svm + gradient boosting + ech ROSA

library(e1071)  
library(dplyr)

##   
## Attaching package: 'dplyr'

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

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

library(caTools)  
library(ROSE)

## Loaded ROSE 0.0-3

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(xgboost)

##   
## Attaching package: 'xgboost'

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

set.seed(123)

load(file = "D:/Téléchargements/M2/fouille de données/Xtrain.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/ytrain.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/Xtest.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/ytest.Rdata")  
  
TX= cbind(Xtrain,ytrain)  
TXtrain = TX[ytrain==1,]  
ech = sort(sample(nrow(TX), 5000))  
p = TX[ech,]  
TXtrain = rbind(TXtrain,p)  
Tytrain = TXtrain$ytrain ; TXtrain = TXtrain[,-18]  
  
  
ech = sort(sample(nrow(Xtrain), nrow(Xtrain)\*.01))  
ech2 = sort(sample(nrow(Xtest), nrow(Xtest)\*.1))  
Xtrain=Xtrain[ech,]  
ytrain = ytrain[ech]  
  
Xtest = Xtest[ech2,]  
ytest = ytest[ech2]  
  
Xytrain = cbind(Xtrain,ytrain)  
table(Xytrain$ytrain)

##   
## 0 1   
## 19942 58

SVM

On commence par une méthode d’échantillonage qui combine l’oversampling ainsi que l’undersampling via le package ROSE pour rééquilibrer la classe minoritaire à 50% On ne prendra que 20.000 données étant donné qu’il est compliqué de traiter énormément de données avec des SVM de part la matrice que cela crée

Xytrain2 <- ovun.sample(ytrain ~ ., data = Xytrain, method = "both", p=0.5, seed=1)$data  
table(Xytrain2$ytrain)

##   
## 0 1   
## 10041 9959

Xtrain2 = Xytrain2[,1:17]  
ytrain2 = Xytrain2[,18]

Nous utilisons le package ’e1071’ pour l’implémentation des SVM. Nous demandons à la procédure svm() de construire un classifieur dont on centre les valeurs avec un noyau de type linéaire.

On applique cette fonction à nos données d’apprentissage.

model2 <- svm(Xtrain2, ytrain2, scale=T, type= "C-classification",kernel='linear')  
summary(model2)

##   
## Call:  
## svm.default(x = Xtrain2, y = ytrain2, scale = T, type = "C-classification",   
## kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 9361  
##   
## ( 4636 4725 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

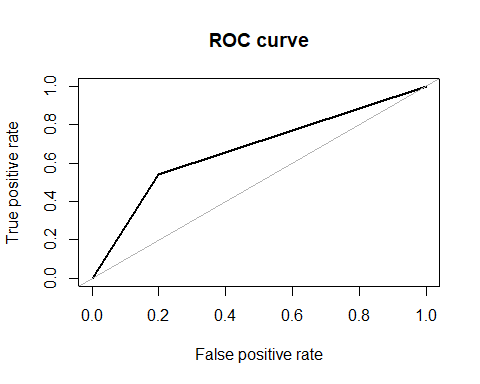
#Prédiction sur les données test  
pred2 = predict(model2, newdata = Xtest)  
#Matrice de confusion  
cm = table(pred2, ytest); cm

## ytest  
## pred2 0 1  
## 0 18473 38  
## 1 4581 45

#Taux d'erreur  
err2 = (cm[1,2] + cm[2,1])/sum(cm); err2

## [1] 0.1996369

roc.curve(ytest, pred2)



## Area under the curve (AUC): 0.672

On observe un taux d’erreur de %, ce qui est globalement assez élevé.

Cependant l’aire sous la courbe est de , ce qui est assez bon pour un premier modèle simple.

#F mesure  
cf <- confusionMatrix(pred2, ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 1.91

La F mesure est de %, cette mesure est faible, on pourrait essayer de l’améliorer en tunant le modèle.

Avant cela, nous allons essayer une autre technique d’échantillonage qui dont la fonction s’appelle ROSE du package du même nom.

Méthode ROSE du package :

Les données générées par le suréchantillonnage ont prévu une quantité d’observations répétées. Les données générées par le sous-échantillonnage sont privées d’informations importantes par rapport aux données d’origine.

Ce qui entraine des inexactitudes dans les performances résultantes. Pour faire face à ces problémes, ROSE nous aide à générer des données de manière synthétique également. Les données générées par ROSE sont considérées comme fournissant une meilleure estimation des données originales.

Xytrain3 <- ROSE(ytrain ~ ., data = Xytrain, seed = 1)$data  
table(Xytrain3$ytrain)

##   
## 0 1   
## 10041 9959

Cet ensemble nous fournit également des méthodes pour vérifier l’exactitude du modèle en utilisant la méthode de bagging et holdout.

Cela nous permet de nous assurer que nos prévisions résultantes ne souffrent pas d’une variance élevée.

ROSE.holdout <- ROSE.eval(ytrain ~ ., data = Xytrain3, learner = svm, method.assess = "holdout", extr.pred = function(obj)obj, seed = 1)  
ROSE.holdout

##   
## Call:   
## ROSE.eval(formula = ytrain ~ ., data = Xytrain3, learner = svm,   
## extr.pred = function(obj) obj, method.assess = "holdout",   
## seed = 1)  
##   
## Holdout estimate of auc: 0.899

Nous constatons que notre précision se maintient à ~ 0,89 et montre que nos prévisions ne souffrent pas d’une variance élevée.

Xtrain = Xytrain3[,1:17]  
ytrain = Xytrain3[,18]

On va donc retenter un SVM avec la méthode d’échantillonage ROSE, toujours avec un noyau linéaire et en centrant les données.

##########Kernel linéaire  
model <- svm(Xtrain, ytrain, scale=T, type= "C-classification",kernel='linear')  
summary(model)

##   
## Call:  
## svm.default(x = Xtrain, y = ytrain, scale = T, type = "C-classification",   
## kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 11803  
##   
## ( 5899 5904 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

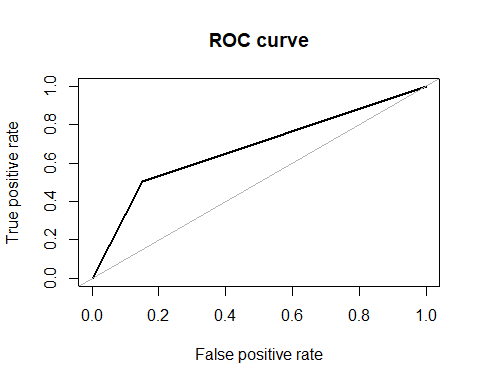
#Prédiction sur les données test  
pred = predict(model, newdata = Xtest)  
#Matrice de confusion  
cm = table(pred, ytest); cm

## ytest  
## pred 0 1  
## 0 19597 41  
## 1 3457 42

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.1511864

roc.curve(ytest, pred)



## Area under the curve (AUC): 0.678

On constate que le taux d’erreur est de %, ce qui est nettement meilleur que l’ancien modèle. Cependant l’aire sous la courbe est de , ce qui est légèrement en baisse mais cette baisse est négligeable.

#F mesure  
cf <- confusionMatrix(pred, ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 2.35

On voit que la F mesure est de , toujours meilleur que l’autre modèle. On peut donc conclure que cette méthode d’échantillonage serait meilleur pour le SVM avec des données réduites.

Nous allons tester un autre modèle de SVM simple mais cette fois avec les vraies données pour la classe minoritaire. Il y a environ 5000 données de classe minoritaire et 5000 de classe majoritaire.

##########Kernel linéaire  
model <- svm(TXtrain, Tytrain, scale=T, type= "C-classification",kernel='linear')  
summary(model)

##   
## Call:  
## svm.default(x = TXtrain, y = Tytrain, scale = T, type = "C-classification",   
## kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 6440  
##   
## ( 3223 3217 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

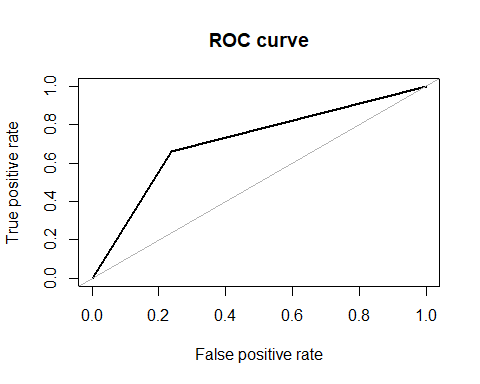
#Prédiction sur les données test  
pred = predict(model, newdata = Xtest)  
#Matrice de confusion  
cm = table(pred, ytest); cm

## ytest  
## pred 0 1  
## 0 17609 28  
## 1 5445 55

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.2365475

roc.curve(ytest, pred)



## Area under the curve (AUC): 0.713

On constate que le taux d’erreur est de %, le modèle avec l’échantillonage de la méthode ROSE était bien meilleur pour cette mesure.

Cependant l’aire sous la courbe est de , légèrement meilleur que notre précédent modèle mais rien de significatif.

#F mesure  
cf <- confusionMatrix(pred, ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 1.97

La F mesure est de , en dessous de l’ancien modèle. On pourrait donc conclure que parmi ces différentes méthodes d’échantillonage pour le SVM réalisé simplement, que celle de la méthode ROSE serait la plus efficace si on cherche à maximiser la F mesure.

On va maintenant essayer de tuner le SVM pour essayer d’améliorer nos prédictions.

Nous allons essayer d’ajuster notre modèle en tunant pour le moment deux hyperparamètres : C et Gamma.

Pour rappel, l’hyperparamètre C est responsable de la taille de la marge du MVC. Cela signifie que les points situés à l’intérieur de cette marge ne sont classés dans aucune des deux catégories. Plus la valeur de C est faible, plus la marge est importante

L’hyperparamètre gamma doit être réglé pour mieux adapter l’hyperplan aux données. Il est responsable du degré de linéarité de l’hyperplan, et pour cela, il n’est pas présent lors de l’utilisation de noyaux linéaires. Plus γ est petit, plus l’hyperplan aura l’air d’une ligne droite, tandis que si γ est trop grand, l’hyperplan sera plus courbé et pourrait trop bien délimiter les données, ce qui entraînerait un overfitting.

#Optimisation de Gamma et C  
tuned = tune.svm(x=TXtrain,  
 y=Tytrain,   
 scale=T, type = "C-classification", kernel='linear',  
 cost = 10^(-1:2),   
 gamma = c(0.1, 1, 10),  
 tunecontrol=tune.control(cross=5))  
tuned$performances

## gamma cost error dispersion  
## 1 0.1 0.1 0.2639391 0.008858174  
## 2 1.0 0.1 0.2639391 0.008858174  
## 3 10.0 0.1 0.2639391 0.008858174  
## 4 0.1 1.0 0.2634634 0.007956343  
## 5 1.0 1.0 0.2634634 0.007956343  
## 6 10.0 1.0 0.2634634 0.007956343  
## 7 0.1 10.0 0.2635585 0.008355932  
## 8 1.0 10.0 0.2635585 0.008355932  
## 9 10.0 10.0 0.2635585 0.008355932  
## 10 0.1 100.0 0.2638440 0.007653312  
## 11 1.0 100.0 0.2638440 0.007653312  
## 12 10.0 100.0 0.2638440 0.007653312

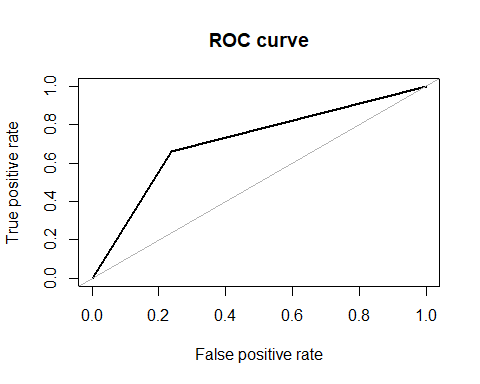
svmfit = tuned$best.model  
#Prédiction sur les données test  
pred = predict(svmfit, newdata = Xtest)  
#Matrice de confusion  
cm = table(pred, ytest); cm

## ytest  
## pred 0 1  
## 0 17609 28  
## 1 5445 55

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.2365475

roc.curve(ytest, pred)



## Area under the curve (AUC): 0.713

#F mesure  
cf <- confusionMatrix(pred, ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 1.97

XGBOOST Gradient Boosting

rm(list = ls())

load(file = "D:/Téléchargements/M2/fouille de données/Xtrain.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/ytrain.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/Xtest.Rdata")  
load(file = "D:/Téléchargements/M2/fouille de données/ytest.Rdata")  
  
TX= cbind(Xtrain,ytrain)  
TXtrain = TX[ytrain==1,]  
ech = sort(sample(nrow(TX), 5000))  
p = TX[ech,]  
TXtrain = rbind(TXtrain,p)  
Tytrain = TXtrain$ytrain ; TXtrain = TXtrain[,-18]  
Txy=cbind(TXtrain,Tytrain)  
  
rm(ech,p,TXtrain,Tytrain,TX)  
  
ech = sort(sample(nrow(Xtrain), nrow(Xtrain)\*.25))  
ech2 = sort(sample(nrow(Xtest), nrow(Xtest)\*.1))  
XYtrain=cbind(Xtrain[ech,],ytrain[ech])  
colnames(XYtrain)[18] = "ytrain"  
  
XYtest = cbind(Xtest[ech2,], ytest[ech2])  
colnames(XYtest)[18] = "ytest"  
  
table(XYtrain$ytrain)

##   
## 0 1   
## 498647 1353

rm(ech,ech2)

Gradient boosting avec 50/50 mais vraies données

xgb\_Xtrain = xgb.DMatrix(as.matrix(Txy %>% select(-Tytrain)))  
xgb\_ytrain = Txy$Tytrain  
  
#Recode en char car XGboost ne prend pas 0 et 1 en binaire  
xgb\_ytrain = recode(xgb\_ytrain, "0"="A", "1"="B")  
  
xgb\_Xtest = xgb.DMatrix(as.matrix(XYtest %>% select(-ytest)))  
xgb\_ytest = XYtest$ytest

On va donc commencer par définir un objet trainControl, qui permet de contrôler la manière dont se fait l’entraînement du modèle, assuré par la fonction train().

Ici, nous choisissons une validation croisée (method = ‘cv’) à 2 folds (number = 2). On choisit également d’autoriser la parallélisation des calculs (allowParallel = TRUE), de réduire la verbosité (verboseIter = FALSE).

xgb\_trcontrol = trainControl(method = "cv", number = 2, allowParallel = TRUE,   
 verboseIter = FALSE, returnData = FALSE, summaryFunction = twoClassSummary,classProbs = TRUE)

On définit ensuite une grille de paramètres du modèle XGBoost appelée xgbGrid

xgbGrid <- expand.grid(nrounds = c(100,200),   
 max\_depth = c(3, 5, 10, 15, 20),  
 colsample\_bytree = seq(0.5, 0.9, length.out = 5),  
 ## valeurs par défaut :   
 eta = 0.1,  
 gamma=0,  
 min\_child\_weight = 1,  
 subsample = 1  
 )

nrounds: nombre d’itérations de boosting à effectuer. Plus il est grand, et plus c’est lent max\_depth: profondeur d’arbre maximale. Risque d’over-fit si trop grand, et d’under-fit si trop petit colsample\_bytree: pourcentage des colonnes pris pour construire un arbre (rappelle-toi, un arbre est construit avec un sous-ensemble des données: lignes et colonnes) eta: ou learning rate, ce paramètre contrôle la vitesse à laquelle on convergence lors de la descente du gradient fonctionnelle (par défaut = 0.3) gamma: diminution minimale de la valeur de la loss (fonction objectif) pour prendre la décision de partitionner une feuille

xgb\_model = train(xgb\_Xtrain, xgb\_ytrain, trControl = xgb\_trcontrol, tuneGrid = xgbGrid,   
 method = "xgbTree",metric = "ROC")

xgb\_model$bestTune

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 11 100 5 0.1 0 0.5 1 1

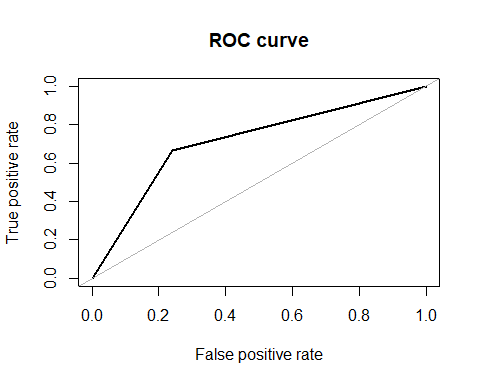
pred = predict(xgb\_model, xgb\_Xtest)  
pred = recode(pred, "A"=0,"B"=1)  
pred=as.factor(pred)  
#Matrice de confusion  
cm = table(pred, xgb\_ytest); cm

## xgb\_ytest  
## pred 0 1  
## 0 17509 26  
## 1 5550 52

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.2409993

roc.curve(xgb\_ytest, pred)



## Area under the curve (AUC): 0.713

#F mesure  
cf <- confusionMatrix(pred, xgb\_ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 1.83

Test avec méthode d’échantillonage over/undersampling avec autant de classe minoritaire que majoritaire

XYtrain <- ovun.sample(ytrain ~ ., data = XYtrain, method = "both", p=0.5, seed=1)$data  
table(XYtrain$ytrain)

##   
## 0 1   
## 249890 250110

xgb\_Xtrain = xgb.DMatrix(as.matrix(XYtrain %>% select(-ytrain)))  
xgb\_ytrain = XYtrain$ytrain  
  
#Recode en char car XGboost ne prend pas 0 et 1 en binaire  
xgb\_ytrain = recode(xgb\_ytrain, "0"="A", "1"="B")  
  
xgb\_Xtest = xgb.DMatrix(as.matrix(XYtest %>% select(-ytest)))  
xgb\_ytest = XYtest$ytest

xgb\_trcontrol = trainControl(method = "cv", number = 2, allowParallel = TRUE,   
 verboseIter = FALSE, returnData = FALSE, summaryFunction = twoClassSummary,classProbs = TRUE)

xgbGrid <- expand.grid(nrounds = c(100,200),   
 max\_depth = c(3, 5, 10, 15, 20),  
 colsample\_bytree = seq(0.5, 0.9, length.out = 5),  
 ## valeurs par défaut :   
 eta = 0.1,  
 gamma=0,  
 min\_child\_weight = 1,  
 subsample = 1  
 )

xgb\_model = train(xgb\_Xtrain, xgb\_ytrain, trControl = xgb\_trcontrol, tuneGrid = xgbGrid,   
 method = "xgbTree",metric = "ROC")

xgb\_model$bestTune

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 42 200 20 0.1 0 0.5 1 1

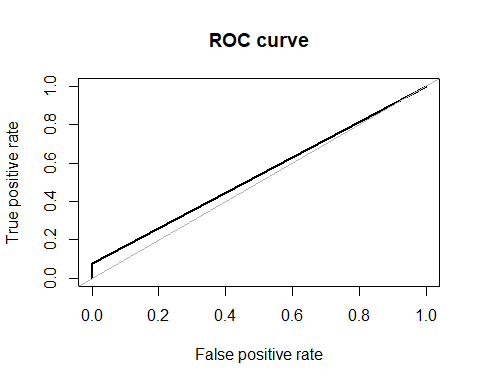
pred = predict(xgb\_model, xgb\_Xtest)  
pred = recode(pred, "A"=0,"B"=1)  
pred=as.factor(pred)  
#Matrice de confusion  
cm = table(pred, xgb\_ytest); cm

## xgb\_ytest  
## pred 0 1  
## 0 23039 72  
## 1 20 6

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.003976315

roc.curve(xgb\_ytest, pred)



## Area under the curve (AUC): 0.538

#F mesure  
cf <- confusionMatrix(pred, xgb\_ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 11.54

Test avec méthode d’échantillonage over/undersampling avec un ratio de 75/25

ech = sort(sample(nrow(Xtrain), nrow(Xtrain)\*.25))  
ech2 = sort(sample(nrow(Xtest), nrow(Xtest)\*.1))  
XYtrain=cbind(Xtrain[ech,],ytrain[ech])  
colnames(XYtrain)[18] = "ytrain"

XYtrain <- ovun.sample(ytrain ~ ., data = XYtrain, method = "both", p=0.25, seed=1)$data  
table(XYtrain$ytrain)

##   
## 0 1   
## 375268 124732

xgb\_Xtrain = xgb.DMatrix(as.matrix(XYtrain %>% select(-ytrain)))  
xgb\_ytrain = XYtrain$ytrain  
  
#Recode en char car XGboost ne prend pas 0 et 1 en binaire  
xgb\_ytrain = recode(xgb\_ytrain, "0"="A", "1"="B")  
  
xgb\_Xtest = xgb.DMatrix(as.matrix(XYtest %>% select(-ytest)))  
xgb\_ytest = XYtest$ytest

xgb\_trcontrol = trainControl(method = "cv", number = 2, allowParallel = TRUE,   
 verboseIter = FALSE, returnData = FALSE, summaryFunction = twoClassSummary,classProbs = TRUE)

xgbGrid <- expand.grid(nrounds = c(100,200),   
 max\_depth = c(3, 5, 10, 15, 20),  
 colsample\_bytree = seq(0.5, 0.9, length.out = 5),  
 ## valeurs par défaut :   
 eta = 0.1,  
 gamma=0,  
 min\_child\_weight = 1,  
 subsample = 1  
 )

xgbGrid <- expand.grid(nrounds = 200,   
 max\_depth = 20,  
 colsample\_bytree = 0.5,  
 ## valeurs par défaut :   
 eta = 0.1,  
 gamma=0,  
 min\_child\_weight = 1,  
 subsample = 1  
 )

xgb\_model = train(xgb\_Xtrain, xgb\_ytrain, trControl = xgb\_trcontrol, tuneGrid = xgbGrid,   
 method = "xgbTree",metric = "ROC")

xgb\_model$bestTune

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 1 200 20 0.1 0 0.5 1 1

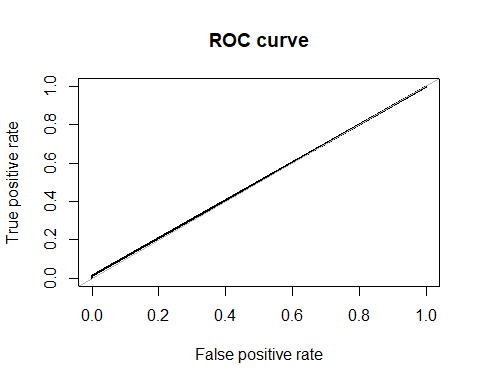
pred = predict(xgb\_model, xgb\_Xtest)  
pred = recode(pred, "A"=0,"B"=1)  
pred=as.factor(pred)  
#Matrice de confusion  
cm = table(pred, xgb\_ytest); cm

## xgb\_ytest  
## pred 0 1  
## 0 23050 77  
## 1 9 1

#Taux d'erreur  
err = (cm[1,2] + cm[2,1])/sum(cm); err

## [1] 0.00371699

roc.curve(xgb\_ytest, pred)



## Area under the curve (AUC): 0.506

#F mesure  
cf <- confusionMatrix(pred, xgb\_ytest, mode = "prec\_recall", positive = '1')$byClass[7]  
print(round(cf\*100,2))

## F1   
## 2.27