomes(), one_hot = TRUE) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes())
```

```r
#-----------Resampling Method----------------------------#

cv <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  classProbs = TRUE,
  summaryFunction = twoClassSummary)
#-------Hyperparameters and Gridsearch---------------------------#

hyper_grid <- expand.grid(
  k = floor(seq(1, nrow(churn_train)/3, length.out = 20))
)

# Fit knn model and perform grid search
knn_grid <- train(
  blueprint_attr,
  data = churn_train,
  method = "knn",
  trControl = cv,
  tuneGrid = hyper_grid,
  metric = "ROC"
)

ggplot(knn_grid)
```



```r
#----------Variable Importance----------------#

varimpo <- varImp(knn_grid)
varimpo
```

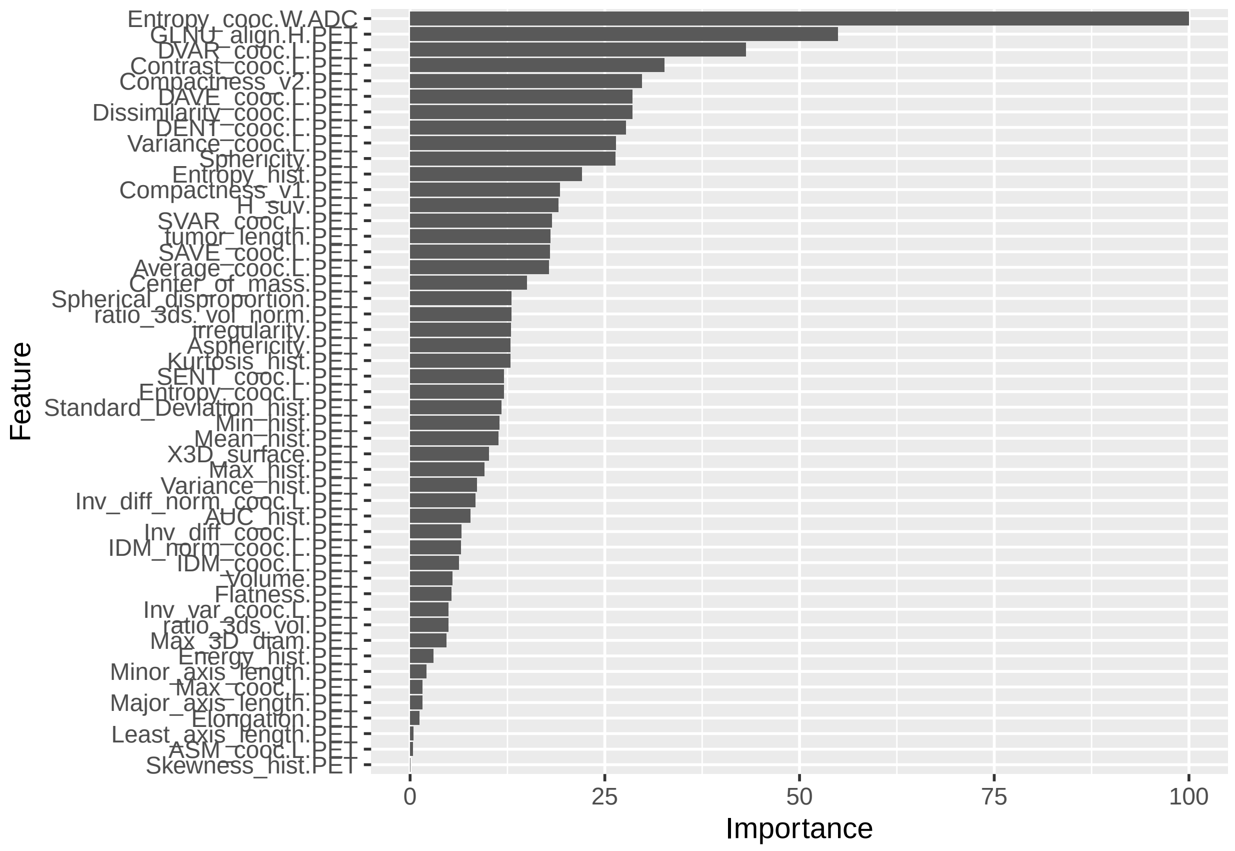
```
## ROC curve variable importance
##
##   only 20 most important variables shown (out of 49)
##
##                              Importance
## Entropy_cooc.W.ADC               100.00
## GLNU_align.H.PET                  54.91
## DVAR_cooc.L.PET                   43.14
## Contrast_cooc.L.PET               32.67
## Compactness_v2.PET                29.76
## DAVE_cooc.L.PET                   28.52
## Dissimilarity_cooc.L.PET          28.52
## DENT_cooc.L.PET                   27.69
## Variance_cooc.L.PET               26.43
## Sphericity.PET                    26.35
## Entropy_hist.PET                  22.03
## Compactness_v1.PET                19.20
## H_suv.PET                         19.02
## SVAR_cooc.L.PET                   18.18
## tumor_length.PET                  18.01
## SAVE_cooc.L.PET                   17.93
## Average_cooc.L.PET                17.84
## Center_of_mass.PET                14.99
## Spherical_disproportion.PET       12.98
## ratio_3ds_vol_norm.PET            12.98
```

```r
pred_knngrid <- predict(knn_grid, churn_train)

confusionMatrix(
  data = relevel(pred_knngrid, ref = "Success"),
  reference = relevel(churn_train$Failure.binary, ref = "Success")
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Success Failure
##    Success      18       4
##    Failure      35     100
##
##                Accuracy : 0.7516
##                  95% CI : (0.6764, 0.817)
##     No Information Rate : 0.6624
##     P-Value [Acc > NIR] : 0.01003
##
##                   Kappa : 0.3516
##
##  Mcnemar's Test P-Value : 1.556e-06
```

```
##
##             Sensitivity : 0.3396
##             Specificity : 0.9615
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.7407
##              Prevalence : 0.3376
##          Detection Rate : 0.1146
##    Detection Prevalence : 0.1401
##       Balanced Accuracy : 0.6506
##
##        'Positive' Class : Success
##
```

```
ggplot(varimpo)
```



```
par(mfrow = c(1,2))



# § Plot the training data performance while print the AUC values.


knngrid_prob <- predict(knn_grid, churn_train, type = "prob")$Success
roc(churn_train$Failure.binary ~ knngrid_prob, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = Failure, case = Success

## Setting direction: controls < cases

##
## Call:
## roc.formula(formula = churn_train$Failure.binary ~ knngrid_prob,     plot = TRUE, legacy.axes = FALSI
##
## Data: knngrid_prob in 104 controls (churn_train$Failure.binary Failure) < 53 cases (churn_train$Failu
## Area under the curve: 79.23%
```

```
title(main = "Model Performance during Training", line = 2.5)


# § Use the PREDICT function to predict using the testing data.

knntest <- predict(knn_grid, churn_test)
confusionMatrix(
  data = relevel(knntest, ref = "Success"),
  reference = relevel(churn_test$Failure.binary, ref = "Success")
)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction Success Failure
##    Success       0       0
##    Failure      14      26
##
##                Accuracy : 0.65
##                  95% CI : (0.4832, 0.7937)
##     No Information Rate : 0.65
##     P-Value [Acc > NIR] : 0.572082
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.000512
##
##             Sensitivity : 0.00
##             Specificity : 1.00
##          Pos Pred Value :  NaN
##          Neg Pred Value : 0.65
##              Prevalence : 0.35
##          Detection Rate : 0.00
##    Detection Prevalence : 0.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : Success
##
```

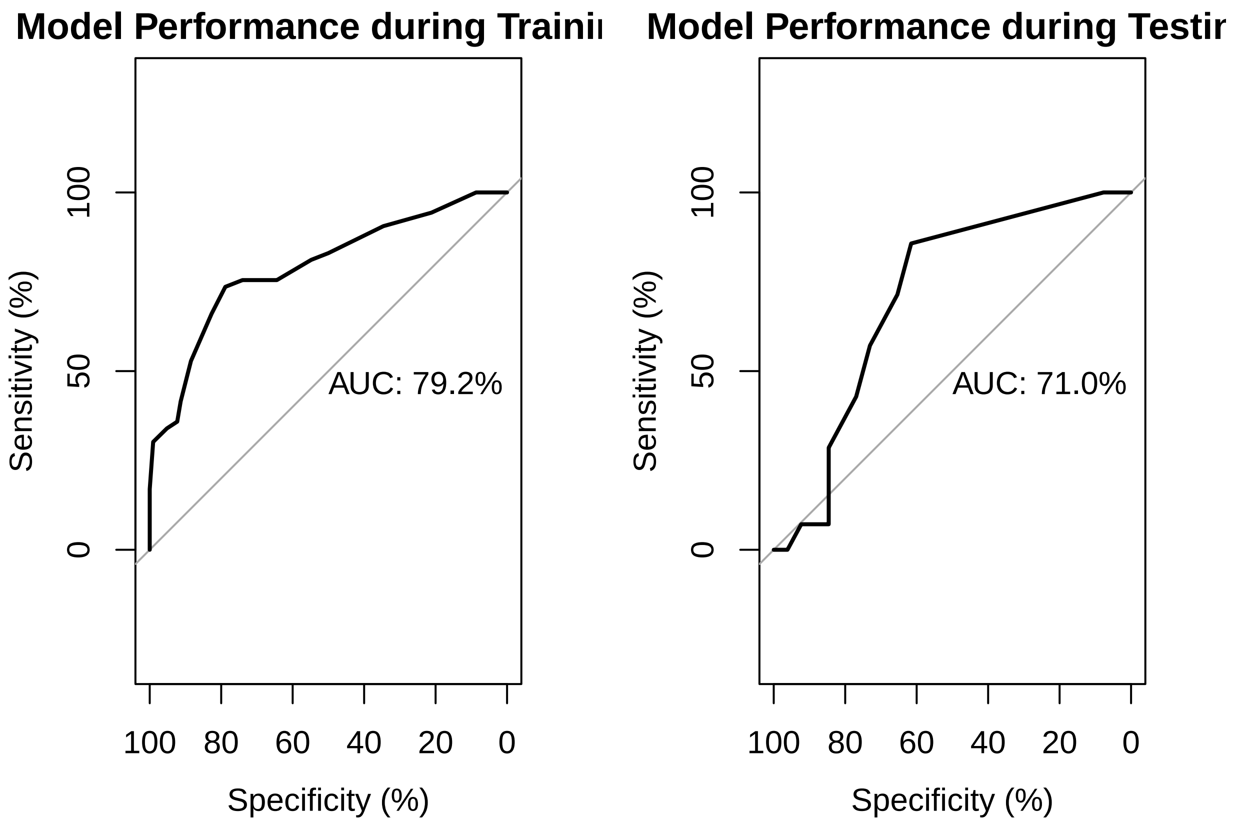
```
# § Plot the testing data performance while print the AUC values.

knngrid_probtest <- predict(knn_grid, churn_test, type = "prob")$Success
```

```
roc(churn_test$Failure.binary ~ knngrid_probtest, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```

```
##
## Call:
## roc.formula(formula = churn_test$Failure.binary ~ knngrid_probtest,     plot = TRUE, legacy.axes = FA
##
## Data: knngrid_probtest in 26 controls (churn_test$Failure.binary Failure) < 14 cases (churn_test$Fail
## Area under the curve: 71.02%
```

```
title(main = "Model Performance during Testing", line = 2.5)
```

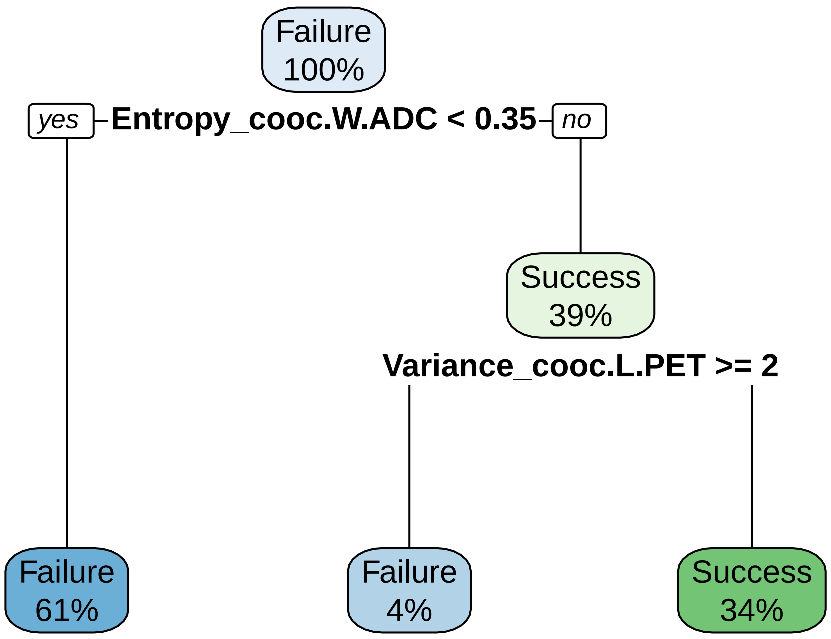


The training data performance is just a little better than testing, but the AUC values for both aren't as good as the ones from LR.

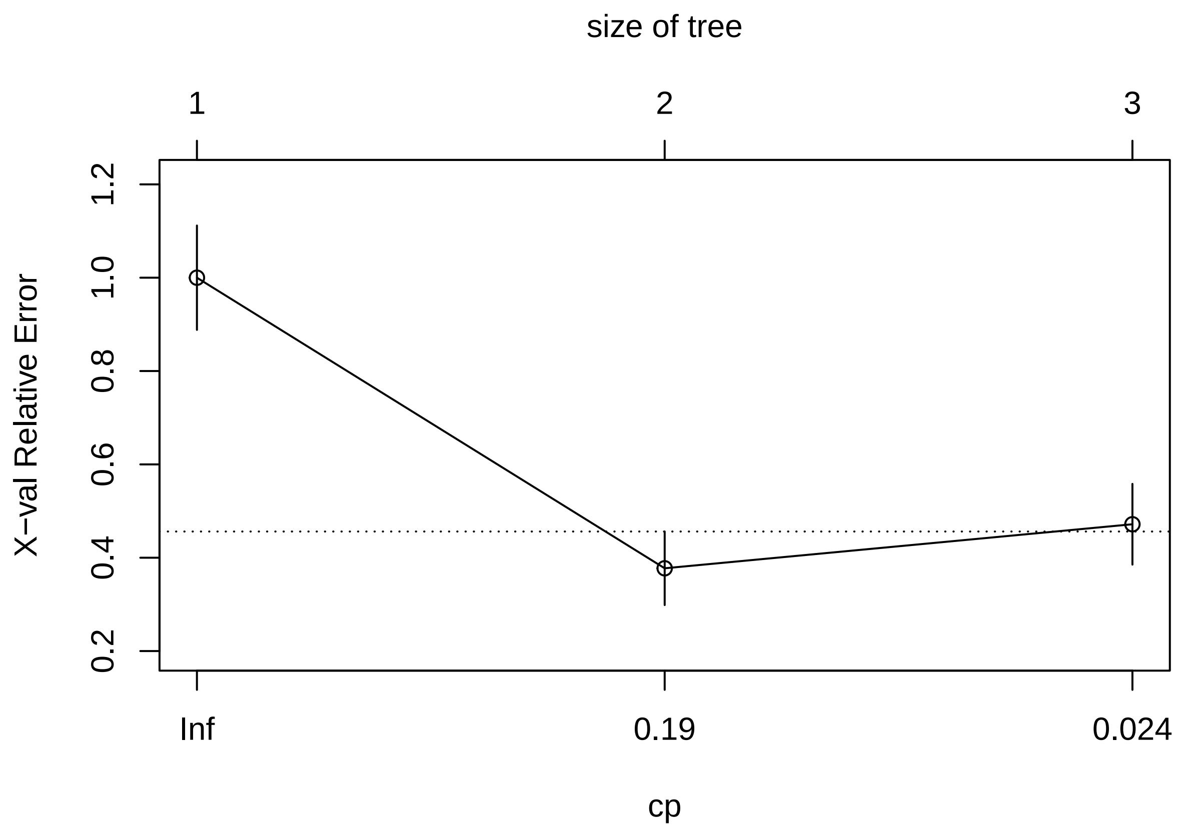
Model 3: Decision Tree

```
##modeling
fit <- rpart(Failure.binary~., data = churn_train, method = 'class')
```

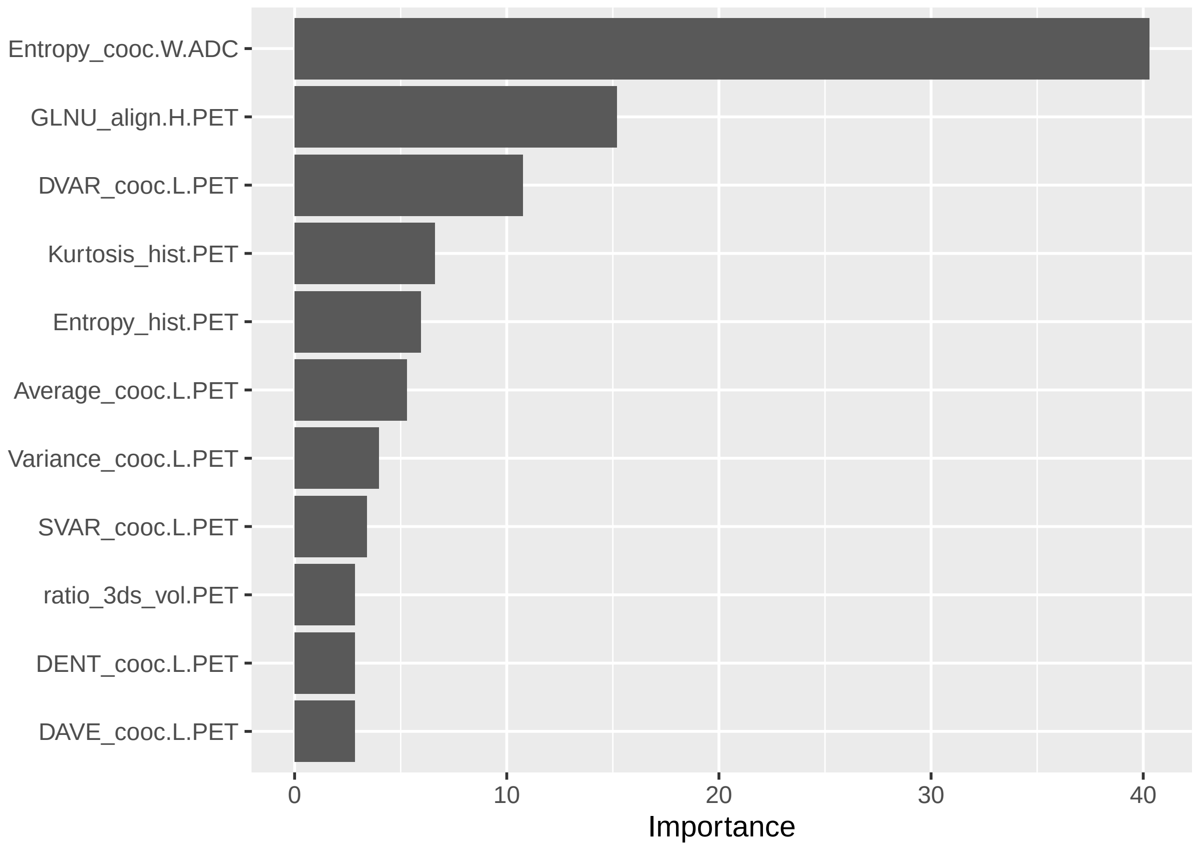
```
#plotting
rpart.plot(fit, extra = 100)
```



```
#plotting
plotcp(fit)
```



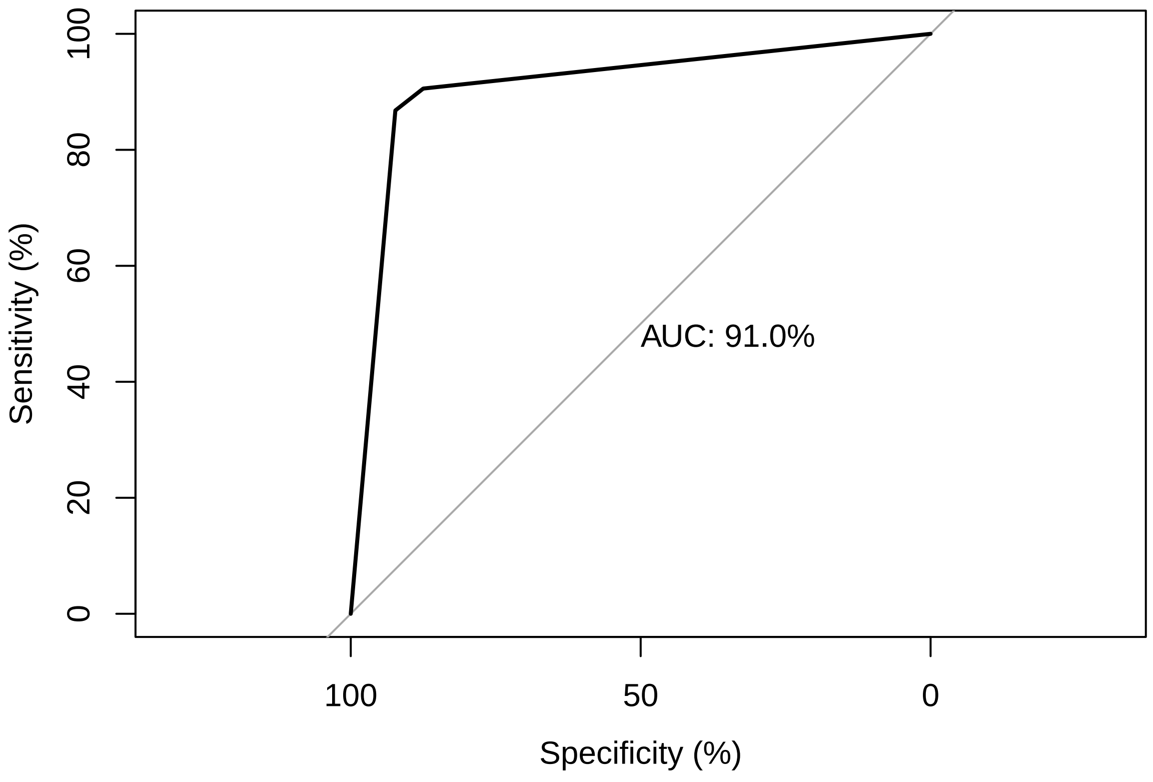
```
#feature importance
vip(fit, num_features = 20, bar = FALSE)
```



```
# Compute predicted probabilities on training data
dt1_prob <- predict(fit, churn_train, type = "prob")

# ROC plot for training data
roc(churn_train$Failure.binary ~ dt1_prob[,2], plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

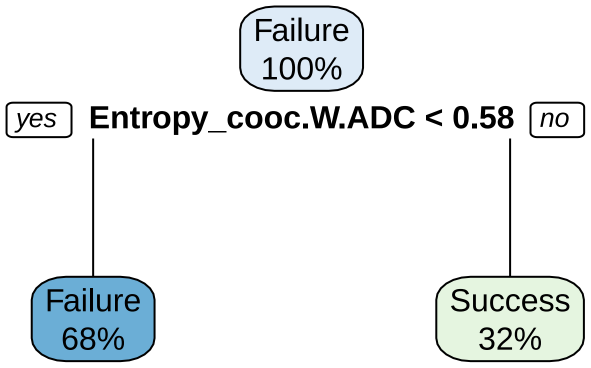
```
## Setting levels: control = Failure, case = Success
```

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

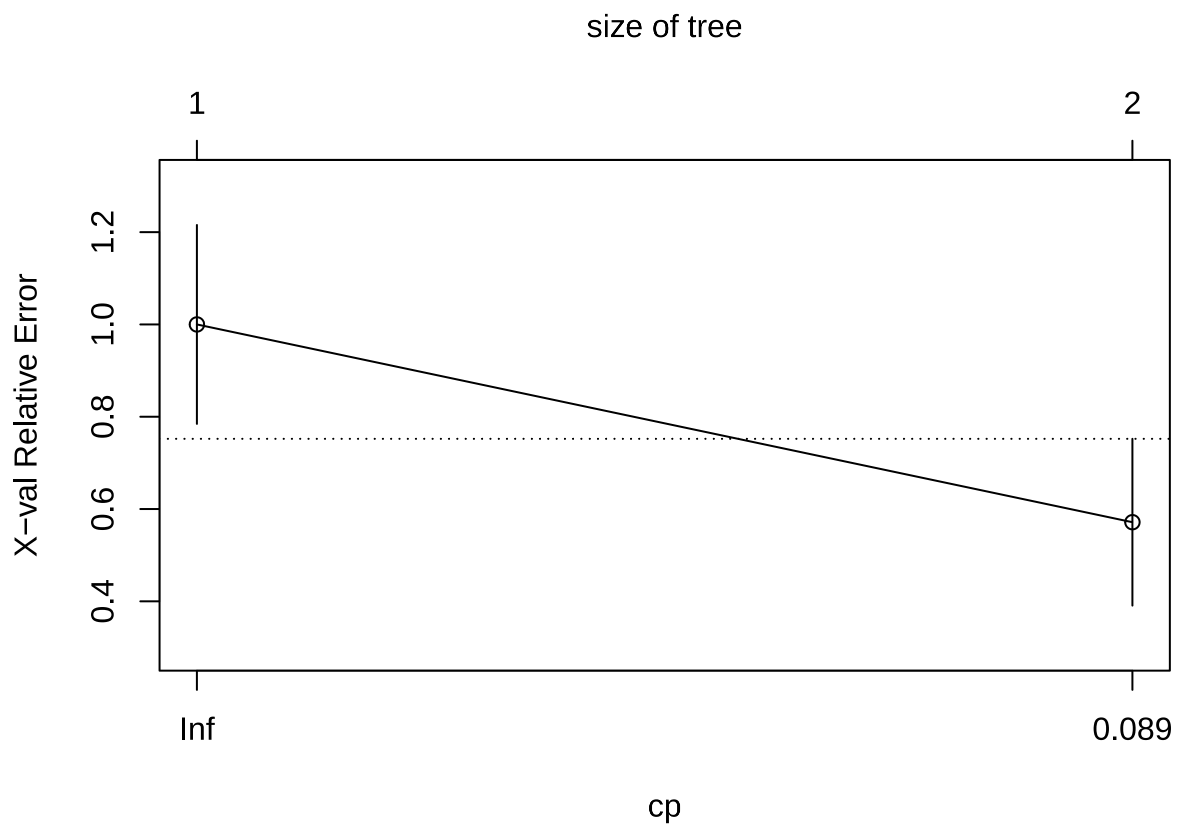


```
##
## Call:
## roc.formula(formula = churn_train$Failure.binary ~ dt1_prob[,     2], plot = TRUE, legacy.axes = FALS
##
## Data: dt1_prob[, 2] in 104 controls (churn_train$Failure.binary Failure) < 53 cases (churn_train$Fail
## Area under the curve: 90.97%
```

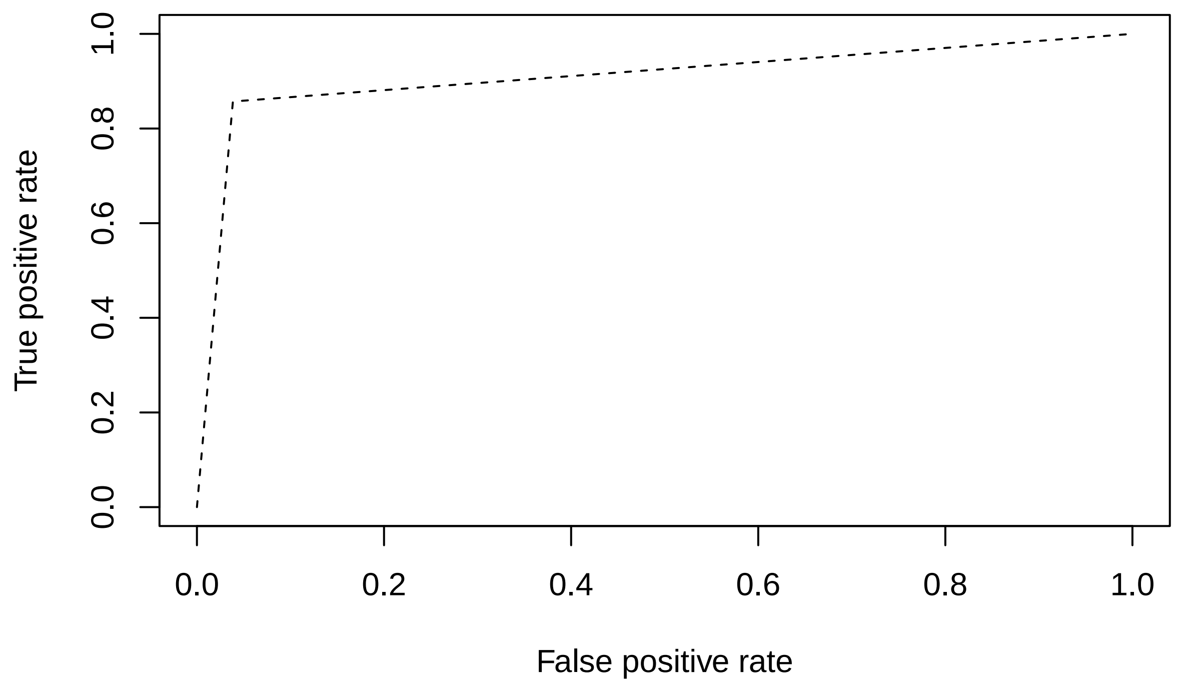
```
test_fit <- rpart(Failure.binary~., data = churn_test, method = 'class')
failure_predict <- predict(test_fit,churn_test)


#    Use the RPART.PLOT and PLOTCP function to identify the trees
rpart.plot(test_fit, extra =  100)
```



```
plotcp(test_fit)
```

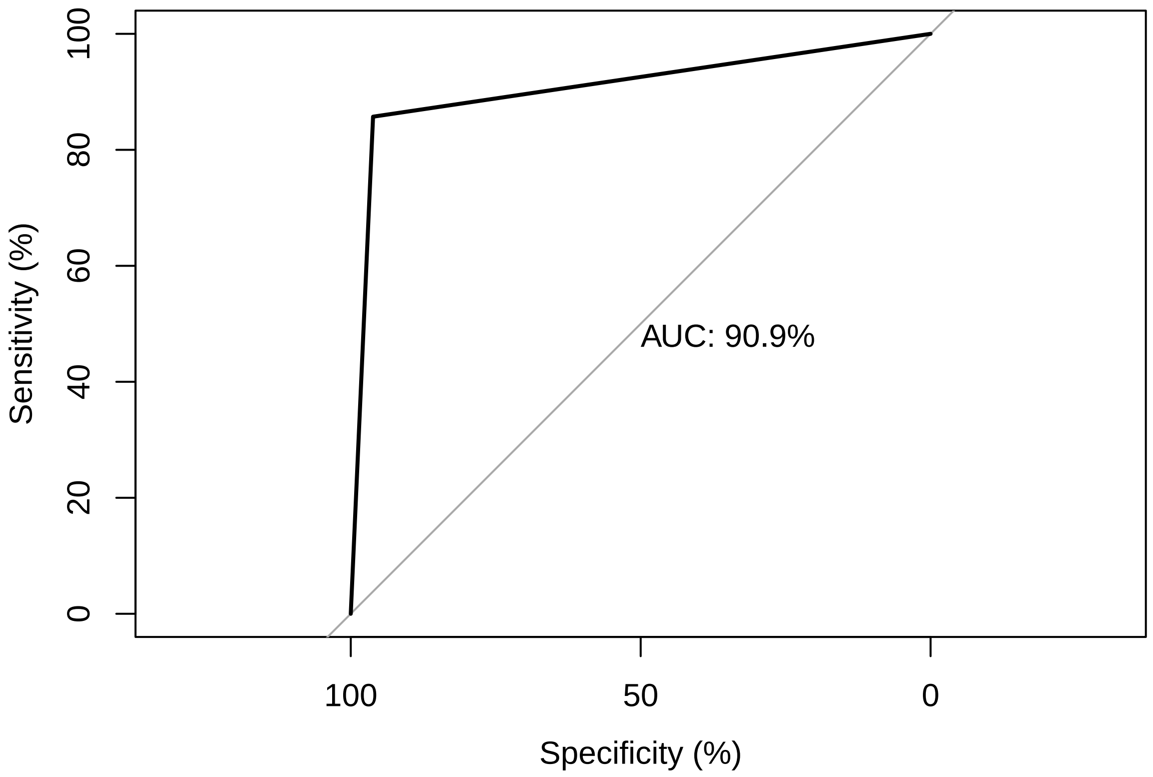


```
#    Plot the testing data performance while print the AUC values


dt2_prob <- predict(test_fit, churn_test, type = "prob")

perf1 <- prediction(dt2_prob[,2], churn_test$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
plot(perf1, col = "black", lty = 2)
```



```
roc(churn_test$Failure.binary ~ dt2_prob[,2], plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = Failure, case = Success
## Setting direction: controls < cases
```



```
##
## Call:
## roc.formula(formula = churn_test$Failure.binary ~ dt2_prob[,     2], plot = TRUE, legacy.axes = FALS
##
## Data: dt2_prob[, 2] in 26 controls (churn_test$Failure.binary Failure) < 14 cases (churn_test$Failur
## Area under the curve: 90.93%
```

The Decision Tree ROC curves and AUC values are excellent considering both training and testing are over 90%.

Conclusion

Based on the above AUC results, KNN has performed the least well compared to both Linear Regression and Decision Tree. If we consider the ROC models that had Failure.binary be predicted against the full (sampled) dataset, Decision Tree had the highest accuracy.