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

Barou Axelle - Delzant Tanguy - Picard Amelie

20/01/2021

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

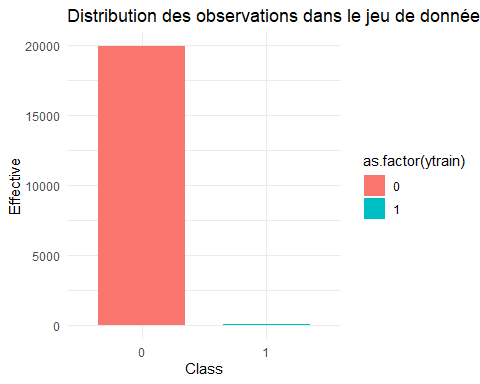
Le jeu de données est composé de 2 231 369 transactions et 23 variables. Lors du chargement des données, il a fallu effectué un travail d’hamonisation.

## '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 ...

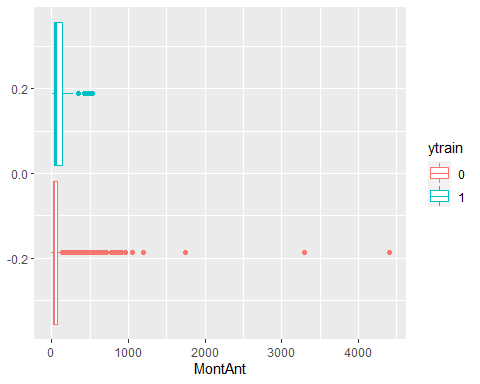
En effet, comme le montre le résultat précédent, certaines variables numériques sont de type chaîne de caratère. Nous avons donc commencé par remplacer les virgules par des points pour les variables numériques et nous les avons convertit en type “numéric”. Nous avons ensuite arrondi les montant à deux 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:

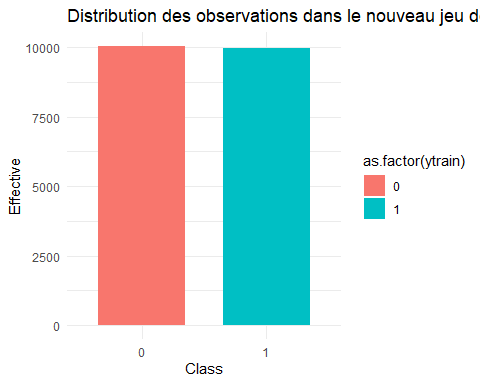
## 'data.frame': 20000 obs. of 18 variables:  
## $ MontAnt : num 37.5 36 34.8 18.5 65.1 ...  
## $ VerifiAnceCPT1: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ VerifiAnceCPT2: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ VerifiAnceCPT3: num 0 0 0 0 0 1 1 1 0 0 ...  
## $ D2CB : int 264 543 1 457 335 466 270 482 529 530 ...  
## $ 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 ...

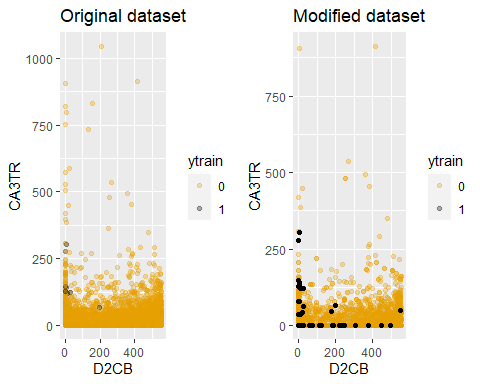
# Analyse synthétique et pre-processing

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 prendrais 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 ré-échantillonnage pour tenter de résoudre le problème de désèquilibre. Il existe plusieurs stratégies de ré-échantillonnage 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 génére de nouvelles observations entre des individus de la plus petite classe.  
L’undersampling consiste à ré-échantilloner 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 options en ré-échantillonnant les données à l’aide de la fonction “ovun.sample” avec une probabilié de rééchantillonnage à partir de la classe minoritaire de 50%. Nous nous sommes donc retrouvé 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. 

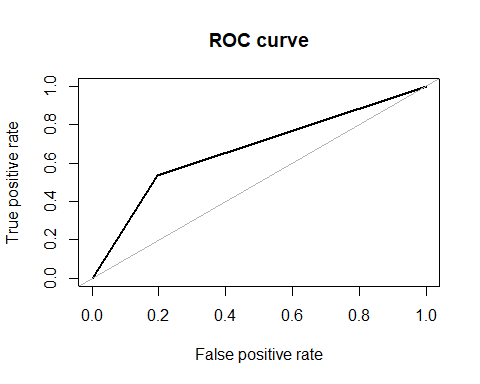
Nous pouvons ensuite observer la génération de nouvelles données avec un graphique de comparaison. 

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

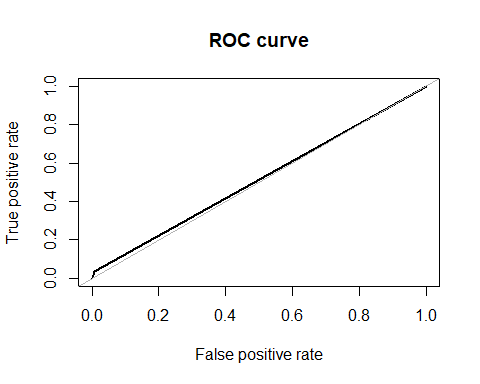
Pour résoudre ce problème de classification, nous allons tester plusieurs méthodes supervisés et non supervisés que nous allons entraîner sur le jeu d’apprentissage précédemment ré-échantilloné contenant 20000 données. Nous allons tenter d’optimiser chacune de nos méthodes puis nous sélectionnerons celle 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 meilleurs 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. 

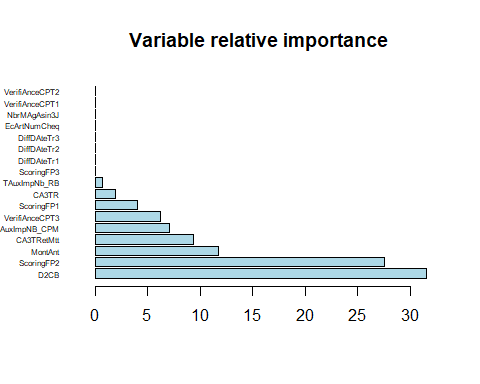
## Area under the curve (AUC): 0.669

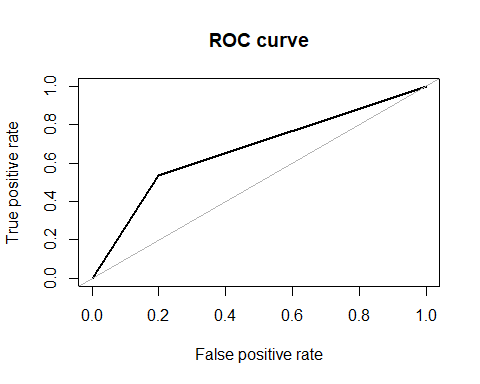
Nous obtenons une F mesure de 0.0198 et une AUC de 0.6693.

Nous avons donc essayé de créer des arbres plus profond 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

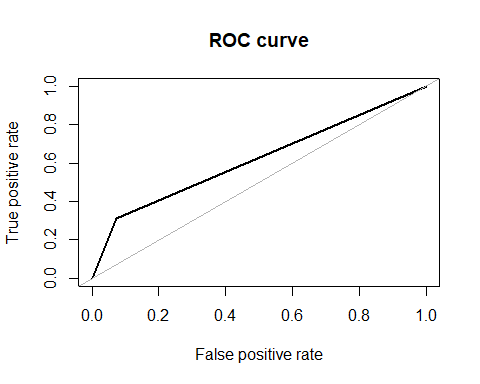
Nous obtenons alors une F-mesure de 0.0243 mais un AUC de 0.514. L’AUC désigne la capacité à retrouvé la classe positive. Notre modèle détecte donc moins bien la classe positive que le précédent. D’ailleurs, il détecte aussi bien que ferait l’aléatoire.

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 arbres 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 bon résultats sur notre jeu de données. Nous allons donc tenter d’autres modèles.