R Notebook

# Load packages and libraries

# install.packages("moments")  
# install.packages("ggplot2")  
# install.packages("corrplot")  
# install.packages("tidyr")  
# install.packages("dplyr")  
# install.packages("ggridges")  
# install.packages("mlbench")  
# install.packages("lattice")  
# install.packages("caret")  
# install.packages("broom")  
# install.packages("C50")  
# install.packages("rpart")  
# install.packages("pROC")  
# install.packages("DMwR")  
# install.packages("ROSE")  
# install.packages('rattle')  
# install.packages('rpart.plot')  
# install.packages('RColorBrewer')  
# install.packages("glmboost")  
# install.packages("ROCR")  
library(moments)  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.4.4

library("corrplot")

## Warning: package 'corrplot' was built under R version 3.4.4

## corrplot 0.84 loaded

library(tidyr)

## Warning: package 'tidyr' was built under R version 3.4.4

library(dplyr)

## Warning: package 'dplyr' was built under R version 3.4.4

##   
## 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(ggridges)

## Warning: package 'ggridges' was built under R version 3.4.4

library(mlbench)

## Warning: package 'mlbench' was built under R version 3.4.4

library(caret)

## Warning: package 'caret' was built under R version 3.4.4

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.4.4

library("C50")

## Warning: package 'C50' was built under R version 3.4.4

library(rpart)  
library(pROC)

## Warning: package 'pROC' was built under R version 3.4.4

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

##   
## Attaching package: 'pROC'

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

library(DMwR)

## Warning: package 'DMwR' was built under R version 3.4.4

## Loading required package: grid

library(ROSE)

## Warning: package 'ROSE' was built under R version 3.4.4

## Loaded ROSE 0.0-3

library(rattle)

## Warning: package 'rattle' was built under R version 3.4.4

## Rattle: A free graphical interface for data science with R.  
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(rpart.plot)  
library(RColorBrewer)  
library(ROCR)

## Warning: package 'ROCR' was built under R version 3.4.4

## Loading required package: gplots

## Warning: package 'gplots' was built under R version 3.4.4

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

# Load the dataset

ccfraud <- read.csv("C:/Users/Kiran Kandhola/Documents/creditcardfraud/creditcard.csv",stringsAsFactors = FALSE)

# Reorder the levels of the “Class” attribute

cc <- ccfraud # Make a copy of the data  
cc$Class <- as.factor(cc$Class) # Convert the class to factors  
cc$Class <- factor(cc$Class, levels = c("1", "0")) # Change the order of levels   
ccfd <- cc   
levels(ccfd$Class) <- c("Fraud", "Genuine") # Change the name of levels to Fraud and Genuine

## Stratified 80%- 20% splitting of the dataset into training and testing (time excluded)

set.seed(7)  
library(caret)  
train\_index <- createDataPartition(ccfd$Class,times = 1, p=0.8, list=FALSE)  
ccfd <- ccfd[ ,-c(1)]  
ccfd\_train <- ccfd[train\_index, ]  
ccfd\_test <- ccfd[-train\_index, ]  
  
zscorenormalize <- function(x){   
 return((x-mean(x))/(sd(x)))}  
  
Amount\_trn <- as.data.frame(lapply(ccfd\_train[29], zscorenormalize))  
ccfd\_train <- as.data.frame(c(ccfd\_train[1:28],Amount\_trn,ccfd\_train[30]))  
Amount\_tst <- as.data.frame(lapply(ccfd\_test[29], zscorenormalize))  
ccfd\_test <- as.data.frame(c(ccfd\_test[1:28],Amount\_tst,ccfd\_test[30]))  
  
X\_trn <- ccfd\_train[ ,-30]  
X\_tst <- ccfd\_test[ ,-30]  
  
Y\_trn <- ccfd\_train[ ,30]  
Y\_tst <- ccfd\_test[ ,30]  
  
## stratified folds for cross-validation:  
foldInd <- createFolds(Y\_trn, k = 10, list = TRUE, returnTrain = TRUE)  
lapply(foldInd, function(ii) table(Y\_trn[ii]))

## $Fold01  
##   
## Fraud Genuine   
## 354 204706   
##   
## $Fold02  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold03  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold04  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold05  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold06  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold07  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold08  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold09  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold10  
##   
## Fraud Genuine   
## 355 204706

set.seed(7)  
library(caret)  
train\_index <- createDataPartition(cc$Class,times = 1, p=0.8, list=FALSE)  
  
cc <- cc[ ,-c(1)]  
cc\_train <- cc[train\_index, ]  
cc\_test <- cc[-train\_index, ]  
  
zscorenormalize <- function(x){   
 return((x-mean(x))/(sd(x)))}  
Amount\_train <- as.data.frame(lapply(cc\_train[29], zscorenormalize))  
cc\_train <- as.data.frame(c(cc\_train[1:28],Amount\_train,cc\_train[30]))  
Amount\_test <- as.data.frame(lapply(ccfd\_test[29], zscorenormalize))  
cc\_test <- as.data.frame(c(ccfd\_test[1:28],Amount\_test,cc\_test[30]))  
  
X\_train <- cc\_train[ ,-30]  
X\_test <- cc\_test[ ,-30]  
  
Y\_train <- cc\_train[ ,30]  
Y\_test <- cc\_test[ ,30]  
  
## stratified folds for cross-validation:  
foldInds <- createFolds(Y\_train, k = 10, list = TRUE, returnTrain = TRUE)  
lapply(foldInds, function(ii) table(Y\_train[ii]))

## $Fold01  
##   
## 1 0   
## 355 204706   
##   
## $Fold02  
##   
## 1 0   
## 354 204707   
##   
## $Fold03  
##   
## 1 0   
## 355 204707   
##   
## $Fold04  
##   
## 1 0   
## 354 204707   
##   
## $Fold05  
##   
## 1 0   
## 354 204706   
##   
## $Fold06  
##   
## 1 0   
## 354 204707   
##   
## $Fold07  
##   
## 1 0   
## 355 204707   
##   
## $Fold08  
##   
## 1 0   
## 355 204707   
##   
## $Fold09  
##   
## 1 0   
## 355 204707   
##   
## $Fold10  
##   
## 1 0   
## 355 204707

# Training scheme for unbalanced dataset

ctr\_ub <- trainControl(index = foldInd,  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 verboseIter = TRUE,  
 classProbs = TRUE,  
 summaryFunction = twoClassSummary,  
 savePredictions = TRUE)

# Training scheme for dataset using “smote” resampling

ctr\_s <- trainControl(index = foldInd,  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 verboseIter = TRUE,  
 classProbs = TRUE,  
 sampling = "smote",  
 summaryFunction = twoClassSummary,  
 savePredictions = TRUE)

# Training scheme for dataset using “up” resampling

ctr\_up <- trainControl(index = foldInd,  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 verboseIter = TRUE,  
 classProbs = TRUE,  
 sampling = "up",  
 summaryFunction = twoClassSummary,  
 savePredictions = TRUE)

# Training scheme for dataset using “down” resampling

ctr\_down <- trainControl(index = foldInd,  
 method = "repeatedcv",  
 number = 10,  
 repeats = 3,  
 verboseIter = TRUE,  
 classProbs = TRUE,  
 sampling = "down",  
 summaryFunction = twoClassSummary,  
 savePredictions = TRUE)

# Baseline model: Decision Tree

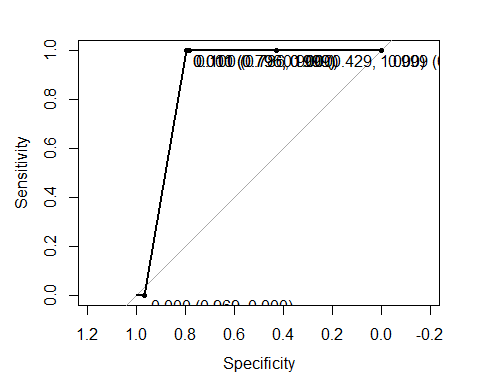
set.seed(7)  
fit\_rpart <- rpart(Class ~ ., data = cc\_train)  
fit\_rpartPred <- predict(fit\_rpart, cc\_test, type = "class")  
confusionMatrix(fit\_rpartPred,cc\_test$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 0  
## 1 77 12  
## 0 21 56851  
##   
## Accuracy : 0.9994   
## 95% CI : (0.9992, 0.9996)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 2.343e-14   
##   
## Kappa : 0.8232   
## Mcnemar's Test P-Value : 0.1637   
##   
## Sensitivity : 0.785714   
## Specificity : 0.999789   
## Pos Pred Value : 0.865169   
## Neg Pred Value : 0.999631   
## Prevalence : 0.001720   
## Detection Rate : 0.001352   
## Detection Prevalence : 0.001562   
## Balanced Accuracy : 0.892752   
##   
## 'Positive' Class : 1   
##

# Predicting Class probabilities  
prob\_fit\_rpart <- predict(fit\_rpart, cc\_test, type = "prob")  
head(prob\_fit\_rpart)

## 1 0  
## 1 0.0003035983 0.9996964  
## 2 0.0003035983 0.9996964  
## 3 0.0003035983 0.9996964  
## 4 0.0003035983 0.9996964  
## 5 0.0003035983 0.9996964  
## 6 0.0003035983 0.9996964

#library(pROC)  
ROC\_fit\_rpart <- roc(Y\_tst, prob\_fit\_rpart[ ,"1"])  
plot(ROC\_fit\_rpart, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.90, 0.999))



ROC\_fit\_rpart

##   
## Call:  
## roc.default(response = Y\_tst, predictor = prob\_fit\_rpart[, "1"])  
##   
## Data: prob\_fit\_rpart[, "1"] in 98 controls (Y\_tst Fraud) > 56863 cases (Y\_tst Genuine).  
## Area under the curve: 0.8825

# rpart on unbalanced full dataset

set.seed(7)  
rpart\_ub <- train(Class ~., data = ccfd\_train, method = "rpart", parms = list(split = "information"), metric = "ROC", trControl=ctr\_ub, tuneLength = 30)

## + Fold01: cp=0   
## - Fold01: cp=0   
## + Fold02: cp=0   
## - Fold02: cp=0   
## + Fold03: cp=0   
## - Fold03: cp=0   
## + Fold04: cp=0   
## - Fold04: cp=0   
## + Fold05: cp=0   
## - Fold05: cp=0   
## + Fold06: cp=0   
## - Fold06: cp=0   
## + Fold07: cp=0   
## - Fold07: cp=0   
## + Fold08: cp=0   
## - Fold08: cp=0   
## + Fold09: cp=0   
## - Fold09: cp=0   
## + Fold10: cp=0   
## - Fold10: cp=0   
## Aggregating results  
## Selecting tuning parameters  
## Fitting cp = 0 on full training set

rpart\_ub

## CART   
##   
## 227846 samples  
## 29 predictor  
## 2 classes: 'Fraud', 'Genuine'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 205060, 205062, 205062, 205062, 205061, 205062, ...   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.00000000 0.9424679 0.7769231 0.9997890  
## 0.01654122 0.9098180 0.7845513 0.9998197  
## 0.03308244 0.9097583 0.8075000 0.9996659  
## 0.04962367 0.8439452 0.6907692 0.9997142  
## 0.06616489 0.8439368 0.6907692 0.9997010  
## 0.08270611 0.8439279 0.6907692 0.9996878  
## 0.09924733 0.8464266 0.6933333 0.9995867  
## 0.11578855 0.8489745 0.6983974 0.9995516  
## 0.13232977 0.8489745 0.6983974 0.9995516  
## 0.14887100 0.8489745 0.6983974 0.9995516  
## 0.16541222 0.8489745 0.6983974 0.9995516  
## 0.18195344 0.8489745 0.6983974 0.9995516  
## 0.19849466 0.8489745 0.6983974 0.9995516  
## 0.21503588 0.8489745 0.6983974 0.9995516  
## 0.23157710 0.8489745 0.6983974 0.9995516  
## 0.24811833 0.8489745 0.6983974 0.9995516  
## 0.26465955 0.8489745 0.6983974 0.9995516  
## 0.28120077 0.8489745 0.6983974 0.9995516  
## 0.29774199 0.8489745 0.6983974 0.9995516  
## 0.31428321 0.8489745 0.6983974 0.9995516  
## 0.33082444 0.8489745 0.6983974 0.9995516  
## 0.34736566 0.8489745 0.6983974 0.9995516  
## 0.36390688 0.8489745 0.6983974 0.9995516  
## 0.38044810 0.8489745 0.6983974 0.9995516  
## 0.39698932 0.8489745 0.6983974 0.9995516  
## 0.41353054 0.8489745 0.6983974 0.9995516  
## 0.43007177 0.8054045 0.6112179 0.9995911  
## 0.44661299 0.7358221 0.4719872 0.9996571  
## 0.46315421 0.6679218 0.3360897 0.9997538  
## 0.47969543 0.5324736 0.0650000 0.9999472  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.

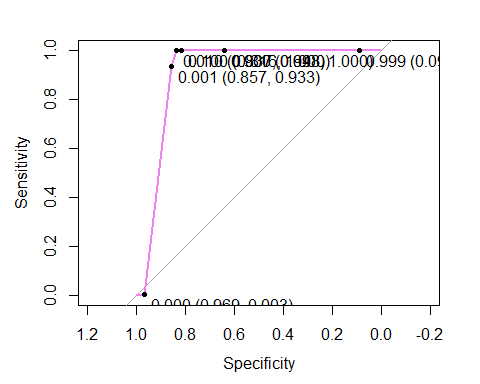
# test set results using confusion matrix  
pred\_rpart\_ub <- predict(rpart\_ub, ccfd\_test)  
confusionMatrix(pred\_rpart\_ub, Y\_tst)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Fraud Genuine  
## Fraud 79 17  
## Genuine 19 56846  
##   
## Accuracy : 0.9994   
## 95% CI : (0.9991, 0.9996)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 5.406e-13   
##   
## Kappa : 0.8141   
## Mcnemar's Test P-Value : 0.8676   
##   
## Sensitivity : 0.806122   
## Specificity : 0.999701   
## Pos Pred Value : 0.822917   
## Neg Pred Value : 0.999666   
## Prevalence : 0.001720   
## Detection Rate : 0.001387   
## Detection Prevalence : 0.001685   
## Balanced Accuracy : 0.902912   
##   
## 'Positive' Class : Fraud   
##

# Predicting Class probabilities  
prob\_rpart\_ub <- predict(rpart\_ub, ccfd\_test, type = "prob")  
head(prob\_rpart\_ub)

## Fraud Genuine  
## 1 0.0001510924 0.9998489  
## 2 0.0001510924 0.9998489  
## 3 0.0012295082 0.9987705  
## 4 0.0001510924 0.9998489  
## 5 0.0001510924 0.9998489  
## 6 0.0001510924 0.9998489

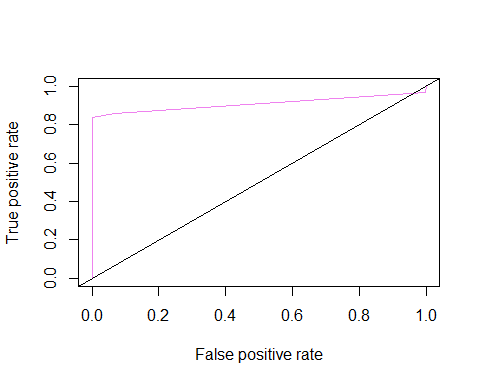
# AUC using pROC package  
ROC\_rpart\_ub <- roc(Y\_tst, prob\_rpart\_ub [ ,"Fraud"])  
plot(ROC\_rpart\_ub, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.90, 0.999), col = "violet")



ROC\_rpart\_ub

##   
## Call:  
## roc.default(response = Y\_tst, predictor = prob\_rpart\_ub[, "Fraud"])  
##   
## Data: prob\_rpart\_ub[, "Fraud"] in 98 controls (Y\_tst Fraud) > 56863 cases (Y\_tst Genuine).  
## Area under the curve: 0.909

# AUC using ROCR package  
pr\_rpart\_ub <- prediction(prob\_rpart\_ub[ , 1], Y\_test)  
prf\_rpart\_ub <- performance(pr\_rpart\_ub, measure = "tpr", x.measure = "fpr")  
plot(prf\_rpart\_ub, col = "violet")  
abline(0,1)



auc\_rub <- performance(pr\_rpart\_ub, measure = "auc")  
auc\_rub <- auc\_rub@y.values[[1]]  
auc\_rub

## [1] 0.9089837

# rpart using “smote” resampling

set.seed(10)  
rpart\_s <- train(Class ~., data = ccfd\_train, method = "rpart", parms = list(split = "information"), metric = "ROC", trControl=ctr\_s, tuneLength = 30)

## + Fold01: cp=0   
## - Fold01: cp=0   
## + Fold02: cp=0   
## - Fold02: cp=0   
## + Fold03: cp=0   
## - Fold03: cp=0   
## + Fold04: cp=0   
## - Fold04: cp=0   
## + Fold05: cp=0   
## - Fold05: cp=0   
## + Fold06: cp=0   
## - Fold06: cp=0   
## + Fold07: cp=0   
## - Fold07: cp=0   
## + Fold08: cp=0   
## - Fold08: cp=0   
## + Fold09: cp=0   
## - Fold09: cp=0   
## + Fold10: cp=0   
## - Fold10: cp=0   
## Aggregating results  
## Selecting tuning parameters  
## Fitting cp = 0 on full training set

rpart\_s

## CART   
##   
## 227846 samples  
## 29 predictor  
## 2 classes: 'Fraud', 'Genuine'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 205060, 205062, 205062, 205062, 205061, 205062, ...   
## Addtional sampling using SMOTE  
##   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.00000000 0.9544226 0.8759615 0.9525130  
## 0.01654122 0.9110890 0.8352564 0.9840581  
## 0.03308244 0.9100925 0.8352564 0.9849286  
## 0.04962367 0.9100925 0.8352564 0.9849286  
## 0.06616489 0.9100925 0.8352564 0.9849286  
## 0.08270611 0.9100925 0.8352564 0.9849286  
## 0.09924733 0.9100925 0.8352564 0.9849286  
## 0.11578855 0.9100925 0.8352564 0.9849286  
## 0.13232977 0.9100925 0.8352564 0.9849286  
## 0.14887100 0.9100925 0.8352564 0.9849286  
## 0.16541222 0.9100925 0.8352564 0.9849286  
## 0.18195344 0.9100925 0.8352564 0.9849286  
## 0.19849466 0.9100925 0.8352564 0.9849286  
## 0.21503588 0.9100925 0.8352564 0.9849286  
## 0.23157710 0.9100925 0.8352564 0.9849286  
## 0.24811833 0.9100925 0.8352564 0.9849286  
## 0.26465955 0.9100925 0.8352564 0.9849286  
## 0.28120077 0.9100925 0.8352564 0.9849286  
## 0.29774199 0.9100925 0.8352564 0.9849286  
## 0.31428321 0.9100925 0.8352564 0.9849286  
## 0.33082444 0.9100925 0.8352564 0.9849286  
## 0.34736566 0.9100925 0.8352564 0.9849286  
## 0.36390688 0.9100925 0.8352564 0.9849286  
## 0.38044810 0.9100925 0.8352564 0.9849286  
## 0.39698932 0.9100925 0.8352564 0.9849286  
## 0.41353054 0.9100925 0.8352564 0.9849286  
## 0.43007177 0.9100925 0.8352564 0.9849286  
## 0.44661299 0.9100925 0.8352564 0.9849286  
## 0.46315421 0.9100925 0.8352564 0.9849286  
## 0.47969543 0.9100925 0.8352564 0.9849286  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.

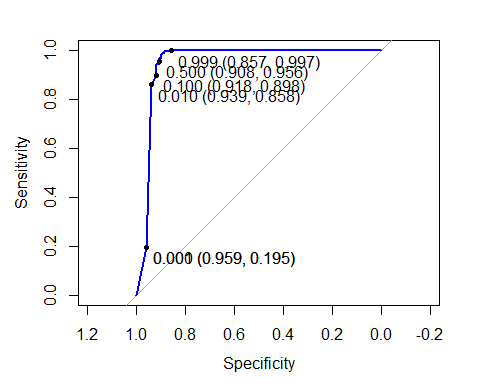
# test set results using confusion matrix  
pred\_rpart\_s <- predict(rpart\_s, ccfd\_test)  
confusionMatrix(pred\_rpart\_s, Y\_tst)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Fraud Genuine  
## Fraud 89 2519  
## Genuine 9 54344  
##   
## Accuracy : 0.9556   
## 95% CI : (0.9539, 0.9573)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0627   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.908163   
## Specificity : 0.955701   
## Pos Pred Value : 0.034126   
## Neg Pred Value : 0.999834   
## Prevalence : 0.001720   
## Detection Rate : 0.001562   
## Detection Prevalence : 0.045786   
## Balanced Accuracy : 0.931932   
##   
## 'Positive' Class : Fraud   
##

# Predicting Class probabilities  
prob\_rpart\_s <- predict(rpart\_s, ccfd\_test, type = "prob")  
head(prob\_rpart\_s)

## Fraud Genuine  
## 1 0.000000000 1.0000000  
## 2 0.857142857 0.1428571  
## 3 0.127272727 0.8727273  
## 4 0.004562044 0.9954380  
## 5 0.004562044 0.9954380  
## 6 0.004562044 0.9954380

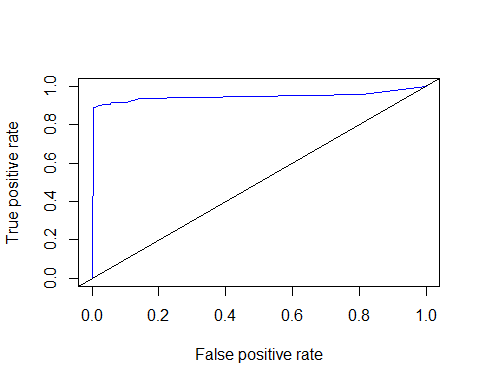
# AUC using pROC package  
ROC\_rpart\_s <- roc(Y\_tst, prob\_rpart\_s [ ,"Fraud"])  
plot(ROC\_rpart\_s, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.5, 0.999), col = "blue")



ROC\_rpart\_s

##   
## Call:  
## roc.default(response = Y\_tst, predictor = prob\_rpart\_s[, "Fraud"])  
##   
## Data: prob\_rpart\_s[, "Fraud"] in 98 controls (Y\_tst Fraud) > 56863 cases (Y\_tst Genuine).  
## Area under the curve: 0.9485

# AUC using ROCR package  
pr\_rpart\_s <- prediction(prob\_rpart\_s[ , 1], Y\_test)  
prf\_rpart\_s <- performance(pr\_rpart\_s, measure = "tpr", x.measure = "fpr")  
plot(prf\_rpart\_s, col = "blue")  
abline(0,1)



auc\_s <- performance(pr\_rpart\_s, measure = "auc")  
auc\_s <- auc\_s@y.values[[1]]  
auc\_s

## [1] 0.9484651

# rpart Using “up”" resampling

set.seed(7)  
rpart\_up <- train(Class ~., data = ccfd\_train, method = "rpart", parms = list(split = "information"), metric = "ROC", trControl=ctr\_up, tuneLength = 30)

## + Fold01: cp=0   
## - Fold01: cp=0   
## + Fold02: cp=0   
## - Fold02: cp=0   
## + Fold03: cp=0   
## - Fold03: cp=0   
## + Fold04: cp=0   
## - Fold04: cp=0   
## + Fold05: cp=0   
## - Fold05: cp=0   
## + Fold06: cp=0   
## - Fold06: cp=0   
## + Fold07: cp=0   
## - Fold07: cp=0   
## + Fold08: cp=0   
## - Fold08: cp=0   
## + Fold09: cp=0   
## - Fold09: cp=0   
## + Fold10: cp=0   
## - Fold10: cp=0   
## Aggregating results  
## Selecting tuning parameters  
## Fitting cp = 0.0165 on full training set

rpart\_up

## CART   
##   
## 227846 samples  
## 29 predictor  
## 2 classes: 'Fraud', 'Genuine'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 205060, 205062, 205062, 205062, 205061, 205062, ...   
## Addtional sampling using up-sampling  
##   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.00000000 0.8896530 0.7796795 0.9993229  
## 0.01654122 0.9253976 0.8581410 0.9717963  
## 0.03308244 0.9152723 0.8454487 0.9850958  
## 0.04962367 0.9152723 0.8454487 0.9850958  
## 0.06616489 0.9152723 0.8454487 0.9850958  
## 0.08270611 0.9152723 0.8454487 0.9850958  
## 0.09924733 0.9152723 0.8454487 0.9850958  
## 0.11578855 0.9152723 0.8454487 0.9850958  
## 0.13232977 0.9152723 0.8454487 0.9850958  
## 0.14887100 0.9152723 0.8454487 0.9850958  
## 0.16541222 0.9152723 0.8454487 0.9850958  
## 0.18195344 0.9152723 0.8454487 0.9850958  
## 0.19849466 0.9152723 0.8454487 0.9850958  
## 0.21503588 0.9152723 0.8454487 0.9850958  
## 0.23157710 0.9152723 0.8454487 0.9850958  
## 0.24811833 0.9152723 0.8454487 0.9850958  
## 0.26465955 0.9152723 0.8454487 0.9850958  
## 0.28120077 0.9152723 0.8454487 0.9850958  
## 0.29774199 0.9152723 0.8454487 0.9850958  
## 0.31428321 0.9152723 0.8454487 0.9850958  
## 0.33082444 0.9152723 0.8454487 0.9850958  
## 0.34736566 0.9152723 0.8454487 0.9850958  
## 0.36390688 0.9152723 0.8454487 0.9850958  
## 0.38044810 0.9152723 0.8454487 0.9850958  
## 0.39698932 0.9152723 0.8454487 0.9850958  
## 0.41353054 0.9152723 0.8454487 0.9850958  
## 0.43007177 0.9152723 0.8454487 0.9850958  
## 0.44661299 0.9152723 0.8454487 0.9850958  
## 0.46315421 0.9152723 0.8454487 0.9850958  
## 0.47969543 0.9152723 0.8454487 0.9850958  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.01654122.

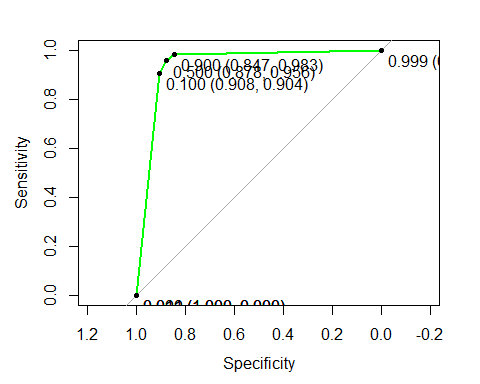
# test set results using confusion matrix  
pred\_rpart\_up <- predict(rpart\_up, ccfd\_test)  
confusionMatrix(pred\_rpart\_up, Y\_tst)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Fraud Genuine  
## Fraud 86 2518  
## Genuine 12 54345  
##   
## Accuracy : 0.9556   
## 95% CI : (0.9539, 0.9573)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0605   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.87755   
## Specificity : 0.95572   
## Pos Pred Value : 0.03303   
## Neg Pred Value : 0.99978   
## Prevalence : 0.00172   
## Detection Rate : 0.00151   
## Detection Prevalence : 0.04572   
## Balanced Accuracy : 0.91663   
##   
## 'Positive' Class : Fraud   
##

# Predicting Class probabilities  
prob\_rpart\_up <- predict(rpart\_up, ccfd\_test, type = "prob")  
head(prob\_rpart\_up)

## Fraud Genuine  
## 1 0.07207488 0.9279251  
## 2 0.19819634 0.8018037  
## 3 0.19819634 0.8018037  
## 4 0.07207488 0.9279251  
## 5 0.07207488 0.9279251  
## 6 0.07207488 0.9279251

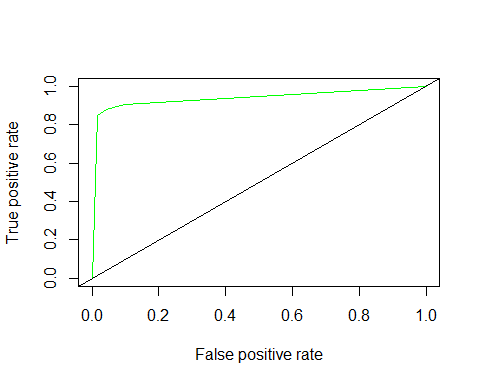
# AUC using pROC package  
ROC\_rpart\_up <- roc(Y\_tst, prob\_rpart\_up [ ,"Fraud"])  
plot(ROC\_rpart\_up, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.5, 0.9, 0.999), col = "green")



ROC\_rpart\_up

##   
## Call:  
## roc.default(response = Y\_tst, predictor = prob\_rpart\_up[, "Fraud"])  
##   
## Data: prob\_rpart\_up[, "Fraud"] in 98 controls (Y\_tst Fraud) > 56863 cases (Y\_tst Genuine).  
## Area under the curve: 0.9395

# AUC using ROCR package  
pr\_rpart\_up <- prediction(prob\_rpart\_up[ , 1], Y\_test)  
prf\_rpart\_up <- performance(pr\_rpart\_up, measure = "tpr", x.measure = "fpr")  
plot(prf\_rpart\_up, col = "green")  
abline(0,1)



auc\_rup <- performance(pr\_rpart\_up, measure = "auc")  
auc\_rup <- auc\_rup@y.values[[1]]  
auc\_rup

## [1] 0.9394688

# rpart Using “down”" resampling

set.seed(7)  
rpart\_down <- train(Class ~., data = ccfd\_train, method = "rpart", parms = list(split = "information"), metric = "ROC", trControl=ctr\_down, tuneLength = 30)

## + Fold01: cp=0   
## - Fold01: cp=0   
## + Fold02: cp=0   
## - Fold02: cp=0   
## + Fold03: cp=0   
## - Fold03: cp=0   
## + Fold04: cp=0   
## - Fold04: cp=0   
## + Fold05: cp=0   
## - Fold05: cp=0   
## + Fold06: cp=0   
## - Fold06: cp=0   
## + Fold07: cp=0   
## - Fold07: cp=0   
## + Fold08: cp=0   
## - Fold08: cp=0   
## + Fold09: cp=0   
## - Fold09: cp=0   
## + Fold10: cp=0   
## - Fold10: cp=0   
## Aggregating results  
## Selecting tuning parameters  
## Fitting cp = 0 on full training set

rpart\_down

## CART   
##   
## 227846 samples  
## 29 predictor  
## 2 classes: 'Fraud', 'Genuine'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 205060, 205062, 205062, 205062, 205061, 205062, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.00000000 0.9408647 0.8961538 0.9176441  
## 0.01654122 0.9249469 0.8706410 0.9666568  
## 0.03308244 0.9167414 0.8580769 0.9754059  
## 0.04962367 0.9167414 0.8580769 0.9754059  
## 0.06616489 0.9167414 0.8580769 0.9754059  
## 0.08270611 0.9167414 0.8580769 0.9754059  
## 0.09924733 0.9167414 0.8580769 0.9754059  
## 0.11578855 0.9167414 0.8580769 0.9754059  
## 0.13232977 0.9167414 0.8580769 0.9754059  
## 0.14887100 0.9167414 0.8580769 0.9754059  
## 0.16541222 0.9167414 0.8580769 0.9754059  
## 0.18195344 0.9167414 0.8580769 0.9754059  
## 0.19849466 0.9167414 0.8580769 0.9754059  
## 0.21503588 0.9167414 0.8580769 0.9754059  
## 0.23157710 0.9167414 0.8580769 0.9754059  
## 0.24811833 0.9167414 0.8580769 0.9754059  
## 0.26465955 0.9167414 0.8580769 0.9754059  
## 0.28120077 0.9167414 0.8580769 0.9754059  
## 0.29774199 0.9167414 0.8580769 0.9754059  
## 0.31428321 0.9167414 0.8580769 0.9754059  
## 0.33082444 0.9167414 0.8580769 0.9754059  
## 0.34736566 0.9167414 0.8580769 0.9754059  
## 0.36390688 0.9167414 0.8580769 0.9754059  
## 0.38044810 0.9167414 0.8580769 0.9754059  
## 0.39698932 0.9167414 0.8580769 0.9754059  
## 0.41353054 0.9167414 0.8580769 0.9754059  
## 0.43007177 0.9167414 0.8580769 0.9754059  
## 0.44661299 0.9167414 0.8580769 0.9754059  
## 0.46315421 0.9167414 0.8580769 0.9754059  
## 0.47969543 0.9167414 0.8580769 0.9754059  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.

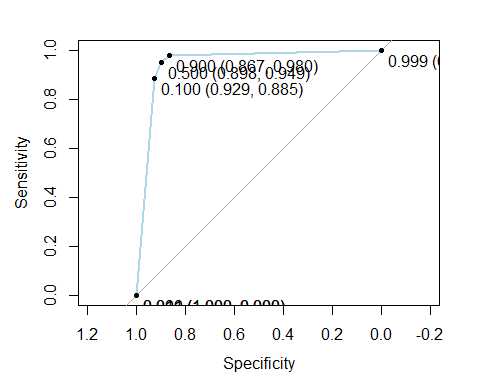
# test set results using confusion matrix  
pred\_rpart\_down <- predict(rpart\_down, ccfd\_test)  
confusionMatrix(pred\_rpart\_down, Y\_tst)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Fraud Genuine  
## Fraud 88 2879  
## Genuine 10 53984  
##   
## Accuracy : 0.9493   
## 95% CI : (0.9474, 0.9511)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0543   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.897959   
## Specificity : 0.949370   
## Pos Pred Value : 0.029660   
## Neg Pred Value : 0.999815   
## Prevalence : 0.001720   
## Detection Rate : 0.001545   
## Detection Prevalence : 0.052088   
## Balanced Accuracy : 0.923664   
##   
## 'Positive' Class : Fraud   
##

# Predicting Class probabilities  
prob\_rpart\_down <- predict(rpart\_down, ccfd\_test, type = "prob")  
head(prob\_rpart\_down)

## Fraud Genuine  
## 1 0.05612245 0.9438776  
## 2 0.21428571 0.7857143  
## 3 0.21428571 0.7857143  
## 4 0.05612245 0.9438776  
## 5 0.05612245 0.9438776  
## 6 0.05612245 0.9438776

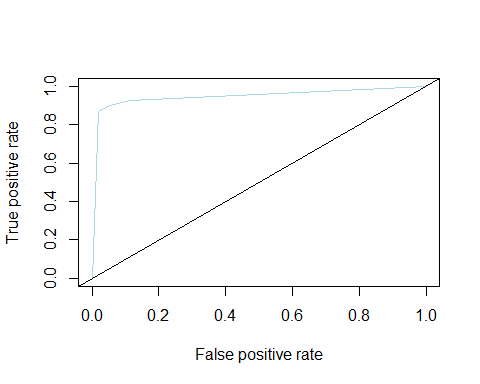
# AUC using pROC package  
ROC\_rpart\_down <- roc(Y\_tst, prob\_rpart\_down [ ,"Fraud"])  
plot(ROC\_rpart\_down, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.5, 0.9, 0.999), col = "lightblue")



ROC\_rpart\_down

##   
## Call:  
## roc.default(response = Y\_tst, predictor = prob\_rpart\_down[, "Fraud"])  
##   
## Data: prob\_rpart\_down[, "Fraud"] in 98 controls (Y\_tst Fraud) > 56863 cases (Y\_tst Genuine).  
## Area under the curve: 0.9478

# AUC using ROCR package  
pr\_rpart\_down <- prediction(prob\_rpart\_down[ , 1], Y\_test)  
prf\_rpart\_down <- performance(pr\_rpart\_down, measure = "tpr", x.measure = "fpr")  
plot(prf\_rpart\_down, col = "lightblue")  
abline(0,1)



auc\_rd <- performance(pr\_rpart\_down, measure = "auc")  
auc\_rd <- auc\_rd@y.values[[1]]  
auc\_rd

## [1] 0.9478227

# Stratified 80% - 20% splitting of the dataset into training and testing (V4, V8, V10, V12, V14, V16, V17, V21, V27, and Amount)

set.seed(7)  
train\_id <- createDataPartition(ccfd$Class,times = 1, p=0.8, list=FALSE)  
  
ccf <- ccfd[ , c("V4", "V8", "V10", "V12", "V14", "V16", "V17", "V21", "V27", "Amount", "Class")]  
ccf\_train <- ccf[train\_id, ]  
ccf\_test <- ccf[-train\_id, ]  
  
zscorenormalize <- function(x){   
 return((x-mean(x))/(sd(x)))}  
  
Amount\_tr <- as.data.frame(lapply(ccf\_train[10], zscorenormalize))  
ccf\_train <- as.data.frame(c(ccf\_train[1:9],Amount\_tr,ccf\_train[11]))  
Amount\_ts <- as.data.frame(lapply(ccf\_test[10], zscorenormalize))  
ccf\_test <- as.data.frame(c(ccf\_test[1:9],Amount\_ts,ccf\_test[11]))  
  
X\_tr <- ccf\_train[ ,-11]  
X\_ts <- ccf\_test[ ,-11]  
  
Y\_tr <- ccf\_train[ ,11]  
Y\_ts <- ccf\_test[ ,11]  
  
## stratified folds for cross-validation:  
foldId <- createFolds(Y\_tr, k = 10, list = TRUE, returnTrain = TRUE)  
lapply(foldId, function(ii) table(Y\_tr[ii]))

## $Fold01  
##   
## Fraud Genuine   
## 354 204706   
##   
## $Fold02  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold03  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold04  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold05  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold06  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold07  
##   
## Fraud Genuine   
## 355 204707   
##   
## $Fold08  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold09  
##   
## Fraud Genuine   
## 354 204707   
##   
## $Fold10  
##   
## Fraud Genuine   
## 355 204706

# rpart using “smote” resampling with 10 selected features

set.seed(10)  
rpart\_s\_sf <- train(Class ~., data = ccf\_train, method = "rpart", parms = list(split = "information"), metric = "ROC", trControl=ctr\_s, tuneLength = 30)

## + Fold01: cp=0   
## - Fold01: cp=0   
## + Fold02: cp=0   
## - Fold02: cp=0   
## + Fold03: cp=0   
## - Fold03: cp=0   
## + Fold04: cp=0   
## - Fold04: cp=0   
## + Fold05: cp=0   
## - Fold05: cp=0   
## + Fold06: cp=0   
## - Fold06: cp=0   
## + Fold07: cp=0   
## - Fold07: cp=0   
## + Fold08: cp=0   
## - Fold08: cp=0   
## + Fold09: cp=0   
## - Fold09: cp=0   
## + Fold10: cp=0   
## - Fold10: cp=0   
## Aggregating results  
## Selecting tuning parameters  
## Fitting cp = 0 on full training set

rpart\_s\_sf

## CART   
##   
## 227846 samples  
## 10 predictor  
## 2 classes: 'Fraud', 'Genuine'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 205060, 205062, 205062, 205062, 205061, 205062, ...   
## Addtional sampling using SMOTE  
##   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.00000000 0.9469659 0.8859615 0.9588880  
## 0.01654122 0.9198800 0.8428846 0.9799254  
## 0.03308244 0.9101541 0.8352564 0.9850517  
## 0.04962367 0.9101541 0.8352564 0.9850517  
## 0.06616489 0.9101541 0.8352564 0.9850517  
## 0.08270611 0.9101541 0.8352564 0.9850517  
## 0.09924733 0.9101541 0.8352564 0.9850517  
## 0.11578855 0.9101541 0.8352564 0.9850517  
## 0.13232977 0.9101541 0.8352564 0.9850517  
## 0.14887100 0.9101541 0.8352564 0.9850517  
## 0.16541222 0.9101541 0.8352564 0.9850517  
## 0.18195344 0.9101541 0.8352564 0.9850517  
## 0.19849466 0.9101541 0.8352564 0.9850517  
## 0.21503588 0.9101541 0.8352564 0.9850517  
## 0.23157710 0.9101541 0.8352564 0.9850517  
## 0.24811833 0.9101541 0.8352564 0.9850517  
## 0.26465955 0.9101541 0.8352564 0.9850517  
## 0.28120077 0.9101541 0.8352564 0.9850517  
## 0.29774199 0.9101541 0.8352564 0.9850517  
## 0.31428321 0.9101541 0.8352564 0.9850517  
## 0.33082444 0.9101541 0.8352564 0.9850517  
## 0.34736566 0.9101541 0.8352564 0.9850517  
## 0.36390688 0.9101541 0.8352564 0.9850517  
## 0.38044810 0.9101541 0.8352564 0.9850517  
## 0.39698932 0.9101541 0.8352564 0.9850517  
## 0.41353054 0.9101541 0.8352564 0.9850517  
## 0.43007177 0.9101541 0.8352564 0.9850517  
## 0.44661299 0.9101541 0.8352564 0.9850517  
## 0.46315421 0.9101541 0.8352564 0.9850517  
## 0.47969543 0.9101541 0.8352564 0.9850517  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.

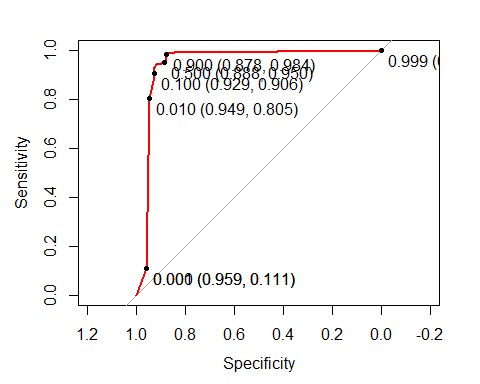
# test set results using confusion matrix  
pred\_rpart\_s\_sf <- predict(rpart\_s\_sf, ccf\_test)  
confusionMatrix(pred\_rpart\_s\_sf, Y\_ts)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Fraud Genuine  
## Fraud 87 2843  
## Genuine 11 54020  
##   
## Accuracy : 0.9499   
## 95% CI : (0.9481, 0.9517)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0543   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.887755   
## Specificity : 0.950003   
## Pos Pred Value : 0.029693   
## Neg Pred Value : 0.999796   
## Prevalence : 0.001720   
## Detection Rate : 0.001527   
## Detection Prevalence : 0.051439   
## Balanced Accuracy : 0.918879   
##   
## 'Positive' Class : Fraud   
##

# Predicting Class probabilities  
prob\_rpart\_s\_sf <- predict(rpart\_s\_sf, ccf\_test, type = "prob")  
head(prob\_rpart\_s\_sf)

## Fraud Genuine  
## 1 0.054263566 0.9457364  
## 2 0.000000000 1.0000000  
## 3 0.266666667 0.7333333  
## 4 0.006167401 0.9938326  
## 5 0.006167401 0.9938326  
## 6 0.006167401 0.9938326

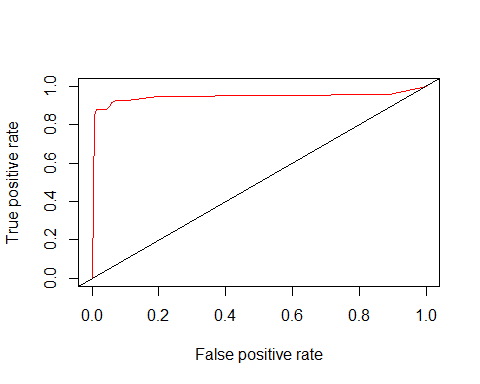
# AUC using pROC package  
ROC\_rpart\_s\_sf <- roc(Y\_ts, prob\_rpart\_s\_sf [ ,"Fraud"])  
plot(ROC\_rpart\_s\_sf, print.thres = c(0.0001,0.001, 0.01, 0.1, 0.5, 0.9, 0.999), col ="red")



ROC\_rpart\_s\_sf

##   
## Call:  
## roc.default(response = Y\_ts, predictor = prob\_rpart\_s\_sf[, "Fraud"])  
##   
## Data: prob\_rpart\_s\_sf[, "Fraud"] in 98 controls (Y\_ts Fraud) > 56863 cases (Y\_ts Genuine).  
## Area under the curve: 0.9464

# AUC using ROCR package  
pr\_rpart\_s\_sf <- prediction(prob\_rpart\_s\_sf[ , 1], Y\_test)  
prf\_rpart\_s\_sf <- performance(pr\_rpart\_s\_sf, measure = "tpr", x.measure = "fpr")  
plot(prf\_rpart\_s\_sf, col = "red" )  
abline(0,1)



auc\_s\_sf <- performance(pr\_rpart\_s\_sf, measure = "auc")  
auc\_s\_sf <- auc\_s\_sf@y.values[[1]]  
auc\_s\_sf

## [1] 0.9463916