````
---
title: "Fraud Detection"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.
#Loading Libraries

```{r loadpackages}
library(tidyverse)
library(dplyr)
library(data.table)
library(e1071)
library(caret)
library(rpart)
library(ROSE)
library(corrplot)
library(DescTools)
library(caTools)
library(class)
library(randomForest)
library(naivebayes)
library(klaR)
library(scales)
options(warn=-1)
```
#Confusion Matrix visualisation Function
```{r confusion matrix function,warning=FALSE}
draw_confusion_matrix <- function(cm) {

  layout(matrix(c(1,1,2)))
  par(mar=c(2,2,2,2))
  plot(c(100, 345), c(300, 450), type = "n", xlab="", ylab="", xaxt='n', yaxt='n')
  title('CONFUSION MATRIX', cex.main=2)

  # create the matrix
  rect(150, 430, 240, 370, col='#3F97D0')
  text(195, 435, 'Class0', cex=1.2)
  rect(250, 430, 340, 370, col='#F7AD50')
  text(295, 435, 'Class1', cex=1.2)
text(125, 370, 'Predicted', cex=1.3, srt=90, font=2)
text(245, 450, 'Actual', cex=1.3, font=2)
rect(150, 305, 240, 365, col='#F7AD50')
rect(250, 305, 340, 365, col='#3F97D0')
text(140, 400, 'Class0', cex=1.2, srt=90)
text(140, 335, 'Class1', cex=1.2, srt=90)

# add in the cm results
res <- as.numeric(cm$table)
text(195, 400, res[1], cex=1.6, font=2, col='white')
text(195, 335, res[2], cex=1.6, font=2, col='white')
text(295, 400, res[3], cex=1.6, font=2, col='white')
text(295, 335, res[4], cex=1.6, font=2, col='white')

# add in the specifics
plot(c(100, 0), c(100, 0), type = "n", xlab="", ylab="", main = "DETAILS", xaxt='n',
````
````

````
yaxt='n')
text(10, 85, names(cm$byClass[1]), cex=1.2, font=2)
text(10, 70, round(as.numeric(cm$byClass[1]), 3), cex=1.2)
text(30, 85, names(cm$byClass[2]), cex=1.2, font=2)
text(30, 70, round(as.numeric(cm$byClass[2]), 3), cex=1.2)
text(50, 85, names(cm$byClass[5]), cex=1.2, font=2)
text(50, 70, round(as.numeric(cm$byClass[5]), 3), cex=1.2)
text(70, 85, names(cm$byClass[6]), cex=1.2, font=2)
text(70, 70, round(as.numeric(cm$byClass[6]), 3), cex=1.2)
text(90, 85, names(cm$byClass[7]), cex=1.2, font=2)
text(90, 70, round(as.numeric(cm$byClass[7]), 3), cex=1.2)

# add in the accuracy information
text(30, 35, names(cm$overall[1]), cex=1.5, font=2)
text(30, 20, round(as.numeric(cm$overall[1]), 3), cex=1.4)
text(70, 35, names(cm$overall[2]), cex=1.5, font=2)
text(70, 20, round(as.numeric(cm$overall[2]), 3), cex=1.4)
}
```

## reading the data

```{r read data, warning=FALSE}
fraud_detection=fread("creditcard.csv")
names(fraud_detection)
str(fraud_detection)
table(fraud_detection$Class)
```

#Class Distribution
```{r class,warning=FALSE}
x=fraud_detection[,-1]
y=fraud_detection[,31]

#visualising the class

y %>%
  mutate(max_class = max(table(Class))) %>%
  ggplot(aes(x=factor(Class)))+
  geom_bar(stat="count", width=0.75, fill="blue", color = "grey40", alpha = .75)+
  xlab("Class") + ylab("Number of Transactions") +
  ggtitle("Class Distributions") +
  theme_minimal()
#visualising the class

y %>%
  mutate(max_class = max(table(Class))) %>%
  ggplot(aes(x=factor(Class)))+
  geom_bar(stat="count", width=0.75, fill="blue", color = "grey40", alpha = .75)+
  xlab("Class") + ylab("Number of Transactions") +
  ggtitle("Class Distributions") +
  theme_minimal()

#money hist

ggplot(fraud_detection, aes(x=Amount)) +
  geom_histogram(aes(y=..density..),stat="bin",bins="30", colour="black", fill="white")+
  geom_density(alpha=.2, fill="#FF6611") +
  ggtitle("Density of Transaction vs Amount")

#time hist
````

````
ggplot(fraud_detection, aes(x=Time)) +
  geom_histogram(aes(y=..density..),stat="bin",bins="30", colour="black", fill="white")+
  geom_density(alpha=.2, fill="#FF6611") +
  ggtitle("Density of Transaction vs Time")

#time amount and class
ggplot(filter(fraud_detection, Class %in% c("0", "1")),
       aes(x=Time,
           y=Amount,
           color=Class))+
  geom_point() +
  ggtitle("Time of Transaction vs Amount by Class")
```
#correlation

```{r distribution,warning=FALSE}
#correlation

fraud_detection_c=fraud_detection
fraud_detection_c=fraud_detection_c[,c(-1,-30)]
#fraud_detection_c$Class=as.numeric(fraud_detection_c$Class)
fraud_detection_cp=as.matrix(fraud_detection_c)
class(fraud_detection_cp)
corr_mat=cor(fraud_detection_cp,method="s")
corr_mat=cor(fraud_detection_cp)
library(scales)
ord=hclust(1-as.dist(corr_mat))$order
co=melt(corr_mat[ord,ord])
ggplot(co, aes(Var1, Var2)) +
  geom_tile(aes(fill = value)) +
  geom_text(aes(fill = co$value, label = round(co$value, 2))) +
  scale_fill_gradient2(low = muted("darkred"),
                       mid = "white",
                       high = muted("midnightblue"),
                       midpoint = 0) +
  theme(panel.grid.major.x=element_blank(),
        panel.grid.minor.x=element_blank(),
        panel.grid.major.y=element_blank(),
        panel.grid.minor.y=element_blank(),
        panel.background=element_rect(fill="white"),
        axis.text.x = element_text(angle=90, hjust = 1,vjust=1,size = 12,face = "bold"),
        plot.title = element_text(size=20,face="bold"),
        axis.text.y = element_text(size = 12,face = "bold")) +
  ggtitle("Correlation Plot") +
  theme(legend.title=element_text(face="bold", size=14)) +
  scale_x_discrete(name="") +
  scale_y_discrete(name="") +
  labs(fill="Corr. Coef.")

```


#Scaling the data
```{r scaling, warning=FALSE}
fraud_detection$scaled_time=RobScale(fraud_detection$Time,center=TRUE,scale = TRUE)
fraud_detection$scaled_amount=RobScale(fraud_detection$Amount,center=TRUE,scale = TRUE)

fraud_detection_1=fraud_detection
fraud_detection_1$Time=NULL
fraud_detection_1$Amount=NULL
fraud_detection_dt=fraud_detection
fraud_detection_1$Class <- as.factor(fraud_detection_1$Class)
````

```
x=fraud_detection[,-1]
y=fraud_detection[,31]
```
```

#Training and Testing Split

```
```{r train and test split, warning=FALSE}
#test and train split

trainindex <- createDataPartition(fraud_detection_1$Class, p=0.8, list= FALSE)
fd_train <- fraud_detection_1[trainindex,]
fd_test <- fraud_detection_1[-trainindex, ]

#for dt test and train

trainindex_dt <- createDataPartition(fraud_detection_dt$Class, p=0.8, list= FALSE)
fd_train_dt <- fraud_detection_dt[trainindex_dt,]
fd_test_dt<- fraud_detection_dt[-trainindex_dt, ]

```
```

#Value for N in Sampling and Correlation

```
```{r value for N, warning=FALSE}
#value for n in sampling

x=as.data.frame(table(fd_train$Class))
x1=x%>%filter(Var1==0)
x11=x1$Freq
x2=x%>%filter(Var1==1)
x22=x2$Freq

#value for n in sampling fpr dt

x_dt=as.data.frame(table(fd_train_dt$Class))
x1_dt=x_dt%>%filter(Var1==0)
x11_dt=x1_dt$Freq
x2_dt=x_dt%>%filter(Var1==1)
x22_dt=x2_dt$Freq



```
```

#Undersampling

```
```{r undersampling, warning=FALSE}
fd_balanced_under <- ovun.sample(Class ~ ., data = fd_train, method = "under",
                                 N = (2*x22), seed = 1)$data
table(fd_balanced_under$Class)
str(fd_balanced_under)

```
```

#Undersapling correlation

```
```{r distribution,warning=FALSE}

fraud_detection_c=fd_balanced_under
fraud_detection_c=fraud_detection_c[,c(-1,-30)]
fraud_detection_c$Class=as.numeric(fraud_detection_c$Class)
fraud_detection_cp=as.matrix(fraud_detection_c)
corr_mat=cor(fraud_detection_cp,method="s")
corr_mat=cor(fraud_detection_cp)
ord=hclust(1-as.dist(corr_mat))$order
co=melt(corr_mat[ord,ord])
```

```
ggplot(co, aes(Var1, Var2)) +
  geom_tile(aes(fill = value)) +
  geom_text(aes(fill = co$value, label = round(co$value, 2))) +
  scale_fill_gradient2(low = muted("darkred"),
                       mid = "white",
                       high = muted("midnightblue"),
                       midpoint = 0) +
  theme(panel.grid.major.x=element_blank(),
        panel.grid.minor.x=element_blank(),
        panel.grid.major.y=element_blank(),
        panel.grid.minor.y=element_blank(),
        panel.background=element_rect(fill="white"),
        axis.text.x = element_text(angle=90, hjust = 1,vjust=1,size = 12,face = "bold"),
        plot.title = element_text(size=20,face="bold"),
        axis.text.y = element_text(size = 12,face = "bold")) +
  ggtitle("Under Sampling Correlation Plot") +
  theme(legend.title=element_text(face="bold", size=14)) +
  scale_x_discrete(name="") +
  scale_y_discrete(name="") +
  labs(fill="Corr. Coef.")

```
#Algorithms
#Logistic Regression
```{r logistic, warning=FALSE}
classifier_us = glm(formula = Class ~ .,family = binomial,data =fd_balanced_under)
pred_log_us = predict(classifier_us, type = 'response', newdata = fd_test)
pred_log_us_1 = ifelse(pred_log_us > 0.5, 1, 0)

```
#Logistic Confusion matrix and ROC
```{r confusion matrix, warning=FALSE}

cm1=confusionMatrix(table(pred_log_us_1,fd_test$Class))
cm1
draw_confusion_matrix(cm1)
x1=roc.curve(fd_test$Class, pred_log_us_1,curve = TRUE)
x1
plot(x1)
a1=(pr.curve(fd_test$Class, pred_log_us_1,curve = TRUE))
a1
plot(a1)
```

#Knn
#Hyperparameter tunning
```{r knn hyperparameter tuning, warning=FALSE}
knn_us <- train(Class~., data=fd_balanced_under, method='knn',
  tuneGrid=expand.grid(.k=1:25),metric='Accuracy',
  trControl=trainControl(method='repeatedcv', number=10,repeats=3))
knn_us
knn_us_df=as.data.frame(knn_us$results)
knn_us_optimal=max(knn_us_df$k)
plot(knn_us)
```

#Train, Confusion matrix and ROC
```{r train, warning=FALSE}
pred_knn_us = knn(train = fd_balanced_under[,-29],test = fd_test[,-29],
             cl = fd_balanced_under[,29],
             k = knn_us_optimal,
             prob = TRUE)
```

````
cm2=confusionMatrix(table(pred_knn_us,fd_test$Class))
cm2
draw_confusion_matrix(cm2)
x1=roc.curve(fd_test$Class, pred_knn_us,curve = TRUE)
x1
plot(x1)
a2=(pr.curve(fd_test$Class, pred_knn_us,curve = TRUE))
a2
plot(a2)
```

#Naive Bayes

#Hyperparameter tunning
```{r naive bayes, warning=FALSE}
nb_us = train(x = fd_balanced_under[-29],
               y = fd_balanced_under$Class,method = "nb",
              trControl = trainControl(method='repeatedcv', number=10,repeats=3),
              tuneGrid = expand.grid(usekernel = c(TRUE, FALSE),fL = 0:5,adjust = seq(0,
5, by = 1)))
nb_us

```

#Confusion matrix and ROC
```{r naive bayes train,warning=FALSE}
pred_nb_us = predict(nb_us, newdata = fd_test)
cm3=confusionMatrix(table(pred_nb_us,fd_test$Class))
cm3
draw_confusion_matrix(cm3)
x3=roc.curve(fd_test$Class, pred_nb_us,curve = TRUE)
x3
plot(x3)
```

#recall
```{r recall,warning=False}
a3=(pr.curve(fd_test$Class, pred_nb_us,curve = TRUE))
a3
plot(a3)
```

#Decision Tree
#Hyperparameter tunning
```{r decision tree, warning=FALSE}
dt_us = train(Class~., data=fd_balanced_under, method='rpart',
              tuneGrid=expand.grid(.cp=seq(0.00,0.03,0.001)),metric='Accuracy',
              trControl=trainControl(method='repeatedcv', number=10,repeats=3))
dt_us
plot(dt_us)
```

#Confusion matrix and ROC
```{r confusion matric, warning=FALSE}
fd_balanced_under_dt <- ovun.sample(Class ~ ., data = fd_train_dt, method = "under",
                                    N = (2*x22_dt), seed = 1)$data
table(fd_balanced_under_dt$Class)

tree_us = rpart(Class ~ ., data =fd_balanced_under_dt,
                control=rpart.control(cp=0,minbucket= 8,minsplit = 100))
prune_us <- prune(tree_us, cp = 0.008)
````

```
pred_tree_us <- predict(prune_us, newdata = fd_test_dt)
pred_tree_us_1 = ifelse(pred_tree_us > 0.5, 1, 0)

cm4=confusionMatrix(table(pred_tree_us_1,fd_test_dt$Class))
cm4
draw_confusion_matrix(cm4)
x4=roc.curve(fd_test_dt$Class, pred_tree_us_1,curve=TRUE)
x4
plot(x4)
```

#recall
```{r recall,warning=False}
a4=(pr.curve(fd_test$Class, pred_tree_us_1,curve = TRUE))
a4
plot(a4)
```

#Ramdom Forest
#Hyperparameter tunning

```{r randomforest, warning=FALSE}

rf_us = train(Class~., data=fd_balanced_under, method='rf',
               tuneGrid=expand.grid(.mtry=c(1:15)),
              metric='Accuracy',trControl=trainControl(method='repeatedcv',
number=10,repeats=2))
rf_us

```

#Confusion matrix and ROC
```{r confusion matrix,warning=FALSE}
randomforest = randomForest(x = fd_balanced_under[-29],
                        y = fd_balanced_under$Class,
                        ntree=1500,mtry =9)

pred_rf_us = predict(randomforest, newdata = fd_test)
cm5=confusionMatrix(table(pred_rf_us,fd_test$Class))
cm5
draw_confusion_matrix(cm5)
x5=roc.curve(fd_test$Class, pred_rf_us,curve = TRUE)
x5
plot(x5)
```

#recall
```{r recall,warning=False}
a5=(pr.curve(fd_test$Class, pred_rf_us,curve = TRUE))
a5
plot(a5)
```

#Support Vector Machine
#Hyperparameter tuning
```{r svm, warning=FALSE}

svm_tn_us <- train(Class ~., data = fd_balanced_under,
                method = "svmPoly",
                trControl=trainControl(method = "repeatedcv",
                                       number = 10, repeats = 1),
                preProcess = c("center", "scale"),
```

```
                tuneGrid = expand.grid(.degree = c(2:5),.scale = c(0.1,1,10),
                    .C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75,1, 1.5, 2,5)),
                tuneLength = 10)
svm_tn_us
plot(svm_tn_us)
```

```
#SVM Tuning

```{r svm tuning, warning=FALSE}

tune_out = tune.svm(x = fd_balanced_under[,-29], y = fd_balanced_under[,29],
           type = "C-classification", kernel = "polynomial", degree = 2,
           cost = 0.25, gamma = c(0.1,0.5,1,10), coef0 = c(0.1,1,10))
cost=tune_out$best.parameters$cost
cost
gamma=tune_out$best.parameters$gamma
gamma
coef0=tune_out$best.parameters$coef0
coef0
```
#Confusion matrix and ROC

```{r svm Confusion matrix, warning=FAlSE}
svm_us <- svm(Class~ ., data = fd_balanced_under, type = "C-classification",
                 kernel = "polynomial", degree = 2,scale=0.1,
                 cost = tune_out$best.parameters$cost,
                 gamma = tune_out$best.parameters$gamma,
                 coef0 = tune_out$best.parameters$coef0)
pred_svm_us = predict(svm_us, newdata = fd_test)
cm6=confusionMatrix(table(pred_svm_us,fd_test$Class))
cm6
draw_confusion_matrix(cm6)
x6=roc.curve(fd_test$Class, pred_svm_us,curve = TRUE)
x6
plot(x6)
```

#recall
```{r recall,warning=False}
a6=(pr.curve(fd_test$Class, pred_svm_us,curve = TRUE))
a6
plot(a6)
```

#Neural Network
```{r neural net library,warning=FALSE}
library(neuralnet)
library(GGally)

```

#undersampling

```{r NN US,warning=FALSE}
fd_NN1 = neuralnet(Class~.,
                     data = fd_balanced_under,
                     linear.output = FALSE,
                     err.fct = 'ce',
                     likelihood = TRUE)

plot(fd_NN1, rep = 'best')
```
```

````
```


```{r nn3}
fd_NN3 <- neuralnet(Class~.,
                    data = fd_balanced_under,
                    linear.output = FALSE,
                    err.fct = 'ce',
                    likelihood = TRUE,hidden = c(2,2))
plot(fd_NN3, rep = 'best')

```


```{r nn4}
fd_NN4 <- neuralnet(Class~.,
                    data = fd_balanced_under,
                    linear.output = FALSE,
                    err.fct = 'ce',
                    likelihood = TRUE,hidden = c(1,2))
plot(fd_NN4, rep = 'best')
```

```{r nn2,warning=FALSE}
fd_NN2 <- neuralnet(Class~.,
                    data = fd_balanced_under,
                    linear.output = FALSE,
                    err.fct = 'ce',
                    likelihood = TRUE,hidden = c(2,1))
plot(fd_NN2, rep = 'best')
```

#NN AIC
```{r aic}
Class_NN_ICs <- tibble('Network' = rep(c("NN1", "NN2", "NN3", "NN4"), each = 3),
                       'Metric' = rep(c('AIC', 'BIC', 'ce Error * 100'),
length.out=12),'Value' =c(fd_NN1$result.matrix[4,1],fd_NN1$result.matrix[5,1],
100*fd_NN1$result.matrix[1,1], fd_NN2$result.matrix[4,1],
              fd_NN2$result.matrix[5,1], 100*fd_NN2$result.matrix[1,1],
              fd_NN3$result.matrix[4,1],fd_NN3$result.matrix[5,1],
              100*fd_NN3$result.matrix[1,1], fd_NN4$result.matrix[4,1],
              fd_NN4$result.matrix[5,1],100*fd_NN4$result.matrix[1,1]))

Class_NN_ICs %>%
  ggplot(aes(Network, Value, fill = Metric)) +
  geom_col(position = 'dodge')  +
  ggtitle("AIC, BIC, and Cross-Entropy Error of the Classification ANNs", "Note: ce
Error displayed is 100 times its true value")
```
```{r NN cm}

predict <- predict(fd_NN3, fd_balanced_under[,-29])
predicted.class <- apply(predict,1,which.max)-1

cm19=confusionMatrix(factor(ifelse(predicted.class == "0", "0", "1")),
                factor(fd_balanced_under$Class))
draw_confusion_matrix(cm19)
x7=roc.curve(factor(ifelse(predicted.class == "0", "0", "1")),
                factor(fd_balanced_under$Class),curve = TRUE)
x7
plot(x7)
#recall
````

```
a7=(pr.curve(factor(ifelse(predicted.class == "0", "0", "1")),
                factor(fd_balanced_under$Class),curve = TRUE))
a7
plot(a7)
```

```
#Clustering
```{r k-means,warning=FALSE}
fdk_us=fd_balanced_under[,c(-29,-30)]

kmeans = kmeans(x = fdk_us, centers = 2)
y_kmeans =(kmeans$cluster)

library(cluster)
clusplot(fd_balanced_under,
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste('Clusters of Customers'))
```

##Oversampling

```{r oversample,warning=FALSE}
fd_balanced_over = ovun.sample(Class ~ ., data = fd_train, method = "over",N =
2*x11)$data
fd_balanced_over=fd_balanced_over[sample(nrow(fd_balanced_over), 1500), ]
table(fd_balanced_over$Class)
```
#Logistic Regression
```{r Logistic oversampling, warning=FALSE}
classifier_os = glm(formula = Class ~ .,family = binomial,data =fd_balanced_over)
pred_log_os = predict(classifier_os, type = 'response', newdata = fd_test)
pred_log_os_1 = ifelse(pred_log_os > 0.5, 1, 0)

cm7=confusionMatrix(table(pred_log_os_1,fd_test$Class))
cm7
draw_confusion_matrix(cm7)
x8=roc.curve(fd_test$Class, pred_log_os_1,curve = TRUE)
x8
plot(x8)

a8=(pr.curve(fd_test$Class, pred_log_os_1,curve = TRUE))
a8
plot(a8)

```

#KNN
#hyperparameter tuning
```{r knn hyp, warning=FALSE}
knn_os <- train(Class~., data=fd_balanced_over, method='knn',
  tuneGrid=expand.grid(.k=1:25),metric='Accuracy',
  trControl=trainControl(method='repeatedcv', number=10,repeats=1))
knn_os
```
```

```
knn_os_df=as.data.frame(knn_os$results)
knn_os_optimal=max(knn_os_df$k)
plot(knn_os)
```
```

```{r KNN oversampling, warning=FALSE}
pred_knn_os = knn(train = fd_balanced_over[,-29],test = fd_test[,-29],
                  cl = fd_balanced_over[,29],
                  k = knn_os_optimal,
                  prob = TRUE,use.all = F)

cm8=confusionMatrix(table(pred_knn_os,fd_test$Class))
cm8
draw_confusion_matrix(cm8)
x9=roc.curve(fd_test$Class, pred_knn_os,curve = TRUE)
x9
plot(x9)
a9=(pr.curve(fd_test$Class, pred_knn_os,curve = TRUE))
a9
plot(a9)

```

#Naive Bayes
#tuning
```{r naivebayes oversampling, warning=FALSE}
nb_os = train(x = fd_balanced_over[-29],
               y = fd_balanced_over$Class,method = "nb",
              trControl = trainControl(method='repeatedcv', number=10,repeats=1),
              tuneGrid = expand.grid(usekernel = c(TRUE, FALSE),fL = 0:3,adjust =
seq(0, 3, by = 1)))
nb_os
```

```{r cm, warning=FALSE}
pred_nb_os = predict(nb_os, newdata = fd_test)
cm9=confusionMatrix(table(pred_nb_os,fd_test$Class))
cm9
draw_confusion_matrix(cm9)
x10=roc.curve(fd_test$Class, pred_nb_os,curve=TRUE)
x10
plot(x10)
a10=(pr.curve(fd_test$Class, pred_nb_os,curve = TRUE))
a10
plot(a10)

```

#Decision Tree
#tuning
```{r dt tuning,warning=FALSE}
dt_os = train(Class~., data=fd_balanced_over, method='rpart',
              tuneGrid=expand.grid(.cp=seq(0.00,0.03,0.001)),metric='Accuracy',
              trControl=trainControl(method='repeatedcv', number=10,repeats=3))
dt_os
plot(dt_os)
```

#Confusion matrix
```{r dt oversampling}
fd_balanced_over_dt <- ovun.sample(Class ~ ., data = fd_train_dt, method = "over",
                                   N = 2*x11_dt, seed = 1)$data
```
```

````
fd_balanced_over_dt=fd_balanced_over_dt[sample(nrow(fd_balanced_over_dt), 3000), ]
table(fd_balanced_over_dt$Class)


tree_os = rpart(Class ~ ., data =fd_balanced_over_dt,
                control=rpart.control(cp = 0.0,maxdepth = 8,minsplit = 100))

prune_os <- prune(tree_os, cp = 0.00)
pred_tree_os <- predict(prune_os, newdata = fd_test_dt)
pred_tree_os_1 = ifelse(pred_tree_os > 0.5, 1, 0)

cm10=confusionMatrix(table(pred_tree_os_1,fd_test_dt$Class))
cm10
draw_confusion_matrix(cm10)
x11_=roc.curve(fd_test_dt$Class, pred_tree_os_1,curve = TRUE)
x11_
plot(x11_)
a11=(pr.curve(fd_test$Class, pred_tree_os_1,curve = TRUE))
a11
plot(a11)
```

#Random Forest
#tuning
```{r rf tuning, warning=FALSE}
rf_os = train(Class~., data=fd_balanced_over, method='rf',
               tuneGrid=expand.grid(.mtry=c(1:15)),
              metric='Accuracy',trControl=trainControl(method='repeatedcv',
number=10,repeats=1))
rf_os
```

#confusion matrix
```{r rf oversampling,warning=FALSE}
randomforest_os = randomForest(x = fd_balanced_over[-29],
                               y = fd_balanced_over$Class,
                               mtry = 13)

pred_rf_os = predict(randomforest_os, newdata = fd_test)

cm11=confusionMatrix(table(pred_rf_os,fd_test$Class))
cm11
draw_confusion_matrix(cm11)
x22_=roc.curve(fd_test$Class, pred_rf_os,curve=TRUE)
x22_
plot(x22_)
a12=(pr.curve(fd_test$Class, pred_nb_os,curve = TRUE))
a12
plot(a12)
```

#Support Vector Machine
#tuning
```{r svm tuning,warning=FALSE}
svm_tn_os <- train(Class ~., data = fd_balanced_over,
                method = "svmPoly",
                trControl=trainControl(method = "repeatedcv",
                                       number = 10, repeats = 1),
                preProcess = c("center", "scale"),
                tuneGrid = expand.grid(.degree = c(2:3),.scale = c(0.1,1,10),
                    .C = c(0,0.01, 0.05, 0.1, 0.5,1, 1.5, 2,5)),
                tuneLength = 10)
````

````
svm_tn_os
```

```{r svm oversampling,warning=FALSE}

tune_out_os = tune.svm(x = fd_balanced_over[,-29], y = fd_balanced_over[,29],
                       type = "C-classification", kernel = "polynomial", degree = 3,
                       cost = 1.5, gamma = c(0.1,1,10), coef0 = c(0.1,1,10))
cost1=tune_out$best.parameters$cost
cost1
gamma1=tune_out$best.parameters$gamma
gamma1
coef0_1=tune_out$best.parameters$coef0
coef0_1

svm_os = svm(Class~ ., data = fd_balanced_over, type = "C-classification",
             kernel = "polynomial", degree = 3,scale = 0.1,
             cost = cost1,
             gamma = gamma1,
             coef0 = coef0_1)

pred_svm_os = predict(svm_os, newdata = fd_test)
cm12=confusionMatrix(table(pred_svm_os,fd_test$Class))
cm12
draw_confusion_matrix(cm12)
x13=roc.curve(fd_test$Class, pred_svm_os,curve =TRUE)
x13
plot(x13)
a13=(pr.curve(fd_test$Class, pred_nb_os,curve = TRUE))
a13
plot(a13)

```
#ANN

```{r ANN OS,warning=TRUE}
fd_NN3_o <- neuralnet(Class~.,
                     data = fd_balanced_over,
                     linear.output = FALSE,
                     err.fct = 'ce',
                     likelihood = TRUE,hidden = c(2,2))
plot(fd_NN3_o, rep = 'best')
```

```{r ANN cm}
predict_o <- predict(fd_NN3_o, fd_balanced_over[,-29])
predicted.class_o <- apply(predict,1,which.max)-1

```



```{r}
x14=roc.curve(factor(ifelse(predicted.class_o == "0", "0", "1")),
                factor(fd_balanced_over$Class),curve = TRUE)
x14
plot(x14)
#recall
```

```{r}
a14=(pr.curve(factor(ifelse(predicted.class_o == "0", "0", "1")),
                factor(fd_balanced_over$Class),curve = TRUE))
````

```
a14
plot(a14)
```
```

#clustering
```{r kmeans}
fd_balanced_over_k = ovun.sample(Class ~ ., data = fd_train, method = "over",N =
2*x11)$data
fdk_1=fd_balanced_over_k[,c(-29,-30)]

kmeans = kmeans(x = fdk_1, centers = 2)
y_kmeans =(kmeans$cluster)
clusplot(fd_balanced_over_k,
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste('Clusters of customers'))
```


#Combined Sampling

```{r combined sampling, warning=FALSE}
fd_balanced_both = ovun.sample(Class ~ ., data =fd_train, method = "both", p=0.5,N=3000,
seed = 1)$data
table(fd_balanced_both$Class)
```

#Logistic Regression
```{r logistic, warning=FALSE}
classifier_cm = glm(formula = Class ~ .,family = binomial,data =fd_balanced_both)
pred_log_cm = predict(classifier_cm, type = 'response', newdata = fd_test)
pred_log_cm_1 = ifelse(pred_log_cm > 0.5, 1, 0)

cm13=confusionMatrix(table(pred_log_cm_1,fd_test$Class))
cm13
draw_confusion_matrix(cm13)
a15=roc.curve(fd_test$Class, pred_log_cm_1,curve=TRUE)
a15
plot(a15)
x13=(pr.curve(fd_test$Class, pred_log_cm_1,curve = TRUE))
x13
plot(x13)

```

#KNN
#tuning
```{r knn tuning,warning=FALSE}
knn_cm <- train(Class~., data=fd_balanced_both, method='knn',
                tuneGrid=expand.grid(.k=1:25),metric='Accuracy',
                trControl=trainControl(method='repeatedcv', number=10,repeats=1))
knn_cm
knn_cm_df=as.data.frame(knn_cm$results)
knn_cm_optimal=max(knn_cm_df$k)
plot(knn_cm)
```
```

```
#Confusion Matrix
```{r knn cm,warning=FALSE}
pred_knn_cm = knn(train = fd_balanced_both[,-29],test = fd_test[,-29],
                  cl = fd_balanced_both[,29],
                  k = knn_cm_optimal,
                  prob = TRUE)

cm14=confusionMatrix(table(pred_knn_cm,fd_test$Class))
cm14
draw_confusion_matrix(cm14)
x16=roc.curve(fd_test$Class, pred_knn_cm,curve = TRUE)
x16
plot(x16)
a16=(pr.curve(fd_test$Class, pred_nb_os,curve = TRUE))
a16
plot(a16)

```

#Naive Bayes
#tuning
```{r nb tuning,warning=FALSE}
nb_cm = train(x = fd_balanced_both[-29],
              y = fd_balanced_both$Class,method = "nb",
              trControl = trainControl(method='repeatedcv', number=10,repeats=1),
              tuneGrid = expand.grid(usekernel = c(TRUE, FALSE),fL = 0:3,adjust =
seq(0, 3, by = 1)))
nb_cm
```

#confusion matrix
```{r nb cm,warning=FALSE}
pred_nb_cm = predict(nb_cm, newdata = fd_test)
cm15=confusionMatrix(table(pred_nb_cm,fd_test$Class))
cm15
draw_confusion_matrix(cm15)
x17=roc.curve(fd_test$Class, pred_nb_cm,curve = TRUE)
x17
plot(x17)
a17=(pr.curve(fd_test$Class, pred_nb_cm,curve = TRUE))
a17
plot(a17)
```

#Decision Tree
#Tuning
```{r dt tunning, warning=FALSE}
dt_cm = train(Class~., data=fd_balanced_both, method='rpart',
              tuneGrid=expand.grid(.cp=seq(0.00,0.03,0.001)),metric='Accuracy',
              trControl=trainControl(method='repeatedcv', number=10,repeats=3))
dt_cm
plot(dt_cm)
```

#Tuning2
```{r dt ,warning=FALSE}

fd_balanced_both_dt <- ovun.sample(Class ~ ., data = fd_train_dt, method = "both",
                                   N = 1500, seed = 1)$data
table(fd_balanced_both_dt$Class)

tree_cm = rpart(Class ~ ., data =fd_balanced_both_dt,
```
```

```
                control=rpart.control(cp = 0.002,maxdepth = 8,minsplit = 100))
tree_cm
```

#confusion matrix

```{r dt cm,warning=FALSE}
prune_cm <- prune(tree_cm, cp = 0.002)

pred_tree_cm <- predict(prune_cm, newdata = fd_test_dt)
pred_tree_cm_1 = ifelse(pred_tree_cm > 0.5, 1, 0)

cm16=confusionMatrix(table(pred_tree_cm_1,fd_test_dt$Class))
cm16
draw_confusion_matrix(cm16)
x18=roc.curve(fd_test_dt$Class, pred_tree_cm_1,curve=TRUE)
x18
plot(x18)
a18=(pr.curve(fd_test$Class, pred_tree_cm_1,curve = TRUE))
a18
plot(a18)
```

#Random forest
#tuning

```{r rf tuning,warning=FALSE}
rf_cm = train(Class~., data=fd_balanced_both, method='rf',
               tuneGrid=expand.grid(.mtry=c(1:15)),
              metric='Accuracy',trControl=trainControl(method='repeatedcv',
number=10,repeats=1))
rf_cm
plot(rf_cm)
```

#confusion Matrix
```{r fr cm,warning=FALSE}
randomforest_cm = randomForest(x = fd_balanced_both[-29],
                               y = fd_balanced_both$Class,
                               ntree=2000,mtry = 1)

pred_rf_cm = predict(randomforest_cm, newdata = fd_test)
cm17=confusionMatrix(table(pred_rf_cm,fd_test$Class))
cm17
draw_confusion_matrix(cm17)
a=roc.curve(fd_test$Class, pred_rf_cm,curve = TRUE)
a
plot(a)
a19=(pr.curve(fd_test$Class, pred_rf_cm,curve = TRUE))
a19
plot(a19)
```


#Support Vector Machine
#tuning
```{r svm tuning,warning=FALSE}
svm_tn_os <- train(Class ~., data = fd_balanced_over,
                method = "svmPoly",
                trControl=trainControl(method = "repeatedcv",
                                       number = 10, repeats = 1),
                preProcess = c("center", "scale"),
```

```
                tuneGrid = expand.grid(.degree = c(2:3),.scale = c(0.1,1,10),
                    .C = c(0,0.01, 0.05, 0.1, 0.5,1, 1.5, 2,5)),
                tuneLength = 10)
svm_tn_os
```

#tuning and confusion matrix

```{r svm tuning, warning=FALSE}
tune_out_cm = tune.svm(x = fd_balanced_both[,-29], y = fd_balanced_both[,29],
                       type = "C-classification", kernel = "polynomial", degree = 2,
                       cost = 5, gamma = c(0.1,1,10), coef0 = c(0.1,1,10))

svm_cm = svm(Class~ ., data = fd_balanced_both, type = "C-classification",
             kernel = "polynomial", degree = 2,scale =0.1,
             cost = tune_out_cm$best.parameters$cost,
             gamma = tune_out_cm$best.parameters$gamma,
             coef0 = tune_out_cm$best.parameters$coef0)

pred_svm_cm = predict(svm_cm, newdata = fd_test)
cm18=confusionMatrix(table(pred_svm_cm,fd_test$Class))
cm18
draw_confusion_matrix(cm18)
b=roc.curve(fd_test$Class, pred_svm_cm,curve = TRUE)
b
plot(b)
a20=(pr.curve(fd_test$Class, pred_svm_cm,curve = TRUE))
a20
plot(a20)
```

#Neural Network

```{r neural net library,warning=FALSE}
fd_NN_cm <- neuralnet(Class~.,
                    data = fd_balanced_both,
                    linear.output = FALSE,
                    err.fct = 'ce',
                    likelihood = TRUE,hidden = c(2,2))
plot(fd_NN_cm, rep = 'best')
```

```{r NN US,warning=FALSE}
x44=roc.curve(factor(ifelse(predicted.class_c == "0", "0", "1")),
                factor(fd_balanced_both$Class),curve = TRUE)
x44
plot(x44)
#recall

a44=(pr.curve(factor(ifelse(predicted.class_c == "0", "0", "1")),
                factor(fd_balanced_under$Class),curve = TRUE))
a44
plot(a44)
```