Team 5: Project Deliverable

Roozbeh Jafari, Jeffrey Leung, Phoenix Wang, Ying Wu, Yuzhe Zheng

Setup

```
library(data.table)
library(caTools)
library(xgboost)
library(caret)
```

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

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

```
library(e1071)
library(rpart)
library(lattice)
library(ROCR)
library(pROC)
```

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

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

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

```
library(ROSE)
```

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

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
library(ggplot2)
library(dplyr)
```

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

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

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

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

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

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

```
library(solitude)
library(ggplot2)
library(glmnet)
```

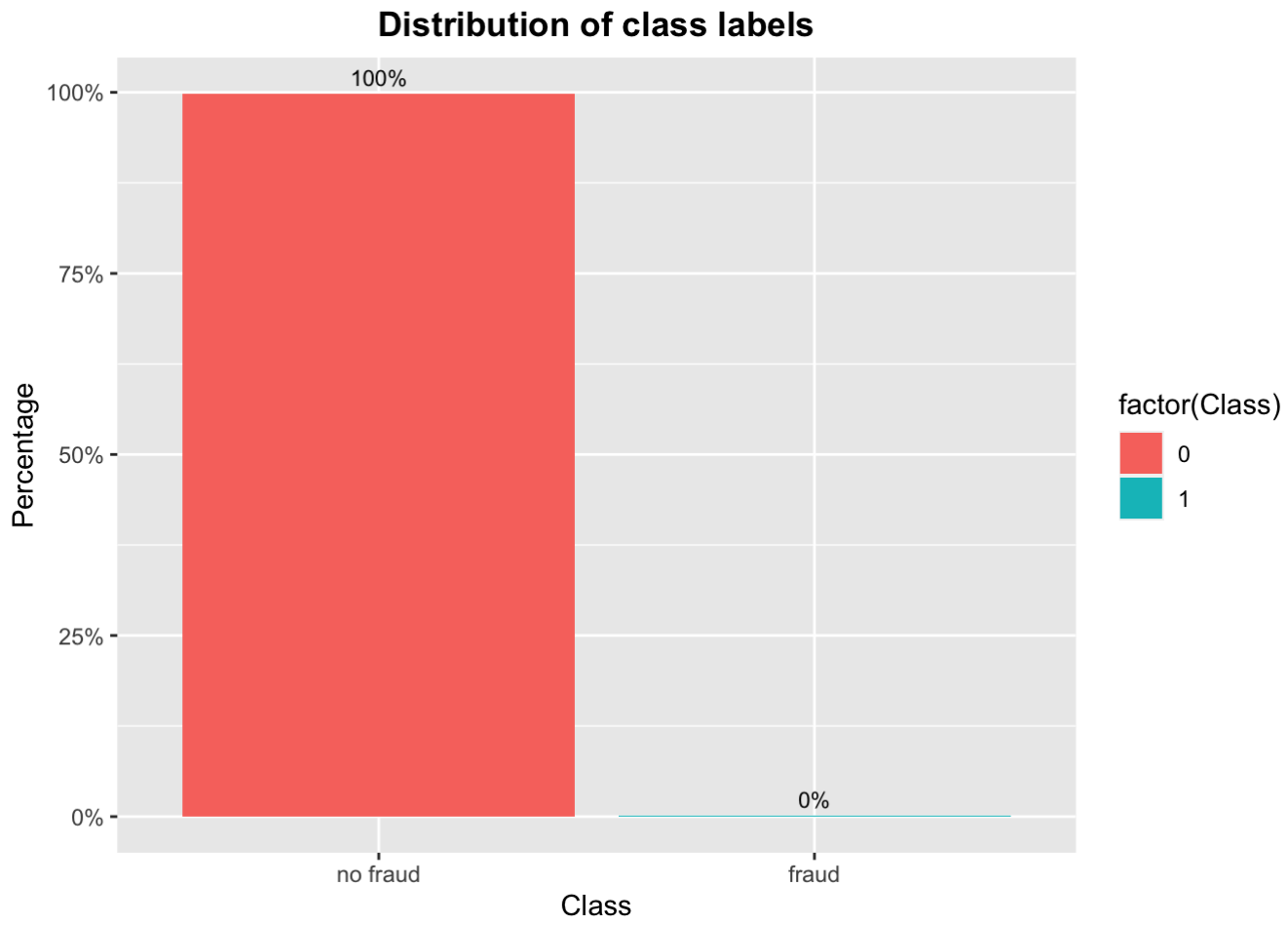
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1
```

```
credit_card_raw = fread("creditcard.csv")
```

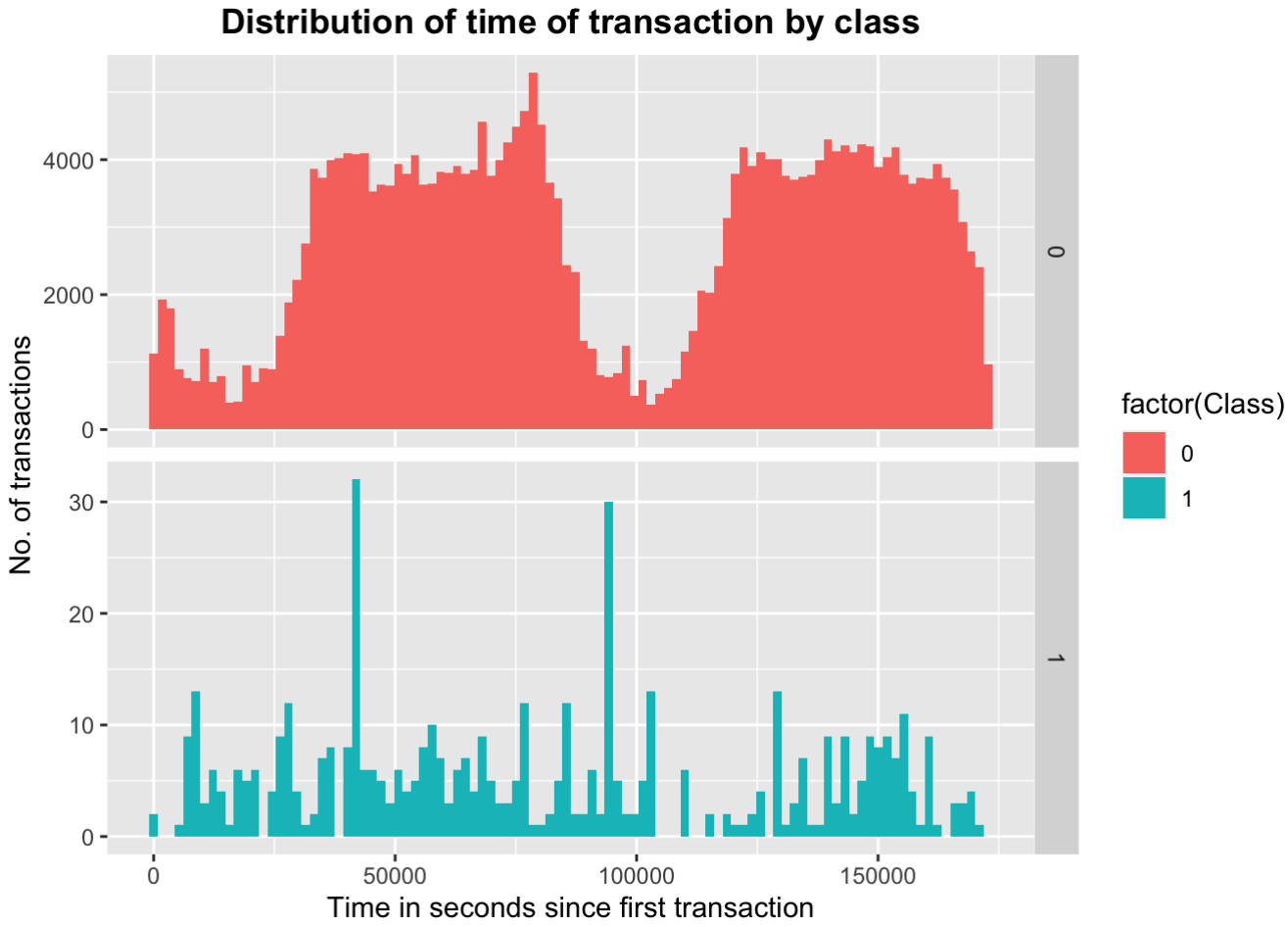
Exploratory Data Analysis

```
common_theme <- theme(plot.title = element_text(hjust = 0.5, face = "bold"))
ggplot(data = credit_card_raw, aes(x = factor(Class),
                          y = prop.table(stat(count)), fill = factor(Class),
                          label = scales::percent(prop.table(stat(count))))) +
    geom_bar(position = "dodge") +
    geom_text(stat = 'count',
              position = position_dodge(.9),
              vjust = -0.5,
              size = 3) +
    scale_x_discrete(labels = c("no fraud", "fraud"))+
    scale_y_continuous(labels = scales::percent)+
    labs(x = 'Class', y = 'Percentage') +
    ggtitle("Distribution of class labels") +
    common_theme
```



Clearly the dataset is very imbalanced with 99.8% of cases being non-fraudulent transactions. A simple measure like accuracy is not appropriate here as even a classifier which labels all transactions as non-fraudulent will have over 99% accuracy. An appropriate measure of model performance here would be AUC (Area Under the Precision-Recall Curve).

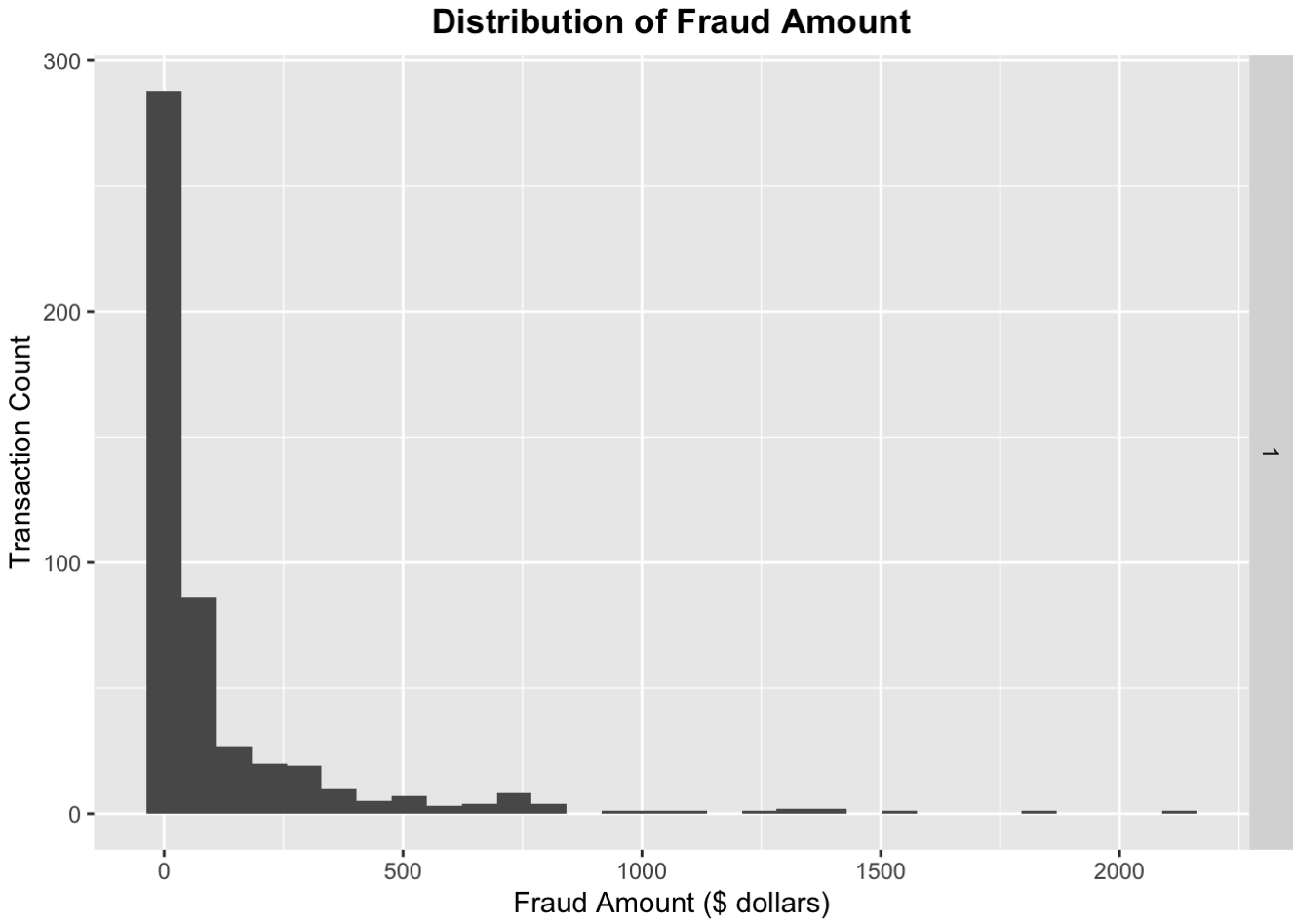
```
ggplot(data=credit_card_raw, aes(x = Time, fill = factor(Class))) + geom_histogram(bins = 100
)+
labs(x = "Time in seconds since first transaction", y = "No. of transactions") +
ggtitle("Distribution of time of transaction by class") +
facet_grid(Class ~ ., scales = "free_y") + common_theme
```



The ‘Time’ feature looks pretty similar across both types of transactions. One could argue that fraudulent transactions are more uniformly distributed.

```
#histogram
fraud_amount <- credit_card_raw[Class == 1]
ggplot(fraud_amount, aes(as.integer(fraud_amount$Amount))) + geom_histogram() +
labs(x = "Fraud Amount ($ dollars)", y = "Transaction Count") +
ggtitle("Distribution of Fraud Amount") +
facet_grid(Class ~ ., scales = "free_y") + common_theme
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



According to this distribution, the fraud amount is highly skewed to the left.

Splitting Data

For all the models, we will be using the same 80-20 train-test split. There will be our imbalanced sets which are directly taken from the raw data and downsampled sets that even out the non-fraud to fraud transactions.

```
# Create train and test dataset
credit_card_raw[, test:=0]
credit_card_raw[, "Time":= NULL]
credit_card_raw[sample(nrow(credit_card_raw), 284807*0.2), test:=1]
test <- credit_card_raw[test==1]
train <- credit_card_raw[test==0]
train[, "test" := NULL]
test[, "test" := NULL]
credit_card_raw[, "test" := NULL]

# Convert datatables to dataframes for downsampling
setDF(train)
setDF(test)

# Downsample
set.seed(1)
train$Class <- factor(train$Class)
downsample.train <- downSample(train[, -ncol(train)], train$Class)

test$Class <- factor(test$Class)
downsample.test <- downSample(test[, -ncol(test)], test$Class)
```

Lasso Regression

For this model, we will be running a lasso regression. We will perform a 5 fold cross validation to determine which lambda value minimizes MSE the most and use that to make predictions.

We first train a model on our downsampled training dataset.

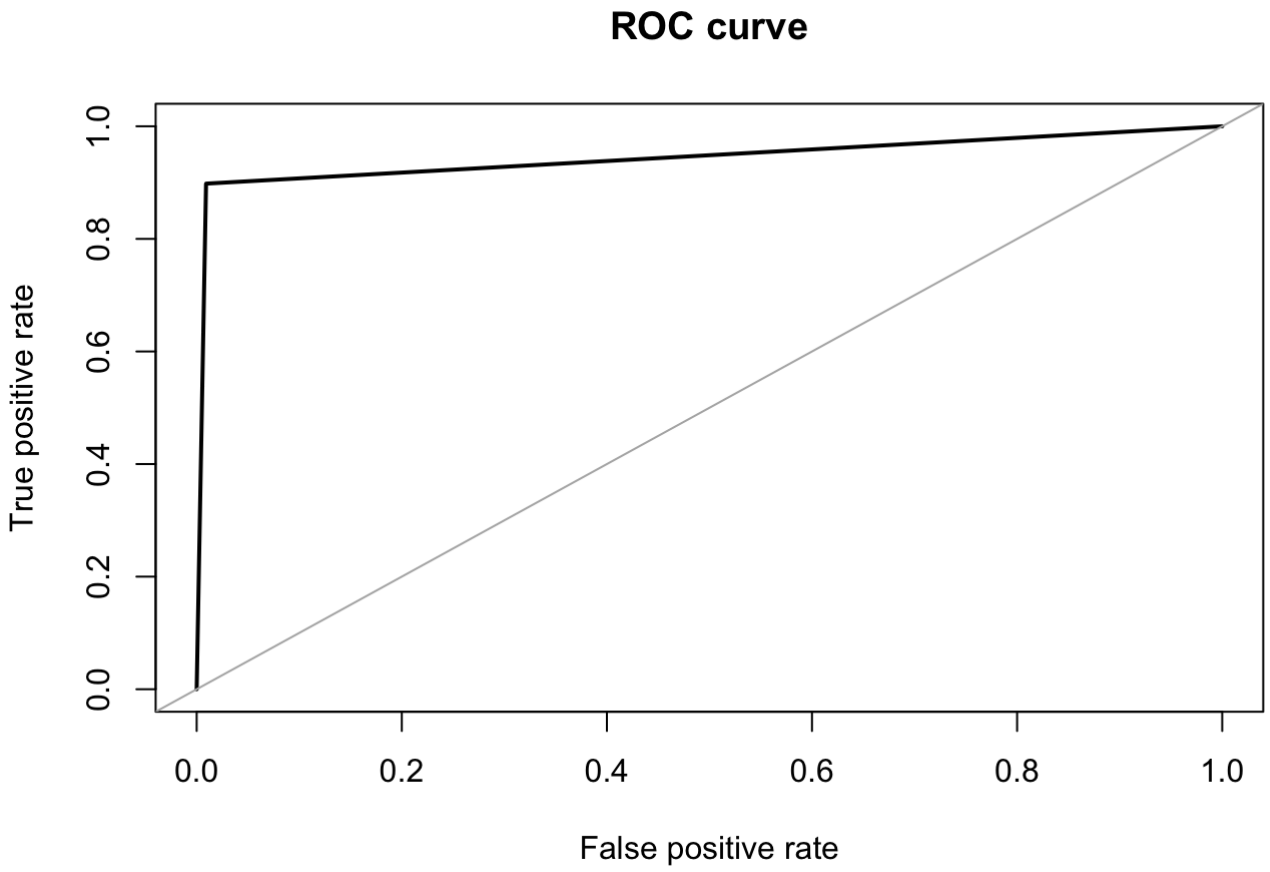
```
# Create formula
formula <- as.formula(Class ~ .)

# Downsample training set modeling
downsample.train.matrix <- model.matrix(formula, downsample.train)[, -1]
y.downsample.train <- downsample.train$Class
downsample.fit <- cv.glmnet(downsample.train.matrix, y.downsample.train, family = "binomial",
alpha = 1, nfolds = 5)

# Create testing matrices
downsample.test.matrix <- model.matrix(formula, downsample.test) [, -1]
imbalanced.test.matrix <- model.matrix(formula, test)[, -1]
```

Next, we predict on a downsampled testset and the original imbalanced testset.

```
# Predicting Downsample test data
downsample.test.predictions <- predict(downsample.fit, downsample.test.matrix, s = downsampl
e.fit$lambda.min)
predicted.classes <- ifelse(downsample.test.predictions > 0, 1, 0)
confusionMatrix(as.factor(predicted.classes), downsample.test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 107  11
##          1   1  97
##
##                Accuracy : 0.9444
##                  95% CI : (0.905, 0.971)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8889
##
##  Mcnemar's Test P-Value : 0.009375
##
##             Sensitivity : 0.8981
##             Specificity : 0.9907
##          Pos Pred Value : 0.9898
##          Neg Pred Value : 0.9068
##              Prevalence : 0.5000
##          Detection Rate : 0.4491
##    Detection Prevalence : 0.4537
##       Balanced Accuracy : 0.9444
##
##        'Positive' Class : 1
##
```

```
roc.curve(as.numeric(downsample.test$Class), as.numeric(predicted.classes), plotit = TRUE)
```

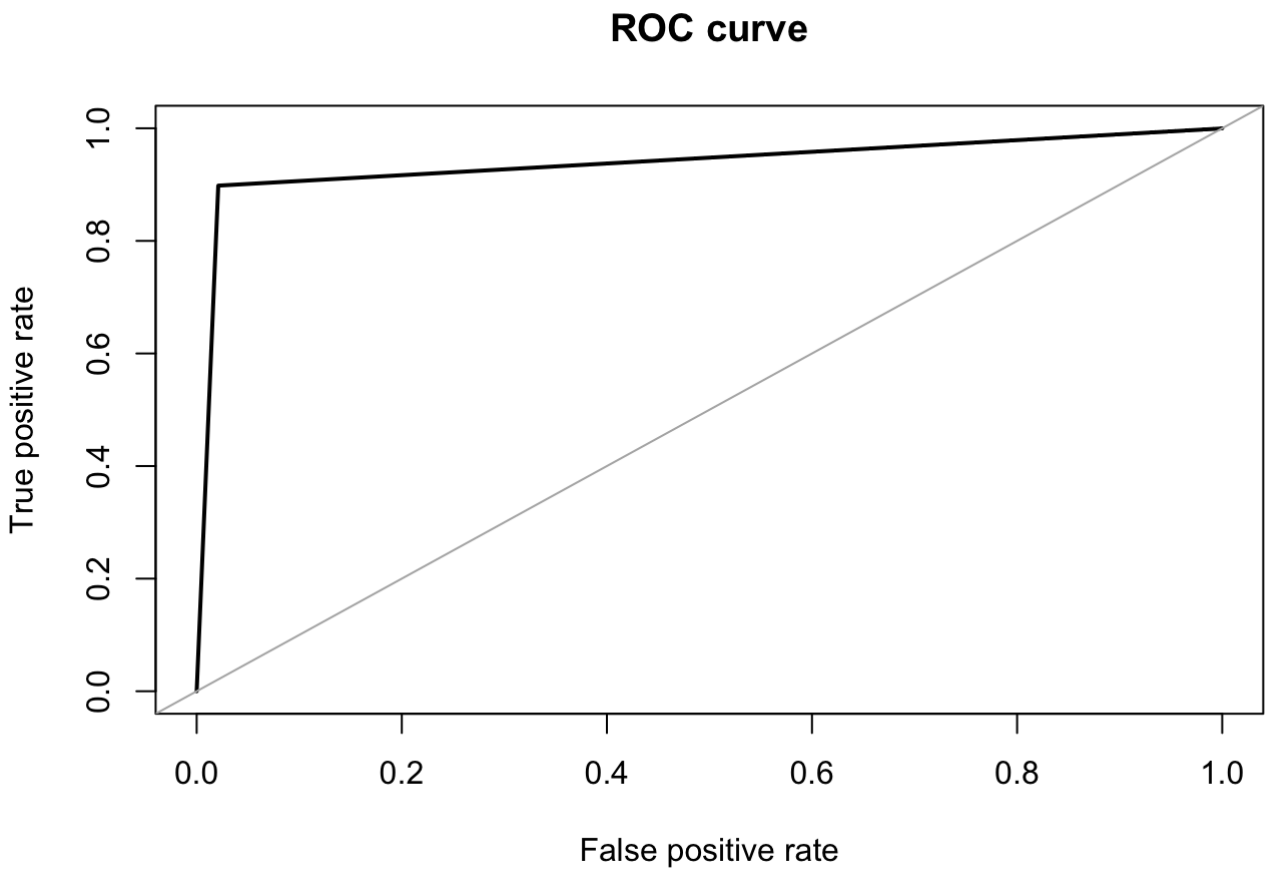



```
## Area under the curve (AUC): 0.944
```

```
# Predicting imbalanced test data
test.predictions <- predict(downsample.fit, imbalanced.test.matrix, s = downsample.fit$lambd
a.min)
predicted.classes <- ifelse(test.predictions > 0, 1, 0)
confusionMatrix(as.factor(predicted.classes), test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 55651    11
##          1  1202    97
##
##                Accuracy : 0.9787
##                  95% CI : (0.9775, 0.9799)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1349
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.898148
##             Specificity : 0.978858
##          Pos Pred Value : 0.074673
##          Neg Pred Value : 0.999802
##              Prevalence : 0.001896
##          Detection Rate : 0.001703
##    Detection Prevalence : 0.022805
##       Balanced Accuracy : 0.938503
##
##        'Positive' Class : 1
##
```

```
roc.curve(as.numeric(test$Class), as.numeric(predicted.classes), plotit = TRUE)
```



```
## Area under the curve (AUC): 0.939
```

Now we train a model on our imbalanced training dataset. Due to the limitations of our computational power, our “imbalanced” set’s non-fraud transaction count was reduced to 4000. This would reduce computational strain but still retain the imbalanced aspect of the original dataset.

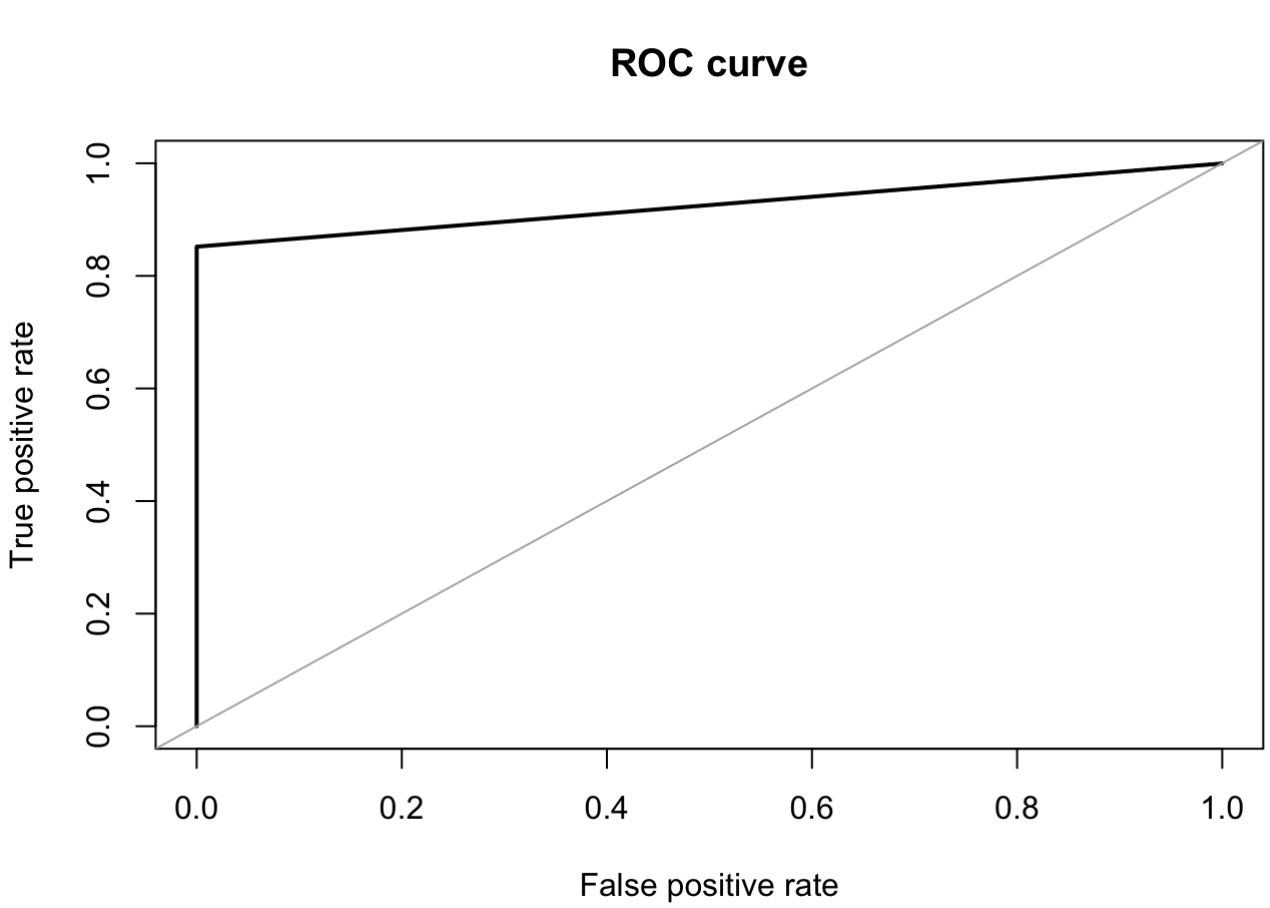
```
# Imbalanced training set modeling
class_0 = copy(train[train$Class == 0,])
x <- copy(class_0[sample(nrow(class_0), 4000),])
imbalanced.train <- rbind(x, train[train$Class == 1,])
train.matrix <- model.matrix(formula, imbalanced.train)[, -1]
y.train <- imbalanced.train$Class
imbalanced.fit <- cv.glmnet(train.matrix, y.train, family = "binomial", alpha = 1, nfolds = 5
)
```

As we did before, we will predict on a downsampled testset and the original imbalanced test set.

```
# Predicting Downsample test data
downsample.test.predictions <- predict(imbalanced.fit, downsample.test.matrix, s = imbalance
d.fit$lambda.min)
predicted.classes <- ifelse(downsample.test.predictions > 0, 1, 0)
confusionMatrix(as.factor(predicted.classes), downsample.test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 108  16
##          1   0  92
##
##                Accuracy : 0.9259
##                  95% CI : (0.8825, 0.9571)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8519
##
##  Mcnemar's Test P-Value : 0.0001768
##
##             Sensitivity : 0.8519
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8710
##              Prevalence : 0.5000
##          Detection Rate : 0.4259
##    Detection Prevalence : 0.4259
##       Balanced Accuracy : 0.9259
##
##        'Positive' Class : 1
##
```

```
roc.curve(as.numeric(downsample.test$Class), as.numeric(predicted.classes), plotit = TRUE)
```

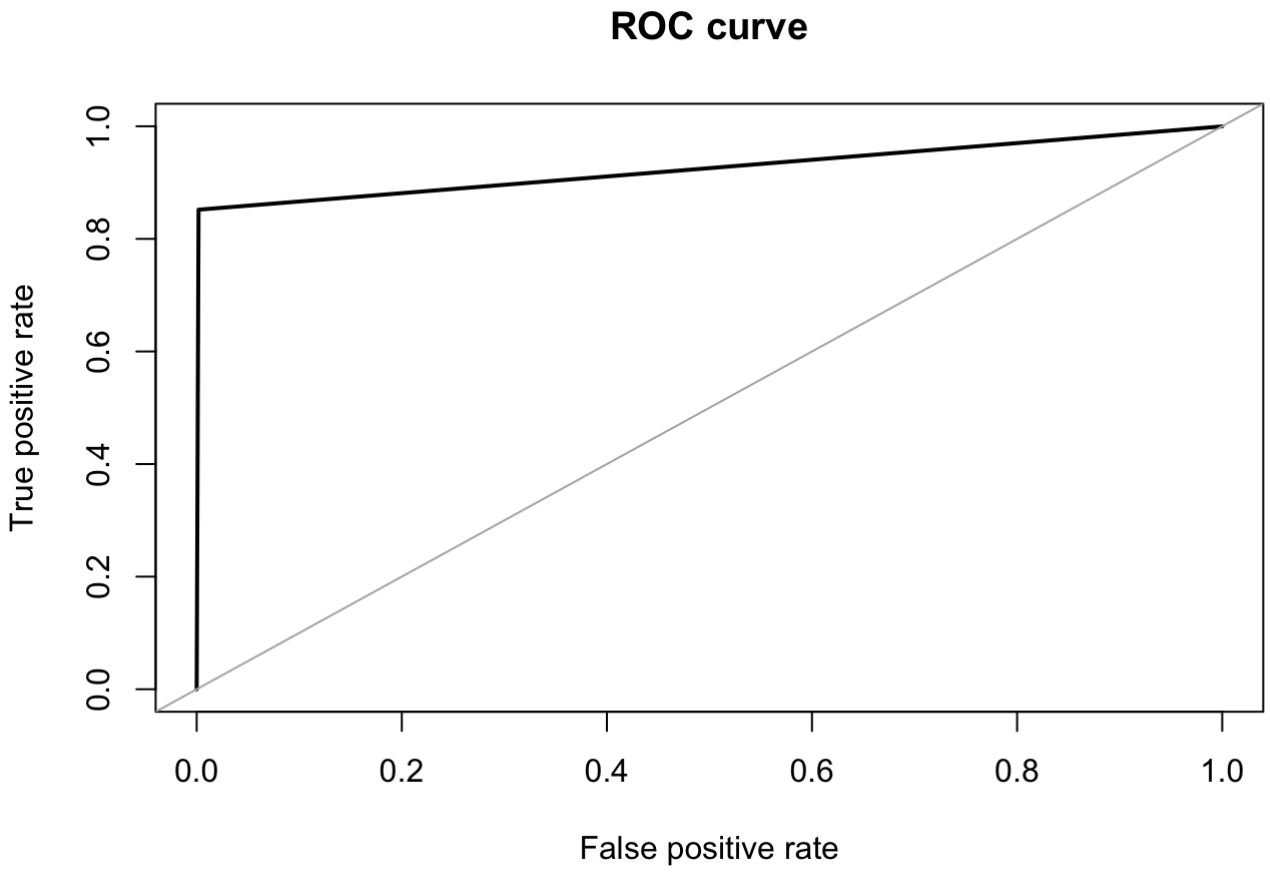



```
## Area under the curve (AUC): 0.926
```

```
#Predicting imbalanced test data
test.predictions <- predict(imbalanced.fit, imbalanced.test.matrix, s = imbalanced.fit$lambd
a.min)
predicted.classes <- ifelse(test.predictions > 0, 1, 0)
confusionMatrix(as.factor(predicted.classes), test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56749    16
##          1   104    92
##
##                Accuracy : 0.9979
##                  95% CI : (0.9975, 0.9983)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 0.8845
##
##                   Kappa : 0.6043
##
##  Mcnemar's Test P-Value : 1.99e-15
##
##             Sensitivity : 0.851852
##             Specificity : 0.998171
##          Pos Pred Value : 0.469388
##          Neg Pred Value : 0.999718
##              Prevalence : 0.001896
##          Detection Rate : 0.001615
##    Detection Prevalence : 0.003441
##       Balanced Accuracy : 0.925011
##
##        'Positive' Class : 1
##
```

```
roc.curve(as.numeric(test$Class), as.numeric(predicted.classes), plotit = TRUE)
```



```
## Area under the curve (AUC): 0.925
```

Logistic Regression

Logistic regression is a simple regression model whose output is a score between 0 and 1. This is achieved by using the logistic function.

Fit logistic regression model by building two models on downsampl(balanced) train data and original(imbalanced) train data, then run each model on both orginal(imbalanced) and downsample(balanced) test data

```
# Fit logistic regression model
set.seed(1)

#fit the model on balanced data(downsampling)
down_fit <- glm(Class ~ ., family = "binomial" , data = downsample.train)
```

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

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

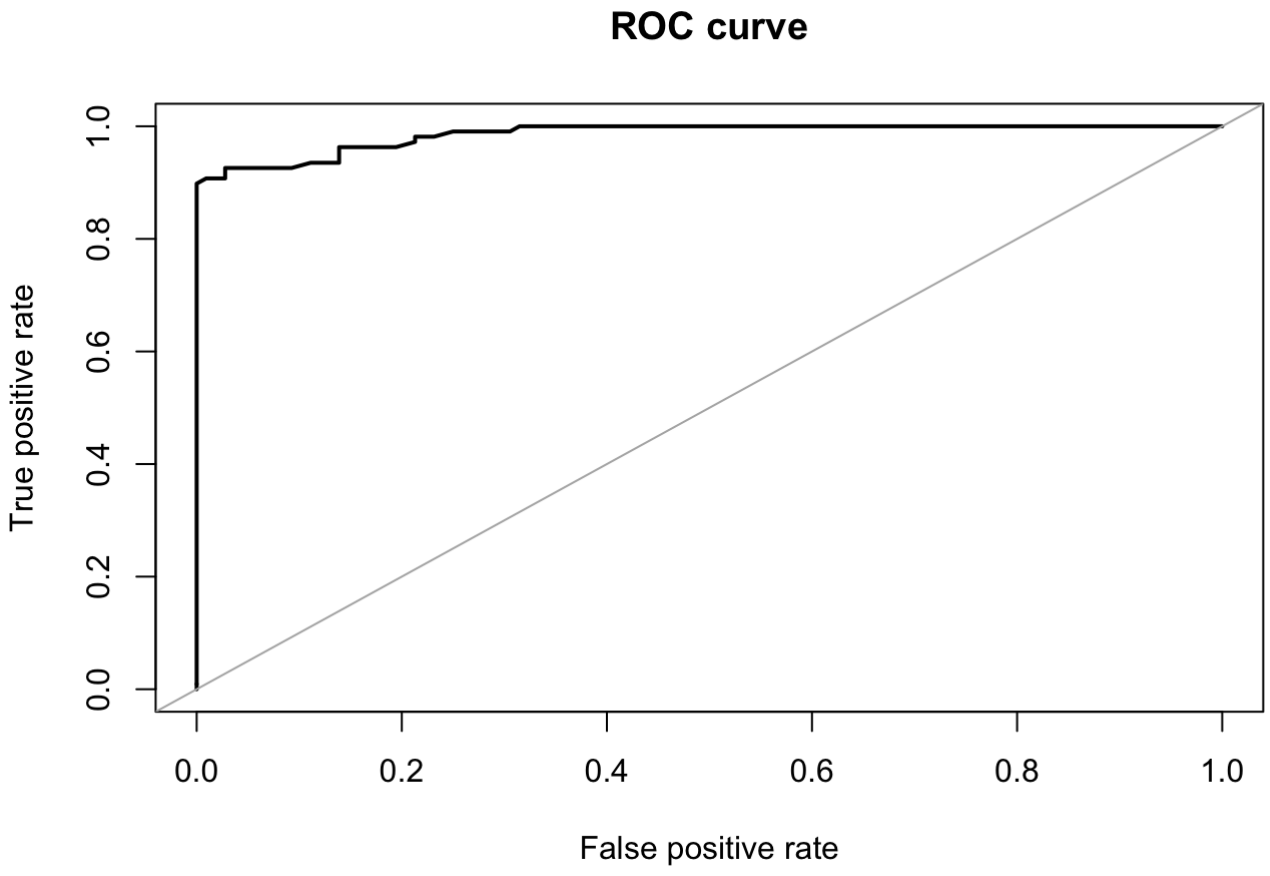
```
summary(down_fit,)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = downsample.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8411  -0.2203   0.0000   0.0000   2.6890
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -21.2616    37.1107  -0.573    0.567
## V1            -3.6614    10.0028  -0.366    0.714
## V2            37.6039    54.4862   0.690    0.490
## V3           -29.7309    24.1930  -1.229    0.219
## V4            21.5692    18.5723   1.161    0.245
## V5           -11.0647     9.4889  -1.166    0.244
## V6           -17.9879    24.7840  -0.726    0.468
## V7           -71.3257    86.0037  -0.829    0.407
## V8            12.8044    14.7743   0.867    0.386
## V9           -26.4364    26.9377  -0.981    0.326
## V10          -60.9212    61.9817  -0.983    0.326
## V11           47.4800    51.9131   0.915    0.360
## V12          -85.3580    93.2289  -0.916    0.360
## V13           -0.6123     2.4420  -0.251    0.802
## V14          -91.6474   101.4642  -0.903    0.366
## V15           -2.3096     3.5696  -0.647    0.518
## V16          -81.0330    89.6241  -0.904    0.366
## V17         -143.7199   157.5899  -0.912    0.362
## V18          -54.6196    60.1371  -0.908    0.364
## V19           21.3943    24.7104   0.866    0.387
## V20           -7.7323    17.0759  -0.453    0.651
## V21            6.4011     5.1763   1.237    0.216
## V22            6.8802    10.8328   0.635    0.525
## V23           17.6765    32.8072   0.539    0.590
## V24           -2.0493     3.1382  -0.653    0.514
## V25           10.4763    14.8989   0.703    0.482
## V26            2.7018     3.7719   0.716    0.474
## V27           12.4241    12.3435   1.007    0.314
## V28           22.7686    39.5462   0.576    0.565
## Amount         0.2173     0.3788   0.574    0.566
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1064.67  on 767  degrees of freedom
## Residual deviance:  176.04  on 738  degrees of freedom
## AIC: 236.04
##
## Number of Fisher Scoring iterations: 25
```

```
pred_down <- predict(down_fit, downsample.test) #balanced

#Evaluate model performance on test set
confusionMatrix(data = as.factor(as.numeric(pred_down >0.5)), reference = as.factor(downsampl
e.test$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 105  10
##          1   3  98
##
##                Accuracy : 0.9398
##                  95% CI : (0.8993, 0.9676)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8796
##
##  Mcnemar's Test P-Value : 0.09609
##
##             Sensitivity : 0.9074
##             Specificity : 0.9722
##          Pos Pred Value : 0.9703
##          Neg Pred Value : 0.9130
##              Prevalence : 0.5000
##          Detection Rate : 0.4537
##    Detection Prevalence : 0.4676
##       Balanced Accuracy : 0.9398
##
##        'Positive' Class : 1
##
```

```
roc.curve(downsample.test$Class, pred_down, plotit=TRUE)
```

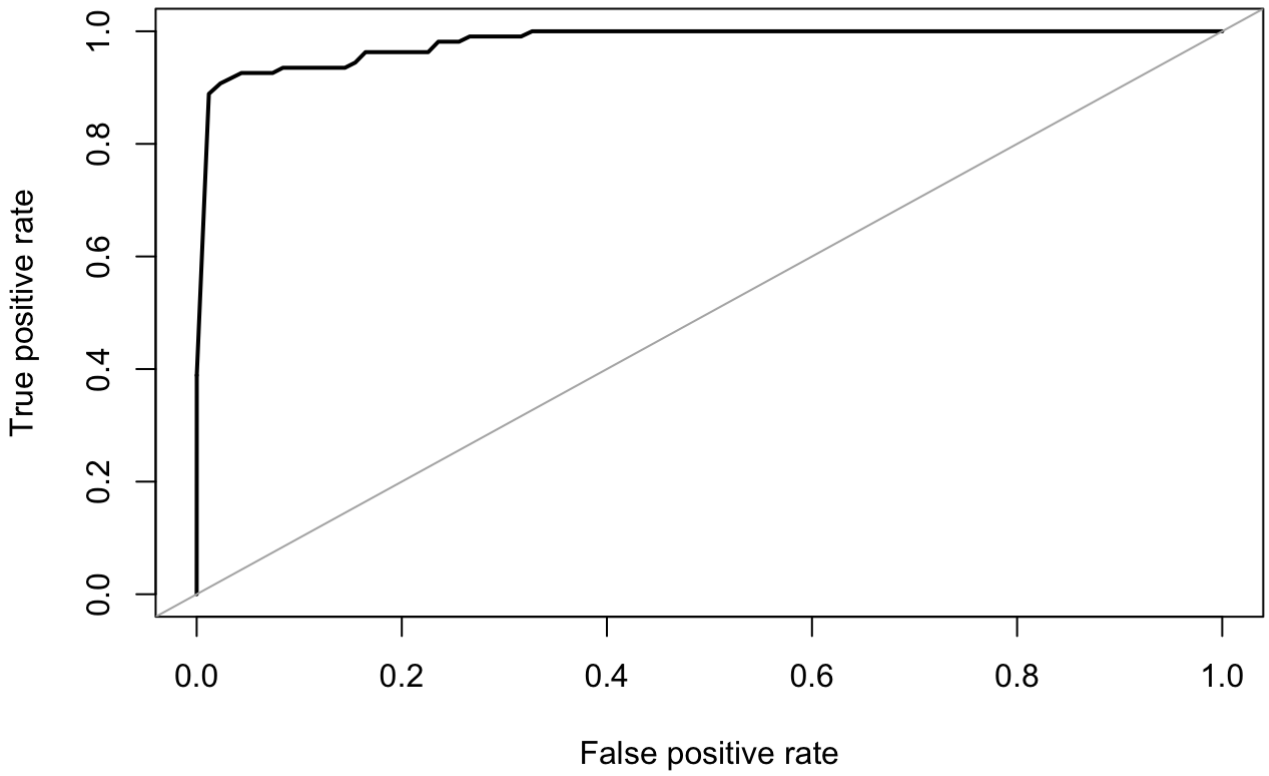


```
## Area under the curve (AUC): 0.986
```

```
#predict on imbalanced test set
pred_imbalanced_down <- predict(down_fit, test) #imbalanced
confusionMatrix(data = as.factor(as.numeric(pred_imbalanced_down >0.5)), reference = as.facto
r(test$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 55065    10
##          1  1788    98
##
##                Accuracy : 0.9684
##                  95% CI : (0.967, 0.9699)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.095
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.907407
##             Specificity : 0.968550
##          Pos Pred Value : 0.051962
##          Neg Pred Value : 0.999818
##              Prevalence : 0.001896
##          Detection Rate : 0.001720
##    Detection Prevalence : 0.033110
##       Balanced Accuracy : 0.937979
##
##        'Positive' Class : 1
##
```

```
roc.curve(test$Class, pred_imbalanced_down, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.981
```

Apply the model on imbalanced train data(original), fit the model to imbalanced and balanced

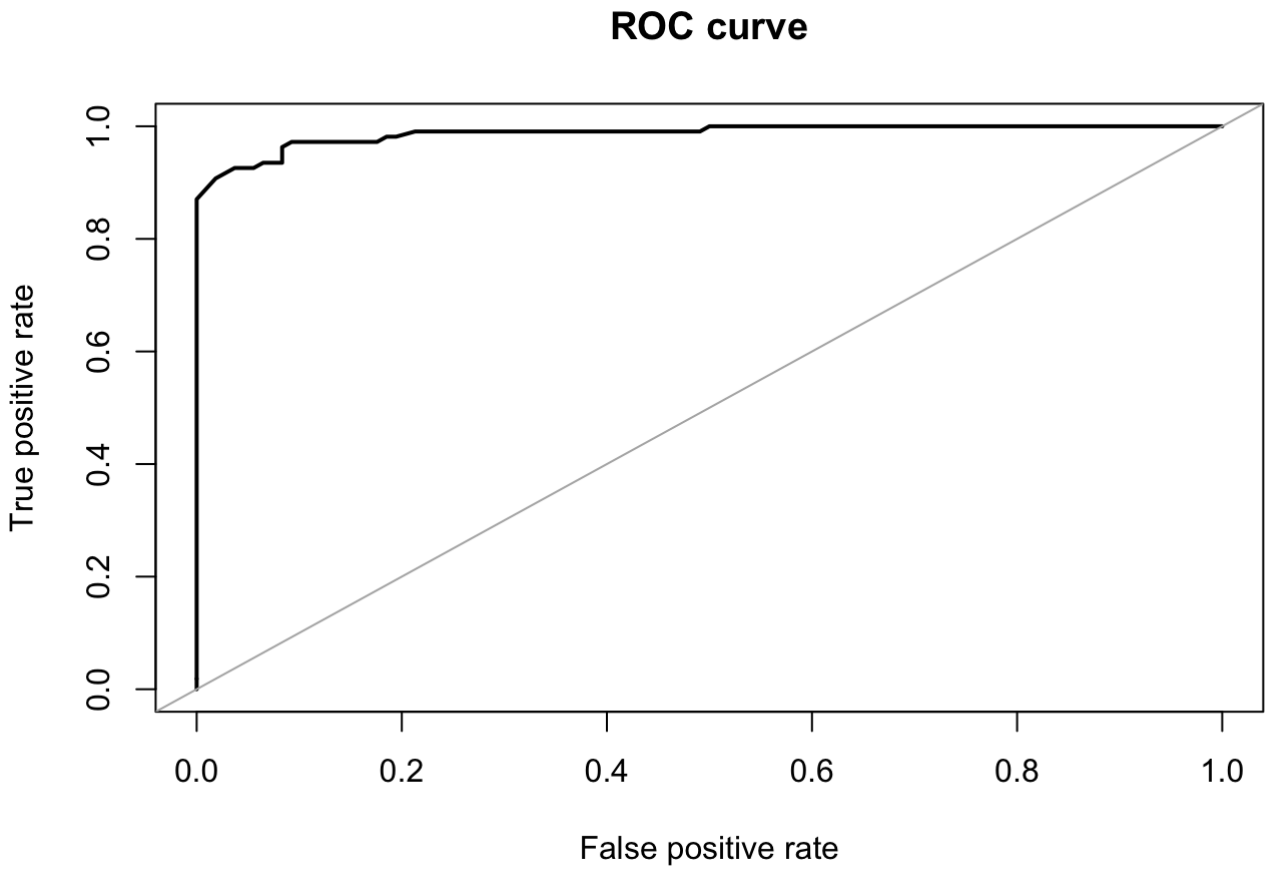
```
org_fit <- glm(Class ~ .,family = "binomial" ,data = train)
summary(org_fit,)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.9904  -0.0287  -0.0194  -0.0129   4.5941
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.7459867  0.1745112 -50.117  < 2e-16 ***
## V1           0.0878636  0.0496028   1.771  0.07650 .
## V2           0.0535863  0.0832303   0.644  0.51968
## V3           0.0408736  0.0529907   0.771  0.44051
## V4           0.6677762  0.0845165   7.901 2.76e-15 ***
## V5           0.1537216  0.0794293   1.935  0.05295 .
## V6          -0.1008230  0.0823870  -1.224  0.22104
## V7          -0.1269826  0.0837522  -1.516  0.12948
## V8          -0.1639058  0.0357064  -4.590 4.42e-06 ***
## V9          -0.2286206  0.1245456  -1.836  0.06641 .
## V10         -0.8353246  0.1097630  -7.610 2.74e-14 ***
## V11         -0.0117536  0.0861907  -0.136  0.89153
## V12          0.0306091  0.0947891   0.323  0.74676
## V13         -0.2460305  0.0911933  -2.698  0.00698 **
## V14         -0.5427667  0.0679758  -7.985 1.41e-15 ***
## V15         -0.0762515  0.0946912  -0.805  0.42067
## V16         -0.2018280  0.1427174  -1.414  0.15731
## V17          0.0101923  0.0775877   0.131  0.89549
## V18         -0.0185827  0.1454898  -0.128  0.89837
## V19          0.1713988  0.1095914   1.564  0.11782
## V20         -0.4470229  0.0995160  -4.492 7.06e-06 ***
## V21          0.3543738  0.0673374   5.263 1.42e-07 ***
## V22          0.5685667  0.1459624   3.895 9.81e-05 ***
## V23         -0.0927912  0.0802150  -1.157  0.24736
## V24          0.1012719  0.1644846   0.616  0.53810
## V25          0.0309728  0.1487510   0.208  0.83506
## V26          0.0111550  0.2124036   0.053  0.95812
## V27         -0.7873522  0.1403283  -5.611 2.01e-08 ***
## V28         -0.2362251  0.1075584  -2.196  0.02807 *
## Amount       0.0011832  0.0006199   1.909  0.05630 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5671.6  on 227845  degrees of freedom
## Residual deviance: 1739.8  on 227816  degrees of freedom
## AIC: 1799.8
##
## Number of Fisher Scoring iterations: 12
```

```
pred_org <- predict(org_fit, downsample.test)

#Evaluate model performance on test set
confusionMatrix(data = as.factor(as.numeric(pred_org >0.5)), reference = as.factor(downsampl
e.test$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 108  43
##          1   0  65
##
##                Accuracy : 0.8009
##                  95% CI : (0.7414, 0.852)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6019
##
##  Mcnemar's Test P-Value : 1.504e-10
##
##             Sensitivity : 0.6019
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7152
##              Prevalence : 0.5000
##          Detection Rate : 0.3009
##    Detection Prevalence : 0.3009
##       Balanced Accuracy : 0.8009
##
##        'Positive' Class : 1
##
```

```
roc.curve(downsample.test$Class, pred_org, plotit=TRUE)
```

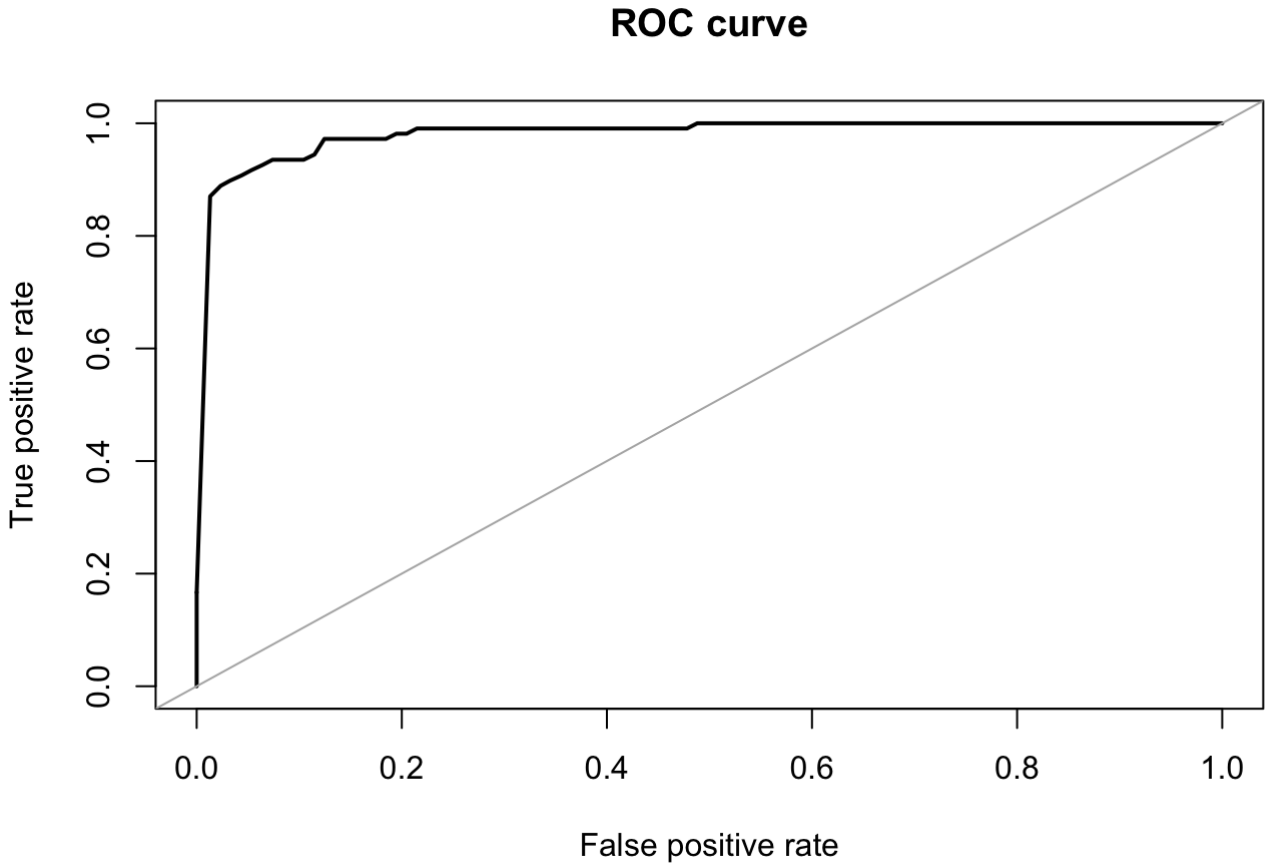


```
## Area under the curve (AUC): 0.987
```

```
pred_imbalanced_org <- predict(org_fit, test)

#Evaluate model performance on test set
confusionMatrix(data = as.factor(as.numeric(pred_imbalanced_org >0.5)), reference = as.factor
(test$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56845    43
##          1     8    65
##
##                Accuracy : 0.9991
##                  95% CI : (0.9988, 0.9993)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 7.392e-10
##
##                   Kappa : 0.7178
##
##  Mcnemar's Test P-Value : 1.927e-06
##
##             Sensitivity : 0.601852
##             Specificity : 0.999859
##          Pos Pred Value : 0.890411
##          Neg Pred Value : 0.999244
##              Prevalence : 0.001896
##          Detection Rate : 0.001141
##    Detection Prevalence : 0.001282
##       Balanced Accuracy : 0.800856
##
##        'Positive' Class : 1
##
```

```
roc.curve(test$Class, pred_imbalanced_org, plotit=TRUE)
```




```
## Area under the curve (AUC): 0.980
```

Decision Tree

Apply 5-folds cross validation to find the best parameter cp for decision tree

```
ctrl <- trainControl(method = "cv", number = 5)
```

Use downsample training set to fit model

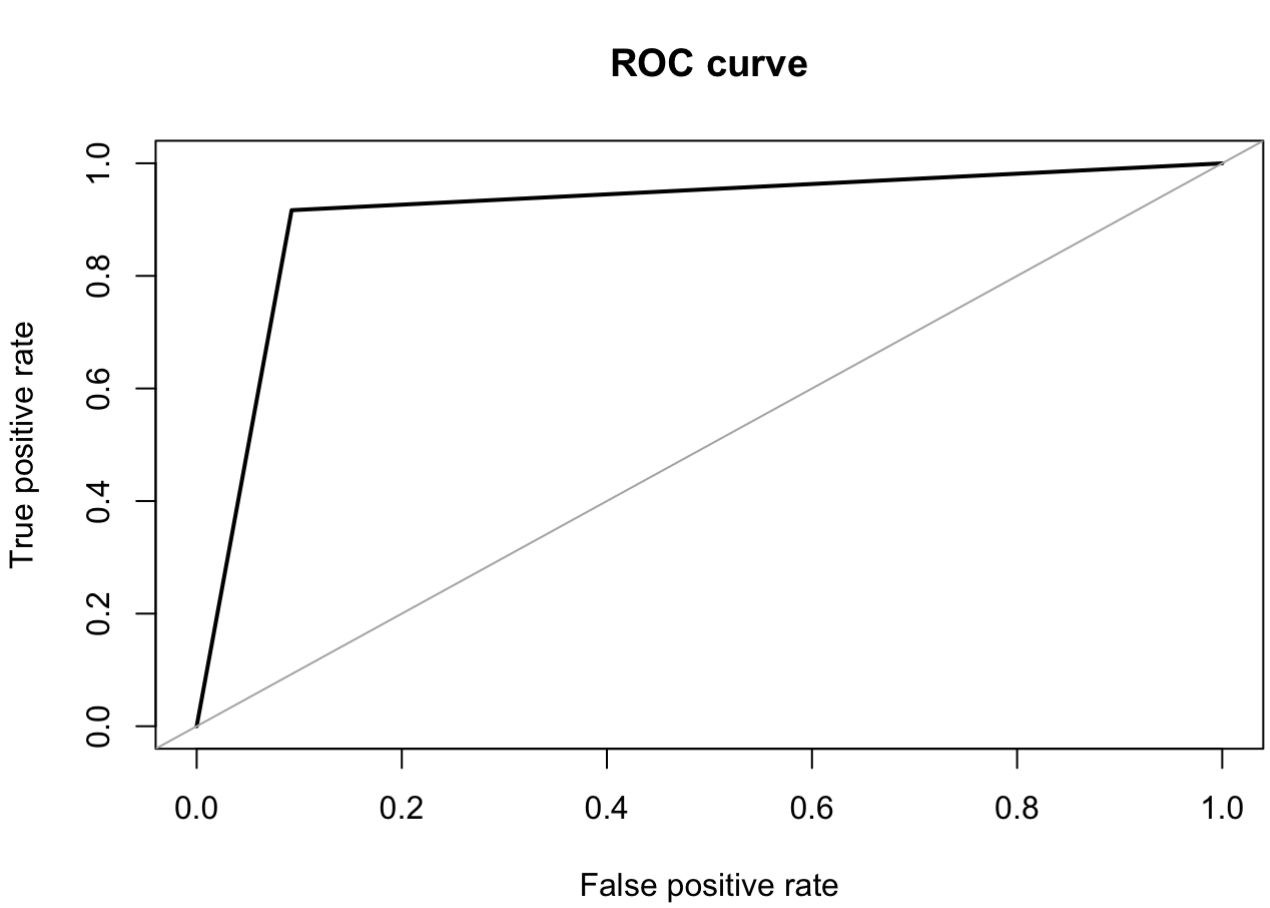
```
dt <- train(Class ~ ., data = downsample.train,
               method = 'rpart',
               trControl = ctrl)
```

Find best cp for decision model which is cp = 0.015 Then evaluate the model using downsample test dataset and output the confusion matrix and ROC curve.

```
pred <- predict(dt, downsample.test)

#performance
confusionMatrix(pred, downsample.test$Class, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 98  9
##          1 10 99
##
##                Accuracy : 0.912
##                  95% CI : (0.866, 0.9462)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8241
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9167
##             Specificity : 0.9074
##          Pos Pred Value : 0.9083
##          Neg Pred Value : 0.9159
##              Prevalence : 0.5000
##          Detection Rate : 0.4583
##    Detection Prevalence : 0.5046
##       Balanced Accuracy : 0.9120
##
##        'Positive' Class : 1
##
```

```
#ROC curve
roc.curve(downsample.test$Class,pred , plotit=TRUE)
```



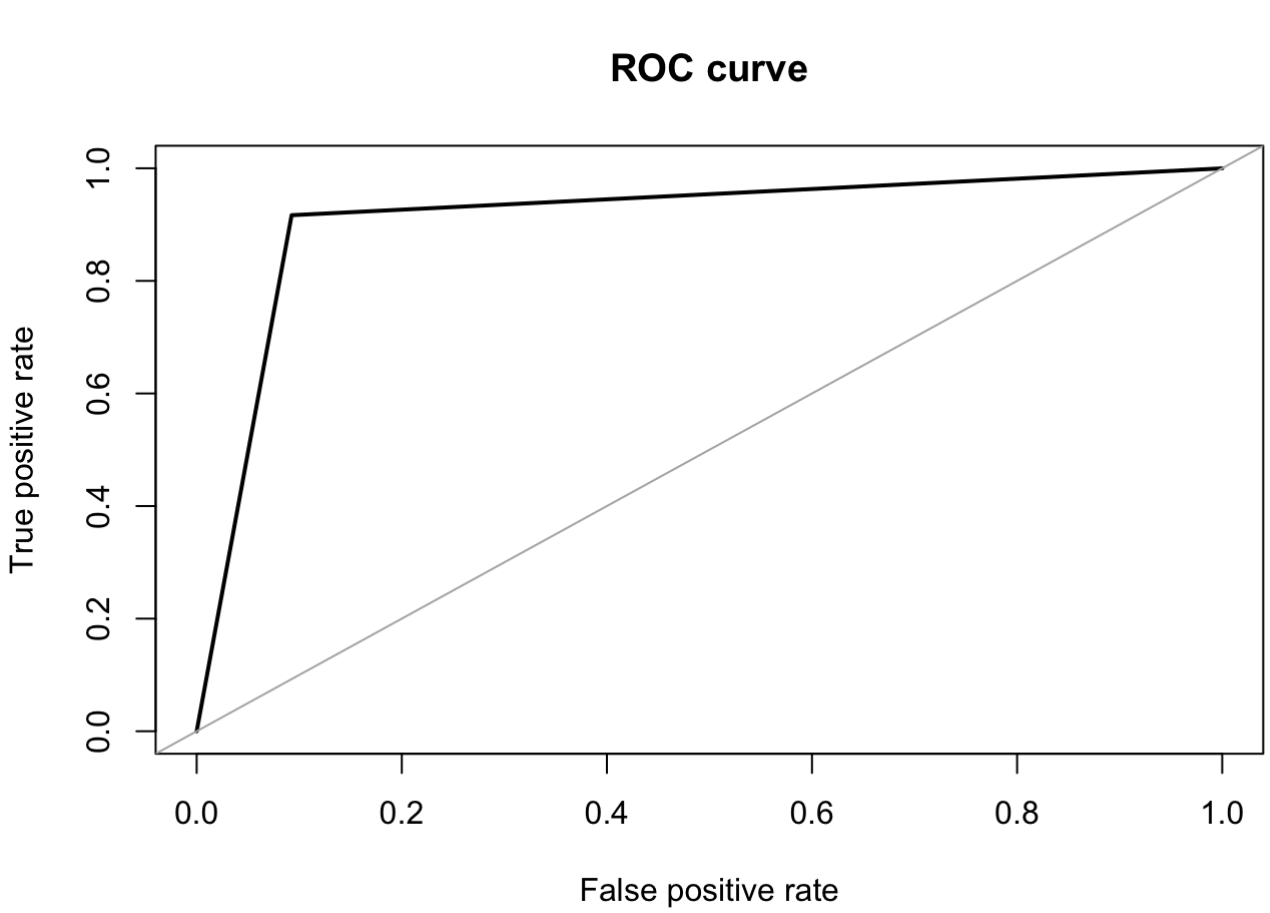

```
## Area under the curve (AUC): 0.912
```

Evaluate the model using imbalanced test dataset and output the confusion matrix and ROC curve.

```
pred.imbalanced <- predict(dt, test)

#performance
confusionMatrix(pred.imbalanced, test$Class, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 51587     9
##          1  5266    99
##
##                Accuracy : 0.9074
##                  95% CI : (0.905, 0.9098)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0326
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.916667
##             Specificity : 0.907375
##          Pos Pred Value : 0.018453
##          Neg Pred Value : 0.999826
##              Prevalence : 0.001896
##          Detection Rate : 0.001738
##    Detection Prevalence : 0.094187
##       Balanced Accuracy : 0.912021
##
##        'Positive' Class : 1
##
```

```
#ROC curve
roc.curve(test$Class, pred.imbalanced, plotit = TRUE)
```




```
## Area under the curve (AUC): 0.912
```

Use imbalanced training set to fit model

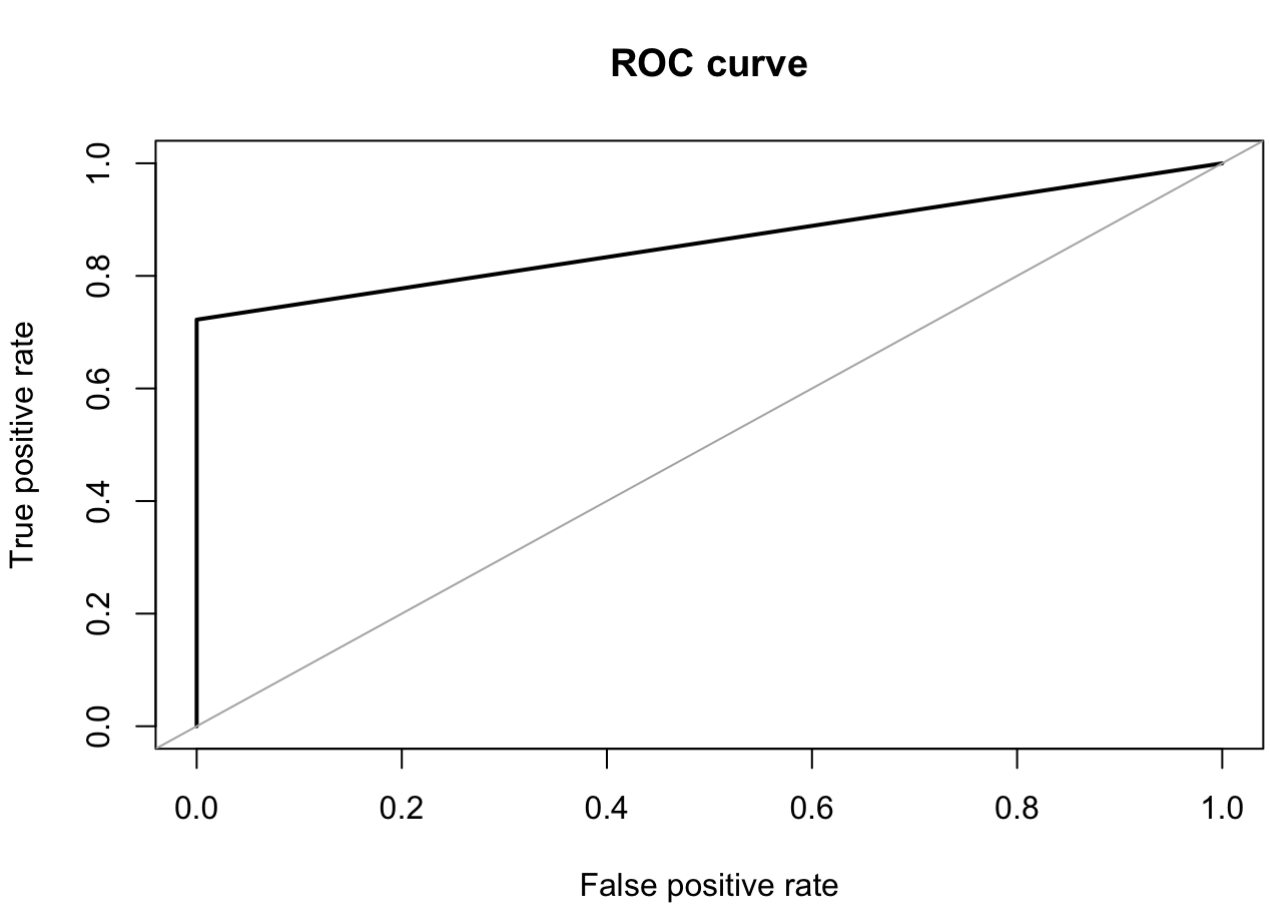
```
dt_imbalanced <- train(Class ~ ., data = train,
                method = 'rpart',
                trControl = ctrl)
```

Evaluate the model using downsample test dataset and output the confusion matrix and ROC curve.

```
pred <- predict(dt_imbalanced, downsample.test)

#performance
confusionMatrix(pred, downsample.test$Class, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 108  30
##          1   0  78
##
##                Accuracy : 0.8611
##                  95% CI : (0.8077, 0.9043)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7222
##
##  Mcnemar's Test P-Value : 1.192e-07
##
##             Sensitivity : 0.7222
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.7826
##              Prevalence : 0.5000
##          Detection Rate : 0.3611
##    Detection Prevalence : 0.3611
##       Balanced Accuracy : 0.8611
##
##        'Positive' Class : 1
##
```

```
#ROC curve
roc.curve(downsample.test$Class, pred, plotit=TRUE)
```



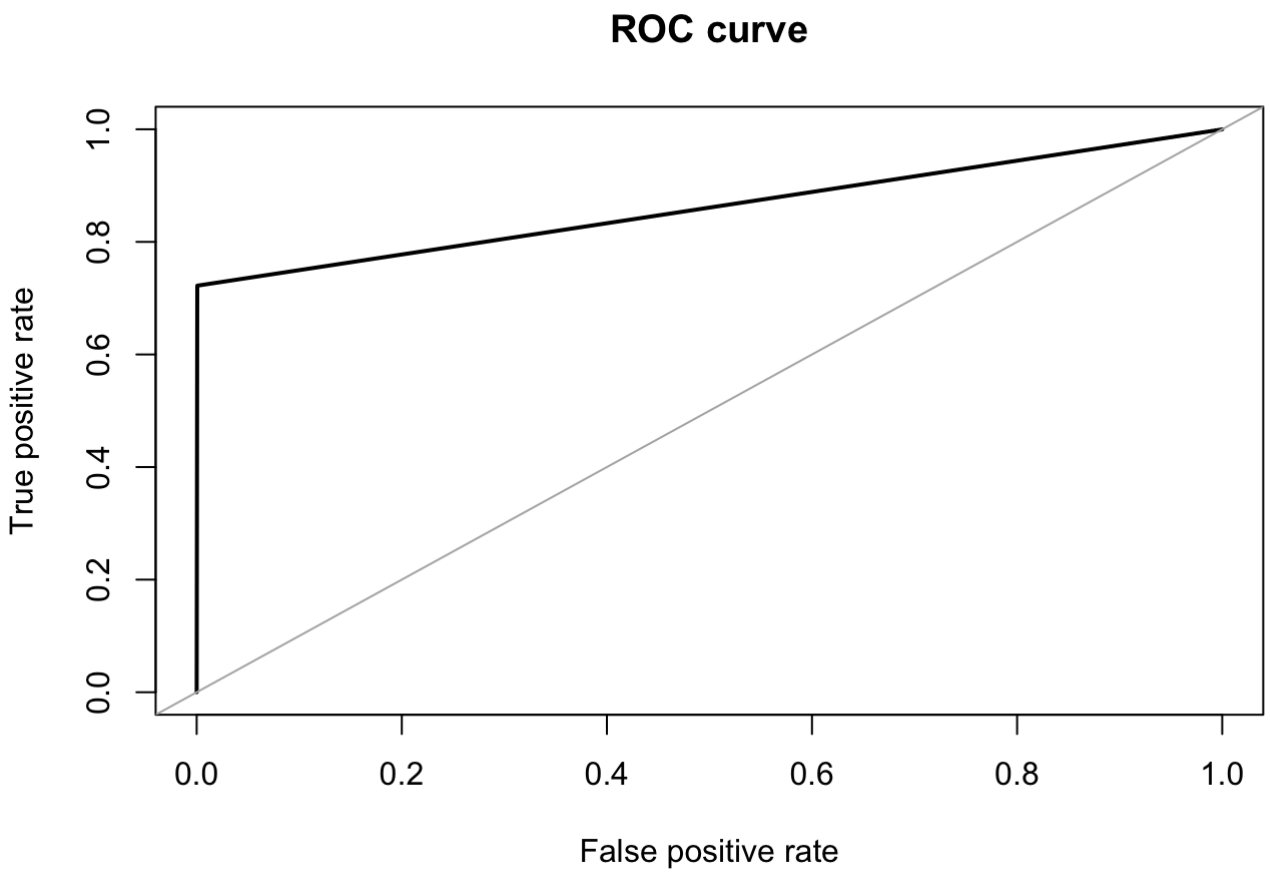

```
## Area under the curve (AUC): 0.861
```

Evaluate the model using imbalanced test dataset and output the confusion matrix and ROC curve.

```
#predict on imbalanced test set
pred.imbalanced <- predict(dt_imbalanced, test)

#performance
confusionMatrix(pred.imbalanced, test$Class, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56817    30
##          1    36    78
##
##                Accuracy : 0.9988
##                  95% CI : (0.9985, 0.9991)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 9.018e-06
##
##                   Kappa : 0.7021
##
##  Mcnemar's Test P-Value : 0.5383
##
##             Sensitivity : 0.722222
##             Specificity : 0.999367
##          Pos Pred Value : 0.684211
##          Neg Pred Value : 0.999472
##              Prevalence : 0.001896
##          Detection Rate : 0.001369
##    Detection Prevalence : 0.002001
##       Balanced Accuracy : 0.860795
##
##        'Positive' Class : 1
##
```

```
#ROC curve
roc.curve(test$Class, pred.imbalanced, plotit = TRUE)
```

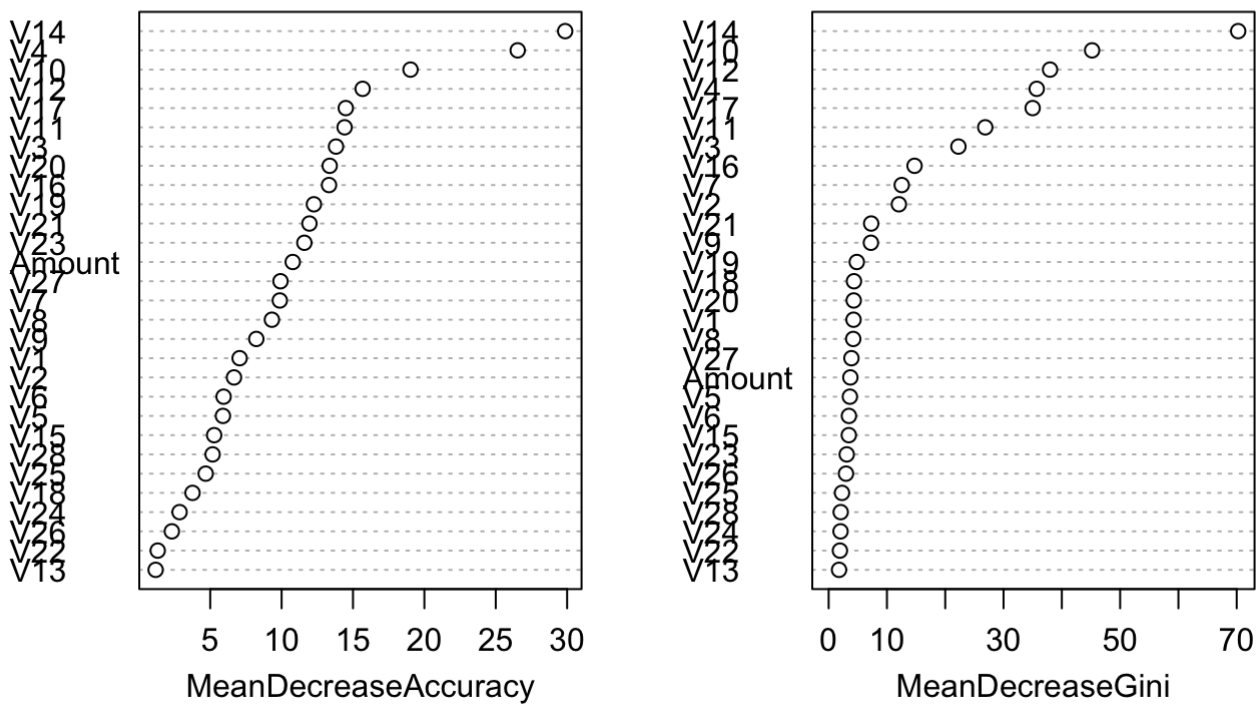



```
## Area under the curve (AUC): 0.861
```

Random Forest

First Fit the random Forest with the downsampled train dataset(balanced) and plot the feature importance graph.

```
# Fit random forest model
fit_rndfor <- randomForest(downsample.train$Class~., data=downsample.train, ntree = 500, impo
rtance = TRUE)

varImpPlot(fit_rndfor)
```



Then we make predictions by using the downsampled test set and the original test set and compute their confusion matrix.

```
#make predictions
pd.test <- predict(fit_rndfor, downsample.test[,-ncol(downsample.test)])
table(observed = downsample.test[,ncol(downsample.test)], predicted = pd.test)
```

```
##          predicted
## observed   0   1
##        0 105   3
##        1  10  98
```

```
confusionMatrix(pd.test, downsample.test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 105  10
##          1   3  98
##
##                Accuracy : 0.9398
##                  95% CI : (0.8993, 0.9676)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8796
##
##  Mcnemar's Test P-Value : 0.09609
##
##             Sensitivity : 0.9074
##             Specificity : 0.9722
##          Pos Pred Value : 0.9703
##          Neg Pred Value : 0.9130
##              Prevalence : 0.5000
##          Detection Rate : 0.4537
##    Detection Prevalence : 0.4676
##       Balanced Accuracy : 0.9398
##
##        'Positive' Class : 1
##
```

```
pd.test.original <- predict(fit_rndfor, test[,-ncol(test)])
table(observed = test[,ncol(test)], predicted = pd.test.original)
```

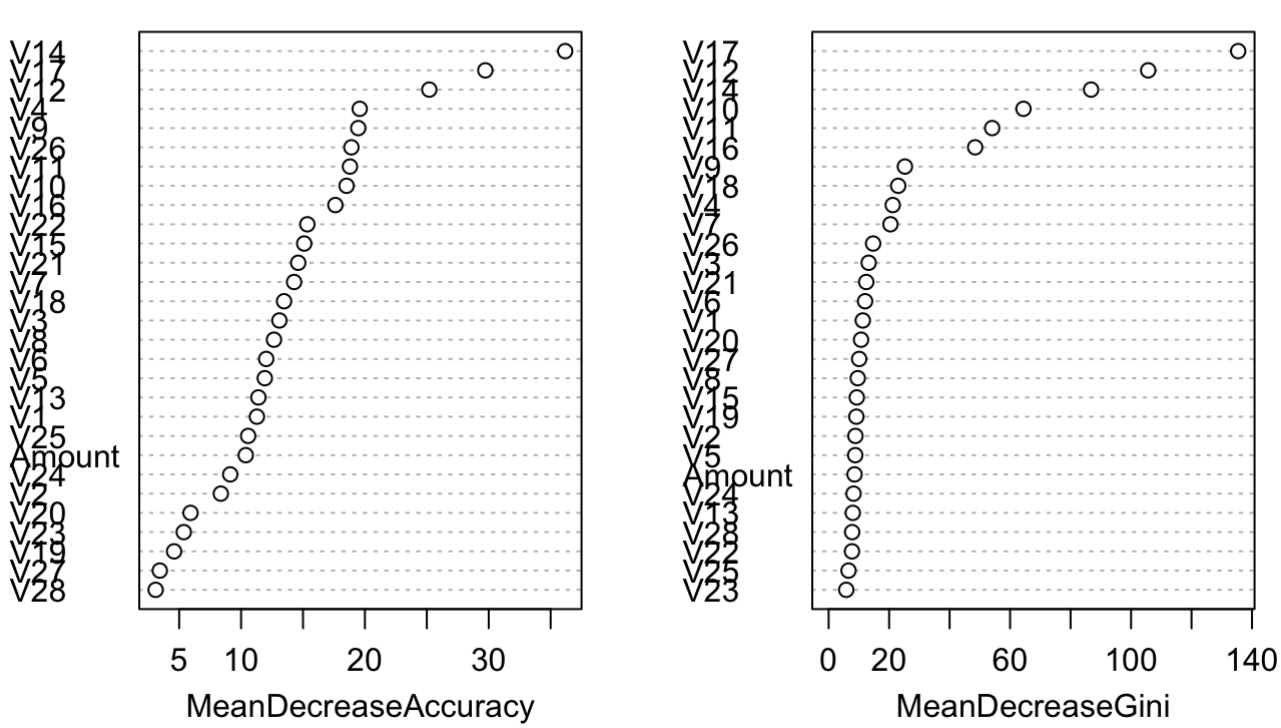
```
##         predicted
## observed     0     1
##        0 54479  2374
##        1    10    98
```

```
confusionMatrix(pd.test.original, test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 54479    10
##          1  2374    98
##
##                Accuracy : 0.9581
##                  95% CI : (0.9565, 0.9598)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0726
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.907407
##             Specificity : 0.958243
##          Pos Pred Value : 0.039644
##          Neg Pred Value : 0.999816
##              Prevalence : 0.001896
##          Detection Rate : 0.001720
##    Detection Prevalence : 0.043398
##       Balanced Accuracy : 0.932825
##
##        'Positive' Class : 1
##
```

We want to see how the model works with the imbalance dataset which is the original train set.

```
#Random Forest fit with original dataset
fit_rndfor_origin <- randomForest(train$Class~., data=train, ntree = 500, importance = TRUE)
varImpPlot(fit_rndfor_origin)
```



Following with making predictions with both downsampled test set and the original test set and compute the confusion matrix.

```
#make predictions
pd.test.original2 <- predict(fit_rndfor_origin, test[,-ncol(test)])
table(observed = test[,ncol(test)], predicted = pd.test.original2)
```

```
##          predicted
## observed     0     1
##        0 56846     7
##        1    23    85
```

```
pd.test.original3 <- predict(fit_rndfor_origin, downsample.test[,-ncol(downsample.test)])
table(observed = downsample.test[,ncol(downsample.test)], predicted = pd.test.original3)
```

```
##          predicted
## observed   0   1
##        0 108   0
##        1  23  85
```

```
confusionMatrix(pd.test.original2, test$Class, positive = "1")
```

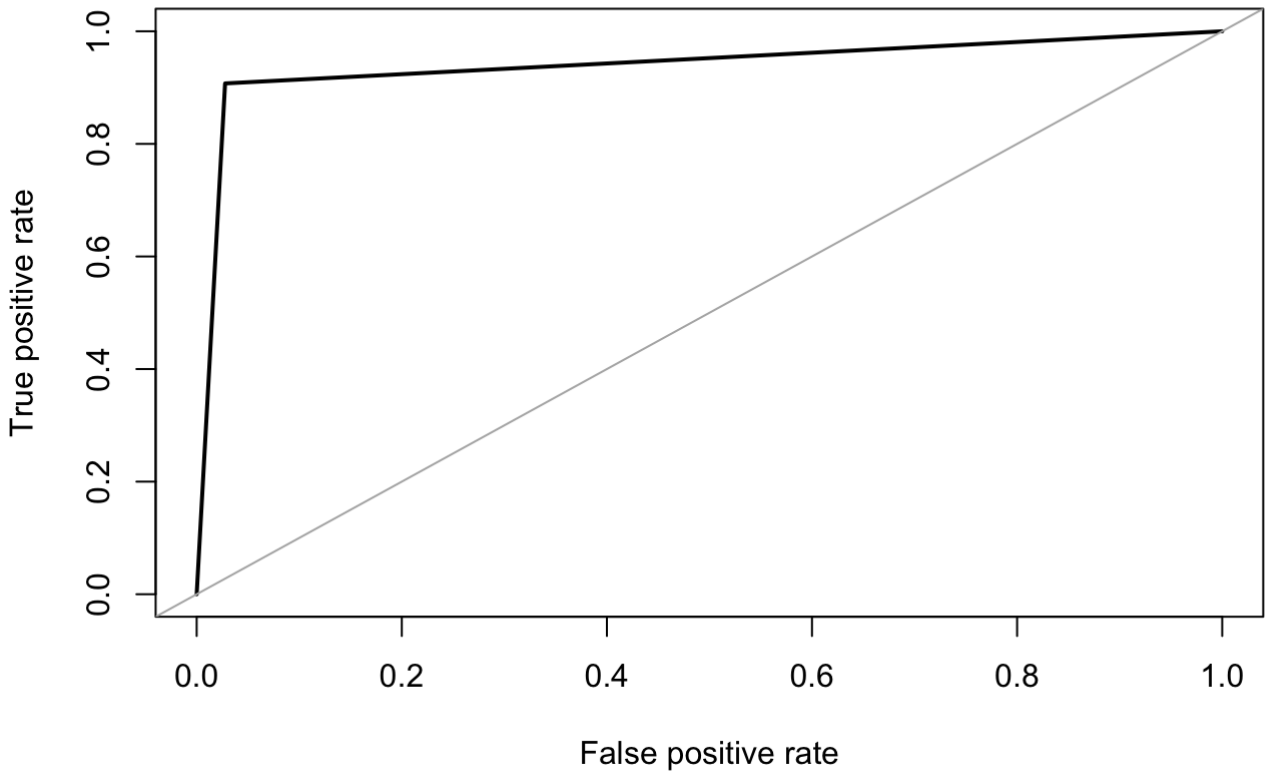
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56846    23
##          1     7    85
##
##                Accuracy : 0.9995
##                  95% CI : (0.9992, 0.9996)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8497
##
##  Mcnemar's Test P-Value : 0.00617
##
##             Sensitivity : 0.787037
##             Specificity : 0.999877
##          Pos Pred Value : 0.923913
##          Neg Pred Value : 0.999596
##              Prevalence : 0.001896
##          Detection Rate : 0.001492
##    Detection Prevalence : 0.001615
##       Balanced Accuracy : 0.893457
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(pd.test.original3, downsample.test$Class, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 108  23
##          1   0  85
##
##                Accuracy : 0.8935
##                  95% CI : (0.8445, 0.9313)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.787
##
##  Mcnemar's Test P-Value : 4.49e-06
##
##             Sensitivity : 0.7870
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8244
##              Prevalence : 0.5000
##          Detection Rate : 0.3935
##    Detection Prevalence : 0.3935
##       Balanced Accuracy : 0.8935
##
##        'Positive' Class : 1
##
```

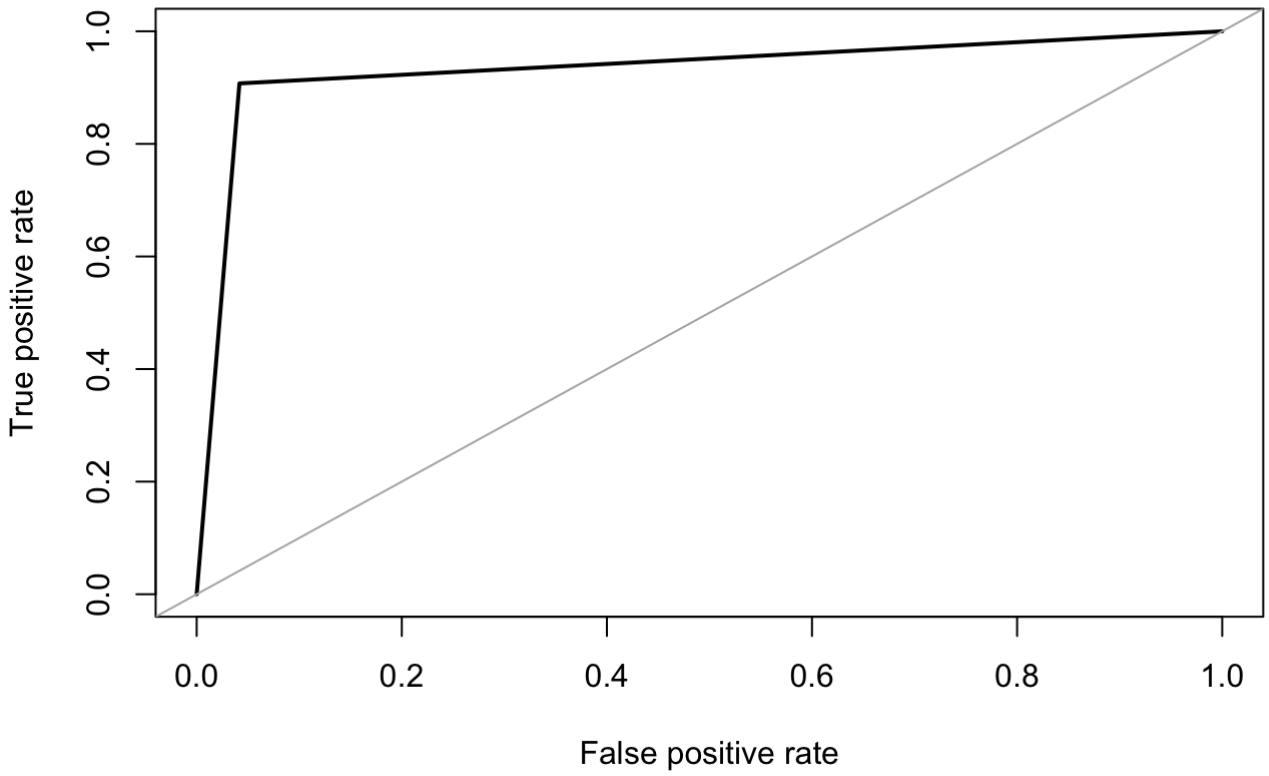
To better inspect the model accuracy, we also calculate the roc and auc for four models.

```
#ROC curve and AUC
roc.curve(downsample.test$Class, pd.test, plotit=TRUE)
```



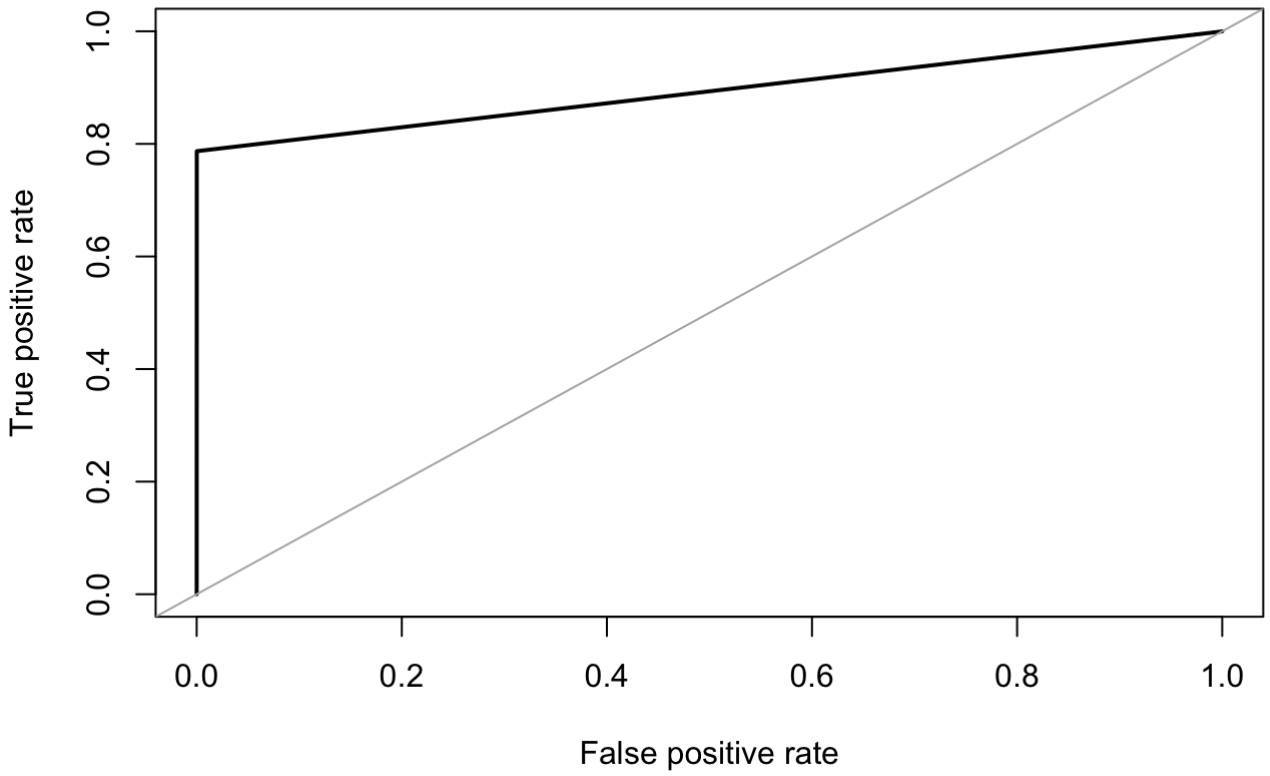

```
## Area under the curve (AUC): 0.940
```

```
roc.curve(test$Class, pd.test.original, plotit=TRUE)
```



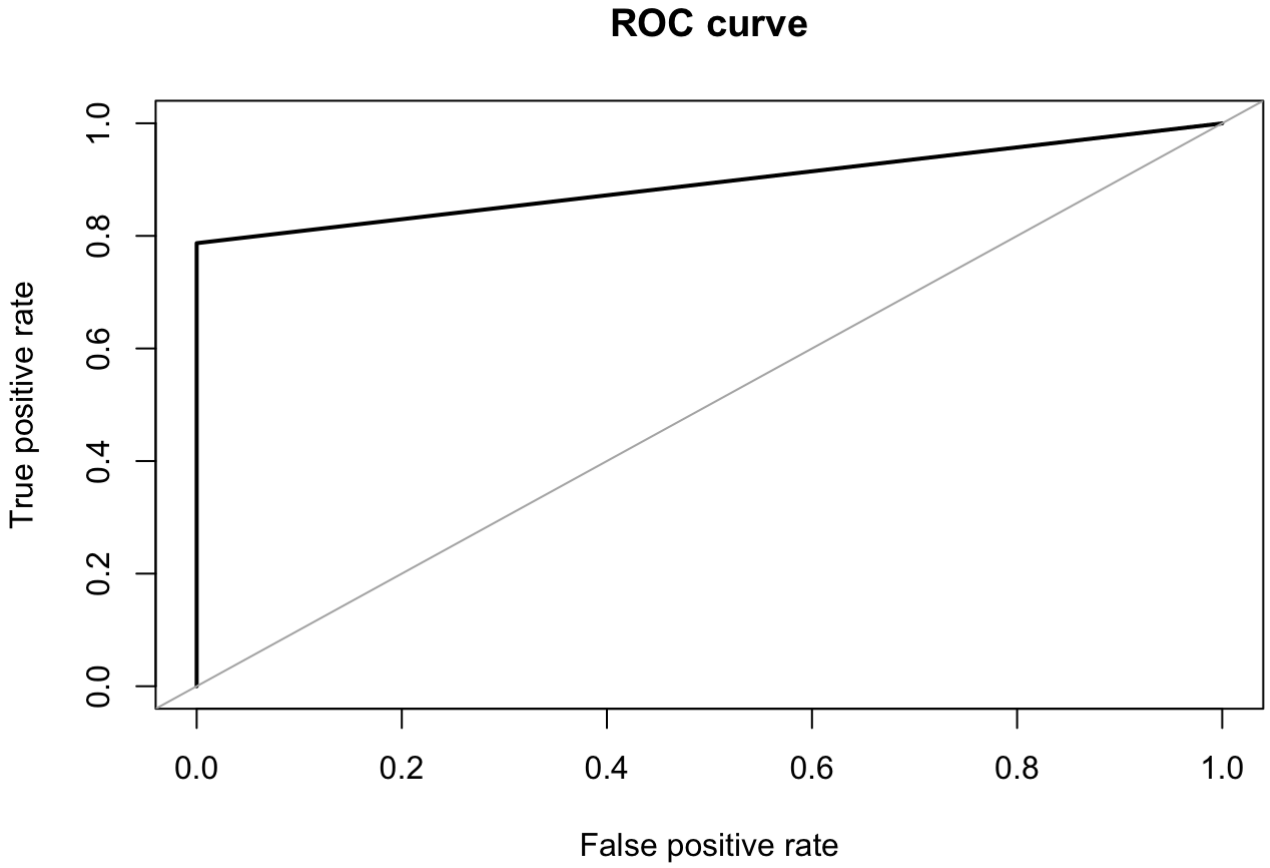

```
## Area under the curve (AUC): 0.933
```

```
roc.curve(test$Class, pd.test.original2, plotit=TRUE)
```




```
## Area under the curve (AUC): 0.893
```

```
roc.curve(downsample.test$Class, pd.test.original3, plotit=TRUE)
```




```
## Area under the curve (AUC): 0.894
```

XGBoost

In order to find the best parameters to fit the XGBoost model, we set randomly chosen values to the parameters and ran k-fold cross-validation. Each time, a set of parameters that maximized AUC was returned. We then created a loop to repeat this process 10 times. We selected the best set of parameters from the 10 iterations and used it to build the XGBoost model.

```
# Cross-validation (downsample.train)
dtrain = data.matrix(downsample.train[,1:29])
best_param = list()
best_seednumber = 1234
best_auc = Inf
best_auc_index = 0

for (iter in 1:10) {
  param <- list(objective = "binary:logistic", eval_metric = "auc")
  cv.nround = 1000
  cv.nfold = 5
  seed.number = sample.int(10000, 1)
  set.seed(seed.number)
  mdcv <- xgb.cv(data=dtrain, params = param,
                 nfold=cv.nfold, nrounds=cv.nround, verbose=0,
                 early_stopping_rounds=8, maximize=TRUE,
                 label=as.numeric(downsample.train$Class)-1)

  min_auc = min(mdcv$evaluation_log[, test_auc_mean])
  min_auc_index = which.min(mdcv$evaluation_log[, test_auc_mean])

  if (min_auc < best_auc) {
    best_auc = min_auc
    best_auc_index = min_auc_index
    best_seednumber = seed.number
    best_param = param
  }
}

nround = best_auc
set.seed(best_seednumber)
```

We first trained a model on our downsampled training set.

```
# Fit XGBoost model on downsampled training set
xgb = xgboost(data = dtrain,
              params = best_param,
              nround = nround,
              label=as.numeric(downsample.train$Class)-1)
```

```
## [1]  train-auc:0.984656
```

```
# Feature importance
xgb.importance(model=xgb)
```

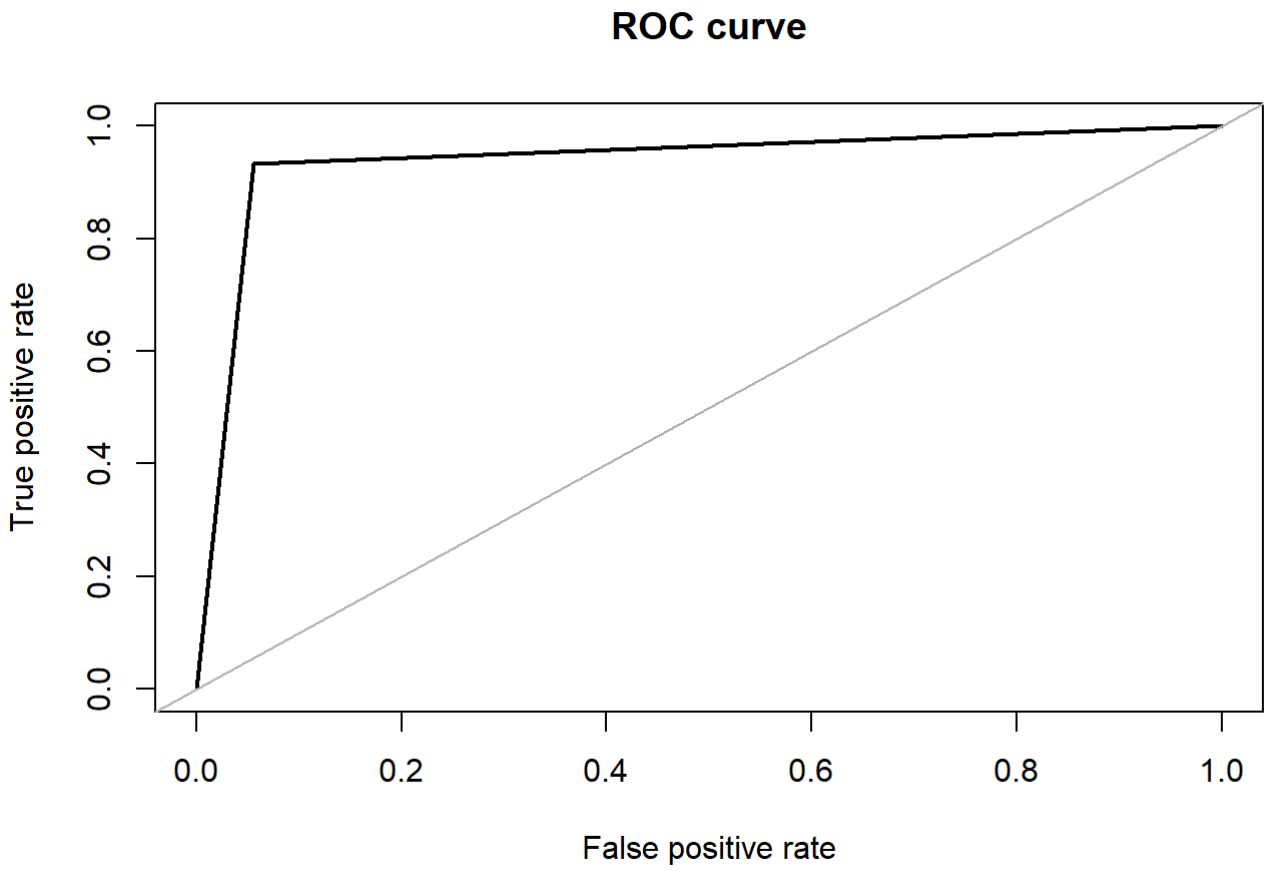
```
##     Feature        Gain       Cover  Frequency
##  1:     V14 0.868850198 0.430487137 0.27777778
##  2:      V4 0.059356455 0.128899836 0.11111111
##  3:      V8 0.020893995 0.013957307 0.05555556
##  4:      V7 0.015908938 0.206075534 0.16666667
##  5:     V13 0.007589904 0.004926108 0.05555556
##  6:     V24 0.006619848 0.004378763 0.05555556
##  7:     V19 0.006491439 0.005747126 0.05555556
##  8:      V6 0.004309146 0.002736727 0.05555556
##  9:     V21 0.004297199 0.104542967 0.05555556
## 10:     V23 0.003756931 0.093869732 0.05555556
## 11:     V18 0.001925948 0.004378763 0.05555556
```

Then, we predicted on our downsampled test set and the imbalanced test set.

```
# Apply XGBoost model on downsampled test set
predictions = predict(xgb, data.matrix(downsample.test[,1:29]))
# Transform predictions to binary results
predictions = as.numeric(predictions>0.5)
predictions = as.factor(predictions)
# Confusion matrix
cm1 = confusionMatrix(predictions, downsample.test$Class
                      ,dnn=c("Prediction", "Reference")
                      ,positive='1')
print(cm1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 84  6
##          1  5 83
##
##                Accuracy : 0.9382
##                  95% CI : (0.8921, 0.9688)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8764
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9326
##             Specificity : 0.9438
##          Pos Pred Value : 0.9432
##          Neg Pred Value : 0.9333
##              Prevalence : 0.5000
##          Detection Rate : 0.4663
##    Detection Prevalence : 0.4944
##       Balanced Accuracy : 0.9382
##
##        'Positive' Class : 1
##
```

```
# Plot ROC curve
roc1 = roc.curve(downsample.test$Class, as.factor(predictions), plotit = TRUE)
```



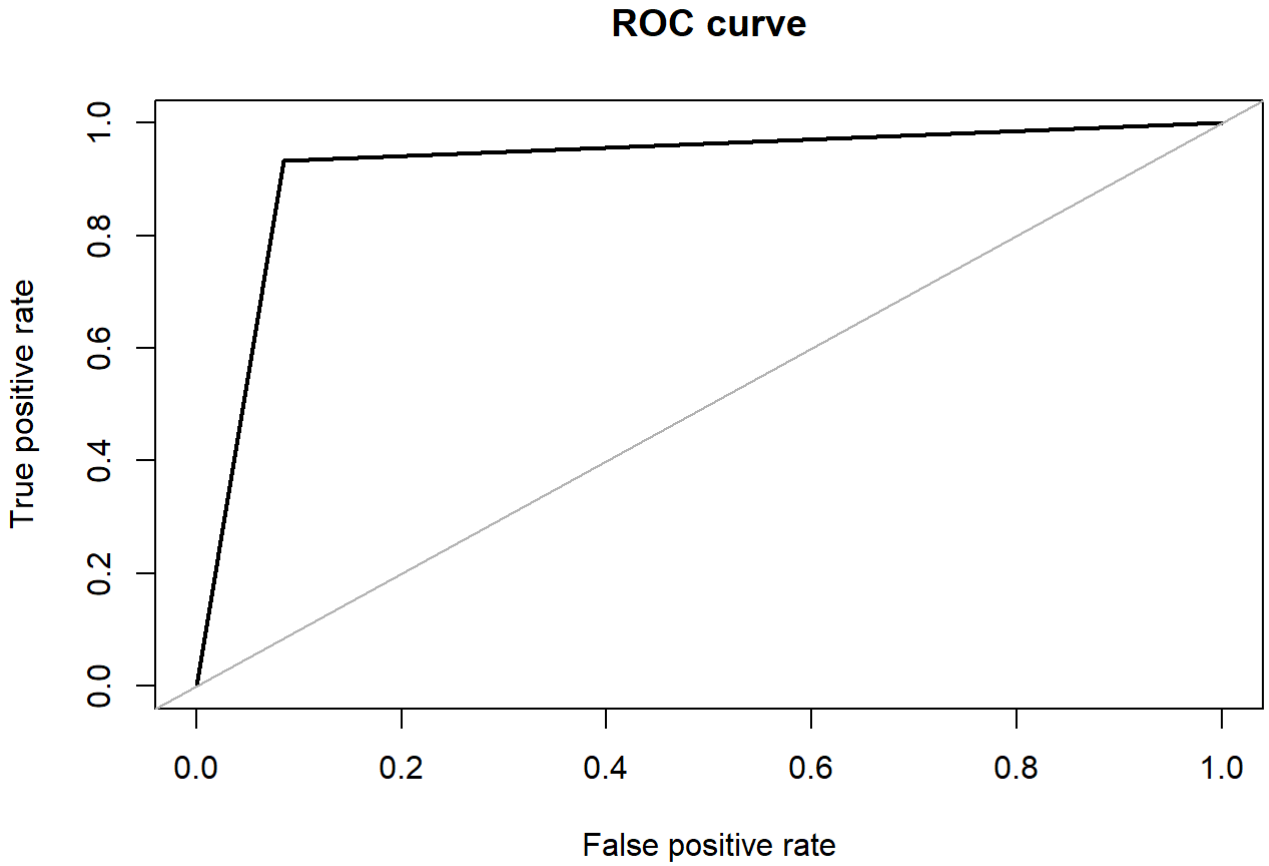

```
print(paste("Area under the curve (AUC):", round(roc1$auc, digits=3)))
```

```
## [1] "Area under the curve (AUC): 0.938"
```

```
# Apply XGBoost model on imbalanced test set
predictions2 = predict(xgb, data.matrix(test[,1:29]))
predictions2 = as.numeric(predictions2>0.5)
predictions2 = as.factor(predictions2)
# Confusion matrix
cm2 = confusionMatrix(predictions2, test$Class
                      ,dnn=c("Prediction", "Reference")
                      ,positive='1')
print(cm2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 52054     6
##          1  4818    83
##
##                Accuracy : 0.9153
##                  95% CI : (0.913, 0.9176)
##     No Information Rate : 0.9984
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0303
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.932584
##             Specificity : 0.915283
##          Pos Pred Value : 0.016935
##          Neg Pred Value : 0.999885
##              Prevalence : 0.001562
##          Detection Rate : 0.001457
##    Detection Prevalence : 0.086041
##       Balanced Accuracy : 0.923934
##
##        'Positive' Class : 1
##
```

```
# Plot ROC curve
roc2 = roc.curve(test$Class, as.factor(predictions2), plotit = TRUE)
```




```
print(paste("Area under the curve (AUC):", round(roc2$auc, digits=3)))
```

```
## [1] "Area under the curve (AUC): 0.924"
```

Next, we ran k-fold cross-validation to find the optimal parameters like before and used them to train our second model on the imbalanced training set.

```
# Cross-validation (imbalanced training set)
dtrain2 = data.matrix(train[,1:29])
best_param2 = list()
best_seednumber2 = 1234
best_auc2 = Inf
best_auc_index2 = 0

for (iter in 1:10) {
  param <- list(objective = "binary:logistic", eval_metric = "auc")
  cv.nround = 1000
  cv.nfold = 5
  seed.number = sample.int(10000, 1)
  set.seed(seed.number)
  mdcv <- xgb.cv(data=dtrain2, params = param,
                 nfold=cv.nfold, nrounds=cv.nround, verbose=0,
                 early_stopping_rounds=8, maximize=TRUE,
                 label=as.numeric(train$Class)-1)

  min_auc = min(mdcv$evaluation_log[, test_auc_mean])
  min_auc_index = which.min(mdcv$evaluation_log[, test_auc_mean])

  if (min_auc < best_auc2) {
    best_auc2 = min_auc
    best_auc_index2 = min_auc_index
    best_seednumber2 = seed.number
    best_param2 = param
  }
}

nround2 = best_auc2
set.seed(best_seednumber2)

# Fit XGBoost model on imbalanced training set
xgb2 = xgboost(data = dtrain2,
               params = best_param2,
               nround = nround2,
               label=as.numeric(train$Class)-1)
```

```
## [1]  train-auc:0.908057
```

```
# Feature importance
xgb.importance(model=xgb2)
```

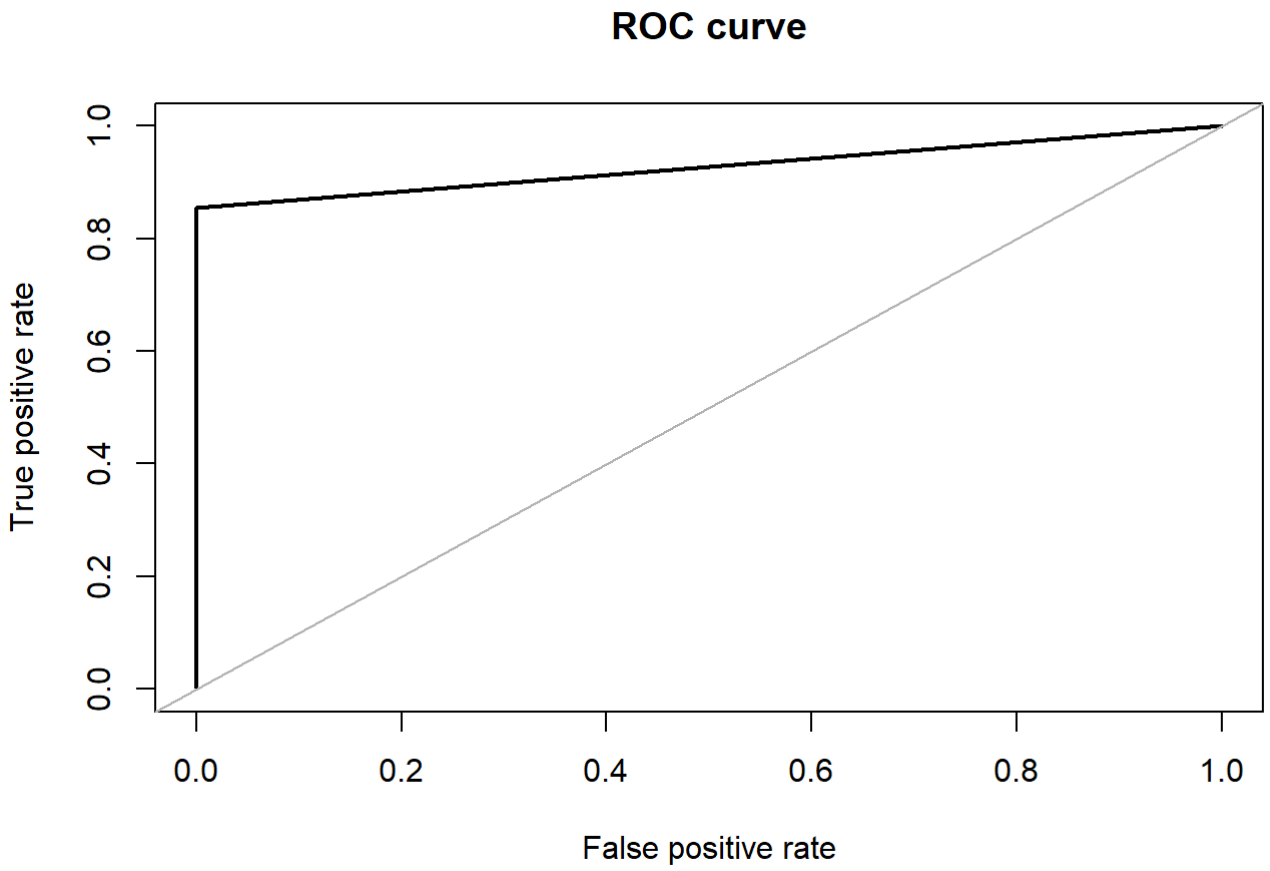
```
##     Feature        Gain        Cover Frequency
##  1:     V17 0.696389669 2.497509e-01      0.05
##  2:     V14 0.119711008 4.992135e-01      0.20
##  3:     V12 0.048647158 2.494955e-01      0.10
##  4:     V10 0.040337834 2.192278e-04      0.05
##  5:     V15 0.025840458 3.266494e-04      0.10
##  6:     V16 0.016212419 4.713398e-05      0.05
##  7:     V27 0.015119489 3.310340e-04      0.05
##  8:     V20 0.009896555 1.940166e-04      0.10
##  9:      V3 0.009069054 3.617259e-05      0.05
## 10:      V4 0.008369918 5.590309e-05      0.10
## 11:      V1 0.005427167 2.882846e-04      0.05
## 12:      V2 0.002814885 1.205753e-05      0.05
## 13:      V7 0.002164387 2.959575e-05      0.05
```

Again, we predicted on our downsampled test set and the imbalanced test set.

```
# Apply XGBoost model on downsampled test set
pred = predict(xgb2, data.matrix(downsample.test[,1:29]))
pred = as.numeric(pred>0.5)
pred = as.factor(pred)
# Confusion matrix
cm3 = confusionMatrix(pred, downsample.test$Class
                      ,dnn=c("Prediction", "Reference")
                      ,positive='1')
print(cm3)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 89 13
##          1  0 76
##
##                Accuracy : 0.927
##                  95% CI : (0.8783, 0.9605)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8539
##
##  Mcnemar's Test P-Value : 0.0008741
##
##             Sensitivity : 0.8539
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8725
##              Prevalence : 0.5000
##          Detection Rate : 0.4270
##    Detection Prevalence : 0.4270
##       Balanced Accuracy : 0.9270
##
##        'Positive' Class : 1
##
```

```
# Plot ROC curve
roc3 = roc.curve(downsample.test$Class, as.factor(pred), plotit = TRUE)
```



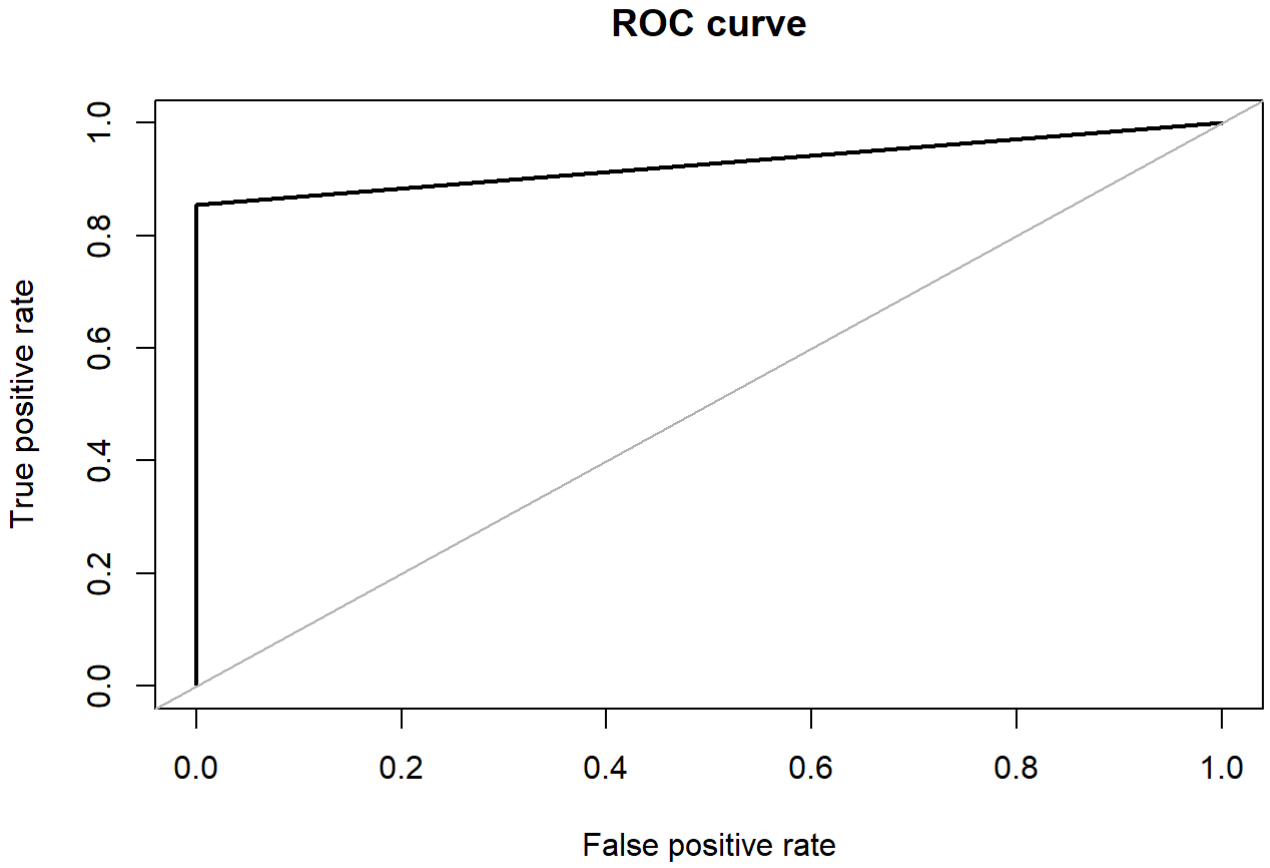

```
print(paste("Area under the curve (AUC):", round(roc3$auc, digits=3)))
```

```
## [1] "Area under the curve (AUC): 0.927"
```

```
# Apply XGBoost model on imbalanced test set
pred2 = predict(xgb2, data.matrix(test[,1:29]))
pred2 = as.numeric(pred2>0.5)
pred2 = as.factor(pred2)
# Confusion matrix
cm4 = confusionMatrix(pred2, test$Class
                      ,dnn=c("Prediction", "Reference")
                      ,positive='1')
print(cm4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 56865    13
##          1     7    76
##
##                Accuracy : 0.9996
##                  95% CI : (0.9995, 0.9998)
##     No Information Rate : 0.9984
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8835
##
##  Mcnemar's Test P-Value : 0.2636
##
##             Sensitivity : 0.853933
##             Specificity : 0.999877
##          Pos Pred Value : 0.915663
##          Neg Pred Value : 0.999771
##              Prevalence : 0.001562
##          Detection Rate : 0.001334
##    Detection Prevalence : 0.001457
##       Balanced Accuracy : 0.926905
##
##        'Positive' Class : 1
##
```

```
# Plot ROC curve
roc4 = roc.curve(test$Class, as.factor(pred2), plotit = TRUE)
```




```
print(paste("Area under the curve (AUC):", round(roc4$auc, digits=3)))
```

```
## [1] "Area under the curve (AUC): 0.927"
```

```
# Model comparison
sets = list("1" = c("downsampled training set", "downsampled test set.")
    ,"2" = c("downsampled training set", "imbalanced test set.")
    ,"3" = c("imbalanced training set", "downsampled test set.")
    ,"4" = c("imbalanced training set", "imbalanced test set."))

i = which.max(c(round(roc1$auc, digits=3)
           ,round(roc2$auc, digits=3)
           ,round(roc3$auc, digits=3)
           ,round(roc4$auc, digits=3)))

cat("Downsampled Training & Downsampled Test"
   ,paste("Sensitivity:"
          , cm1$byClass["Sensitivity"]
          ,"Specificity:"
          , cm1$byClass["Specificity"]
          , " AUC:"
          , round(roc1$auc, digits=3))
   , ""
   ,"Downsampled Training & Imbalanced Test"
   ,paste("Sensitivity:"
          , cm2$byClass["Sensitivity"]
          ,"Specificity:"
          , cm2$byClass["Specificity"]
          , " AUC:"
          , round(roc2$auc, digits=3))
   ,""
   ,"Imbalanced Training & Downsampled Test"
   ,paste("Sensitivity:"
          , cm3$byClass["Sensitivity"]
          ,"Specificity:"
          , cm3$byClass["Specificity"]
          , " AUC:"
          , round(roc3$auc, digits=3))
   ,""
   ,"Imbalanced Training & Imbalanced Test"
   ,paste("Sensitivity:"
          , cm4$byClass["Sensitivity"]
          ,"Specificity:"
          , cm4$byClass["Specificity"]
          , " AUC:"
          , round(roc4$auc, digits=3))
   ,""
   ,paste("As shown above, the model trained on the",
          sets[[i]][1],
          "produced the highest AUC when it was used to predict on the",
          sets[[i]][2])
   ,sep = '\n')
```

```
## Downsampled Training & Downsampled Test
## Sensitivity: 0.932584269662921 Specificity: 0.943820224719101  AUC: 0.938
##
## Downsampled Training & Imbalanced Test
## Sensitivity: 0.932584269662921 Specificity: 0.915283443522296  AUC: 0.924
##
## Imbalanced Training & Downsampled Test
## Sensitivity: 0.853932584269663 Specificity: 1  AUC: 0.927
##
## Imbalanced Training & Imbalanced Test
## Sensitivity: 0.853932584269663 Specificity: 0.999876916584611  AUC: 0.927
##
## As shown above, the model trained on the downsampled training set produced the highest AUC
when it was used to predict on the downsampled test set.
```

Isolation Forest

As an additional bonus, we wanted to try Isolation Forest. This is an unsupervised model that was developed specifically to detect anomalies.

```
# Copy new data as to not disturb other models
iforest_train <- copy(train)
iforest_test <- copy(test)
```

```
# initiate an isolation forest
iso <- isolationForest$new(sample_size = length(iforest_train))
# fit for data
iso$fit(iforest_train)
```

```
## INFO  [17:52:03.620] dataset has duplicated rows
## INFO  [17:52:03.669] Building Isolation Forest ...
## INFO  [17:52:07.685] done
## INFO  [17:52:07.685] Computing depth of terminal nodes ...
## INFO  [17:52:08.114] done
## INFO  [17:52:18.106] Completed growing isolation forest
```

Next, we obtain the anomally scores. According to the documentation of this package, scores that are closer to 1 are likely outliers, while if all the scores hover around 0.5, then there is a low likelihood of outliers.

With this in mind, we set the threshold to 0.6.

```
iforest_scores_train = iso$predict(iforest_train)
iforest_scores_train[order(anomaly_score, decreasing = TRUE)]
```

```
##             id average_depth anomaly_score
##      1: 219752          3.21     0.6882572
##      2:  52228          3.36     0.6763461
##      3:   1303          3.37     0.6755594
##      4: 175688          3.39     0.6739888
##      5: 123810          3.40     0.6732048
##     ---
## 227842:  40523          5.00     0.5588243
## 227843:  53393          5.00     0.5588243
## 227844:  73517          5.00     0.5588243
## 227845:  81035          5.00     0.5588243
## 227846:  96989          5.00     0.5588243
```

```
iforest_train$predictions <- as.factor(ifelse(iforest_scores_train$anomaly_score >=0.6, 1, 0
))

iforest_scores_test = iso$predict(iforest_test)
iforest_scores_test[order(anomaly_score, decreasing = TRUE)]
```

```
##           id average_depth anomaly_score
##     1: 30819          3.42     0.6716396
##     2: 43881          3.45     0.6692987
##     3:  1349          3.46     0.6685202
##     4:  9579          3.50     0.6654152
##     5: 39047          3.50     0.6654152
##    ---
## 56957: 17613          4.99     0.5594751
## 56958: 18670          4.99     0.5594751
## 56959: 19160          4.99     0.5594751
## 56960:   392          5.00     0.5588243
## 56961: 22603          5.00     0.5588243
```

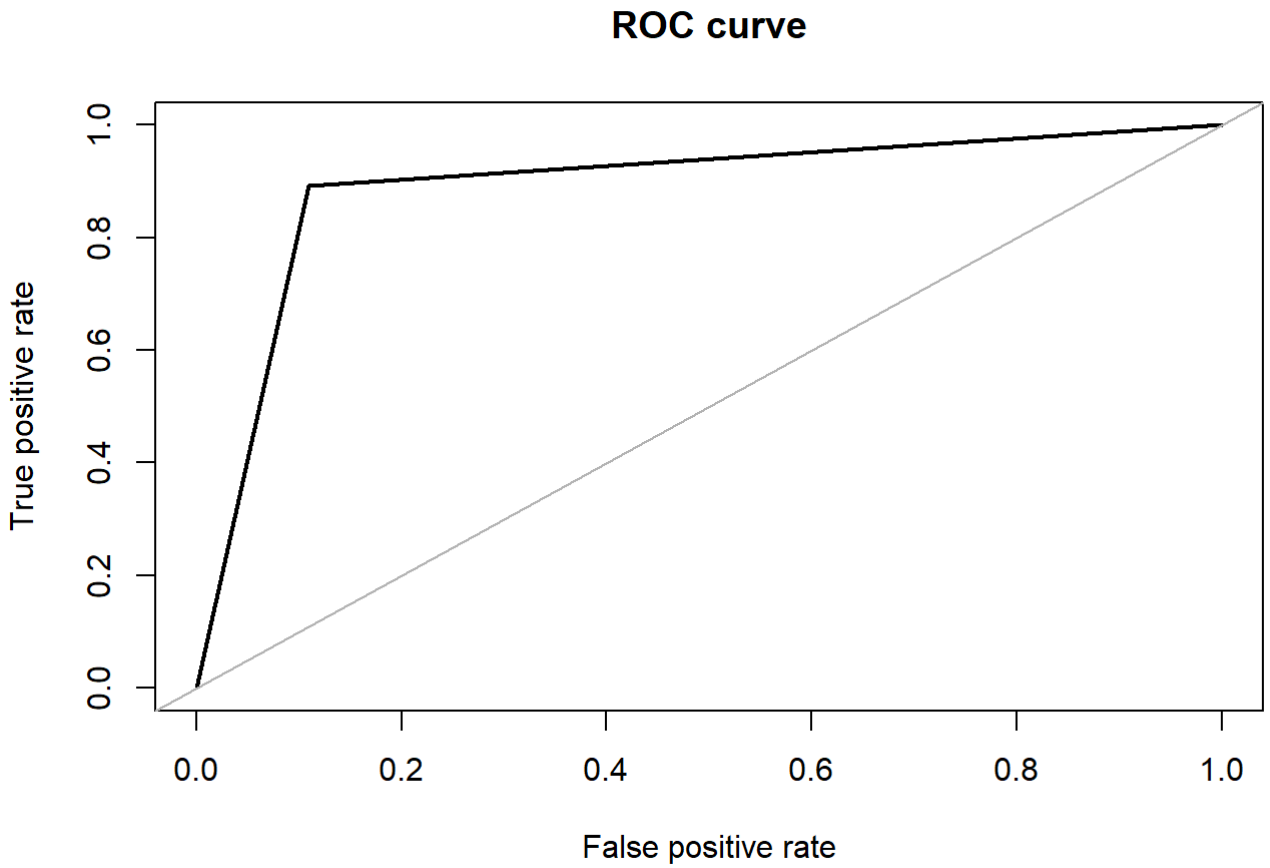
```
iforest_test$predictions <- as.factor(ifelse(iforest_scores_test$anomaly_score >=0.6, 1, 0))
```

ROC and AUC results

```
# Confusion Matrix and ROC curve of training data
confusionMatrix(iforest_train$predictions, as.factor(train$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 202641     43
##          1  24807    355
##
##                Accuracy : 0.8909
##                  95% CI : (0.8896, 0.8922)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0244
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.891960
##             Specificity : 0.890933
##          Pos Pred Value : 0.014109
##          Neg Pred Value : 0.999788
##              Prevalence : 0.001747
##          Detection Rate : 0.001558
##    Detection Prevalence : 0.110434
##       Balanced Accuracy : 0.891447
##
##        'Positive' Class : 1
##
```

```
roc.curve(train$Class, iforest_train$predictions, plotit = TRUE)
```

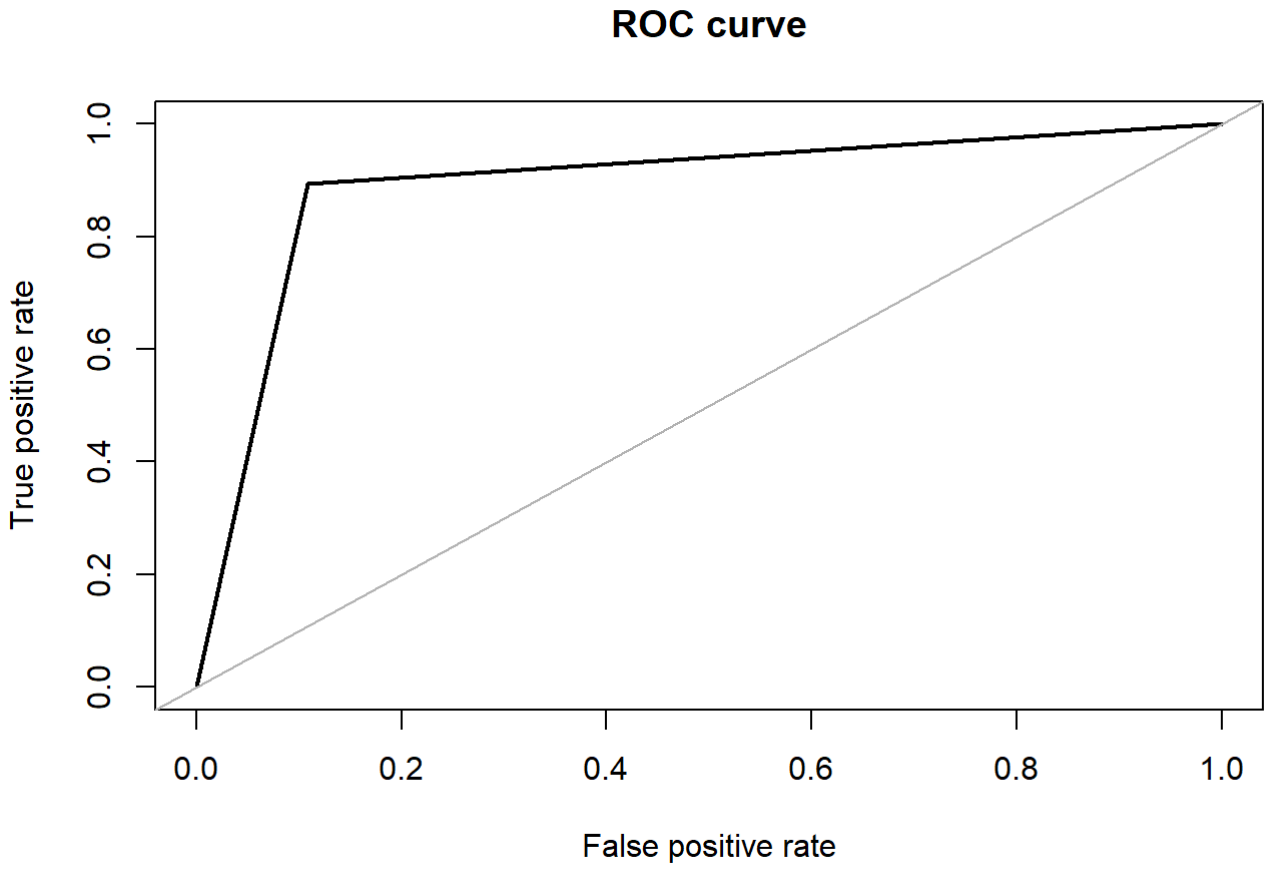



```
## Area under the curve (AUC): 0.891
```

```
# Confusion Matrix and ROC curve of test data
confusionMatrix(iforest_test$predictions, as.factor(test$Class), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 50701    10
##          1  6166    84
##
##                Accuracy : 0.8916
##                  95% CI : (0.889, 0.8941)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0233
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.893617
##             Specificity : 0.891572
##          Pos Pred Value : 0.013440
##          Neg Pred Value : 0.999803
##              Prevalence : 0.001650
##          Detection Rate : 0.001475
##    Detection Prevalence : 0.109724
##       Balanced Accuracy : 0.892594
##
##        'Positive' Class : 1
##
```

```
roc.curve(test$Class, iforest_test$predictions, plotit = TRUE)
```



```
## Area under the curve (AUC): 0.893
```