Credit card fraud Detection Using Random Forest & Logistic Regression

Setting the Working Directory

```
setwd("D:/Data science")
```

Load required libraries

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
library(caTools)
library(smotefamily)
library(caret)
```

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

```
##
## Attaching package: 'ggplot2'
```

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

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

```
library(ggplot2)
```

Read the dataset in csv format

```
df_og <- read.csv("creditcard.csv")
```

Checking the imbalance in the class through plot

```
cat("\n Checking Class Imbalance...")
```

```
##
##  Checking Class Imbalance...
```

```r
0
```

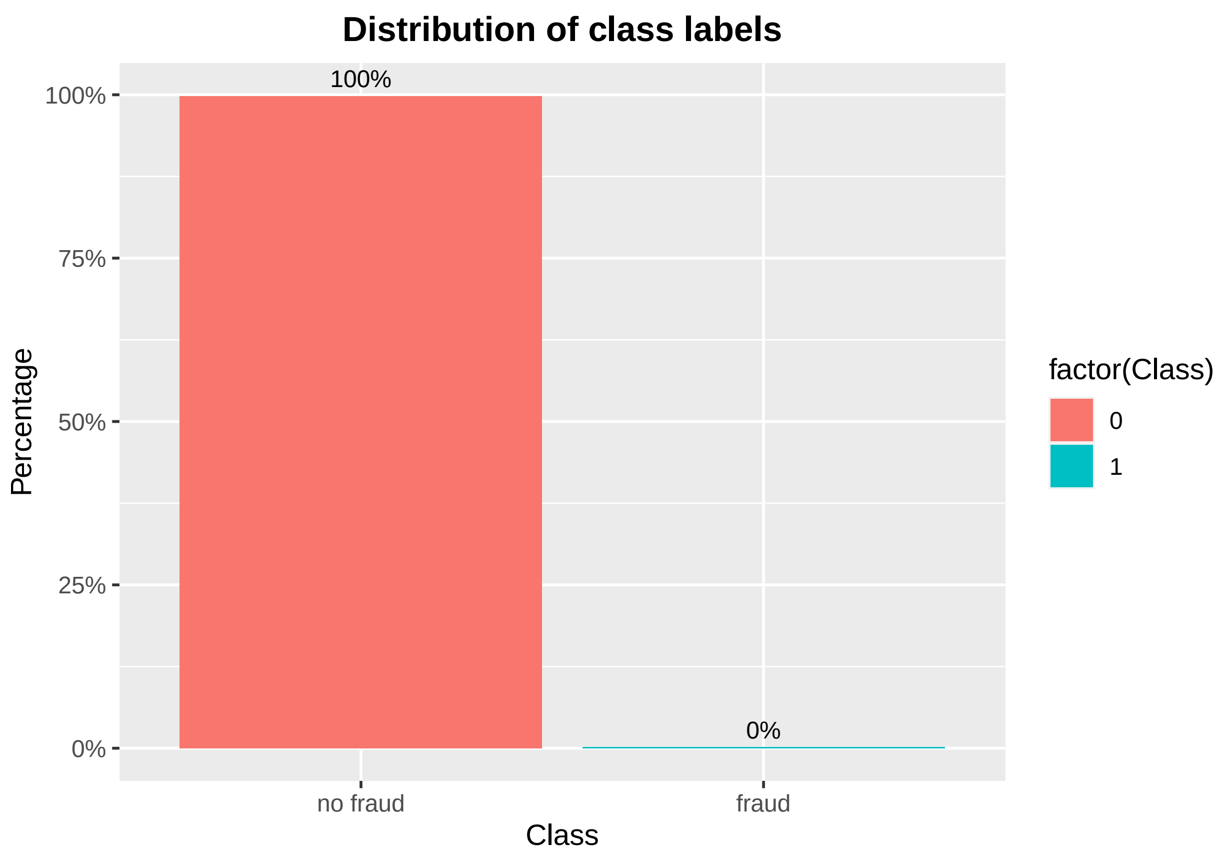
```
## [1] 0
```

```r
common_theme <- theme(plot.title = element_text(hjust = 0.5, face = "bold"))

plot1 <- ggplot(data = df_og, 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

print(plot1)
```

```
## Warning: `stat(count)` was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
```



```
cat("\n Data highly imbalanced. \n SMOTE being implemented...")
```

```
##
## Data highly imbalanced.
## SMOTE being implemented...
```

set number of fraud and legitimate cases and desired % of legitimate cases

```
n0 <- nrow(subset(df_og,Class==0))
n1 <- nrow(subset(df_og,Class==1))
r0 <- 0.65
```

Calculate value for dup_size parameter of SMOTE

```
ntimes <- ((1 - r0) / r0) * (n0/n1) - 1
```

Create synthetic fraud cases with SMOTE

```
set.seed(1234)
smote_output = SMOTE(X = df_og[ , -c(1,31)], target = df_og$Class, K = 5, dup_size = ntimes)
```

SMOTE output

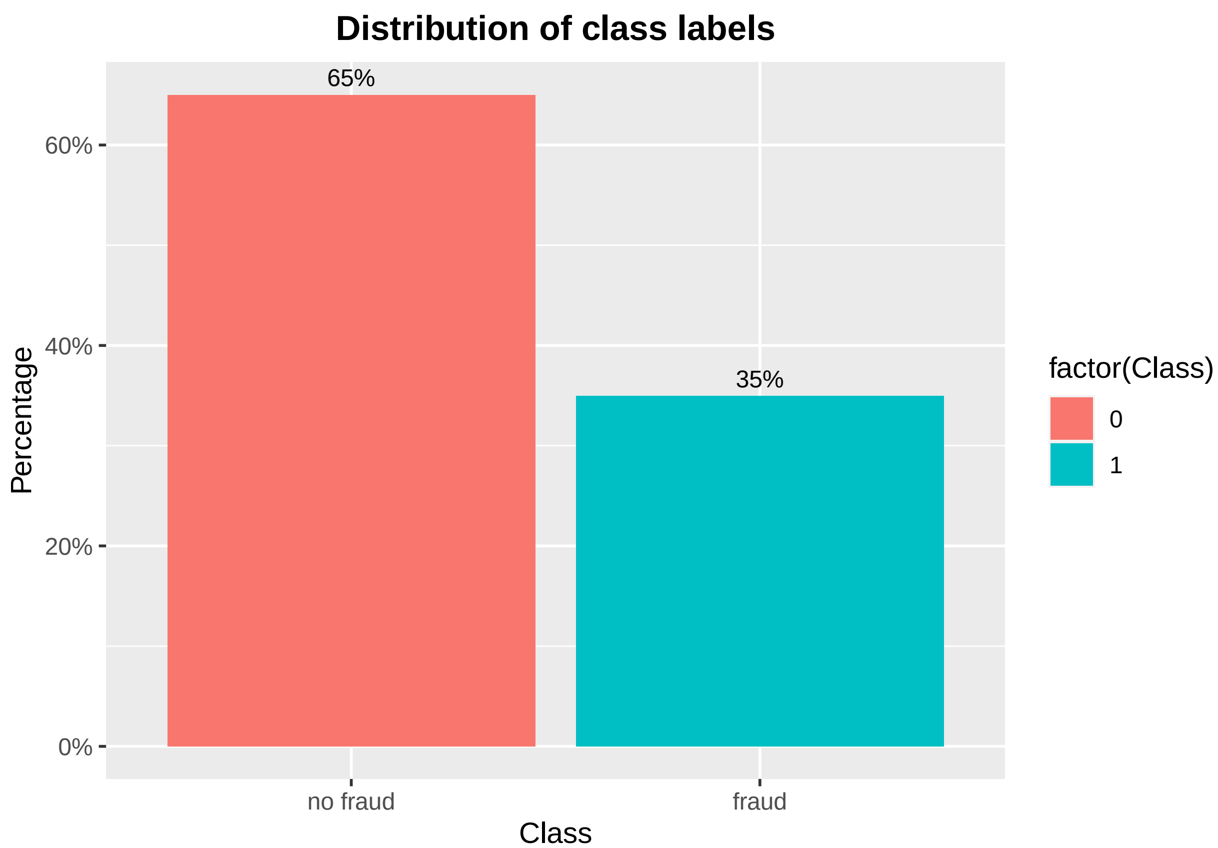
```
df_new <- smote_output$data
colnames(df_new)[30] <- "Class"

df_new$Class <- as.factor(df_new$Class)
```

Plot of SMOTE output

```
plot2 <- ggplot(data = df_new, aes(x = factor(Class),
                         y = prop.table(after_stat(count)), fill = factor(Class),
                         label = scales::percent(prop.table(after_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

print(plot2)
```



sample data randomly

```
set.seed(333)
x <- sample(1:nrow(df_new),50000)

df <- df_new[x, ]
```

Splitting data in train and test data

```
set.seed(444)
split <- sample.split(df, SplitRatio = 0.7)
train <- subset(df, split == "TRUE")
test <- subset(df, split == "FALSE")
```

Training Random Forest model

```
trControl = trainControl(method = "cv", number = 10, allowParallel = TRUE, verboseIter = FALSE, savePre
modfit <- train(Class ~ ., data = train, method = "rf", trControl = trControl)
```

```
cat("Model trained successfully!")
```

```
## Model trained successfully!
```

Predict the class

```r
testclass <- predict(modfit,test)
```

Creating the confusion matrix

```r
cfMatrix <- confusionMatrix(testclass, as.factor(test$Class))
print(cfMatrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9746   55
##          1   18 5180
##
##                Accuracy : 0.9951
##                  95% CI : (0.9939, 0.9962)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9893
##
##  Mcnemar's Test P-Value : 2.515e-05
##
##             Sensitivity : 0.9982
##             Specificity : 0.9895
##          Pos Pred Value : 0.9944
##          Neg Pred Value : 0.9965
##              Prevalence : 0.6510
##          Detection Rate : 0.6498
##    Detection Prevalence : 0.6534
##       Balanced Accuracy : 0.9938
##
##        'Positive' Class : 0
##
```

Training Logistic Regression model

```r
set.seed(766)

reguarlized_model <- train(Class ~ ., data = train,
                           method = "glmnet",
                           metric = "Accuracy",

                           trControl = trainControl(method = "cv",
                                                    number = 10,
                                                    search = "random",
                                                    verboseIter = T))
```

```
## + Fold01: alpha=0.6793, lambda=0.015533
## - Fold01: alpha=0.6793, lambda=0.015533
## + Fold01: alpha=0.3265, lambda=0.027892
## - Fold01: alpha=0.3265, lambda=0.027892
```

```
## + Fold01: alpha=0.4396, lambda=0.001984
## - Fold01: alpha=0.4396, lambda=0.001984
## + Fold02: alpha=0.6793, lambda=0.015533
## - Fold02: alpha=0.6793, lambda=0.015533
## + Fold02: alpha=0.3265, lambda=0.027892
## - Fold02: alpha=0.3265, lambda=0.027892
## + Fold02: alpha=0.4396, lambda=0.001984
## - Fold02: alpha=0.4396, lambda=0.001984
## + Fold03: alpha=0.6793, lambda=0.015533
## - Fold03: alpha=0.6793, lambda=0.015533
## + Fold03: alpha=0.3265, lambda=0.027892
## - Fold03: alpha=0.3265, lambda=0.027892
## + Fold03: alpha=0.4396, lambda=0.001984
## - Fold03: alpha=0.4396, lambda=0.001984
## + Fold04: alpha=0.6793, lambda=0.015533
## - Fold04: alpha=0.6793, lambda=0.015533
## + Fold04: alpha=0.3265, lambda=0.027892
## - Fold04: alpha=0.3265, lambda=0.027892
## + Fold04: alpha=0.4396, lambda=0.001984
## - Fold04: alpha=0.4396, lambda=0.001984
## + Fold05: alpha=0.6793, lambda=0.015533
## - Fold05: alpha=0.6793, lambda=0.015533
## + Fold05: alpha=0.3265, lambda=0.027892
## - Fold05: alpha=0.3265, lambda=0.027892
## + Fold05: alpha=0.4396, lambda=0.001984
## - Fold05: alpha=0.4396, lambda=0.001984
## + Fold06: alpha=0.6793, lambda=0.015533
## - Fold06: alpha=0.6793, lambda=0.015533
## + Fold06: alpha=0.3265, lambda=0.027892
## - Fold06: alpha=0.3265, lambda=0.027892
## + Fold06: alpha=0.4396, lambda=0.001984
## - Fold06: alpha=0.4396, lambda=0.001984
## + Fold07: alpha=0.6793, lambda=0.015533
## - Fold07: alpha=0.6793, lambda=0.015533
## + Fold07: alpha=0.3265, lambda=0.027892
## - Fold07: alpha=0.3265, lambda=0.027892
## + Fold07: alpha=0.4396, lambda=0.001984
## - Fold07: alpha=0.4396, lambda=0.001984
## + Fold08: alpha=0.6793, lambda=0.015533
## - Fold08: alpha=0.6793, lambda=0.015533
## + Fold08: alpha=0.3265, lambda=0.027892
## - Fold08: alpha=0.3265, lambda=0.027892
## + Fold08: alpha=0.4396, lambda=0.001984
## - Fold08: alpha=0.4396, lambda=0.001984
## + Fold09: alpha=0.6793, lambda=0.015533
## - Fold09: alpha=0.6793, lambda=0.015533
## + Fold09: alpha=0.3265, lambda=0.027892
## - Fold09: alpha=0.3265, lambda=0.027892
## + Fold09: alpha=0.4396, lambda=0.001984
## - Fold09: alpha=0.4396, lambda=0.001984
## + Fold10: alpha=0.6793, lambda=0.015533
## - Fold10: alpha=0.6793, lambda=0.015533
## + Fold10: alpha=0.3265, lambda=0.027892
## - Fold10: alpha=0.3265, lambda=0.027892
```

```
## + Fold10: alpha=0.4396, lambda=0.001984
## - Fold10: alpha=0.4396, lambda=0.001984
## Aggregating results
## Selecting tuning parameters
## Fitting alpha = 0.44, lambda = 0.00198 on full training set
```

Predict class

```
t2 <- predict(reguarlized_model,test)
```

Creating the confusion matrix

```
cm <- confusionMatrix(t2,as.factor(test$Class))
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9684  416
##          1   80 4819
##
##                Accuracy : 0.9669
##                  95% CI : (0.9639, 0.9697)
##     No Information Rate : 0.651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9261
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9918
##             Specificity : 0.9205
##          Pos Pred Value : 0.9588
##          Neg Pred Value : 0.9837
##              Prevalence : 0.6510
##          Detection Rate : 0.6456
##    Detection Prevalence : 0.6734
##       Balanced Accuracy : 0.9562
##
##        'Positive' Class : 0
##
```

Output dataframe comparing evaluation metrics of two algorithms

```
output <- data.frame(metric=rep(c('Accuracy', 'Sensitivity','Specificity', 'Precision'), each=4),
                 position=rep(c('Logistic Regression', 'Random Forest'), times=2),
                 percentage=c(99.51,98.52,98.76,99.84,96.32,99.19,91.01,98.38))
```

Final plot of evaluation metrics

```
plot3 <- ggplot(output, aes(fill=position, y=percentage, x=metric)) +
  geom_bar(position='dodge', stat='identity')

print(plot3)
```