Fouilles Donnees Massives

Pauline Attal

Axel Eutarici

15 Janvier 2023

GitHub

Résumé Les fraudes bancaires représentent un problème de classification difficile dû à l'inégalité des deux classes observées (vraies transactions et fraudes). Après le nettoyage des données et l'analyse descriptive nous nous sommes intéressés à plusieurs méthodes d'under-sampling et d'over-sampling décrites dans la littérature ainsi qu'à plusieurs algorithmes supervisés afin de prédire si une nouvelle transaction sera frauduleuse ou non. Nous nous sommes surtout intéressées à l'application de la combinaison des deux méthodes SMOTE et TomekLinks pour le re-sampling de nos données. Puis nous verrons que l'algorithme du Gradient Tree Boosting à été le plus efficace pour notre problématique de prédiction si l'on se base sur la métrique d'évaluation de la F-mesure.

Table des matières

1 Introduction

Dans ce rapport, nous allons présenter une démarche de fouille de données massives dans le but de déterminer si une transaction par chèque est frauduleuse ou non. Pour ce faire, et à terme, nous allons appliquer un algorithme d'apprentissage automatique sur de nouvelles données. Cet algorithme aura été entraîné et testé avec notre jeu de données qui fait l'étude de ce rapport. L'apprentissage automatique est une branche de l'intelligence artificielle qui se fonde sur des techniques mathématiques, statistiques, et informatiques. Quand nous parlons de fouilles de données, nous faisons aussi référence à l'analyse descriptive de données, au nettoyage de données, au ré-échantillonnage, et également au re-codage de celles-ci pour que nos données soient conformes pour les algorithmes d'apprentissage automatiques.

Les données sur lesquelles nous travaillons sont des données réelles. Elles sont issues d'une enseigne de la grande distribution ainsi que de certains organismes bancaires (FNCI et Banque de France).

D'après le Nilson Report[1] les fraudes bancaires ont fait perdre $28.58 Milliards en 2020 dans le monde. Bien qu'elles soient en baisse depuis le pic de 2016 elles restent une problématique en vogue de nos jours.

Nous nous concentrons dans ce rapport sur la détection de la fraude par chèque qui consiste à utiliser de faux chèques pour payer des produits coûteux. Les fraudeurs modifient les informations des vrais chèques pour en produire de faux. Il existe quatre types de chèques frauduleux :

- **Le faux chèque** : il s'agit d'un chèque vierge, perdu ou volé, sur lequel une fausse signature est apposée par le fraudeur ;
- **Le chèque falsifié** : il s'agit d'un chèque régulièrement émis par le titulaire du compte et falsifié par l'escroc, généralement après avoir été volé. La falsification peut porter sur le montant, le nom du bénéficiaire et/ou la ligne magnétique (zone de chiffres en bas du chèque). Elle peut être apparente (rature, surcharge, gommage...) ou quasi-indécelable ;
- **Le chèque détourné** : c'est un chèque régulièrement émis par le titulaire, perdu ou volé lors de son acheminement vers le bénéficiaire, et remis à l'encaissement sur le compte de la personne qui l'a détourné (sans falsification) ;
- **Le chèque contrefait** : c'est la fabrication ou la reproduction, par des faussaires, d'un chèque à partir d'un modèle existant.

La détection de fraude dans une enseigne de grande distribution connue cause deux principales difficultés :

1. Le déséquilibre des classes pour notre variable cible à prédire. Beaucoup plus de transactions bancaires ne sont pas frauduleuses. Nous allons donc penser à mettre en place des techniques de ré-échantillonnage afin d'équilibrer au mieux les classes de notre variable cible.

2. La quantité de données que nous disposons. Une problématique d'optimisation sera donc à résoudre. Nous décrivons comment optimiser un modèle dans l'introduction de la section 3. Nous prendrons également soins d'optimiser l'écriture de notre algorithme pour la partie de protocole expérimental. Cela passera par un choix minutieux du nombre et des valeurs des hypermètres à tester, de l'application de validation croisée ou non, du nombre de données passées pour l'entraînement avec l'utilisation de certaines méthodes de ré-échantillonnage, du calcul des métriques d'évaluation et de la façon dont nous enregistrons les performances de nos modèles.

Afin d'aborder au mieux ces enjeux et de résoudre le plus rigoureusement notre problématique, nous parlerons dans une première partie de l'analyse descriptive des variables à travers une description de la nature des données et des statistiques descriptives sur le jeu de données. Ensuite nous avons une partie plus conséquente qui est consacrée à la présentation de plusieurs méthodes et démarches utiles pour résoudre ce problème : comment mesurer les performances des modèles que nous testons, puis le pré-processus à appliquer sur nos données avant de les présenter en entrée de divers algorithmes, et les détails de ces derniers. La section suivante décrira notre protocole expérimental. Nous présenterons plus en détail le code que nous avons mis en place. Nous détaillerons également la méthode du Gradient Boosting qui a donné les meilleurs résultats lors de nos expériences. Nous finirons par conclure ce rapport avec des pistes d'améliorations que nous n'avons pas pu appliquer et expliquer.

2 Analyse descriptive des données

2.1 Nature des données

Plusieurs critères sur les données sont à prendre en compte, à analyser et à traiter pour le choix de l'algorithme de prédiction.

Premièrement, nous avons une base de données avec des valeurs connues et historiées pour la variable à prédire (dite variable cible) et des variables descriptives de cette variable à prédire. Cela nous indique que nous faisons face à un schéma supervisé. Des algorithmes spécifiques pour les bases de données étiquetées sont plus nombreux et plus performants que les algorithmes non supervisés. Cependant, rien ne nous empêche de penser à l'application d'un algorithme non supervisé (type clustering : K-means) pour résoudre notre problème.

Deuxièmement, la prédiction va dépendre également de certaines caractéristiques de la variable à prédire. Nous avons une variable cible qui est qualitative binaire : fraude ou non fraude, ce qui nous conduit à appliquer des algorithmes de classification.

Ensuite, il faut aussi prendre soin de regarder la nature de nos variables descriptives. Certains algorithmes ne considèrent que des variables quantitatives et donc des re-codages disjonctifs sont donc nécessaires à faire en amont pour ces algorithmes. Ce ne sera pas notre cas car notre jeu de données n'est composé que de variables quantitatives, sauf pour la date mais nous avons décidé de ne pas travailler avec. Celle-ci apparaissait comme non représentative dans les algorithmes de sélection de variables filtres (cf section 3.2.2). Nous l'avons recodée pour avoir deux nouvelles variables : mois et numéro du jour de la semaine (0 :lundi, 1 :mardi, etc) mais ce recodage ne s'est pas avéré utile pour la décision des algorithmes de prédiction, peut être à cause de la courte durée sur laquelle les données ont été récolté (seulement 7 mois d'une même année pour la base d'entraînement et 3 mois pour le test sur la même année également).

Par ailleurs, nous faisons face à un jeu de données dit déséquilibré car notre classe positive que nous cherchons à prédire : fraudeur, est significativement moins représentée que la classe négative : non fraudeur. C'est un cas de figure très courant pour les jeux de données traitant des cas de fraudes. On utilise souvent "l'Imbalance ratio" pour caractériser le degré de déséquilibre :

$$Imbalance\ ratio = \frac{\text{Nb d'observations majoritaires}}{\text{Nb d'observations minoritaires}}$$

Avec cette définition, *Imbalance ratio* ≥ 1 conclura d'un jeu de données déséquilibré, sachant que la valeur 1 correspond à des données parfaitement équilibrées. Dans notre jeu de données, il vaut :

$$\textit{Imbalance ratio} = \frac{3\ 788\ 984}{23\ 055} = 164.35$$

Les jeux de données déséquilibrés nous sont difficiles à appréhender dans des problématiques de prédiction. En effet beaucoup d'algorithmes vont faire de grosses erreurs de prédiction avec des données déséquilibrées, par exemple on pourra avoir un modèle qui fait du sur-apprentissage car nous avons essayé de lui réduire son erreur en entraînement, et qui par conséquent possède un biais faible et une variance élevée. Ce qui n'est donc pas un bon modèle (par définition de la problématique du compromis de biais-variance, qui sera décrit dans la section 3.1.1).

Enfin, nous avons un jeu de données que l'on peut considérer comme volumineux : plusieurs millions de lignes. Il est donc impératif d'optimiser toutes nos approches dans la résolution de notre problème de prédiction.

2.2 Statistiques descriptives

Quelques statistiques descriptives de notre base initiale sont importantes à faire afin de mieux appréhender le jeu de données sur lequel nous travaillons.

Avant de faire ces statistiques, nous avons effectué un rapide nettoyage de la base : suppression des lignes dupliquées, recodage de la variable date en deux nouvelles variables : numéro du mois et numéro du jour de la semaine et information du bon type toutes nos variables.

Le jeu de données comporte 3 812 039 observations et 21 variables. Notre variable cible a le label "FlagImpaye", elle est binaire : 0 ou 1. Comme vu précédemment, la variable cible contient 0.9% de la modalité 1. La variable cible est décrite par 20 variables explicatives : 18 sont quantitatives et 2 sont qualitatives (les deux variables de renseignement sur le mois et le jour).

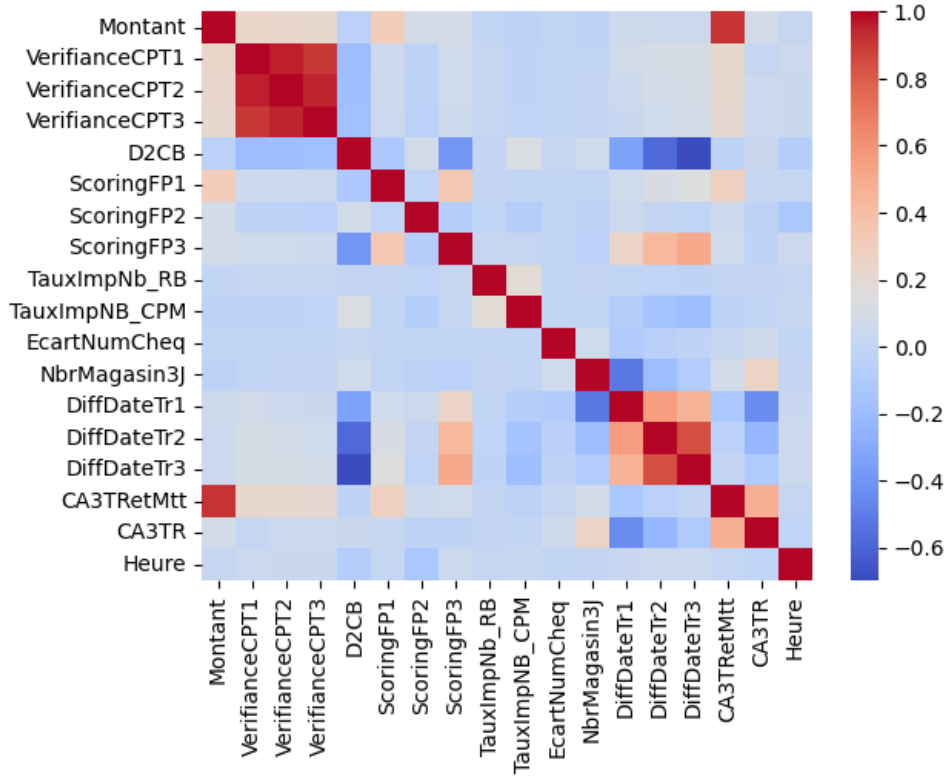

FIGURE 1 – Matrice de corrélation entre toutes les variables descriptives

La figure 1 représente la matrice de corrélation entre les variables explicatives. On peut dire de cette matrice que les 3 variables VerifianceCPTx, qui comptent le nombre de transactions effectuées par le même identifiant bancaire à x intervalles de temps, sont très corrélées entre elles. On peut en dire de même pour les variables de DiffDateTrx.

Enfin, les variables $Montant$ et $CA3TR$ sont aussi très corrélées, cela parait logique car la variable $CA3TR$ inclut dans sa valeur le $Montant$.

Pour toutes ces variables très corrélées que l'on vient de citer, on peut penser qu'il est intéressant d'appliquer une sélection de variables filtre basé sur la corrélation, pour ne garder qu'une variable parmi les DiffDateTrx, une parmi les VérifianceCPTx et une parmi $Montant$ ou $CA3TR$, nous en parlons dans la section 3.2.2.

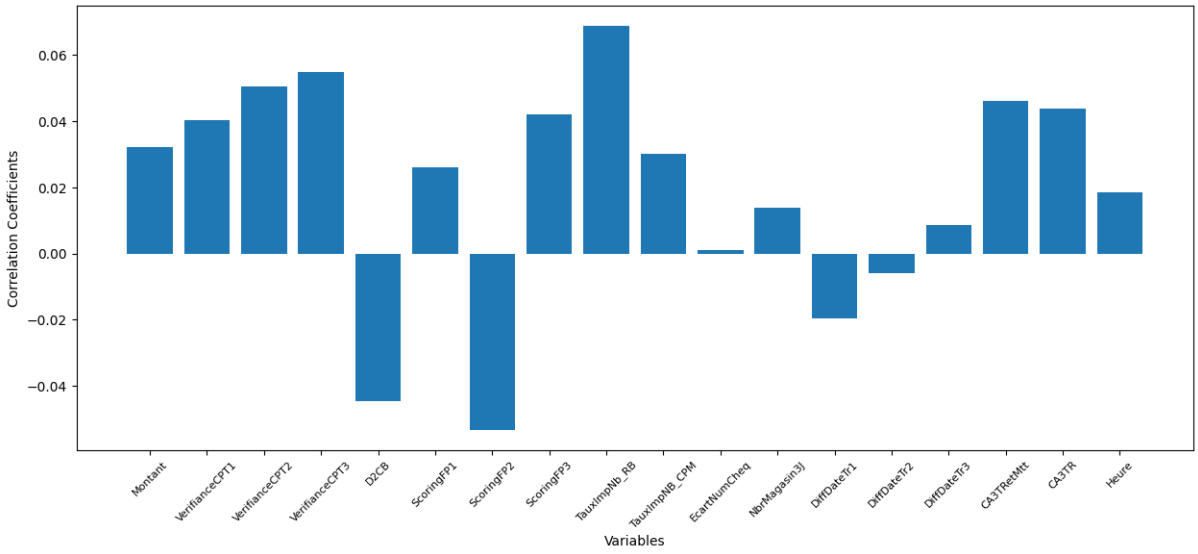

FIGURE 2 – Graphique des indicateurs de corrélation entre la variable cible et chacune des variables descriptives

La Figure 2 représente un diagramme en barre des indicateurs de corrélation entre la variable cible *FlagImpaye* et chacune des variables explicatives quantitatives. On peut voir ici que les variables : **TauxImpNb_RB** : taux impayés enregistrés selon la région où a lieu la transaction, **ScoringFP2** : score d'anormalité du panier relatif à la famille de produits électroniques et **VérifianceCPT3** : nombre de transactions effectuées par le même identifiant bancaire au cours des sept derniers jours sont les trois variables les plus corrélées a notre cible. On peut également affirmer que la variable **EcartNumCheq** : différence entre les numéros de chèques n'est pas du tout corréler avec notre cible.
Pour mettre en relation ces deux analyses sur la corrélation, on peut penser à une sélection de variable instinctive : retirer les deux variables VérifianceCPT1 et 2 ainsi que les deux variables DiffDateTr2 et 3.

Ensuite, la détection de la fraude permet à l'enseigne de minimiser sa perte financière. Observons alors la variable *Montant* qui décrit le montant des chèques en euros afin de visualiser l'impact de la fraude bancaire par chèque sur le revenu total des chèques.

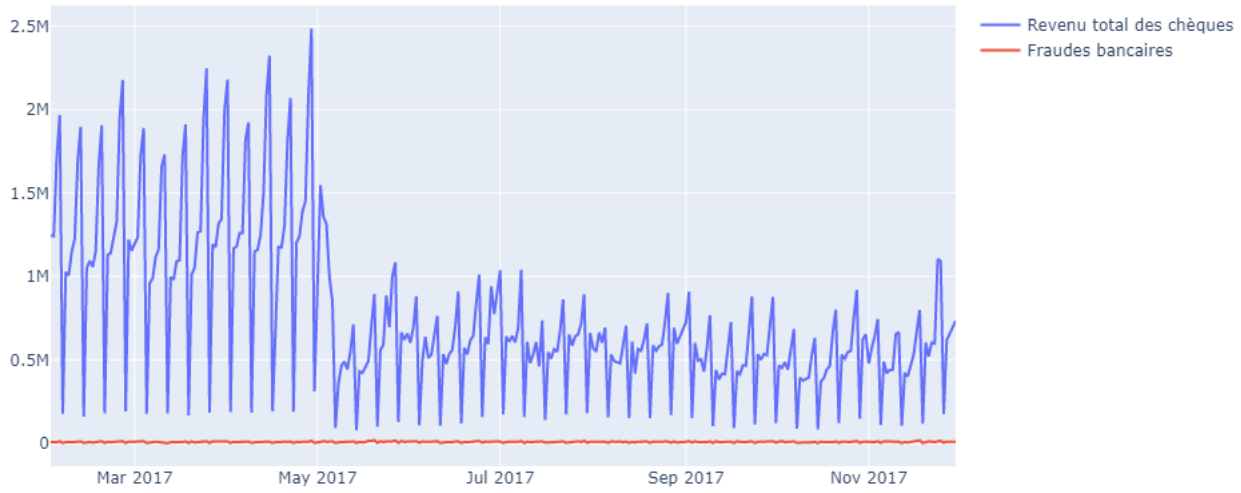

FIGURE 3 – Montant des fraudes par chèques et recettes par chèque par jour

(a) Box plots des montant des fraudes par chèques et recettes par chèque

(b) Zoom sur les box plots des montant des fraudes par chèques et recettes par chèque

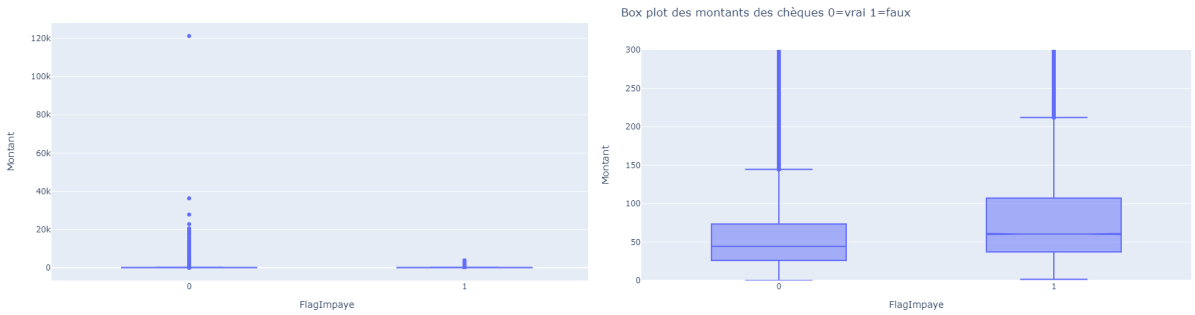

FIGURE 4 – Boxplot des Montants

Ce graphique nous montre que beaucoup de données sont outliers. Grâce à la fonction zscore() du module stats de python nous avons pu calculer le score z de chaque observation. Le score z ou score standard, représente le nombre d'écarts types d'une observation par rapport à la moyenne d'une distribution. La formule pour calculer le score z est $\frac{(x-moyenne)}{\text{écart type}}$. Cette détection nous informe que

449760 observations seraient considérées comme des outliers.

3 Méthodologie

Plusieurs algorithmes de classifications existent pour résoudre notre problème. Nous allons en présenter quelques-uns qui paraissent pertinents pour notre problématique de classification binaire déséquilibrée.

Nous pouvons appliquer ces méthodes de classification en faisant varier les hyperparamètres de celles-ci. Un hyperparamètre est un paramètre qui ne pourra pas être appris ou déduit par l'algorithme. Il doit être choisi et testé par le développeur. Le but est donc de développer plusieurs modèles avec différentes configurations de paramètres pour chaque méthode de classification. Ensuite il faudra calculer des indicateurs de performance pour chaque modèle, et les comparer afin de choisir le modèle qui fonctionne le mieux sur nos données. Il est important également de prendre soin de choisir un modèle qui a une faible complexité afin d'améliorer la performance du modèle et de gagner en temps de calcul et en mémoire.

La complexité dépend :

- De la régularité du modèle (une forte variance et un sur-apprentissage produisent un modèle complexe) ;
- Du nombre de variables explicatives du modèle (un grand nombre de variables ajoute de la complexité) ;
- Du nombre de d'hyperparamètres du modèle et le choix de ceux-ci.

Il est d'ailleurs important de souligner que, pour le gradient boosting notamment, le choix des hyperparamètres est très important car ils vont influer sur le processus d'apprentissage : fonction de perte utilisée, nombre d'itération, taux d'apprentissage, ...

Certaines méthodes de classification fournissent des indications sur la pertinence des variables, qui peuvent être exploitées dans un mécanisme de sélection des variables. On peut se servir de cette sélection de variables dans le but de réduire la complexité du modèle et améliorer l'efficacité de l'apprentissage en obtenant un modèle plus simple et global, ce qui entraînera également une réduction de la variance (et éviter le sur-apprentissage).

Dans la sous-section 3.3 nous allons expliquer quelques algorithmes supervisés, de manière verbeuse, qui nous paraissaient utiles à tester lors de notre protocole expérimental. Cela nous permettra de voir le spectre d'applications pour la résolution de notre problème, et également de mieux comprendre les méthodes que nous appliquons à travers les outils de python. Nous allons ensuite détailler mathématiquement dans la section 4.3 , l'algorithme qui aura ressorti les meilleurs résultats, et que nous définirons comme l'algorithme de résolution à notre problématique de classification de fraude : le Gradient Boosting.

Notations utilisées :

Notation	Description
S	Échantillon
n	Nombre d'observations
$\mathbf{x}$	Vecteur d'une variable explicative
x_i	Observation i du vecteur $\mathbf{x}$
y	Vecteur d'étiquettes ou variable cible
y_i	Étiquette du vecteur y
$L(.)$	Fonction de perte
$\|\|.\|\|$	Norme d'un vecteur
$< ., . >$	Produit scalaire entre deux vecteurs
$f(.)$	Fonction
$\hat{f}(.)$	Fonction de prédiction
$\hat{y}$	Vecteur de prédiction de la variable cible
$k(.)$	Noyau

3.1 Mesure de performance

Afin de déterminer si notre modèle fera de bonnes prédictions, il est important de mesurer sa performance. Nous allons définir ici deux mesures : la F-mesure et la courbe AUC-PR, qu'il serait idéal de maximiser dans notre cadre d'étude. Certaines mesures ne nous sont pas utiles à considérer seules comme la précision car elle vaut 100% quand on prédit tout à la classe majoritaire.

Pour les deux mesures que nous allons présenter, elles prennent en compte à la fois la précision et le rappel dans leurs formules. Ces deux mesures simples (rappel et précision) sont obtenues à l'aide de la matrice de confusion. La matrice de confusion permet de recenser des résultats de prédiction pour les problèmes de classification. Elle compare les données réelles de la variable cible à celles prédites par le modèle.

	Prédiction positive	Prédiction négative
Vraie positive	True Positive (TP)	False Negative (FN)
Vraie négative	False Positive (TP)	True Negative (TN)

La Précision correspond au taux de prédictions correctes de notre classe positive parmi toutes les prédictions positives. Elle mesure la capacité du modèle à ne pas faire d'erreur lors d'une prédiction positive :

$$Précision = \frac{TP}{TP + FP}$$

Le Rappel correspond au taux d'individus positifs bien détecté par le modèle. Il mesure la capacité du modèle à détecter l'ensemble des individus positifs :

$$Rappel = \frac{TP}{TP + FN}$$

3.1.1 Arbitrage biais-variance

Le dilemme biais-variance est le conflit qui consiste à simultanément essayer de minimiser ces deux sources d'erreurs qui empêchent de généraliser les résultats des algorithmes d'apprentissage supervisé au-delà de leur jeu de données d'entraînement :

- L'erreur de biais est une erreur d'hypothèse.s erronée.s dans l'algorithme d'apprentissage. Un biais élevé peut empêcher un algorithme de trouver des relations pertinentes entre les caractéristiques et les sorties cibles (sous-apprentissage) ;
- La variance est une erreur provenant de la sensibilité aux petites fluctuations du jeu d'entraînement. Une variance élevée peut amener un algorithme à modéliser le bruit aléatoire dans les données d'apprentissage. (sur-apprentissage).

Idéalement, on veut choisir un modèle qui reflète avec précision les régularités dans les données d'apprentissage, mais qui se généralise aussi aux données tests. Il est très difficile de minimiser le biais et la variance. Les modèles avec une variance élevée représentent assez bien l'échantillon d'apprentissage, mais il y a un risque non négligeable de sur-apprentissage. En revanche, les algorithmes avec une variance faible produisent généralement des modèles plus simples mais peuvent être en sous-apprentissage sur le jeu de données test.

Supposons que nous avons un ensemble d'apprentissage constitué d'un ensemble de points $x_1, \ldots, x_n$ et de valeurs réelles y_i associées à chaque point x_i. Nous supposons qu'il existe une relation fonctionnelle bruitée $y_i = f(x_i) + \epsilon_i$, où le bruit, ϵ_i, a une moyenne nulle et une variance σ^2.

Trouver une fonction $\hat{f}$ qui se généralise à des points extérieurs à l'ensemble d'apprentissage peut être fait avec l'un des nombreux algorithmes utilisés pour l'apprentissage supervisé. Selon la fonction $\hat{f}$ que nous choisissons, son erreur attendue sur un échantillon test x peut se décomposer comme suit :

$$\mathrm{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathrm{Biais}\left[\hat{f}(x)\right]^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

où

$$\mathrm{Biais}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x) - f(x)\right]$$

et

$$\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}\left[\left(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\right)^2\right]$$

3.1.2 La F-mesure

La F-mesure est une bonne mesure de performance à observer sur un modèle qui essaie de faire une prédiction binaire avec une classe minoritaire. Elle est très utilisée dans la détection des fraudes et des anomalies. Elle est définie comme la moyenne harmonique de la Précision et du Rappel.

$$\text{F-mesure} = \frac{(2 * TP)}{2 * TP + FN + FP}$$

Par rapport à la précision, la F-mesure est difficile à optimiser. La F-mesure présente deux inconvénients majeurs :

1. Elle est non linéaire.

   Il est difficile de dériver des garanties de généralisation pour une telle mesure. En outre, nous ne pouvons pas utiliser les algorithmes standard de descente de gradient pour l'optimiser.

2. Elle est non convexe.

   Un algorithme d'optimisation peut tomber dans des optimums locaux qui peuvent être éloignés de la réalité.

3.1.3 L'AUC de la courbe Précision-Rappel

Le terme AUC signifie **Area Under the Curve**, aire sous la courbe en français. La métrique AUCconsiste donc à calculer l'aire sous la courbe Précision-Rappel.

Le modèle parfait a un rappel maximal pour n'importe quelle valeur de Précision et une Précision maximale pour n'importe quelle valeur de Rappel. L'AUC Précision Rappel vaut 100% pour un modèle parfait.

Comme elle est global – calculée sur l'ensemble des seuils – l'AUC Précision Rappel peut être une bonne métrique pour l'optimisation des hyperparamètres d'un modèle, où pour faire de l'early stopping en boosting (Gradient Boosting Machine, XGBoost etc.).

3.2 Pré-process sur les données

Comme nous l'avons vu en introduction, notre problématique fait ressortir deux enjeux majeurs qui sont :

1. Le traitement de données déséquilibrées.

   Afin d'appréhender le déséquilibre de la modalité de notre variable cible, nous avons à dispositions plusieurs méthodes pour rééquilibrer nos données afin que les deux classes soient mieux équilibrées. Pour ce pré-processus de rééquilibrage des classes de notre variable cible nous allons utiliser ce package de la librairie scikit-learn qui propose des fonctions pour appliquer les méthodes que nous allons expliquer.

2. Le traitement d'un gros volume de données.

   Pour l'aspect volume de données, nous allons procéder a une sélection de variable filtre, c'est-à-dire en amont de l'application de quelconque algorithme. Pour ce pré-processus de rééquilibrage

des classes de notre variable cible nous allons utiliser ce package de la librairie scikit-learn qui propose des fonctions pour appliquer les méthodes que nous allons expliquer.

3.2.1 Re-échantillonnage au niveau des données

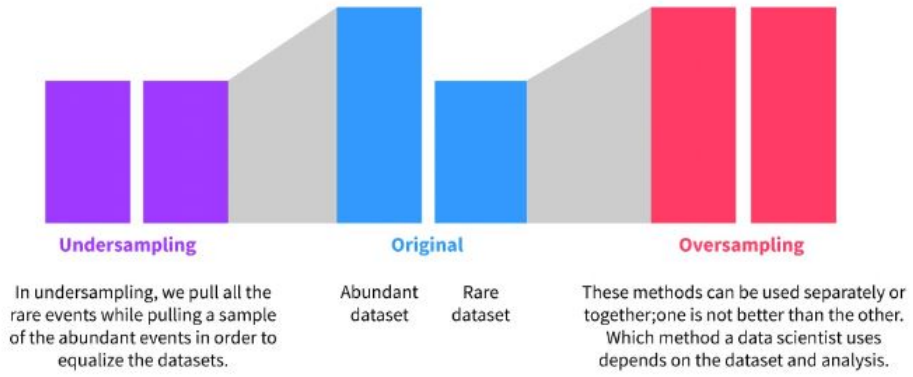

FIGURE 5 – Explications de l'Undersampling et de l'Oversampling
source : https ://www.mastersindatascience.org/learning/statistics-data-science/undersampling/

Undersampling

L'Undersampling est un ensemble de méthodes qui consistent à supprimer des observations de la classe majoritaire pour rééquilibrer les classes dans le jeu de données. Ces méthodes sont intéressantes dans le cas des gros jeux de données car elles permettent de diminuer de façon non négligeable le temps de calcul des algorithmes.
Il existe plusieurs algorithmes d'Undersampling (random - Tomek Links - Edited Nearest Neighbour - Near Miss - ...), pour notre analyse nous allons nous intéresser aux Tomek Links.

Tomek Links :
Cette méthode effectue des liens par paires de données qui sont les plus proches autour de la ligne de séparation. Cela signifie que ce sont les données qui vont être les plus problématiques pour la plupart des algorithmes de classification. En supprimant ces paires de données, la séparation entre les deux classes sera élargie, de sorte que notre l'algorithme pourra faire moins d'erreurs.

Oversampling

Inversement à l'Undersampling, L'Oversampling est un ensemble de méthodes qui consistent à ajouter des observations de la classe minoritaire pour rééquilibrer les classes du jeu de données. Les méthodes d'Oversampling sont toujours préférées aux méthodes d'Undersampling car elles ne suppriment pas d'information.
Il existe plusieurs algorithmes d'Oversampling (random - SMOTE - Adasyn - ...) , pour notre analyse nous allons nous intéresser à SMOTE

SMOTE : Cette méthode de sur-échantillonnage se concentre sur la classe minoritaire, qui est augmentée en créant des exemples «synthétiques». C'est l'un des algorithmes les plus utilisés pour améliorer la performance de classifieurs appliqués sur les données déséquilibrées. L'algorithme fournit un ensemble de règles simples pour générer ces nouvelles données. Bien que chaque nouvelle donnée synthétique soit construite à partir de ses parents (la donnée choisie et l'un de ses voisins les plus proches), la donnée générée n'est jamais un double exact de l'un de ses parents.

Combinaison des deux

SMOTE-TomekLinks :La méthode hybride combine les approches de sur-échantillonnage et celles de sous-échantillonnage en éliminant des données dans la classe majoritaire et ajoutant des données dans la classe minoritaire afin de rééquilibrer la distribution des classes (Santoso et al., 2017). L'approche Smote-TL est la combinaison des algorithmes Smote et Tomek Links. Elle a d'abord été utilisée pour améliorer la classification des exemples sur le problème de l'annotation des protéines en bioinformatique (Batista et al., 2003).

3.2.2 Sélection de variable filtre

Afin d'alléger notre jeu de données, et dans le but d'optimiser en temps nos algorithmes, nous allons appliquer un algorithme de sélection de variable filtre ou ranking sur notre jeu de données. Cette pratique nous amène à supposer que la sélection faite en amont sera la bonne pour n'importe quel algorithme, ce qui est faux en pratique. En effet, il est plus minutieux d'appliquer les sélections de variables intégrées à chaque méthode, mais cela demande beaucoup de temps de calcul et dans le cadre de notre projet, au vu du temps imparti et de la capacité de nos machines, nous appliquerons simplement une sélection de variable en amont une seule fois pour tous les algorithmes que nous testons

dans notre protocole.

Comme nous sommes dans un cas de classification, avec des variables explicatives quantitatives, il existe des méthodes spécifiques avec des critères spécifiques. Ces méthodes utilisent comme critère :

1. **Critère de pertinence.** Les variables explicatives doivent êtres liées à la variable cible. Ce critère mesure le degré de liaison de chaque variable explicative avec la variable cible. Ce critère se base sur les calculs de la loi du chi-deux.

2. **Critère de redondance.** Les variables explicatives doivent être faiblement liée entre elles. Ce critère mesure le degré de liaison entre 2 variables explicatives (2 à 2).

La question que l'on se pose est : Combien de variables allons-nous sélectionner ? Les k meilleures ? Les méthodes proposées dans scikit-learn sont dotées de critères de sélection qui se basent sur des calculs statistiques. En d'autres termes, l'algorithme va calculer un degré de pertinence/seuil de significativité de la variable, qu'il va comparer en fonction d'un seuil que nous fixons. Ce seuil de significativité est la p-valeur, calculée a l'issue de tests statistiques tels que l'ANOVA ou le Chi-deux. Nous verrons dans la section 4.1 que l'application de ces méthodes sur notre jeu de données n'a pas donné de flagrants résultats. Nous pourrons suggérer comme piste d'amélioration alors l'application de sélection de variables à l'issue des entraînements de chaque modèle, avec des indications de pertinence des variables propres à chaque modèle.

3.2.3 Apprentissage sensible aux coûts

L'apprentissage sensible aux coûts est un type d'apprentissage qui prend en compte le coût des erreurs de classification . Quand on se trouve dans le cadre d'un jeu de données déséquilibré on peut accorder un poids plus fort aux faux négatifs qu'aux faux positifs lors de l'apprentissage. Le modèle va être entraîné pour savoir que mal classer une fraude va entraîner une hausse de la loss $L()$ mais, mal classer un vrai chèque va avoir moins d'importance.

3.3 Algorithmes supervisés

3.3.1 Arbres

L'arbre de décision est un modèle simple dans sa structure algorithmique. Il est construit par un ensemble de noeuds de décision et de feuilles. Chaque feuille représente une décision finale sur la variable

cible, et les noeuds représentent des décisions intermédiaires sur les valeurs des variables descriptives afin de construire un chemin vers une feuille. Un arbre de décision est construit en commençant par la racine qui représente l'ensemble entier des données d'entrée, puis en divisant récursivement les données en sous-ensembles plus petits en fonction des résultats des tests des noeuds. Nous pouvons tuner notre arbre en définissant un critère particulier de séparation des feuilles, comme par exemple le critère de Gini, le critère d'Entropie, ou encore la Variance à minimiser. On peut aussi définir comme hyerparamètres la profondeur maximal de l'arbre, le nombre minimum d'exemples dans une feuille, le nombre minimum d'exemples pour effectuer une séparation, ou encore le gain minimum à chaque noeud.

3.3.2 KNN

Avec un algorithme de KNN, on cherche à prédire la classe d'une nouvelle observation en observant ses k plus proches voisins pour lesquels on connaît déjà l'étiquette. On donnera comme classe à cette nouvelle observation, la classe la plus représentée : résultat majoritaire des statistiques des classes d'appartenance des k plus proches voisins de cette nouvelle observation. Cette méthode est non paramétrique, donc nous n'avons pas besoin de vérifier des conditions sur nos données pour utiliser la méthode. L'avantage est que cette méthode est très simple.
Attention, si le jeu de données est déséquilibrée au niveau des classes, cette méthode fera de mauvaise prédiction car les KNN vont toujours prédire la classe majoritaire. Il existe une solution pour pallier à ce problème :

- Mettre en place un système de pondération sur les voisins pour mesurer leur influence contributive (par exemple : $poids = \frac{1}{d}$, où d est la distance de l'élément, à classer ou à pondérer, de ce voisin). Les différentes distances possibles pour les KNN :
  - Distances métriques disponibles pour les données quantitatives : Euclidienne, Minkowski, Manhatan, Tchebychev, Canberra.
  - Distances noyaux disponibles pour les données quantitatives : linéaire, sigmoïde, logarithmique, puissance, Gaussienne, Laplacienne.

Cet algorithme est tunable par le nombre de voisins que l'on veut considérer.

3.3.3 Modèles linéaires

SVM

Le modèle SVM permet de construire un hyperplan qui sépare le mieux des données en 2 parties. Cette façon de faire permet d'avoir une méthode avec une qualité de prévision qui est la plus grande possible. La marge géométrique m d'un échantillon linéairement séparable est donnée par la plus petite distance d'un point de l'échantillon à la frontière de décision. On cherche à maximiser cette distance car ça nous permettra de maximiser la confiance et donc minimiser la probabilité d'erreur associée au classifieur. Cet algorithme possède comme hyperparametres :

1. C, qui contrôle la force de la régularisation. Plus C est grand, plus la régularisation est forte, et moins il y a de chances que le modèle soit sur-ajusté.

2. γ, qui est utilisé uniquement avec les noyaux radiaux et contrôle la largeur de la fonction Gaussienne utilisée pour la transformation des données d'entrée.

3. ϵ qui contrôle la tolérance à l'erreur pour les données qui ne sont pas correctement classifiées.

Régression logistique

La régression logistique est une technique de classification qui utilise la fonction sigmoïde pour renvoyer la probabilité d'une étiquette binaire. La fonction sigmoïde génère une sortie de probabilité. En comparant la probabilité avec un seuil prédéfini, l'objet est affecté à une étiquette en conséquence. La fonction sigmoïde est définie comme suit :

$$\sigma(\mathrm{x}) = \frac{1}{1 + e^{-\mathrm{x}}}$$

La performance de ce modèle peut être adaptée avec les hyperparamètres suivants :

1. La régularisation : pour éviter le sur-apprentissage en limitant la complexité des modèles. Il existe deux types de régularisation couramment utilisés dans les modèles de régression logistique : la régularisation L1 et la régularisation L2.

2. Le paramètre de pénalisation : Il est utilisé pour contrôler la force de la régularisation dans le modèle. Plus le paramètre de pénalisation est élevé, plus la régularisation est forte.

3. Le nombre d'itérations : Il est utilisé pour contrôler le nombre de fois que l'algorithme d'optimisation doit être exécuté pour arriver à une solution optimale.

3.3.4 Les modèles non linéaires : méthodes à noyau

Si notre problème de classification ne peut pas se résoudre avec une méthode linéaire, on utilise des fonctions à noyaux pour ces méthodes. Le principe est d'intégrer à notre modèle, plusieurs fonctions de bases, par ex :

- Noyau polynomial :

  Le noyau polynomial permet de déterminer des similitudes et des combinaisons des propriétés des échantillons d'entrée. La forme générale de la fonction de noyau polynomial est :

  $$k(x_i, x_i') = (< x_i, x_i' > +c)^M$$

  où x_i et x_i' sont les points de données, c est un coefficient de régularisation qui permet de limiter la complexité du modèle et M est un hyperparamètre qui détermine le degré du polynôme utilisé pour transformer les données. Plus M est grand plus le modèle a de la capacité, et plus le problème est facile à résoudre. L'enjeu est de trouver la valeur de M pour laquelle on a l'erreur minimale en test. Attention si M est trop grand, on ne sera trop loin sur la fonction d'erreur pour être à son minimum car on aura une transformation trop complexe avec un modèle qui capture beaucoup de relations non linéaires dans les données et on aura donc un modèle qui sur-apprend sur les données d'entraînement et qui ne sera pas généraliste.

- Noyau Gaussien :

  Ce noyau permet de mesurer la similarité entre deux points de données dans un espace non linéaire. La forme générale de la fonction de noyau Gaussien est :

  $$k(x_i, x_i') = exp(\frac{-\|x_i - x_i'\|^2}{2\sigma^2})$$

  où x_i et x_i' sont les points de données, $\|x_i - x_i'\|$ est la distance entre x_i et x_i' et σ^2 est un paramètre de réglage qui contrôle la largeur de la fonction de noyau. σ^2 permet de mettre de l'importance sur les points proches : distance entre x_i et x_i', quand on calcule le noyau, et donc

quand on fera la prédiction. Plus σ^2 est petit, plus le modèle a de la capacité. Si σ^2 est trop grand, on va toujours prédire la même chose et on aura donc avec un modèle qui sous-apprend. Si σ^2 est très petit, on va utiliser donc très peu de données x en entrée pour faire une prédiction et ça entraîne une fonction de prédiction très variable et nous risquons d'avoir un modèle qui fera du sur-apprentissage si nous choisissons un σ^2 trop petit.

- D'autres noyaux comme linéaire ou encore sigmoïde.

Ces fonctions de bases seront passées en paramètres de notre algorithme de classification pour que celui-ci modifie notre jeu de données pour ramener notre problème de classification en quelque chose de résolvable avec une fonction linéaire. Dans ces méthodes on cherche à déterminer K la matrice de Gram, qui se compose de tous les noyaux calculés grâce à la comparaison/ combinaison de toutes les observations 2 à 2. L'analyse des valeurs de la matrice de Gram permet de voir l'influence des hyperparamètres du modèle sur sa capacité d'apprentissage.

3.3.5 Modèles ensemblistes

Parallèles

Ces modèles ont comme avantage d'être parallélisables au niveau des calculs lors de leurs entraînements. En effet les modèles parallèles sont construits sur la base de plusieurs petits modèles qui sont combinés à la fin de leur apprentissage pour créer le modèle final.

1. **Le bagging**

   Le bagging est une méthode qui utilise des sous-échantillons aléatoires de données d'entraînement, puis combine les prévisions de ces modèles pour produire une prévision finale. Les sous-échantillons sont générés en utilisant une technique appelée bootstrap, qui consiste à extraire des échantillons aléatoires avec remplacement à partir des données d'entraînement. Cela permet de créer des sous-échantillons qui contiennent des observations répétées, ce qui capture des informations supplémentaires sur la variabilité des données d'entraînement.

2. **Les fôrets aléatoires**

   Le modèle de foret aléatoire est une extension du bagging et a comme avantage de pouvoir minimiser la variance. La forêt aléatoire arrive à minimiser la variance, contrairement au bagging, en utilisant une technique de sélection aléatoire des caractéristiques pour construire chaque arbre.

Elle consiste à choisir aléatoirement un sous-ensemble de caractéristiques à utiliser pour chaque noeud de décision dans l'arbre. On apprend plusieurs arbres différents en utilisant des informations différentes en tirage aléatoire. Plus on a de diversité, plus on gagne en stabilité et meilleur on sera en généralisation. Ce modèle est tunable avec des hyperparametres comme le nombre de variables à considérer dans chaque arbre, le nombre d'arbres à construire.

Séquentiels

Ce type de modèle se construit aussi sur la base de plusieurs petits apprenants, mais cette fois-ci, les petits modèles sont construits les uns à la suite des autres car chaque nouveau petit modèle apprendra des erreurs du petit modèle précèdent. On peut tout de suite constater que ces modèles seront plus coûteux en temps à entraîner que les modèles parallèles car ils ne seront justement pas parallélisables. Il existe plusieurs modèles séquentiels, ils se basent pour la plupart sur la technique de boosting, mais ils se diffèrent sur les fonctions qu'ils cherchent à optimiser et sur la façon dont ils cherchent à les optimiser notamment.

1. **Le boosting**

   Le boosting est une méthode qui consiste à combiner plusieurs modèles "faibles" pour créer un seul modèle "fort". Il existe plusieurs méthodes qui utilisent des fonctions de perte différentes : ADABOOST (fonction de perte exponentielle et un mauvais résultat reçoit un poids accru), Gradient Boosting (fonction de perte logistique), XGBoost, ...Ici nous nous concentrons sur le Gradient Boosting.

2. **Le Gradient Tree Boosting**

   Le Gradient Boosting est une méthode ensembliste séquentielle à base d'arbres. Pour la problématique de classification avec le Gradient Boosting, la fonction de perte que nous chercherons à optimiser sera la perte logistique : $L(y, \hat{y}) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y}))$. Nous allons voir dans la suite de ce rapport que ce modèle s'avèrera être très efficace pour notre problématique.

   Les hyperparamètres de ce modèle sont :

   1. Le nombre d'arbres : Plus le nombre d'apprenant est élevé, plus le modèle est complexe, attention au risque du sur-apprentissage.
   2. Profondeur des arbre : Plus la profondeur est élevée, plus le modèle est complexe, en général nous voulons de très petits arbres de profondeur 2.

3. Learning rate : La vitesse à laquelle les arbres convergent vers le minimum d'erreurs.
4. La fonction de perte : La fonction de coût utilisée pour mesurer l'erreur entre les prévisions et les valeurs réelles. Nous fixerons toujours la log-loss pour notre part.
5. Nombre minimum d'échantillons requis pour diviser un noeud d'un arbre
6. Nombre minimum d'échantillons requis pour être considéré comme une feuille

Comme nous pouvons le voir, le Gradient Tree Boosting est une méthode très tunable, c'est cela qui fait sa force d'adaptation et de généralisation pour la plupart des problèmes de classification (et régression) et notamment pour notre cas d'un jeu de données déséquilibré.

4 Prototype expérimental

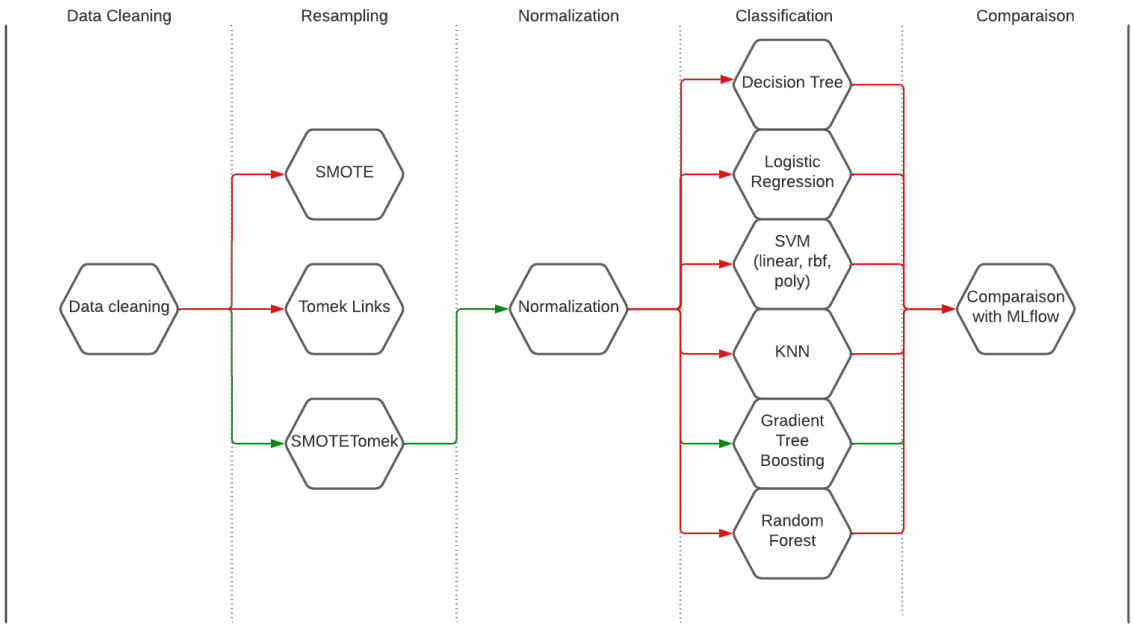

FIGURE 6 – Prototype expérimental

4.1 Pré-process

Le pré-process des données a débuté avec le nettoyage des données en modifiant les types des variables et en enlevant une des lignes qui correspondait au header. Le jeu de données ne comporte

pas de données manquantes nous n'avons donc pas à les imputer. La colonne 'CodeDecision' a été supprimé car c'est une variable qui n'est pas connue au moment de l'encaissement du chèque. Nous avons décidé également de ne pas garder les variables 'Date' et 'Heure'.

Ensuite nous avons procédé à une sélection de variables grâce à la fonction *SelectFwe()* de la librairie *sklearn.feature_selection*. Pour cette méthode nous avons normalisé les données. Les méthodes $f\_classif$ et $chi2$ ont retourné toutes les variables comme significatives. C'est pourquoi nous avons gardé toutes les variables pour construire les modèles.

Le jeu de données a ensuite été séparé en données train, qui serviront pour l'apprentissage des modèles, et en données test qui serviront pour tester nos modèles et les évaluer avec la F-mesure et l'AUC. Les données ont été séparées par la date pour assurer une certaine reproductibilité des expérimentations.

Comme nous l'avons expliqué en introduction, la répartition de la variable cible "FlagImpaye" est déséquilibrée, c'est à dire qu'il y a 99% de 0 (vrai chèque) et 1% de 1 (chèque frauduleux). Pour rééquilibrer les classes nous avons utilisé plusieurs algorithmes de re-sampling.

- Undersampling avec Tomek Links ;
- Oversampling avec SMOTE ;
- Combinaison des deux avec SMOTETomek.

Dataset	Shape Counter
Original	(0 : 3 788 984, 1 : 23 055)
TomekLinks	(0 : 3 041 058, 1 : 22 296)
SMOTE	(0 : 3 788 984, 1 : 325 296)
SMOTETomekLinks	(0 : 3 039 794, 1 : 297 142)

TABLE 1 – Comptage des classes de la variable 'FlagImpayé' dans le jeu de données d'apprentissage

D'après G. Batista, B. Bazzan, M. Monard (2003) la combinaison d'un over- et under-sampling donne les meilleurs résultats c'est donc sur le jeu de données rééquilibré par cet algorithme que nous avons décidé d'apprendre les modèles.

4.2 Protocole expérimental

Afin d'appliquer les modèles qui semblent intéressant pour résoudre notre problème, nous avons décidé de développer un code en python. Comme dit précédemment nous avons essentiellement utilisé la librairie scikit-learn. Dans le pseudo-code qui suit nous avons verbalisé les principales instructions qui permettent de rechercher le meilleur modèle avec les meilleurs hyperparamètres.

Algorithm 1 Pseudo-code recherche du meilleur modèle

```
procedure SelectModel
    params_models ← Listes d'hyperparamètres a tester
    models_list ← Instances de modèles a tester
    for Tous les modèles dans models_list do
        Initialisation d'un run MLFlow
        Recherche du meilleur paramétrage du modèle avec GridSearchCV()
        Entraînement du modèle avec les données d'entraînement
        Calcul du temps d'entraînement du modèle
        Prédiction sur les variables explicatives du jeu d'entraînement
        Prédiction sur les variables explicatives du jeu de test
        Calcul des métriques liées a la matrice de confusion
        Calcul de la métrique AUC Rappel-Précision
        Stockage d'information diverses dans un DataFrame
        Stockage d'information diverses dans notre run MLFlow
        Stopper le run MLFlow
    end for
end procedure
```

Comme dit précédemment nous avons divisé notre jeu de données en deux parties : un jeu d'entraînement et un jeu de test. On fera ensuite une validation croisée sur le jeu d'entraînement a l'aide de la fonction de scikit-learn *GridSearchCV()*. Cela nous permet de choisir un modèle, qui a la meilleure performance grâce à son choix minutieux des hyperparamètres. Ce modèle sera ensuite entraîné sur la totalité du jeu d'entraînement, puis testé sur le jeu de test.

La validation croisée est une méthode qui se base sur des échantillonnages successifs. Ici nous parlerons de la k-fold cross validation. Cette méthode consiste à diviser les jeux de données d'entraînement en k échantillons à peu près égaux. Tour à tour, chacune des k parties est utilisée comme jeu de test, le reste pour l'entraînement. À la fin, chaque observation a servi une fois dans un jeu de test et (k-1) fois dans un jeu d'entraînement. Pour notre algorithme nous fixons le nombre de folds à 5, par convention. Dans la fonction GridSearchCV() nous donnons une liste de tous les hyperparamètres a tester par validation croisées.

4.3 Modèle retenu

Comme nous le verrons dans la section 4.4 qui compare les résultats des métriques des différents modèles testés, le modèle qui a les meilleures performances est le Gradient Tree Boosting. Nous allons donc faire une description un peu plus mathématique de l'algorithme du Gradient Boosting :

Algorithm 2 Pseudo-code Gradient Boosting

1: **Input :** $L(y, \hat{y})$: fonction de perte, $S = \{x_i, y_i\}_{i=1}^{n}$: jeu d'entraînement, M : nombre d'apprenants faibles
2: **Output :** $F(\mathrm{x})$: modèle final
3: $F_0(\mathrm{x}) \leftarrow \arg\min \sum_{i=1}^{n} L(y, \gamma)$
4: **for** $m \leftarrow 1$ to M **do**
5: $\quad r_m \leftarrow -\frac{\partial L(y, F_{m-1})}{\partial f}$
6: $\quad$ entraîner $h_m(\mathrm{x})$ basé sur $\hat{r}_m$
7: $\quad \gamma_m \leftarrow \arg\min \sum_{i=1}^{n} L(y, F_{m-1}(\mathrm{x}) + \gamma h_m(\mathrm{x}))$
8: $\quad F_m(\mathrm{x}) \leftarrow F_{m-1}(\mathrm{x}) + \gamma_m h_m(\mathrm{x})$
9: **end for**
10: **Return :** $F(\mathrm{x})$

[1] La fonction de perte, qui permet de calculer les résidus est à fixer par l'utilisateur. Dans le cadre de la classification, nous prenons en général la fonction d'erreur logistique :

$$L(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Cette fonction mesure la performance de la prédiction en comparant la sortie prédite de la méthode de Boosting (un score compris entre 0 et 1) avec la valeur réelle de la cible (0 ou 1).

S correspond au jeu de données d'entraînement.

M correspond au nombre d'itération, ou au nombre d'apprenant faibles sur lesquels va se baser notre modèle F final.

[3] Ensuite, avant de boucler, on initialise le modèle additif (fonction de prédiction $F(\mathrm{x})$).

Dans la boucle :

[5] r_m correspond aux pseudo-résidus. Ça calcule l'erreur de prédiction avec la fonction de perte.

[6] on entraîne un apprenant faible qui décrit, de façon simple, l'erreur de prédiction. On prend souvent des petits arbres.

[7] calcul γ ou learning rate, qui est la longueur du pas vers lequel on tend a minimiser la loss.

[8] on met à jour notre modèle additif F qui prend en compte tous les apprenants faibles : F_{m-1} est la combinaison linéaire des m-1 premiers apprenant faibles avec leurs poids respectifs.

4.4 Résultats

Tous les résultats ont été comparés grâce au package *MLflow*. MLflow est une plateforme open source qui permet d'organiser les résultats de machine learning grâce à plusieurs fonctions principales :

1. Suivre les expériences en enregistrant et comparant les hyperparamètres et les résultats (MLflow Tracking)

2. Empaqueter son code d'une manière à ce qu'il soit reproductible par d'autres data scientists (MLflow Projects)

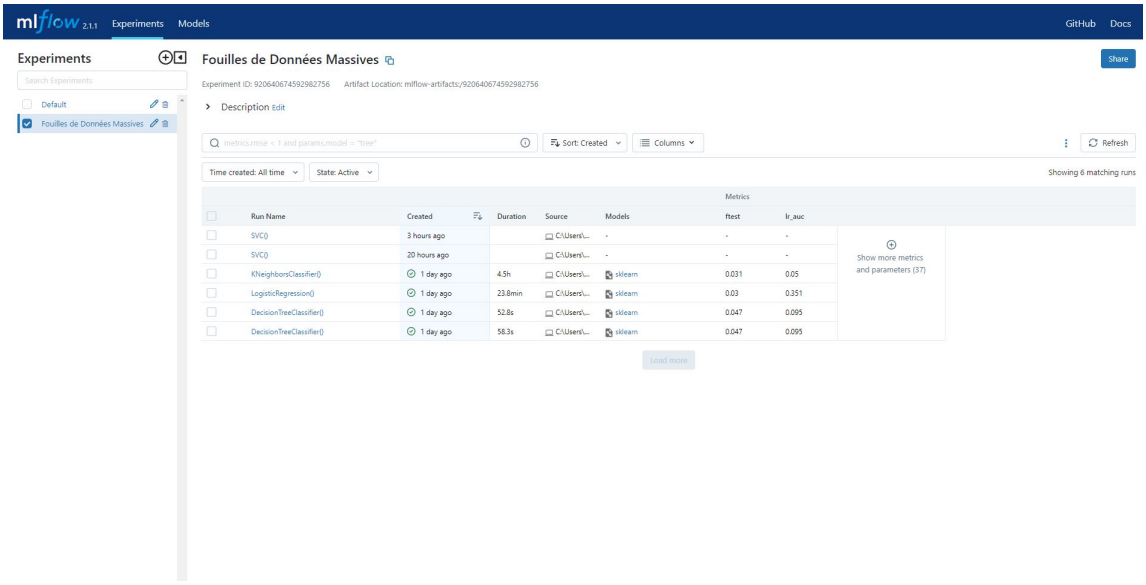

FIGURE 7 – Interface de MLflow avec certains résultats d'algorithmes de classification

Modèle	F-score	AUC
Régression logistique	0.03	0.351
Arbre de décision	0.047	0.095
SVM linéaire	0.02	0.03
SVM à noyau gaussien	0.062	0.12
SVM à noyau polynomial	/	/
KNN (k plus proches voisins)	0.031	0.05
GradientTreeBoosting	0.1067	0.139
Forêts aléatoires	0.091	0.107

TABLE 2 – Résultats des algorithmes de classification avec le jeu de données rééquilibré par SMOTE-Tomek

5 Conclusion

Conclusion Nous avions lu dans les travaux de ceux qui se sont attelé à la problématique, que la classification dans un jeu de données déséquilibrés n'est pas facilement résolue.

La plupart des algorithmes de classification sont codés dans l'idée que les différentes classes de la variable cible sont équilibrées.

Quand nous comparons nos résultats entre le jeu de données original et celui que nous avons rééquilibré avec SMOTETomek nous nous rendons compte que dans le deuxième cas nous avons de bien meilleurs résultats en moyenne sur les méthodes de classification.

Le meilleur modèle que nous avons trouvé est le Gradient Tree Boosting, ce qui est en accord avec les publications que nous avons lu. Avec plus de temps nous pourrions mieux tuner les hyperparamètres pour obtenir le modèle optimal.

Pistes d'amélioration Nous pourrions tester d'autres modèles ensemblistes qui semblent être ceux qui ont les meilleures performances et continuer d'utiliser le code que nous avons pour la recherche des meilleures hyperparamètres.

Nous avons utilisé des méthodes de sélection de variables avant de lancer les algorithmes mais de nombreux modèles proposent une méthode de sélection de variables interne. Nous pourrions donc les utiliser et quantifier les différences de performances.

Nous n'avons pas eu le temps de nous intéresser aux modèles non supervisés car ils ne ressortaient pas dans la littérature que nous avons lu et nous avons préféré nous concentrer sur des modèles supervisés.

Table des figures

Liste des tableaux

6 Références

Références

[1] HSN Consultants, Inc : *The Nilson Report* ISSUE 1209 (Dec 2021) URL

[2] Bahnsen, A.C., Villegas, S., Aouada, D., Ottersten, B., Correa, A.M., Villegas, S. : *Fraud detection by stacking cost-sensitive decision trees.* DSCS (2017) URL

[3] Sahin, Y., Bulkan, S., Duman, E. : *A cost-sensitive decision tree approach for fraud detection.* Expert Syst. Appl. 40(15), 5916–5923 (Nov 2013) URL

[4] Viola, R., Emonet R., Habrardi A., Metzler G., Sebban M., *Learning from Few Positives : a Provably Accurate Metric Learning Algorithm to Deal with Imbalanced Data* In Proceedings in the 17th International Symposium on Intelligent Data Analysis (IDA), (Oct 2018)

[5] Lin, Chi-Fan , Wang, Sheng-De., *Fuzzy Support Vector Machines.* IEEE Transactions on Neural Networks. 13. 464-471. 10.1109/72.991432. (2002)

[6] Tomek Ivan *An Experiment with the Edited Nearest-Neighbor Rule.* IEEE Transactions on Systems, Man, and Cybernetics 6 : 448-452 (1976)

[7] Jerome H. Friedman *Greedy function approximation : A gradient boosting machine.* Ann. Statist. 29(5) : 1189-1232 (Oct 2001). DOI : 10.1214/aos/1013203451

[8] https ://www.math.univ-toulouse.fr/ besse/Wikistat/pdf/st-m-app-agreg.pdf

[9] https ://towardsdatascience.com/simple-logistic-regression-using-python-scikit-learn-86bf984f61f1