Classification dans un contexte déséquilibré Une application à la fraude bancaire M2 SISE

Barou Axelle - Delzant Tanguy - Picard Amelie

25/01/2021

Table des matières

[Introduction 3](#_Toc62417093)

[Travail préliminaire 3](#_Toc62417094)

[Analyse synthétique et pre-processing 4](#_Toc62417095)

[Protocole expérimentale 8](#_Toc62417096)

[La méthode Bagging 8](#_Toc62417097)

[La méthode SVM 13](#_Toc62417098)

[Principe 13](#_Toc62417099)

[Test de la méthode sur divers échantillons 13](#_Toc62417100)

[Gradient Tree Boosting 18](#_Toc62417101)

[Principe 18](#_Toc62417102)

[Test de la méthode sur divers échantillons 18](#_Toc62417103)

# Introduction

Le but de ce projet est d’étudier des données issues d’une enseigne de la grande distribution ainsi que de certains organismes bancaires. Les données représentent des transactions effectuées par chèque dans un magasin de l’enseigne quelque part en France.  
La variable à prédire est la variable FlagImpaye, il s’agit d’une variable binaire qui peut prendre les valeurs suivantes : 0 la transaction est acceptée et considérée comme “normale”, 1 la transaction est refusée car considérée comme “frauduleuse”.

# Travail préliminaire

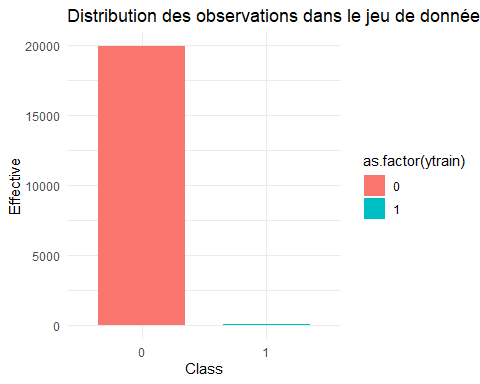
Le jeu de données est composé de 2 231 369 transactions et 23 variables. Lors du chargement des données, il a fallu effectuer un travail d’harmonisation.

## 'data.frame': 2231369 obs. of 23 variables:  
## $ ZIBZIN : chr "A034010041908012010710730" "A035010041908006493331734" "A013010003908005150136747" "A013010041908025639221029" ...  
## $ IDAvisAutorisAtionCheque: int 71051532 71051533 71051534 71051536 71051538 71051539 71051541 71051542 71051544 71051548 ...  
## $ MontAnt : chr "40,170000000000002" "20" "35" "20" ...  
## $ DAteTrAnsAction : chr "2016-03-21 07:47:38" "2016-03-21 08:04:57" "2016-03-21 08:06:45" "2016-03-21 08:11:38" ...  
## $ CodeDecision : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ VerifiAnceCPT1 : int 1 0 0 0 0 0 0 1 1 0 ...  
## $ VerifiAnceCPT2 : int 1 0 0 0 0 0 0 1 1 0 ...  
## $ VerifiAnceCPT3 : int 1 0 0 0 0 0 0 1 1 0 ...  
## $ D2CB : int 535 358 199 38 26 459 500 389 538 13 ...  
## $ ScoringFP1 : chr "0" "0" "0" "0" ...  
## $ ScoringFP2 : chr "0" "0" "0" "0" ...  
## $ ScoringFP3 : chr "0" "0" "0" "0" ...  
## $ TAuxImpNb\_RB : chr "53,937432578209275" "44,742729306487696" "17,094017094017094" "41,878552198623993" ...  
## $ TAuxImpNB\_CPM : chr "21,834061135371179" "12,586532410320956" "39,274924471299094" "39,274924471299094" ...  
## $ EcArtNumCheq : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ NbrMAgAsin3J : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ DiffDAteTr1 : chr "4" "4" "4" "4" ...  
## $ DiffDAteTr2 : chr "4" "4" "4" "4" ...  
## $ DiffDAteTr3 : chr "4" "4" "4" "4" ...  
## $ CA3TRetMtt : chr "40,170000000000002" "20" "35" "20" ...  
## $ CA3TR : chr "0" "0" "0" "0" ...  
## $ Heure : int 28058 29097 29205 29498 29831 29846 29996 30042 30064 30176 ...  
## $ FlAgImpAye : int 0 0 0 0 0 0 0 0 0 0 ...

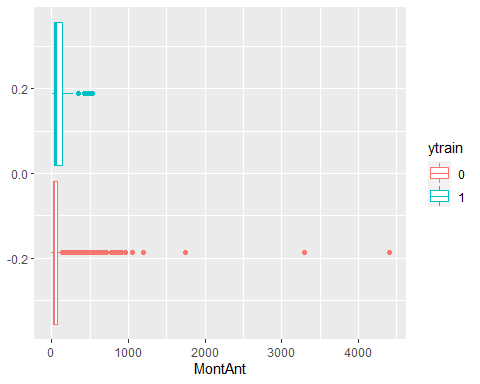
En effet, comme le montre le résultat précédent, certaines variables numériques sont de type chaîne de caratère. Nous avons donc commencé par remplacer les virgules par des points pour les variables numériques et nous les avons convertis en type “numeric”. Nous avons ensuite arrondi les montant à deux décimales. Nous avons aussi recodé la variable d’intéret, FlAgImpAye, en variable de type facteur.  
De plus, nous avons décidé de supprimer les variables que nous n’avons pas considéré pertinentes pour notre étude. Ainsi, la nouvelle structure des données avec les variables sélectionnées est la suivante :

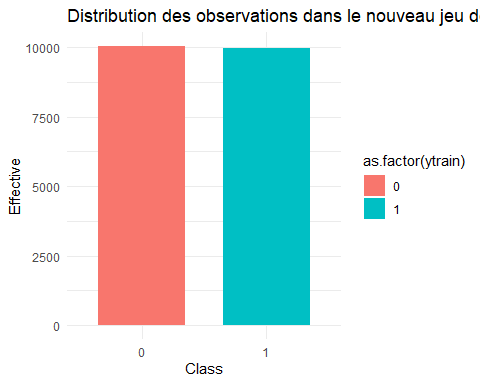
## 'data.frame': 20000 obs. of 18 variables:  
## $ MontAnt : num 37.5 36 34.8 18.5 65.1 ...  
## $ VerifiAnceCPT1: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ VerifiAnceCPT2: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ VerifiAnceCPT3: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ D2CB : int 264 543 1 457 335 466 270 482 529 530 ...  
## $ ScoringFP1 : num 0.02 0 4.98 0 0.01 0 0.27 0 0 0 ...  
## $ ScoringFP2 : num 6.29 0 12.72 -1.49 12.73 ...  
## $ ScoringFP3 : num 0 0 0.47 0 0 0 0 0 0 0 ...  
## $ TAuxImpNb\_RB : num 23.7 75.5 41.9 28.2 0 ...  
## $ TAuxImpNB\_CPM : num 39.3 24.2 13.7 258.8 24.2 ...  
## $ EcArtNumCheq : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ NbrMAgAsin3J : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ DiffDAteTr1 : num 4 4 5 4 4 4 4 4 4 4 ...  
## $ DiffDAteTr2 : num 4 4 5 4 4 4 4 4 4 4 ...  
## $ DiffDAteTr3 : num 4 4 5 4 4 4 4 4 4 4 ...  
## $ CA3TRetMtt : num 37.5 36 34.8 18.5 65.1 ...  
## $ CA3TR : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ ytrain : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...

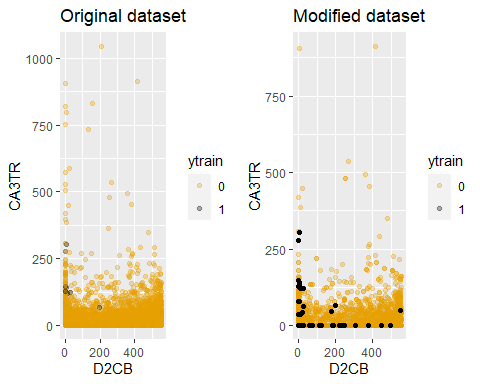
# Analyse synthétique et pre-processing

Les données sur lesquelles nous travaillons sont dites déséquilibrées. En effet, le ratio des observations de la classe positive (opération frauduleuse) est très faible. Une approche naïve de classification qui ne prendrait pas en compte ce déséquilibre des classes et risquerais fortement de biaiser le modèle. 

En effet, seulement 3% des observations sont frauduleuses, et donc appartiennent à la classe positive. Pour repérer ces opérations, nous aurions pu penser que les montants des opérations frauduleuses sont plus élevés que les autres mais le graphique suivant montre que ce n’est pas le cas.



Nous allons donc procéder à un rééchantillonnage pour tenter de résoudre le problème de déséquilibre. Il existe plusieurs stratégies de rééchantillonnage pour ajuster la distribution des classes d’un jeu de données : l’oversampling et l’undersampling.  
L’oversampling consiste à générer de nouvelles observations de la classe minoritaire. L’algorithme le plus utilisé est SMOTE qui génère de nouvelles observations entre des individus de la plus petite classe.  
L’undersampling consiste à ré-échantillonner la classe majoritaire de manière à obtenir un effectif proche de la classe minoritaire. L’idée est donc de supprimer les observations de la classe majoritaire.  
Il est aussi possible de combiner ces deux approches. Nous avons donc choisi cette dernière option en rééchantillonnant les données à l’aide de la fonction “ovun.sample” avec une probabilité de rééchantillonnage à partir de la classe minoritaire de 50%. Nous nous sommes donc retrouvés avec un jeu de données avec une répartition identique de la classe majoritaire et de la classe minoritaire. Le temps de calcul étant très long et les données très volumineuses, nous avons au préalable isolé les 2 000 000 premières lignes pour le jeu d’apprentissage et 231 370 lignes pour le jeu de test. Nous avons ensuite tiré aléatoirement 20000 lignes du jeu d’apprentissage et avons rééquilibré ce jeu de données de 20000 lignes. 

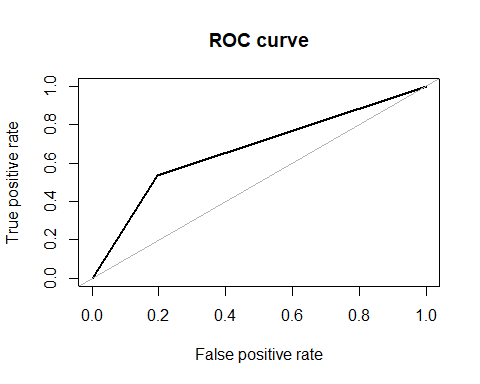
Nous pouvons ensuite observer la génération de nouvelles données avec un graphique de comparaison. 

La première image montre les données initiales. On remarque beaucoup de données de la classe 0 et très peu de la classe 1. Dans la deuxième image, on peut voir que certaines données de la classe 0 ont été supprimées et que des données de la classe 1 ont été générées.

# Protocole expérimentale

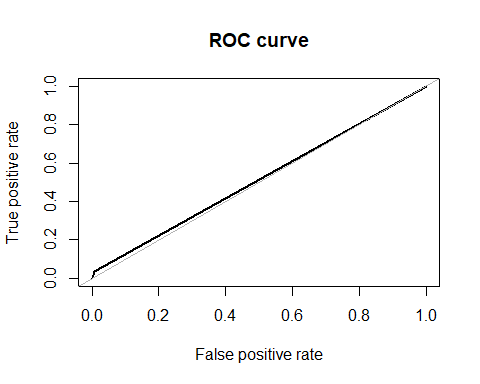
Pour résoudre ce problème de classification, nous allons tester plusieurs méthodes supervisés et non supervisés que nous allons entraîner sur le jeu d’apprentissage précédemment rééchantillonné contenant 20000 données. Nous allons tenter d’optimiser chacune de nos méthodes puis nous sélectionnerons celle qui fonctionne le mieux sur notre jeu de test comportant les 23137 dernières données du jeu de données initial. Nous évaluerons les performances de nos modèles à l’aide de la F-mesure et de l’AUC.

## La méthode Bagging

Le bagging est une méthode générale pour ajuster plusieurs versions d’un modèle de prédiction, puis les combiner en une prédiction agrégée. L’idée est de faire coopérer plusieurs arbres. En effet, le bagging repose sur le fait que l’agrégation d’informations dans de grands groupes diversifiés aboutit souvent à des meilleures décisions que celles qui auraient pu être prises par un seul membre du groupe.  
Nous avons utilisé la fonction bagging du package adabag. Les paramètres de cette fonction sont “mfinal” qui désigne le nombre d’itérations pour lesquelles le boosting est exécuté ou le nombre d’arbres à utiliser et “control” qui désigne les options qui contrôlent les détails de l’algorithme de construction des arbres. Nous avons d’abord tenté de faire un bagging avec les paramètres par défaut de la méthode bagging. C’est-à-dire avec 100 arbres utilisés. 

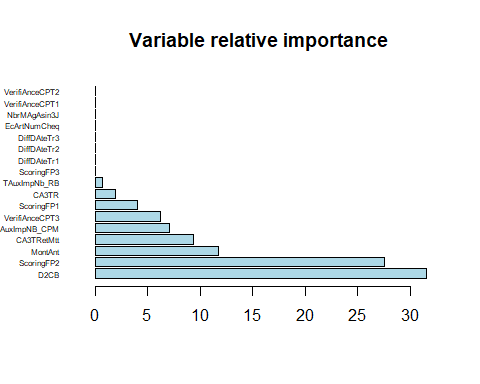
## Area under the curve (AUC): 0.669

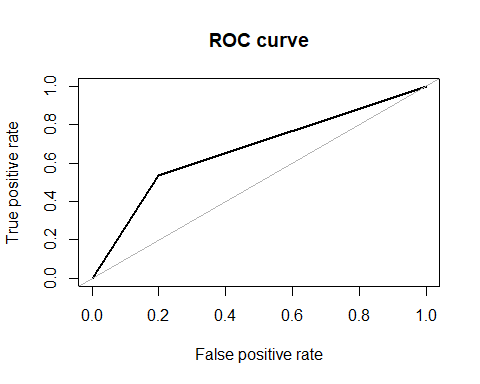
Nous obtenons une F mesure de 0.0198 et une AUC de 0.6693.

Nous avons donc essayé de créer des arbres plus profonds en spécifiant quelques options. Nous définissons une profondeur maximale de 30 et un nombre d’arbre toujours égal à 100. 

## Area under the curve (AUC): 0.514

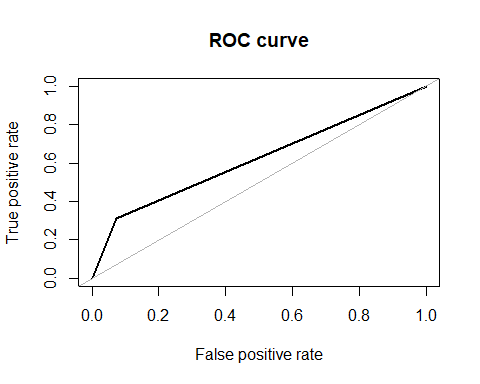
Nous obtenons alors une F-mesure de 0.0243 mais un AUC de 0.514. L’AUC désigne la capacité à retrouver la classe positive. Notre modèle détecte donc moins bien la classe positive que le précédent. D’ailleurs, il détecte aussi bien que ferait l’aléatoire.

Nous allons donc repartir sur le premier modèle et tenter de l’améliorer. En effet, nous remarquons que certaines variables ont une influence nulle sur notre modèle 

Nous allons donc les supprimer du modèle et voir comment évoluent les performances 

## Area under the curve (AUC): 0.668

La F-mesure étant de 0.0194 et l’AUC de 0.6677, les performances n’ont pas été améliorées.

Nous allons donc tenter de combiner les deux premiers modèles. Pour cela, nous récupérons les votes de chaque arbre dans les deux modèles et nous attribuons la classe à l’aide d’un vote de majorité. 

## Area under the curve (AUC): 0.620

Nous obtenons alors une F-mesure de 0.0299 se situant entre les 2 modèles et une AUC de 0.6202.  
Ce modèle semble donc être un bon compromis entre la F-mesure et l’AUC issu des deux premiers modèles. Cependant, nous concluons que la méthode de bagging ne donne pas de très bons résultats sur notre jeu de données. Nous allons donc tenter d’autres modèles.

## La méthode SVM

### Principe

Les SVMs sont une famille d’algorithmes d‘apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d’anomalie. Ils ont pour but de séparer les données en classes à l’aide d’une frontière aussi « simple » que possible, de telle façon que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale. Cette distance est aussi appelée « marge » et les SVMs sont ainsi qualifiés de « séparateurs à vaste marge », les « vecteurs de support » étant les données les plus proches de la frontière.

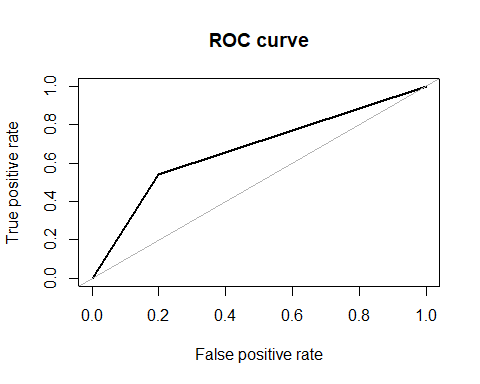
Nous utilisons le package ’e1071’ pour l’implémentation des SVM. Ce package nous permettra d'utiliser les SVM, de faire nos validations croisées et nos prédictions. On pourra y faire varier nos valeurs de Gamma et C.

### Test de la méthode sur divers échantillons

On commence par une méthode d’échantillonnage 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. 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.

## 0 1   
## 10041 9959



## Area under the curve (AUC): 0.672

On observe un taux d’erreur de 19.9%, ce qui est globalement assez élevé.

Cependant l’aire sous la courbe est de 67,2%, ce qui est assez correcte pour un premier modèle simple.

La F mesure est de 1.91%, cette mesure est faible, on pourrait essayer de l’améliorer en tunant le modèle.

Avant cela, nous allons essayer une autre technique d’échantillonnage 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 problèmes, 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.

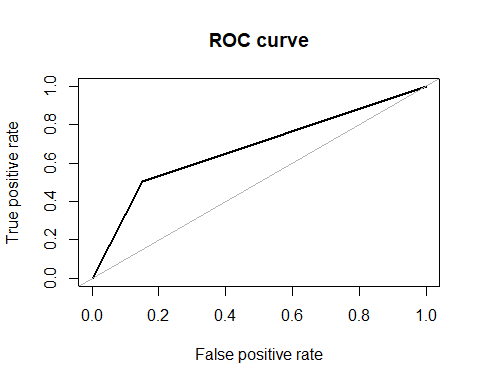
Cet ensemble nous fournit également des méthodes pour vérifier l’exactitude du modèle en utilisant la méthode de bagging et holdout.

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

## 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.

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



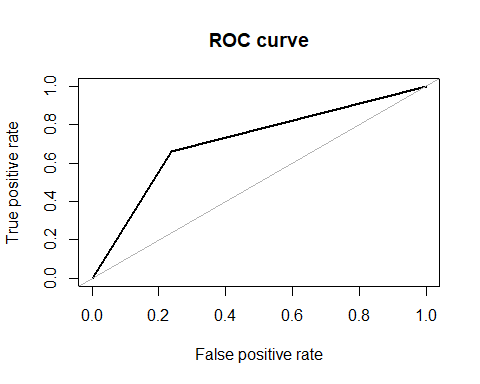
## Area under the curve (AUC): 0.678

On constate que le taux d’erreur est de 15.1%, ce qui est nettement meilleur que l’ancien modèle. Cependant l’aire sous la courbe est de 67.8%, ce qui est très similaire à l’ancien modèle.

## F1   
## 2.35

On voit que la F mesure est de 2.35%, toujours meilleur que l’autre modèle. On peut donc conclure que cette méthode d’échantillonnage serait meilleure 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.



## Area under the curve (AUC): 0.713

On constate que le taux d’erreur est de 23.6%, le modèle avec l’échantillonnage de la méthode ROSE était bien meilleur pour cette mesure.

Cependant l’aire sous la courbe est de 71.3%, légèrement meilleur que notre précédent modèle mais rien de significatif.

## F1   
## 1.97

La F mesure est de 1.97%, en dessous de l’ancien modèle. On pourrait donc conclure que parmi ces différentes méthodes d’échantillonnage 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.

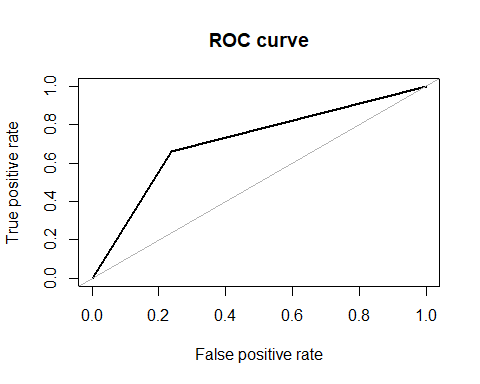
Nous allons essayer d’ajuster notre modèle en tunant pour le moment deux hyperparamètres : C et Gamma.

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.

On fera varier C de manière logarithmique (0.1,1,10,100) et de même pour Gamma (0.1, 1, 10)

#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))



## Area under the curve (AUC): 0.713

## F1   
## 1.97

On constate au vu des métriques, qu’elles sont identiques au modèle precedent. Tuné notre modèle pour trouver nos meilleurs hyperparamètres n’aura donc pas eu d’incidence.

On peut donc conclure que la méthode SVM n’est pas adaptée à ce jeu de données, car cela ne donne pas les résultats escomptés.

## Gradient Tree Boosting

### Principe

Nous allons maintenant nous pencher sur une autre méthode, celle du Gradient Tree Boosting. Pour cela, nous utiliserons la librairie XGBoost.

XGBoost fait partie de la famille des méthodes ensembliste. La différence par rapport aux méthodes classiques, c’est qu’au lieu d’entraîner le meilleur modèle possible sur les données, on va en entraîner des milliers sur des sous-parties diverses du jeu de données d’apprentissage, puis les faire voter pour prendre notre décision.

Le principe du boosting est d’améliorer la qualité de prédiction d’un modèle médiocre (weak learner) en donnant de plus en plus de poids aux valeurs difficiles à prédire au cours de l’apprentissage. Ainsi, on oblige le modèle à s’améliorer.

### Test de la méthode sur divers échantillons

Pour notre premier modèle, nous utiliserons cette fois les vraies données pour la classe minoritaire. Il y a environ 5000 données de classe minoritaire et 5000 de classe majoritaire.

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), pour éviter un trop long temps d’exécution. On est conscient que que cela reste trop faible pour faire une moyenne, mais avec 10 folds, il serait impossible de produire des résultats. 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  
 )

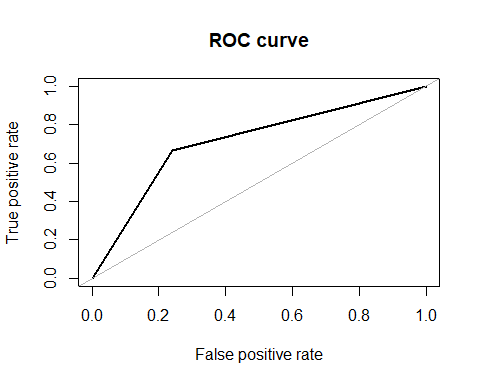
**nrounds**: nombre d’itérations de boosting à effectuer. Plus il est grand, et plus c’est lent.

**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.

**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 pour prendre la décision de partitionner une feuille.



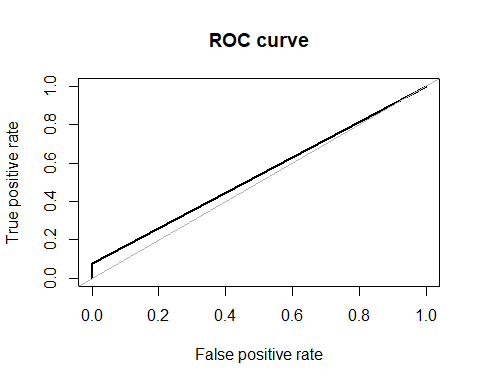
## Area under the curve (AUC): 0.713

## F1   
## 1.83

On obtient un taux d’erreur de 24% et un AUC de 71.3%. La F mesure est de 1.83%. Ce modèle avec ce type de données ne donne pas de bons résultats, il faudrait essayer avec plus de données couple à de l’oversampling et/ou de l’undersampling.

Nous allons maintenant tester un nouveau modèle mais cette fois avec 500.000 données. Nous rééquilibrerons nos données avec une méthode d’oversampling et undersampling combiné pour un ratio de 50/50. Nous gardons les mêmes paramètres que l’ancien modèle pour tester un maximum de valeurs pour les différents hyperparamètres.

## 0 1   
## 249890 250110



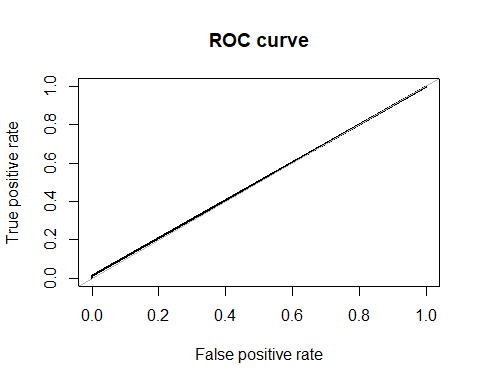
## Area under the curve (AUC): 0.538

## F1   
## 11.54

On obtient un taux d’erreur de 0.03% mais un AUC de 53.8%. L’AUC est très mauvais pour notre modèle, cependant la F mesure est excellente. Dans notre cas, puisqu’on cherche à maximiser la F mesure, on pourrait commencer à retenir ce modèle pour traiter notre problème de fraude.

On va essayer un autre modèle, toujours avec la même plage de valeur pour les hyperparamètres, et le même nombre de données, mais cette fois-ci, nous ferons un ratio de 75% de classe majoritaire et 25% de classe minoritaire.

## 0 1   
## 375268 124732



## Area under the curve (AUC): 0.506

## F1   
## 2.27

Le taux d’erreur est de 0.03%. L’AUC est de 50.6% tandis que la F mesure est de 2.27%. Ce modèle n’est vraiment pas bon par rapport à l’ancien dans notre recherche de maximisation de la F mesure.

Concernant le Gradient Tree Boosting, on pourrait retenir ce modèle pour répondre à notre problème vu qu’il produit de bon résultat. Il faudrait lui faire apprendre un maximum de données d’apprentissage tout en équilibrant parfaitement les deux classes. On pensait déjà avoir de bons résultats avec cette méthode car XGBoost a une excellente précision et s’adapte bien à tous types de données et de problématiques, ce qui en fait l’algorithme idéal quand la performance et la rapidité priment.