Projet Final - Scoring

June 2022

Authors: Cécile Brissard, Maylisse Tso
Programming in : Python

Sorbonne School of Economics, University, Paris, France

Contents

1 Introduction

Afin de garantir la sûreté bancaire à ses clients, toute entité proposant des services de paiement par carte doit pouvoir détecter les transactions frauduleuses. Ces évènements peuvent advenir suite à un vol de la carte elle-même ou à un vol/une copie des informations nécessaires au paiement par carte bancaire, soit les informations présentes sur celle-ci ou contenues dans sa puce. Les méthodes pour y parvenir sont nombreuses : hameçonnage, récupération des informations bancaires via un dispositif placé sur des distributeurs de billets, piratage des données suite aux achats effectués sur internet... et les méthodes de protection des données des clients se développent en conséquence. Pourtant, ce fléau continue de sévir et engendre des pertes pour les banques et assurances, contraintes alors de rembourser leur clientèle. La détection des fraudes présente donc un double intérêt : limiter les pertes explicites dues aux remboursements, et implicites, en d'autres termes, les coûts d'opportunité engendrés par ces fraudes.
L'objectif de notre projet est donc de détecter les transactions bancaires pouvant constituer de potentielles fraudes. Pour ce faire, nous avons accès à une base de données brutes, que nous allons nettoyer et harmoniser. Celle-ci sera utilisée pour prédire les profils présentant les caractéristiques de la fraude avec différentes méthodes de modélisation. Méthodes confrontées les unes aux autres afin de choisir le modèle donnant de meilleurs résultats, compte tenu de notre jeu de données.

Nous commencerons par, dans la Section 2, suivant de cette introduction, détailler le contenu de la donnée et expliciter les traitements préalables à son utilisation. Puis, dans la Section 3, nous expliquerons les fonctionnements des différents modèles, ainsi que l'optimisation qui en a été faites sur nos données. Enfin les Sections 4 et 5, donneront, respectivement, les résultats implicant le choix du modèle et la conclusion.

2 Pre-processing

La base de donnée intutilée *"autorisations"* est composée intialement de 1 151 432 observations et 25 variables, parmi lesquelles nous considèrerons comme essentielles 23 variables réparties comme suit :

1. le numéro de $Carte$,
2. le $Montant$ de la transaction,
3. le code du $Pays$ dans lequel la transaction a été réalisée,
4. MCC : le code du commercant chez qui la transaction a été réalisée,
5. le code réponse $CodeRep$ de la demande d'autorisation où $0 = Accepte$ et $1 = Refus$ (toutes les autres valeurs sont recodées en 1),
6. $Date\_$ la date de la transaction formée de la concaténation des variables $Date$ et $Heure$, convertie au format "string format time" (stfrtime),

respectivement dans les dernières 3, 6, 12 et 24 heures :

7. $FM_Velocity_Condition_$3/6/12/24 : le nombre de transactions réalisées pour chaque carte,
11. $FM_Sum_$3/6/12/24 : le montant cumulé des transactions,
15. $FM_Difference_Pays_$3/6/12/24 : le nombre de transactions réalisées dans des pays différents,
19. $FM_Redondance_MCC_$3/6/12/24 : le nombre de transactions réalisées chez un même type de commerçant,

et enfin la variable utilisée comme **Target** :

23. *fraude* qui signale les transactions frauduleuses (une variable dichotomique prenant 1 le cas échéant et 0 sinon).

Après avoir contrôlé l'absence de valeurs manquantes, nous décidons de ne pas prendre en compte la variable *Date_* car le format n'est pas pris en charge par la plupart des modèles et nous pensons que cette variable présente peu de pouvoir explicatif de la *fraude*.

2.1 Gestion des outliers : Distance de Cook

Afin de parfaire notre base de donnée, nous choisissons d'utiliser, d'une part la **Distance de Cook** pour déterminer les outliers, soit les observations, a exclure des données.

La distance de Cook est une estimation de l'influence d'un point de données. Elle prend en compte à la fois l'effet de levier et le résidu de chaque observation. Elle vise à mesurer la manière dont un modèle de régression change lorsque la i^{eme} observation est supprimée. Si certains points présentent des résidus importants (valeurs aberrantes) et/ou un effet de levier élevé, les résultats de la régression, ou du moins sa précision, peuvent être faussés. Plus la distance de Cook est grande, plus une analyse approfondie des observations concernées est recommandée.
La distance de Cook se définie comme :

$$\underset{n\times 1}{\mathbf{y}} = \underset{n\times p}{\mathbf{X}} \quad \underset{p\times 1}{\boldsymbol{\beta}} \quad + \quad \underset{n\times 1}{\boldsymbol{\varepsilon}} \quad \underset{n\times 1}{\mathbf{y}} = \underset{n\times p}{\mathbf{X}} \quad \underset{p\times 1}{\boldsymbol{\beta}} \quad + \quad \underset{n\times 1}{\boldsymbol{\varepsilon}} \quad ; \quad \boldsymbol{\varepsilon} \sim \mathcal{N}\left(0, \sigma^2 \mathbf{I}\right) \tag{1}$$

où σ est le terme d'erreur, $\boldsymbol{\beta}$ est la matrice des coefficients, p est le nombre de covariables pour chaque observation, et X représente la matrice de design.
La distance de Cook D_iD_i des observations i $(i = 1, \ldots, n)$ est donc la somme des changements induits par la suppression d'une observation i lors de la régression :

$$D_i = \frac{\sum_{j=1}^{n} \left(\widehat{y}_j - \widehat{y}_{j(i)}\right)^2}{ps^2} \quad ; \quad s^2 = \frac{\mathbf{e}^\top \mathbf{e}}{n-p} \tag{2}$$

avec $\widehat{y}_{j(i)}$, la valeur obtenue lorsque i est exclu, et s^2 l'erreur carrée moyenne du modèle de régression.

Exposons ci-dessous les résultats obtenus pour quelques variables :

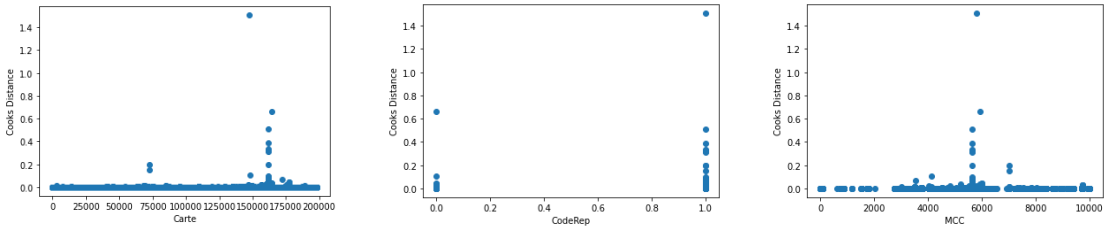

Figure 1: Distances de Cook : *Carte*, *Code_Rep* & *MCC*

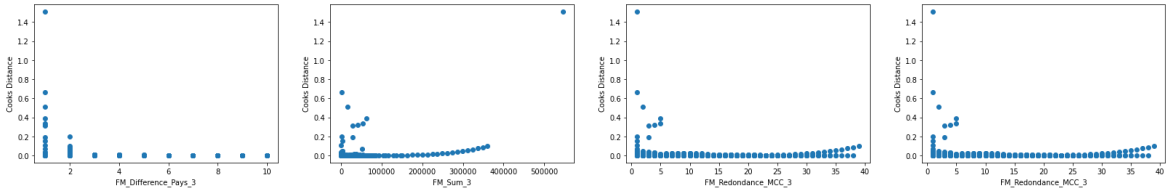

Figure 2: Distances de Cook : $FM\_Difference\_Pays\_3$, FM_Sum_3, $FM\_Redondance\_MCC\_3$ & $FM\_Velocity\_Condition\_3$

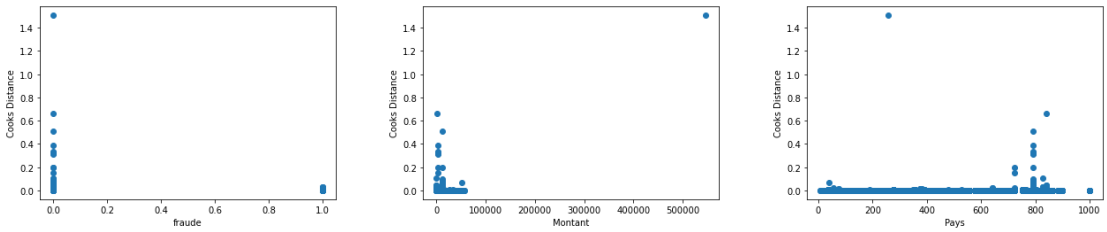

Figure 3: Distances de Cook : *fraude*, *Montant* & *Pays*

Nous choisirons de supprimer ces valeurs. Nous appliquons donc la règle générale qui consiste à étudier tout point dont la distance est supérieure à 3 fois la moyenne de toutes les distances, en supprimant, dans notre cas, ces points. Cette méthode nous permet de détecter 8 174 valeurs aberrantes : nous avons donc maintenant 1 143 258 observations.

2.2 Sélection de variables

D'après la représentation sous forme de carte de chaleur de la matrice des corrélations ci-dessus, quelques variables apportent les mêmes informations.

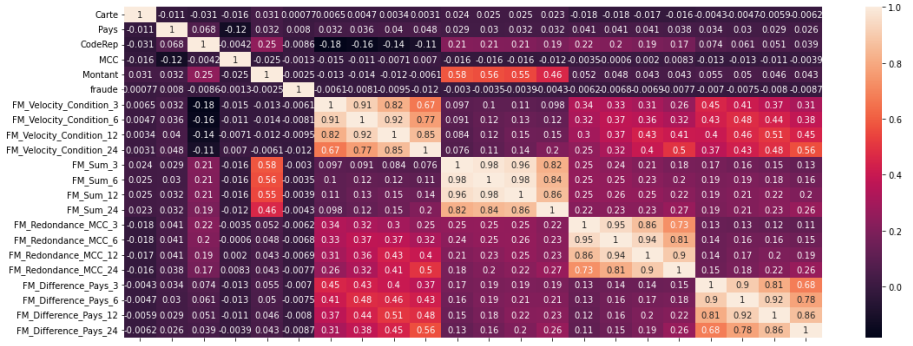

Figure 4: Matrice des corrélations - Heatmap

Pour y pallier, nous tentons différents modèles :

1. Le **Least Absolute Shrinkage and Selection Operator** (*LASSO*) afin de sélectionner les variables apportant le plus d'information et de limiter la répétition de celle ci ;

2. L'**Analyse en Composantes Principales**, afin de concentrer nos sur des axes capturant le maximum d'informations et ainsi avoir une analyse plus pertinente ;

3. La méthode **Weight of Evidence**.

2.2.1 ACP

L'Analyse en Composantes Principales permet de sélectionner les caractéristiques les plus importantes et de capturer le maximum d'informations sur l'ensemble des données. Ainsi, les données sont réarrangées de sorte que sont sélectionnées les caractéristiques dont la part de variance expliquant le modèle est la plus grande. Ainsi, la caractéristique ayant la grande variance est la première composante principale. La caractéristique qui est responsable de la deuxième plus grande variance est considérée comme la deuxième composante principale, et ainsi de suite.
Avant de procéder à l'ACP, il faut commencera normaliser les données. Cela permettra aux variables qui ont des ordres de grandeurs très différents de voir leurs écarts de variances minimisés. Cela permet également d'éviter qu'une trop grande importance soit accordée aux variables ayant un ordre de grandeur plus élevé. Pour ce faire, nous avons utilisé la fonction `StandardScaler` de la librarie Scikit-Learn.

Pour faire l'analyse, nous avons importé la fonction `pca` de la librairie Scikit-Learn. Nous avons toujours nos données partitionnées en deux ensembles : train et test à 80%-20%. On notera cependant que les données utilisées sont celles qui ont été rééquilibrées (voir section 2.2. Rééquilibrage de la base). Nous effectuons ensuite l'analyse en composantes principale sur ces données en lançant la fonction `pca`.

En lançant la fonction `explained_variance_ratio_`, on obtient la variance causée par chacune des composantes principales. Ci-dessous une représentation graphique de ces variances.

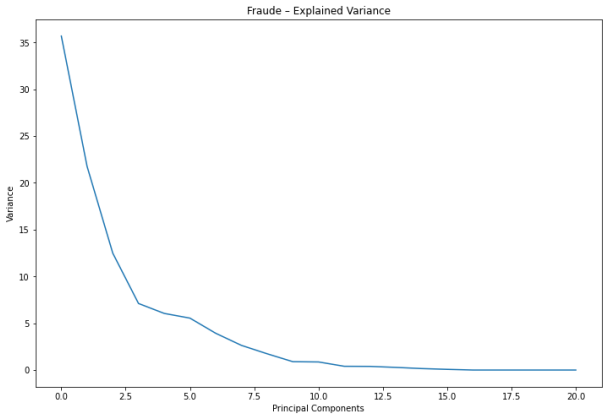

Figure 5: ACP - Variance expliquée

Au niveau de l'interprétation, nous pouvons dire que la première composante principale est responsable de 35.68% de la variance. De même, la deuxième composante principale est responsable de 21.76% de la variance de l'ensemble de données. Ainsi, près de 60% de la variance du modèle est expliquée par les deux premières composantes.
Dans la suite du rapport, les données utilisées dans les modélisations seront celles issues de l'Analyse en Composantes Principales.

2.2.2 LASSO

La capacité du "Least Absolute Shrinkage and Selection Operator", i.e. $LASSO$ à sélectionner un sous-ensemble de variables est due à la nature de la contrainte λ exercée sur les coefficients. La seule différence avec l'utilisation de *Rigde* par exemple, où la norme $l2$ serait appliquée, est l'application de la **norme** $l1$. Cette contrainte va biaiser la capacité de l'algorithme d'apprentissage. $LASSO$ permet de supprimer les paramètres liés aux variables fortement corrélées si nécessaire, et de ne garder ainsi qu'une seule variable parmi celles qui sont corrélées entre elles.

La fonction $LASSO$ est donc :

$$LASSO(w) = \|y - X_w\|_2^2 + \lambda\|w\|_1 \tag{3}$$

avec λ égal à :

$$\lambda = 2\sqrt{2\sigma^2 \frac{\ln\left(\frac{2d}{\nu}\right)}{n}} \tag{4}$$

Le λ est généralement choisi par tatônnement et ou par **Cross-Validation**, nous souhaitons minimiser la variance ; l'idéal est la $10 - fold\ cross - validation$.

Nous procédons par tâtonnement en utilisant la fonction $LASSO$ de la librarie Scikit-Learn, à laquelle nous donnons différentes valeurs pour le paramètre α (correspondant au λ de l'équation). Nous faisons varier ce paramètre de 0.01 à 1000 par pas de 100, afin de voir pour quelle valeur les erreurs de prédiction

seront minimisées.
Les coefficients de chaque variable, en fonction de la valeur de λ, sont représentés dans l'active set ci-dessous. Nous pouvons voir qu'avec un λ plus petit, nous avons des coefficients non nuls. Plus la valeur du λ augmente, et plus le $LASSO$ ramène les poids de chaque variable à zéro.

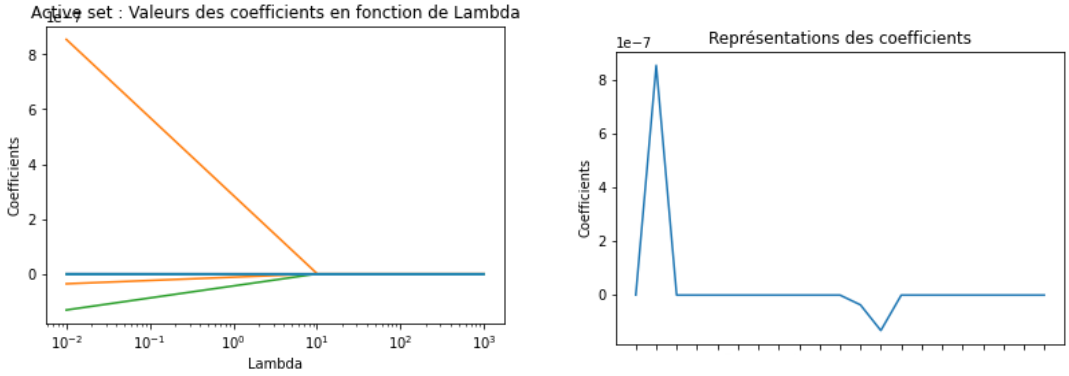

Figure 6: Active set et représentation des coefficents avec $\lambda = 0.01$

Dans notre situation, les erreurs ont été minimisées pour une valeur de λ de 0.01. Nous relançons la régression $LASSO$ avec cette valeur pour le paramètre α. Les coefficients de chaque variable sont disponibles dans le graphique ci-dessus. Nous constatons que la plupart des variables ont un coefficient ramené par le $LASSO$ à 0. Le $LASSO$ a donc effectué une réduction de la dimension en forçant ces coefficients à 0. Si on prend en compte la réduction de variables en résultat, seules les variables $Cartes$, $Pays$, MCC, FM_Sum_12 et FM_Sum_24 seraient encore présente dans nos régressions.

Nous décidons donc de fixer un λ / α plus petit, afin de forcer une sélection plus importante de variables par le modèle : le paramètre varie, cette fois ci, par pas de 10, entre 0.001 et 100. Nous obtenons alors les résultats ci-dessous, pour $\lambda = 0.001$.

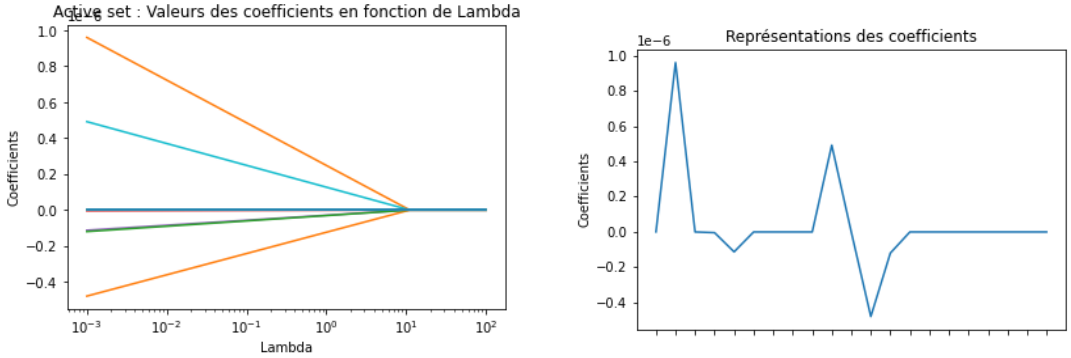

Figure 7: Active set et représentation des coefficents avec $\lambda = 0.001$

Nous comparerons une modélisation englobant toutes les variables, à une seconde ne comprenant que les variables ici sélectionnées par le $LASSO$: $Cartes$, $Pays$, MCC, $Montant$, FM_Sum_3, FM_Sum_12 et FM_Sum_24.

2.2.3 Weight of Evidence

Nos données étant composées de variables qualitatives et quantitatives, nous créons d'abord la base train qui sera utilisée pour la méthode à l'aide du WoE (Weight of Evidence) avant de procéder au rééquilibrage de nos données avec le SMOTE. La méthode du WoE permet d'assigner à chaque variable un poids à la place de leur valeurs d'origine, en se basant sur la variable dépendante. Le WoE est défini comme étant le pouvoir prédictif d'une variable indépendantes par rapport à la variable dépendante (ici fraude). En scoring, il permet également de différencier les mauvais et les bons clients, ou d'évènement et le non-évènement. On a ainsi :

$$WoE = ln(\frac{bons\ clients/non-evenement}{mauvais\ clients/evenement}) \tag{5}$$

Une valeur de WoE > 1 signifie qu'on a un WoE positif (le pourcentage de bons clients est supérieur à celui des mauvais). A l'inverse, une valeur de WoE < 1 signifie qu'on a un WoE négatif (le pourcentage de bons clients est inférieur à celui des mauvais) Le WoE est calculé de la manière suivante.

1. Séparation des données en *bins*, ou groupes (variables continues, sinon étape 2)
2. Calcul du nombre de d'évènements et non-évènements.
3. Calcul du pourcentage de d'évènements et non-évènements
4. Calcul du WoE. Dans notre situation, l'évènement serait la fraude.

$$WoE_{x=i} = ln(\frac{\%of\,y=0}{\%of\,y=1}) \tag{6}$$

Pour les variables continues, une valeur de WoE est attribuée à chaque *bins*, et c'est ces valeurs qui sont gardées. De la même manière pour les variables catégorielles, la valeur du WoE sera gardée à la place des valeurs d'origine. Pour calculer les WoE de nos variables, nous avons utilisé le package scorecardpy (il s'agit de la version python du package scorecard sur R). Nous avons précisé les paramètres concernant la variable dépendante ($fraude$), la valeur de l'évènement (1), et la méthode à utiliser ($chimerge$, car nous avons des variables continues et catégorielles). Nous avons au préalable utilisé la fonction `woebin`, puis divisons nos données en $train\_set$ et $test\_set$ et enfin appliquons `woebin_ply`. La fonction `woebin` permet d'obtenir les différents *bins*, et la fonction `woebin_ply` applique à nos $train\_set$ et $test\_set$ (pour y et x) les *bins* générés précédemment. En sortie, nous obtenons la base avec les valeurs de WoE obtenues, qui nous servira pour le rééquilibrage avec le SMOTE.

2.3 Rééquilibrage de la base : SMOTE (oversampling)

Pour finir cette première partie, nous décidons qu'il est important de rééquilibrer notre base avant de pouvoir y appliquer les modèles de régression destinés à prédire la fraude. En effet, notre base comporte 1 144 186 : **0**, contre seulement 7 246 : **1**.
Les deux graphiques ci-dessous permettent bien de rendre compte du déséquilibre présent :
Pour ce faire, il existe deux possibilités : le sous-échantillonnage (*undersampling*) et le sur-échantillonnage (*oversampling*). Dans le cas du sous-échantillonnage, la classes des individus majoritaires sera réduite (des individus seront supprimés de la base) afin de balancer l'échantillon. Dans notre situation, cela correspondrait à supprimer des individu ayant la valeur 0 pour $fraude$ afin d'accorder plus d'importance à ceux ayant un 1. Le sur-échantillonnage quant à lui, est une méthode qui donne plus d'importance aux individus minoritaires en augmentant leur proportion. Il n'y a dans ce cas pas de suppression d'individus mais création de nouveaux individus, qui sont voulus proches des individus déjà existants (ici, créer des individus ayant $fraude = 1$. C'est vers cette seconde méthode que nous nous orientons, avec la méthode du SMOTE (**S***ynthetic* **M***inority* **O***versampling* **TE***chnique*).

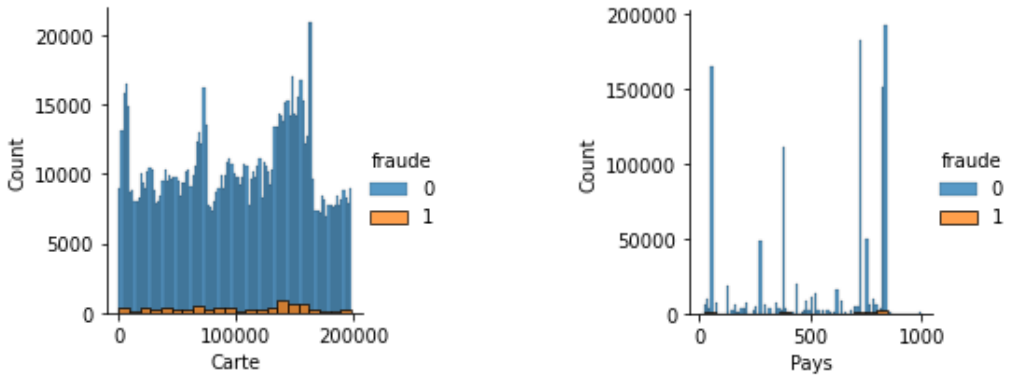

Figure 8: Répartition de la variable $fraude$ en fonction de la $Carte$ utilisé (gauche) et du $Pays$ (droite).

Les nouvelles données obtenues avec le WoE nous servent de base pour le SMOTE. Comme mentionné précédemment, la méthode du SMOTE permet de générer des individus de la catégorie minoritaire dans le but de rééquilibrer la base de données. Il s'agit ici non pas de créer des individus "à l"aveugle", mais en utilisant une technique de "clonage". Ainsi, les nouveaux individus créés seront caractéristiquement proches des individus déjà existants dans la catégorie minoritaire. Ces nouveaux individus sont appelés des individus synthétiques, et créés uniquement pour la base train.
Le processus SMOTE se déroule suivants les étapes suivantes :

1. Sélection aléatoire d'un individu X parmi les individus existants de la catégorie minoritaire
2. Identification des k (paramètre) voisins les plus proches de X
3. Sélection aléatoire d'un des k voisins identifiés
4. Génération aléatoire d'un coefficient α (entre 0 et 1)
5. L'individu synthétique est situé entre l'individu X et le 1 parmi k, en fonction de la valeur de α

Afin d'appliquer le SMOTE à nos données, nous utilisons la fonction `smote` de la librarie imblearn.over_sampling . Les paramètres à fixer sont :

1. **k_neighbors** : le nombre k de plus proches voisins de l'individu X. Pour fixer la valeur de k, nous utilisons une méthode de cross validation avec la fonction `GridSearchCV` de la librarie sklearn.model_selection. Dans notre cas, k devait varier de 4 à 45 par pas de 5, c'est la valeur 45 qui a été sélectionnée.
2. **sampling_strategy** : correspond au nombre d'individus synthétique à générer. Dans notre cas, ce paramètre est $'auto'$, car on ne souhaite ré-échantilloner que la classe minoritaire.

Le SMOTE nous permet d'avoir des des données plus équilibrées, avec en sortie un découpage de 50% de $fraude = 0$ et 50% de $fraude = 1$.

3 Modélisation

Afin de prédire par rapport aux différentes variables à notre disposition, si $fraude = 1$ lorsqu'il y a risque de fraude, $fraude = 0$ sinon, nous appliquons quatre modèles de classification à nos données, avant de choisir celui présentant la meilleure *accuracy*.
Pour ce faire nous séparons tout d'abord notre **Target** du reste des données : la base "Y" ne contient alors que la variable $fraude$, et la base "X" toutes les autres variables. Afin de produire des prédictions, chacune des modélisations ci-dessous, procède d'abord par apprentissage. Ainsi les jeux de donnée X et Y sont tout deux découpés en une partie $test$ (y_test, X_test) et une partie $train$ (X_train & y_train), à l'aide de l'outil `train_test_split` avec une répartition 75%-25%.

3.1 Random Forest

Tout d'abord, nous avons appliqué l'algorithme de RandomForestClassifier de la librairie sklearn.ensemble sur nos données. La méthode du *Random Forest* permet d'obtenir un vote pour chaque arbre de la forêt, en utilisant la méthode du bagging, c'est à dire que la base de données est découpée en plusieurs sous-ensembles constitués d'échantillons tirés aléatoirement. Les arbres de décisions produits sur chacun de ces échantillons sont ensuite combinés dans le but de réduire la variance qui peut être très élevée dans le cas d'usage d'un seul arbre profond.
Le but est ici, après entrainement sur une partie de l'échantillon (X_train, Y_train), de réaliser des prédictions pour des échantillons non vus x' (X_test, Y_test) en procédant, dans notre cas de classification, par vote de la meilleure des prédictions de tous les arbres de régression individuels sur x'.

3.2 xgBoost

Le Boosting est un ensemble d'algorithmes qui s'inscrivent dans la continuité du Bagging, la famille d'algorithmes dont fait partie le *Random Forest*. Le Bagging se défini par un ensemble de petits algorithmes appelés "$Weak\ Learners$" qui votent chacun pour une prédiction. La moyenne de leurs votes donne alors le résultat final de l'algorithme global, nommé "$Strong\ Learner$" et noté S. La différence entre le Bagging est le Boosting naît du passage d'ensemble de "$Weak\ Learners$", notés w, indépendants, dont la diversité fait la force de la première famille d'algorithmes, à des w dépendants de leur précédent où w_i corrige les erreurs de w_{i-1} par apprentissage.

Nous utiliserons ici l'un de ces algorithmes appelé *xgBoost*, qui peut être assimilé à un assemblage des algorithmes *Gradient Boosting* et *Random Forest*. En effet, le fonctionnement est le même que pour le *Gradient Boosting* (i.e. il cherche à prédire la différence entre la moyenne des observations et la réalité, la somme des w_i non pondérée donnant le résultat, soit les **résidus**), excepté pour les "$Weak\ Learner$" qui reposent sur des arbres de décisions. Le but de ce fonctionnement est de fournir à chaque étapes les meilleurs petits algorithmes possibles. Ceci est possible par les multiples paramètres à optimiser qui en font l'algorithme le plus précis.
Ainsi, pour chaque arbre, w, un score de prédiction différent (une probabilité) est fourni selon les informations extraites des données par l'arbre. Le score final, qui permet de classifier la donnée dans un groupe ou un autre, est obtenue par la somme des scores donnée par les $w_{1\ a\ n}$.

Décrivons les étapes mathématiques de l'algorithme :

1. On définit les entrées suivantes : $\{(x_i, y_i)\}_{i=1}^{N}$, l'ensemble d'apprentissage ; α le taux d'apprentissage ; $L(y, F(x))$ la fonction de perte et W le nombre d'apprenants faibles.

Ensuite, l'algorithme XGBClassifier, disponible via la librairie xgboost, se défini par l'enchaînement des étapes suivantes :

2. Il faut initialiser le modèle par une valeur constante, soit : $\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta)$
3. Puis, il faut procéder au calcul des gradients, $\hat{g}_w$, et des hessiens, $\hat{h}_w$, prenant les valeurs suivantes pour $w \in [1, W]$: $\hat{g}_w(x_i) = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]$; $\hat{h}_w(x_i) = [\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}]$
4. Il faut ensuite ajuster les w, en minimisant la fonction suivante :
$$Arg\,min \left[\sum_{i=1}^{n} L(y_i, f^{t-1}(x_i) + \alpha.g(x_i)) \right] \tag{7}$$
5. Enfin, on obtient en sortie le résultat, soit la somme des estimations de chaque w :
$$\hat{f}(x) = \sum_{w=0}^{W} \hat{f}_w(x).$$

3.3 Logistic Regression

Ce modèle permet de prédire les relations entre la variable étudiée et l'ensemble des autres variables de la base de données. Il utilise l'approche de l'*estimateur du maximum de vraisemblance*(EMV) afin d'estimer les coefficients, on souhaite maximiser la log-vraissemblance pour trouver le coefficient optimal. Pour ce faire, on peut notamment mettre à zéro les dérivées de la vraisemblance logarithmique par rapport à chacun des paramètres β :

$$\frac{\partial \ell}{\partial \beta_m} = 0 = \sum_{k=1}^{K} y_k x_{mk} - \sum_{k=1}^{K} p(\boldsymbol{x}_k) x_{mk} \tag{8}$$

La prédiction s'effectue sur la base de seuils. Si la probabilité que l'évènement se produise est supérieure à un certain seuil (souvent fixé à 0.5), alors on attribue 1 à cet évènement, sinon il vaut 0. Plus généralement, la fonction logistique $p : \mathbb{R} \to (0,1) p : \mathbb{R} \to (0,1)$ s'écrit :

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{9}$$

où p est la probabilité qu'un évènement survienne, $\beta_0 \& \beta_1$, respectivement, l'intercept et le coefficient de régression multiplié par une valeur x du prédicteur.

Ici, on utilisera le modèle LogisticRegression du package sklearn.linear_model.

3.4 K-Nearest Neighbors

Enfin, nous avons appliqué le modèle KNNClassifier de la librairie sklearn.neighbors pour prédire le risque de fraude.
La classification par plus proches voisins est une méthode de classification non paramétrique qui stocke les instances des données d'apprentissage. Les KNN sont dérivées du classifieur bayésien, qui se définit comme suit :

$$f_g(x') = \frac{\mathbb{P}[Y' = g \cup X' = x']}{\mathbb{P}[X' = x']} \tag{10}$$

où Y', l'étiquette, et X', l'ensemble des points, sont les nouvelles observations indépendantes et g est le nombre de groupes.
Avec l'estimation des classes par KNN on cherche à estimer les probabilités P, car on ne connaît pas leur loi. Estimer les probabilités *a posteriori* revient à placer les k plus proches voisins de x' dans un ensemble appelé le "voisinage du point x'". Ainsi, plus les points sont proches plus ils se ressemblent : on a l'ensemble des points qui ressemblent le plus à x' parmi les N observations.

$$f_g(x') \approx \frac{\hat{\mathbb{E}}[\mathbb{1}_{\{Y'=g\}} \cdot \mathbb{1}_{\{X' \in V_k(x')\}}]}{\mathbb{P}[X' \in V_k(x')]} \quad \Leftrightarrow \quad \hat{f}_g(x') = \frac{1}{k} \sum_{i=1}^{N} \mathbb{1}_{(Y_i=g)} \cdot \mathbb{1}_{(Xi \in V_k(x'))} \tag{11}$$

où V_k est le voisinage, et $\mathbb{1}$ la fonction indicatrice.

Ajoutons qu'initialement, la classification KNN utilise des poids uniformes : on aura alors weights = 'uniform' par défaut et la valeur attribuée est calculée à partir d'un vote majoritaire simple des plus proches voisins.
L'un des avantages de cette classification est que la frontière entre les classes n'est pas linéaire. D'après sa définition, le clustering par KNN nécessite des données labellisées.

Après application et adaptation (tuning) de ces quatre modèles à nos données, nous en proposons l'évaluation au travers de la mesure *Accuracy*, de la courbe *ROC*.

4 Résultats

Nous présentons ici les résultats obtenus suite à la sélection des variables effectuée par $WeightofEvidence$. En effet, les résultats suite à l'application du $LASSO$ et du PCA pour, respectivement, la sélection des variables et réduction de dimension, étaient moins bons : il n'est pas judicieux de les présenter.

4.1 Métriques de performances

En $MachineLearning$, les métriques utilisées pour évaluer la performances d'un modèle sont pour la plupart issues de la matrice de confusion ($confusionmatrix$). La matrice de confusion donne une vision globale sur la qualité de la prédiction, puisqu'elle est composée des valeurs prédites et des valeurs réelles. Ainsi, elle compare les valeurs qui ont été correctement prédites ou non, et donc si le modèle a été performant.
La matrice de confusion est composée comme telle:

- ***True Positive*/Vrai Positif (TP) :** La valeur réelle et sa prédiction sont positives.
- ***False Positive*/Faux Positif (FP) :** La valeur réelle est négative et sa prédiction est positive ($TypeIError$).
- ***True Negative*/Vrai Négatif (TN) :** La valeur réelle et sa prédiction sont négatives.
- ***False Negative*/Faux Négatif (FN) :** La valeur réelle est positive et sa prédiction est négative (Type II Error).

		Actual	
		Positive	**Negative**
Predicted	**Positive**	TP	FP
	Negative	FN	TN

L'***Accuracy***, ou *Correct Prediction Ratio* (CPR), correspond au nombre de prédictions correctes sur le nombre total de prédictions. Elle mesure donc les évènements qui ont été correctement prédites par le modèle (le modèle a prédit une fraude, et elle a bien eu lieu).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

Le ***Recall***, ou *True Positive Rate* (TPR) ou *Sensitivity*, est une métrique de classsification qui montre le nombre de prédictions corectes sur le total des valeurs positives. Dans le cas de non fraude, le modèle mesure donc la proportion de non fraude prédites sur le nombre actuel de non fraude.

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

4.2 Valeurs des métriques

Comparaison des quatre modélisations

Model	Accuracy	Recall	AUC	Precision	F1 Score	F_β with $\beta = 2$	Confusion matrix
Logistic	**0.912***	**0.516***	**0.794***	0.943	0.667	1.0123	$\begin{pmatrix} 229540 & 1691 \\ 26438 & 28145 \end{pmatrix}$
KNN	0.859	0.469	0.751	0.715	0.566	0.8826	$\begin{pmatrix} 221013 & 10219 \\ 28990 & 25593 \end{pmatrix}$
xgBoost	0.904	0.511	0.530	0.976	0.670	1.011	$\begin{pmatrix} 230539 & 693 \\ 26710 & 27872 \end{pmatrix}$
Random Forest	0.903	0.510	0.727	**0.979***	**0.671***	**1.0124***	$\begin{pmatrix} 230630 & 579 \\ 26729 & 27875 \end{pmatrix}$

4.3 Courbes de performances

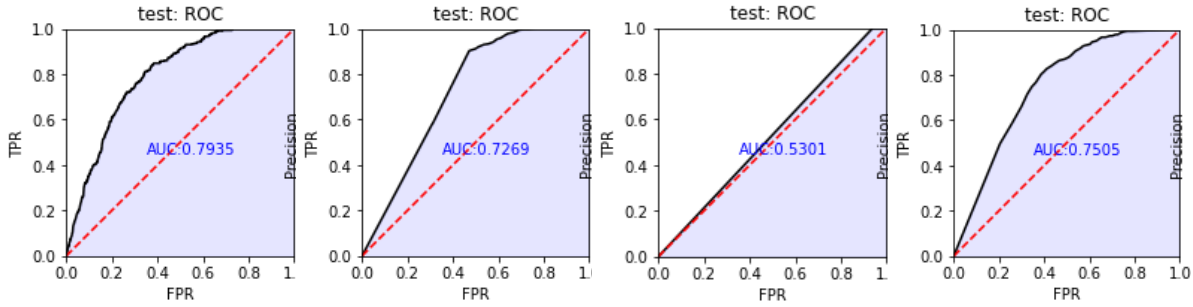

Figure 9: Courbe de performance ROC-AUC sur l'échantillon de test - de gauche à droite : LogisticRegression, RandomForest, xgBoost, KNN.

D'après les courbes ROC, qui permettent de tracer le nombre de vrais positifs en fonction des faux positifs, le meilleure modèle est la *Logistic Regression*, et le classifieur le moins fiable pour nos données et le *xgBoost*.

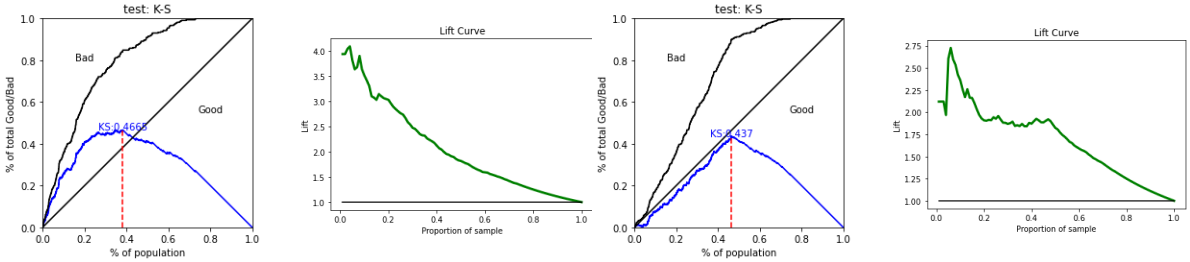

Figure 10: Test de Kolmogorov-Smirnov et Courbe de Lift sur l'échantillon de test - LogisticRegression (gauche) & RandomForest (droite)

Le test de Kolmogorov-Smirnov (KS) mesure la distance entre les distributions des classes positives et négatives, en prenant la plus grande distance entre les CDF - fonction de distribution - de chaque échantillon. Concernant la *Logistic Regression*, on voit qu'elle passe le test de KS tandis que le *Random Forest* est sous la droite de peu. De même, la courbe de Lift de ce premier modèle traduit la supériorité de sa performance. En effet, le score de 4 est le plus élevé de tout nos modèles. Le second modèle ne présente pas un Lift décroissant. Ici, si on choisit 5% de transactions à probabilités de fraude les plus élevées prévues, nous nous attendons à ce qu'il y ait deux fois plus de fraudes que ce qui serait attendu si nous avons contacté 5% des clients au hasard ; alors qu'à l'approche des 10% on est à 2.75 fois plus : notre modèle est donc moins performant des 0 à 5% qu'au niveau des 10% en terme de probabilité prédite. En comparaison, la régression logistique qui nous donne quatre fois plus de fraudes que ce qui serait attendu avec le modèle de hasard, est décroissante, comme attendue.

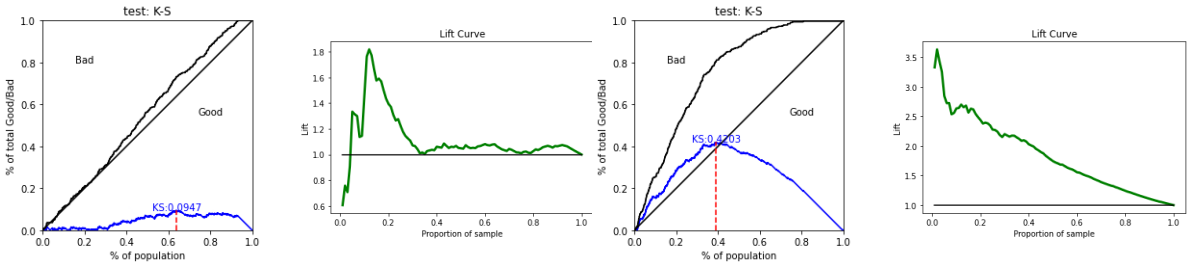

Figure 11: Test de Kolmogorov-Smirnov et Courbe de Lift sur l'échantillon de test - xgBoost (gauche) & KNN (droite)

Sur les graphes ci-dessus, on peut comparer les performances du *xgBoost* et du *K-Nearest Neighbors*. Nous

constatons que, parmi ces deux derniers modèles, seul le KNN passe le test de KS. Notre troisième modèle en est, de fait très loin, et sa courbe de Lift se situe même en partie sous le lift cumulé de 1 : indiquant que pour les 20 % de fraude les plus probable le modèle n'apporte rien de plus qu'un tirage aléatoire ; tandis que le quatrième et dernier modèle apporte une prédiction 3.5 à 2.5 fois plus importante que l'aléatoire sur les 10 premier pourcent des fraudes les plus probables.

5 Conclusion

Dans notre analyse, nous avons utilisé des algorithmes de *Machine Learning* pour analyser et prédire la fraudes dans les transactions bancaires. Au préalable, les données ont été rééquilibrées, puis réduites par *Weight of Evidence*, le LASSO et l'ACP avaient été appliqués aussi, cependant les résultats obtenus avec ceux-ci étaient mauvais. Pour sélectionner la modélisation la plus performante, nous avons utilisé des métriques issues de la matrice de confusion.

5.1 Choix du modèle

On remarque que globalement la *Logistic Regression* donne des résultats très bons, avec la valeur la plus élevée pour les métriques suivantes : l'Accuracy (0.912), Recall (0.516) et l'AUC (0.7935). Il en est de même pour sa courbe de lift, qui est meilleure, d'autant plus qu'elle passe le test de KS.
La modélisation par *Random Forest* quant à elle donne aussi de bons résultats, et arrive juste après la régression logistique en ce qui concerne l'Accuracy (0.903) et Recall (0.510). Au niveau de l'AUC, c'est la modélisation KNN qui arrive en second meilleur modèle après la régression logistique, avec une valeur de 0.7505.
En ce qui concerne le *xgBoost*, malgré de bonnes valeurs d'Accuracy et de Recall, l'AUC n'est pas très bon. Nous avons comparé nos modèles avec XGBoost mais il faut noter qu'il s'agit d'un modèle très complexe avec de nombreux paramètres. Cependant, un tel modèle demande de grandes performances machines afin de *tuner* correctement ces paramètres, ce que nous ne possédons pas. Cela pourrait expliquer pourquoi les performances de ce modèles sont moindres comparées aux autres.
Pour conclure, malgré de bons résultats pour chacune des modélisations, c'est la Logistic Regression qui se distingue comme meilleur choix de modèle. Une autre méthode, le VotingClassifier aurait pu être utilisée pour sélectionner le meilleur modèle de façon automatique et sans avoir à comparer les résultats.

VotingClassifier

Le principe de ce modèle est de proposer, à partir des résultats de chacun des classifieurs, la classe de sortie élue par un système de vote en fonction de la probabilité la plus élevée. Ainsi, le $Voting\ Classifier$ est un modèle d'apprentissage automatique unique qui s'entraîne avec un ensemble de modèles et agrège les résultats de chaque classificateur. Il permet donc de choisir le meilleur résultat sans avoir à déterminer la précision de chacun des modèles qui lui sont passés en argument.
Ce modèle est disponible dans le package `VotingClassifier` de la librairie sklearn.ensemble.