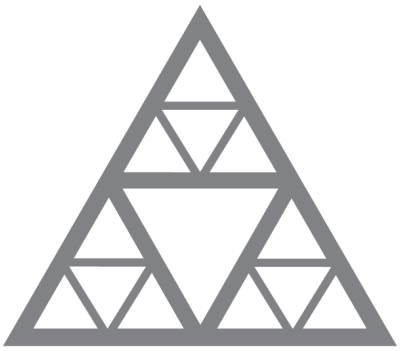
École nationale des ponts et chaussées

Février - Mai 2018



École des Ponts

ParisTech

Projet de département IMI

Outlier detection - Détection de fraudes

Par H. Andres, M. Bouazza, M. Casavane, M. Karpe

sous la direction de

J. Xu, analyste quantitatif à la Société Générale

Résumé

Il existe de nombreux algorithmes de détection d'aberrations. Ces algorithmes ont des comportements différents, et leur efficacité dépend de la distribution des données et du réglage de leurs hyperparamètres. Les 5 algorithmes testés dans le cadre de ce projet, à savoir *One-class SVM, Robust Estimator of Covariance, Isolation Forest, Local Outlier Factor* et *Autoencoder*, ont en effet donné des résultats totalement différents selon les bases de données et les paramètres utilisés dans le cadre de ce projet.

Une fois les algorithmes implémentés, la difficulté du projet réside dans leur paramétrage optimal. Il s'agit d'abord de définir ce qu'est un paramétrage optimal, à savoir quelle quantité on cherche à minimiser ou à maximiser. Dans ce projet, nous nous sommes intéressés à l'application de la détection de points aberrants à la détection de fraudes bancaires. On cherche donc à maximiser le nombre de fraudes détectées, puisque chaque fraude représente un coût pour la banque. Il faut donc modéliser une fonction de coût, qui sera fonction de la bonne ou mauvaise détection de données normales ou aberrantes.

Cette fonction de coût étant modélisée, l'optimisation des hyperparamètres peut être lancée selon des méthodes traditionnelles qui consistent à tester plusieurs fois les algorithmes sur différentes valeurs de paramètres. On cherche alors à minimiser la fonction de coût, ou autrement dit, à maximiser le bénéfice engendrée par la détection correcte des fraudes.

Les résultats obtenus dans le temps imparti du projet sont encourageants, et confirment que certains algorithmes sont plus performants que d'autres sur les données bancaires qui ont été traitées. L'une des difficultés majeures rencontrées lors de ce projet a en effet été le temps d'exécution des algorithmes sur les jeux de données traités. De bien meilleurs résultats pourraient sûrement être obtenus si le projet venait à être continué sur une durée bien plus longue, et était mené sur du matériel informatique plus puissant que des ordinateurs d'étudiants.

Sommaire

1 Introduction **4**

2 Modélisation économique du coût de la fraude **5**
2.1 Evaluation du coût de la fraude pour le client 5
2.2 Evaluation du coût de la fraude pour la banque 5
2.3 Stratégie de minimisation du coût de la fraude 6

3 Algorithmes de détection de données aberrantes **8**
3.1 Robust Estimator of Covariance 8
3.2 One-class SVM 10
3.3 Isolation Forest 12
3.4 Local Outlier Factor 14
3.5 Autoencoder 16

4 Expérimentations et résultats **18**
4.1 Description des jeux de données 18
4.2 Recherche des hyperparamètres optimaux 19
4.3 Description et analyse des résultats 20
4.4 Combinaison des algorithmes de détection d'aberrations 22

5 Conclusion **24**
5.1 Synthèse des résultats 24
5.2 Limites du projet et perspectives d'amélioration 24

1 Introduction

Le projet de département synthétisé par ce rapport traite de la détection d'aberrations (ou *outlier detection*) dans le secteur financier par des méthodes de *data science* (notamment *machine learning*). Ce projet est proposé par la Société Générale et a été encadré par M. Jiali Xu, analyste quantitatif.

Un *outlier* (une observation aberrante) est une observation qui dévie tellement du reste des observations qu'il est probable qu'elle ait été générée par un mécanisme différent. Ces *outliers* sont souvent liés à une erreur de mesure, du bruit, ou à un phénomène financier spécifique. La détection des *outliers* est un des domaines de recherche les plus importants de la *data science* en finance.

Le but de ce projet est de détecter les *outliers* dans les données financières en utilisant les techniques classiques d'apprentissage automatique. Les résultats de ce projet peuvent améliorer la performance d'un modèle prédictif ou directement détecter les fraudes ou les comportements anormaux des clients.

Le projet s'est déroulé en différentes phases de travail :

- La phase décrite dans ce rapport en premier lieu est une phase de modélisation économique du coût de la fraude. Cette phase, bien que placée en début de rapport, s'est déroulée à la fin du projet pour permettre le calibrage et l'amélioration de nos algorithmes à partir d'une fonction de coût dépendant de la détection ou non des données aberrantes. Par souci de clarté, et de façon à comprendre dès le début de la lecture les enjeux de la détection d'aberrations et de la nécessité de limiter les coûts liés à la non détection, cette phase de modélisation est décrite dans la première partie de ce rapport.
- La deuxième partie de ce rapport décrit les différents algorithmes d'apprentissage automatique implémentés pendant la première phase de ce projet. Nous avons implémenté 5 algorithmes, dont 3 proposés par l'encadrant (*One-class SVM*, *Isolation Forest*, *Autoencoder*) et 2 que nous avons découverts lors de nos recherches et qu'il nous a apparu comme pertinent d'implémenter (*Robust Estimator of Covariance* et *Local Outlier Factor*).
- La troisième partie de ce rapport décrit les différents jeux de données utilisés pour expérimenter nos algorithmes, les processus expérimentaux de recherche de paramètres optimaux qui dépendent de la fonction de coût décrite en première partie, les résultats obtenus et analyse ces résultats.

En conclusion, nous procèderons à une synthèse des résultats obtenus dans le cadre de ce projet, et nous indiquerons les limites de ce projet, notamment dues à un temps limité pour la réalisation de celui-ci, alors que ces problématiques constituent le travail à temps plein d'employés du secteur bancaire sur une durée bien plus importante.

2 Modélisation économique du coût de la fraude

Avant de nous intéresser à la description technique des algorithmes de détection de points aberrants que nous avons utilisés, il convient de décrire les enjeux économiques de la détection de fraudes aux moyens de paiement afin de mesurer l'intérêt de notre démarche. Le rapport annuel de l'Observatoire de la sécurité des moyens de paiement [2] fournit à cet effet un certain nombre de données chiffrées exploitables et récentes.

Les moyens de paiement considérés sont la carte de paiement, le chèque, les virements et les prélèvements. Ce sont néanmoins les paiements par carte et par chèque qui sont les plus touchés par la fraude. Les transactions frauduleuses par carte de paiement et par chèque représentent, en effet, plus de 95% du volume des transactions frauduleuses et presque 80% du montant global des fraudes.

Bien que le taux de fraude - tout moyen de paiement confondu - est extrêmement faible (de l'ordre de 0.006 %), le montant total de la fraude aux transactions scripturales est de 800 millions d'euros en France en 2016, ce qui représente environ 4,8 millions de transactions frauduleuses. Ce montant représente un coût non négligeable à la fois pour les utilisateurs et les fournisseurs de services de paiement, que l'on assimilera dans la suite aux banques.

2.1 Evaluation du coût de la fraude pour le client

Une fraude bancaire n'a pas véritablement de coût monétaire pour l'utilisateur qui en est victime. En effet, si la victime d'une fraude bancaire la signale dans les treize mois qui suivent la fraude et que sa responsabilité n'est pas engagée, la banque est légalement contrainte de rembourser l'intégralité des frais liés à la fraude (y compris les frais de découverts et d'opposition).

Le coût de la fraude est donc plutôt un coût en termes de temps nécessaire à faire les démarches de signalement auprès de la banque, et un coût lié aux «dommages collatéraux» de la fraude : le compte peut être effectivement bloqué pendant un certain temps, rendant impossible toute opération bancaire.

2.2 Evaluation du coût de la fraude pour la banque

En raison du fait que la banque soit contrainte de rembourser l'intégralité du coût des fraudes, la fraude représente pour elle un important coût monétaire, compte tenu du montant global des fraudes détaillé en introduction de cette section. Ce montant global étant supporté par l'ensemble des banques françaises, il est plus pertinent de s'intéresser au moyen d'une fraude. Le montant moyen d'une fraude, tout moyen de paiement confondu, est évalué par l'Observatoire de la sécurité des moyens de paiement à environ 160 euros.

Ainsi, si on considère une banque qui concentre 10 % des transactions bancaires, les fraudes représentent un coût d'environ 80 millions d'euros. Les banques possèdent naturellement des assurances contre ce risque donc le coût réel n'est pas aussi élevé mais on peut supposer qu'il reste conséquent. Pour simplifier, on suppposera donc que les banques ne possèdent pas d'assurances pour se protéger des fraudes.

2.3 Stratégie de minimisation du coût de la fraude

Compte tenu de ces coûts importants, l'objectif de notre projet est de détecter les fraudes, qui sont donc vues comme des transactions aberrantes dans un ensemble de transactions, afin de minimiser le coût de la fraude. En effet, si une transaction est détectée comme une fraude, il est possible d'empêcher que la transaction soit effectuée si elle est en cours de validation (ce qui permet d'éviter le remboursement) ou de bloquer le compte si la transaction a déjà été validée (ce qui permet d'éviter de potentielles récidives).

Néanmoins, il est évident que l'algorithme fera des erreurs, ce qui pourrait engendrer des coûts supplémentaires à la banque. Il convient donc d'examiner les 4 cas possibles pour une transaction évaluée par un algorithme de détection d'*outliers*.

- La transaction est détectée comme frauduleuse et elle l'est effectivement. C'est ce qu'on appelle un vrai positif.
- La transaction est détectée comme frauduleuse mais elle ne l'est pas. C'est ce qu'on appelle un faux positif.
- La transaction est détectée comme normale alors qu'elle est frauduleuse. C'est ce qu'on appelle un faux négatif.
- La transaction est détectée comme normale et elle l'est effectivement. C'est ce qu'on appelle un vrai négatif.

La catégorisation de la transaction nécessite néanmoins la connaissance de la vraie nature de la transaction (normale ou frauduleuse). On pourrait demander à du personnel d'effectuer les vérifications mais cela serait très coûteux pour la banque. Une solution bien moins coûteuse et plus efficace serait d'envoyer un mail ou un SMS automatiquement au client concerné, chaque fois qu'une transaction est détectée comme frauduleuse – un peu à la manière de Google lorsqu'il détecte une connexion inhabituelle. Le client pourra ainsi indiquer directement si c'est bien lui qui a fait la transaction ou non.

Ceci étant dit, examinons ce que représente chacun des cas en termes de coût ou de profit :

- Si la détection est un vrai positif, la banque économise le montant du remboursement de la fraude, soit 160€.
- Si la détection est un faux négatif, la banque perd 160€.

— Le cas du faux positif ne coûte presque rien dans la mesure où détecter une transaction comme frauduleuse alors qu'elle ne l'est pas ne coûte que l'envoi d'un mail ou SMS au client. Nous estimons le coût à 1€.
— Si la détection est un vrai négatif, il n'y a ni gain ni perte.

Cette estimation des coûts et des profits est très importante pour la suite. En effet, cela va nous servir de critère pour comparer et optimiser les performances des différents algorithmes. Plus précisément, on cherche en réalité à maximiser pour un ensemble de transactions données le critère suivant :

$$\pi = 160(TPR - FNR) - FPR \quad \text{avec}$$

$$TPR = \frac{\#\text{ Vrais positifs}}{\#\text{ Positifs}} \qquad FNR = \frac{\#\text{ Faux négatifs}}{\#\text{ Positifs}} \qquad FPR = \frac{\#\text{ Faux positifs}}{\#\text{ Négatifs}}$$

Ce critère traduit simplement le fait que l'on veut maximiser le nombre de transactions frauduleuses bien détectées et minimiser le nombre de transactions frauduleuses mal détectées, tout en gardant un nombre de transactions normales détectées comme frauduleuses raisonnable (notamment afin de ne pas inonder les clients de messages).

		PREDICTED	
		"P"	"N"
ACTUAL	P	TRUE POSITIVE Outlier detected as outlier Profit: €160	FALSE NEGATIVE Outlier detected as inlier Cost: €160
	N	FALSE POSITIVE Inlier detected as outlier Cost: €1	TRUE NEGATIVE Inlier detected as inlier Profit: €0

FIGURE 1 – Tableau synthétisant les différents coûts en fonction de la bonne ou mauvaise détection d'*outliers* ou d'*inliers* (données conformes).

3 Algorithmes de détection de données aberrantes

Dans cette section, nous décrivons les 5 algorithmes de détection d'aberrations étudiés dans le cadre de ce projet : *Robust Estimator of Covariance, One-class SVM, Isolation Forest, Local Outlier Factor* et *Autoencoder*. Ces 5 algorithmes ont tous des comportements différents pour procéder à la détection d'aberrations, ce qui justifie de les étudier et de les tester un par par dans cette section, puis d'essayer de les combiner pour aboutir à de meilleurs résultats que par l'utilisation d'un algorithme seul *(Section 4.3.)*.

3.1 Robust Estimator of Covariance

L'application de cet algorithme présuppose que les données sont caractérisées par une distribution dont le contour de densité est elliptique, le cas le plus commode est le cas gaussien. L'estimation est dite robuste car le modèle est imposé par la majorité des données, ce qui correspond à un apprentissage similaire à celui qui se ferait sur le même ensemble de données, privé des *outliers*. Par conséquent, toutes les observations déviant du modèle robuste sont considérées comme étant des *outliers*.

Considérons que notre ensemble de données $X = \{x_1, ..., x_n\}$ avec $x_i \in \mathbb{R}^p$ suit une loi gaussienne de vecteur de moyennes μ et de covariance Σ. Les estimateurs classiques de μ et Σ sont respectivement la moyenne et la matrice de covariance :

$$\overline{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ et } S_n = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(x_i - \overline{x})^T$$

On introduit la distance de Mahalanobis à la moyenne :

$$MD(x_i) = \sqrt{(x_i - \overline{x})^T S_n^{-1}(x_i - \overline{x})}$$

En pratique, μ and Σ sont évalués par des estimateurs. Etant donné que l'estimateur du maximum de vraisemblance est très sensible à la présence d'*outliers* dans notre ensemble de données, la distance de Mahalanobis correspondante l'est aussi. Ainsi serait-il judicieux d'utiliser le *Minimum Covariance Determinant estimator estimation*, qui, lui, est plus résistant aux observations erronnées.

Le principe de cet algorithme est de considérer une donnée comme un *outlier* si elle n'appartient pas à l'ellipsoïde de tolérance défini par :

$$\{x \mid MD(x) \leq \sqrt{\chi^2_{p,0.975}}\}$$

où $\chi^2_{p,0.975}$ désigne le quantile à 97.5% de la loi χ^2 à p degrés de liberté.

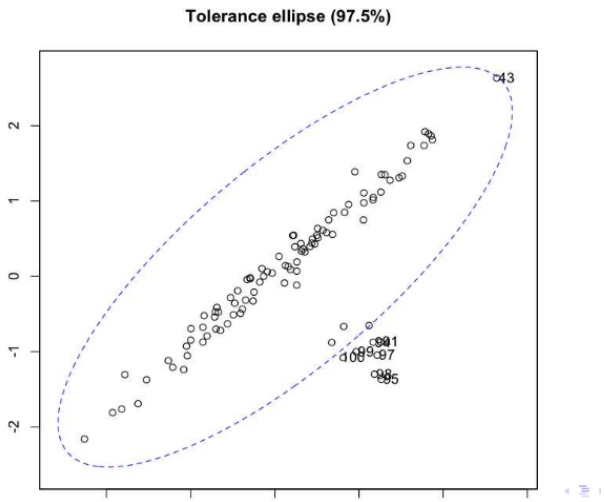


FIGURE 2 – Illustration d'une ellipsoïde de tolérance sur des données plan réparties selon une droite et avec quelques *outliers*.

Les frontières déterminées ayant la forme d'une ellipse, on devine rapidement les limites de l'algorithme si jamais les données étaient réparties de façon originale. Ainsi, on voit sur la figure ci-dessous que la frontière de décision déterminée est beaucoup trop grande, et que des *outliers* situés au centre de l'image seraient considérés comme des *inliers* alors qu'ils n'en sont pas.

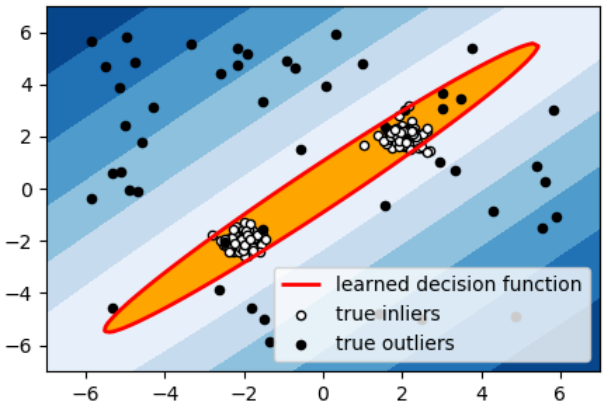


FIGURE 3 – Illustration d'une ellipsoïde de tolérance sur des données plan réparties selon deux gaussiennes excentrées et avec quelques *outliers*.

3.2 One-class SVM

L'algorithme *One-class SVM* consiste à résoudre le programme d'optimisation suivant :

$$\begin{aligned} \min_{w,\xi_i,\rho} \quad & \frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \\ \text{s.t.} \quad & (w \cdot \phi(x_i)) \geq \rho - \xi_i,\ \xi_i \geq 0,\ \forall i = 1, \ldots, n \end{aligned}$$

où ν représente une borne supérieure sur la fraction des *outliers* et une borne inférieure sur le nombre de vecteurs supports, et les ξ_i représentent des variables de contrôle qui représentent la distance des données à l'hyperplan d'équation $w \cdot \phi(x_i) - \rho$.

L'algorithme *One-class SVM* fonctionne selon les mêmes principes que les algorithmes *SVM*, qui calculent également des séparateurs linéaires. Ainsi, lorsqu'il est impossible de séparer des données par un hyperplan, on applique aux données une transformation caractérisée par un noyau K, de façon à augmenter la dimensionnalité des données, et pouvoir ensuite déterminer un hyperplan séparateur dans cet espace de plus grande dimension.

Le noyau généralement utilisé pour *One-class SVM* est le noyau gaussien :

$$K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$$

Les noyaux linéaire et polynomial peuvent également être utilisés, mais leur utilisation est moins courante et est pertinente seulement pour des cas très particuliers.

La figure ci-dessous représente un ensemble de points répartis dans un plan selon une loi gaussienne. Ainsi, la figure de gauche représente les points dans le plan (x, y), et la figure de droite représente la densité de points, telle qu'elle pourrait être représentée selon une dimension z. Il s'agit alors d'appliquer cette densité sur l'ensemble de points, en ajoutant à chaque point (x, y) une troisième coordonnée z d'autant plus grande que la densité est importante.

Ainsi, il sera possible de déterminer un séparateur linéaire, une frontière, qui sera un hyperplan d'équation $z = z_0$, où z_0 dépendra de la valeur de l'hyperparamètre ν. Par exemple, on peut fixer un hyperparamètre ν_0 tel que 5% des points seront situés en-dessous de l'hyperplan $z = z_0$.

Enfin, en projetant la frontière de dimension 3 dans l'espace de dimension 2, on obtient la frontière circulaire tracée en rouge dans la figure de gauche ci-dessous. On identifie alors les *outliers* comme les données situées en dehors de ce cercle, et les *inliers* (données conformes) comme les données situées à l'intérieur de celui-ci.

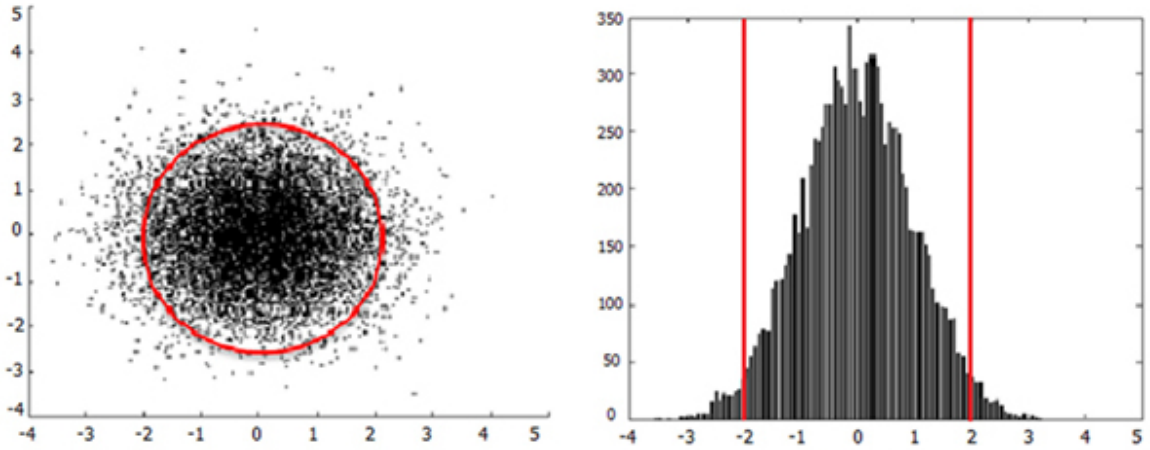


FIGURE 4 – Illustration du principe de séparation par SVM sur un ensemble de points à 2 dimensions.

Dans le cadre de l'algorithme *One-class SVM*, qui est un algorithme de classification, on peut décider d'étiqueter les *inliers* comme des positifs $(+1)$, et les outliers comme des négatifs (-1). La fonction de décision est alors la suivante :

$$f(x) = \text{sgn}((w \cdot \phi(x_i)) - \rho) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i K(x, x_i) - \rho\right)$$

On remarquera que les conventions d'étiquetage choisies par [4] sont le contraire de celle que nous avons définies dans le modèle économique présenté en *Section 2*, et également le contraire des conventions définies par *Scikit-learn* pour les fonctions de *score*.

3.3 Isolation Forest

L'*Isolation Forest* est un algorithme facile à comprendre et à implémenter. Il repose sur deux hypothèses sur la nature des *outliers* :

— Ils sont relativement peu nombreux par rapport aux points normaux

— Certains de leurs attributs ont des valeurs très différentes de celles des points normaux.

En raison de ces deux propriétés, les *outliers* sont très sensibles à un mécanisme d'isolation. Le mécanisme d'isolation utilisé est celui de l'arbre binaire de recherche. L'idée générale de la détection de points aberrants par *Isolation Forest* est le fait que les points aberrants, qui sont plus sensibles à l'isolation, sont donc plus susceptibles d'être isolés plus «rapidement» que les points normaux. Détaillons maintenant l'algorithme et notamment la construction de l'arbre d'isolation ainsi que la façon de détecter les points «rapidement» isolés.

Construction d'un arbre d'isolation : On part d'un jeu de données d'apprentissage $(x_1, \ldots, x_n)$ avec x_i dans $\mathbb{R}^d$. On choisit aléatoirement une dimension q entre 1 et d ainsi qu'un seuil de séparation p entre $\min_i x_i^q$ et $\max_i x_i^q$. On sépare ainsi le jeu de données en deux ensembles :

$$\{x_i, x_i^q \leq p\} \text{ et } \{x_i, x_i^q > p\}$$

Cette étape constitue le premier noeud de l'arbre. Tous les points qui sont dans le premier ensemble sont ajoutés à la branche gauche du noeud et tous les points qui sont dans le deuxième ensemble sont ajoutés à la branche droite du noeud. On poursuit ainsi récursivement la partition du jeu de données en coupant à chaque fois les ensembles obtenus en deux ensembles. On s'arrête lorsque tous les ensembles sont des singletons.

L'arbre obtenu est donc composé de noeuds qui correspondent aux «coupes» effectuées et les feuilles correspondent aux points. Le principe de cet algorithme est représenté sur le graphique ci-dessous sur lequel, les barres violettes sont les coupes, les points bleus sont les points normaux et les points violets sont les points anormaux (les points sont générés selon un mélange de deux gaussiennes).

Détection des points aberrants : On construit un grand nombre (généralement 100) d'arbres d'isolation pour constituer une forêt (d'où le nom d'*Isolation Forest*). Chaque arbre d'isolation n'est par contre pas construit à partir de l'ensemble des données d'apprentissage. On construit en fait chaque arbre à partir d'un échantillon des données d'apprentissage.

Ensuite, on calcule pour chaque point sa profondeur moyenne dans la forêt. Plus un point est en moyenne proche de la racine, plus il est susceptible d'être isolé rapidement et donc plus il est susceptible d'être un point aberrant. En pratique, on calcule un score pour chaque point à partir de la profondeur moyenne qui est d'autant plus grand que le point est proche de la racine. Les points dont les scores sont les plus élevés sont détectés comme des *outliers*.

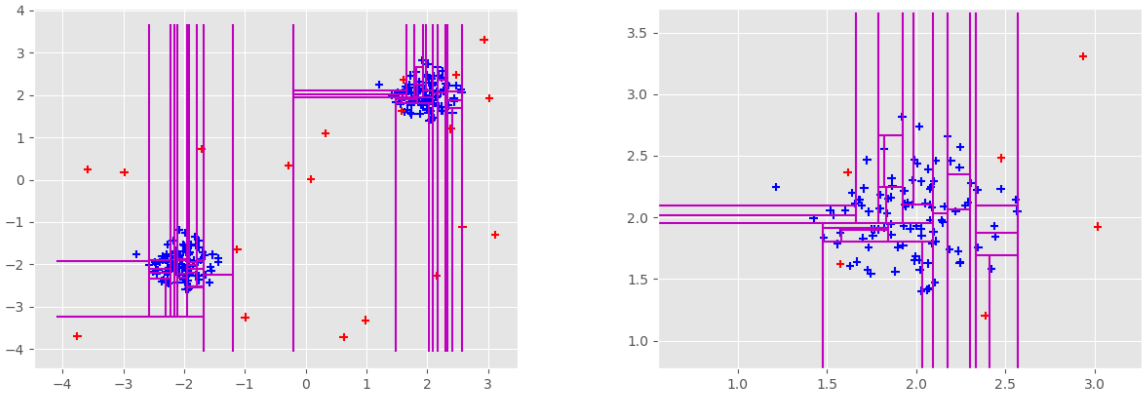


FIGURE 5 – Illustration du principe de l'*Isolation Forest* (l'image de droite est un zoom sur l'image de gauche).

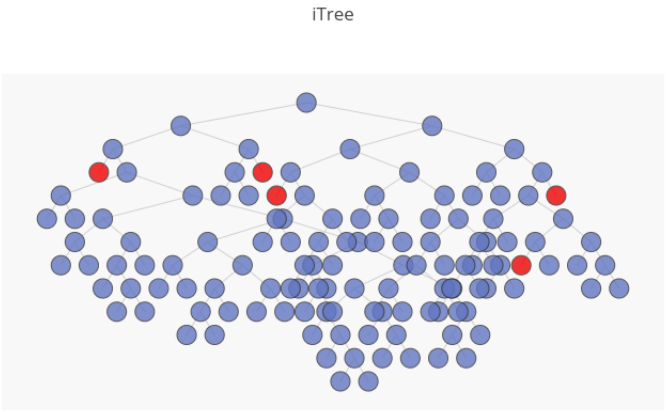


FIGURE 6 – Arbre construit à partir des données présentées à la figure précédente. Les points rouges correspondent aux *outliers*. On constate qu'ils ont globalement une profondeur faible.

L'une des grandes forces de l'*Isolation Forest* est le fait qu'elle ne repose pas sur l'utilisation d'une distance ou d'une mesure de densité pour détecter les points aberrants. De plus, il fonctionne très bien sur de très grands jeux de données en grande dimension. Néanmoins, nous avons pu constater que sur des jeux de données comportant un grand nombre de points aberrants, l'*Isolation Forest* n'est pas très performant. Cela s'explique par le fait que l'*Isolation Forest* exige que les points aberrants soient en faible nombre par rapport aux points normaux.

3.4 Local Outlier Factor

La méthode *Local Outlier Factor* est une méthode de détection de données aberrantes récente (2000). Elle a été mise à disposition dans la bibliothèque *scikit-learn* de Python en Septembre 2016, fruit du travail du même chercheur qui avait contribué à l'implémentation de *"Random Forest"* : Nicolas Goix.

Idée de l'Algorithme : L'algorithme est basé sur une idée assez classique, on va comparer chaque donnée avec ses k plus proches voisins et la donnée sera considérée comme aberrante si on constate une différence. Pour chaque donnée x on notera $N_k(x)$ l'ensemble des k plus proches voisins de x, on définit également la distance de x à son $k^{\text{ème}}$ plus proche voisin par :

$$d_k(x) = \max_{y \in N_k(x)} d(x, y)$$

avec d la métrique choisie dans l'espace des données.

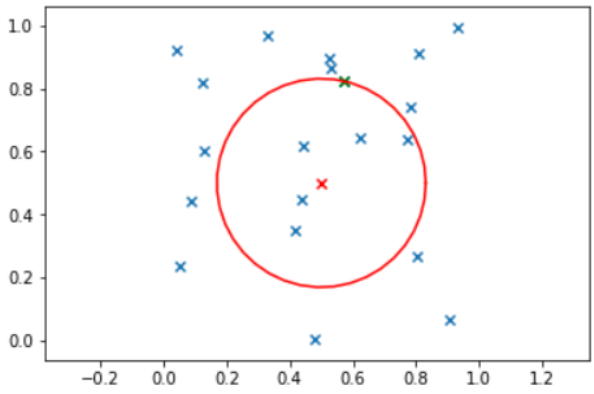


FIGURE 7 – Illustration de la distance au $k^{\text{ème}}$ plus proche voisin du point rouge pour $k = 6$.

Densité locale d'une donnée : Dans l'optique de comparer une donnée avec ses k plus proches voisins, l'algorithme introduit une notion de "densité locale". On définit dans un premier temps la distance d'atteinte d'une donnée x depuis une donnée y par la formule :

$$DA_k(x, y) = max\{d_k(y), d(x, y)\}$$

On remarque notamment que lorsque $x \in N_k(y)$, on a $DA_k(x, y) = d_k(x, y)$ et sinon la distance d'atteinte de x depuis y est la vraie distance entre x et y. À partir de cette notion, on définit la densité locale de x par l'inverse de la moyenne de la distance d'atteinte de x par ses k plus proches voisins :

$$DL_k(x) = \frac{k}{\sum_{y \in N_k(x)} DA_k(x, y)}$$

Remarque : Le nom de densité locale est justifié car pour une donnée appartenant à un cluster contenant un nombre de données supérieur à k, on aura une distance $d_k(y)$ faible pour toutes les données y du cluster, donc une distance d'atteinte $DA_k(x)$ relativement faible et par conséquence une densité locale $DL_k(x)$ élevée.

L'algorithme calcule enfin le score "Local Outlier Factor" de la donnée x qui est égal au quotient de la moyenne des densités locales des k plus proches voisins de x :

$$LOF_k(x) = \frac{\sum_{y \in N_k(x)} DL_k(y)}{k \times DL_(x)}$$

Un score faible ou proche de 1 signifie que la densité locale de x est plus grande, ou du même ordre de grandeur que celles de ses voisins, cela correspond alors à un *inlier*. Au contraire, lorsque x est un *outlier* ce score sera grand, surtout si x est à proximité d'un cluster car ses k plus proches voisins appartiendront à un cluster et leurs densités locales seront bien plus grandes que celle de x.

L'algorithme *Local Outlier Factor* consiste alors à calculer pour chaque donnée x le score $LOF_k(x)$, puis il qualifie d'*outlier* les $\gamma\%$ données dont le score est le plus élévé, γ étant le pourcentage d'*outliers* présents dans le jeu de données et qui est un paramètre de la méthode.

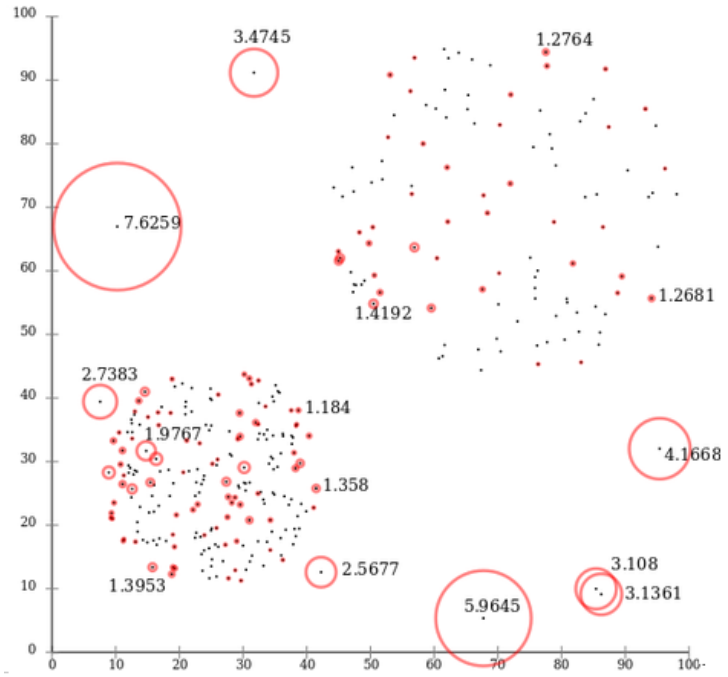


FIGURE 8 – Scores pour la méthode *Local Outlier Factor* sur un jeu de données en 2 dimensions.

3.5 Autoencoder

Perceptron multicouche : Un perceptron multicouche est un type de réseau de neurones, organisé en plusieurs couches, dans lequel l'information circule de la couche d'entrée vers la couche de sortie, en passant par des couches intermédiaires dans lesquelles l'information est cachée. On note n_l le nombre de neurones composant la couche l, c'est-à-dire le nombre d'informations contenues dans la couche l et on note ces informations : $a_1^l, ...a_{n_l}^l$. Entre la couche l et la couche $l+1$, les informations subissent dans un premier temps une transformation linéaire :

$$z_i^{l+1} = \sum_{j=1}^{n_l} W_{i,j}^l a_j^l \ \forall i \in \{1, .., n_{l+1}\}$$

puis une transformation non linéaire via une fonction f appelée fonction d'activation :

$$a_i^{l+1} = f(z_i^{l+1}) \ \forall i \in \{1, .., n_{l+1}\}$$

Dans la littérature, on retrouve souvent les fonctions d'activation "sigmoïde" $f(z) = \frac{1}{1+e^{-z}}$ et "ReLu" (Retified Linear unit) $f(z) = \max(z, 0)$.

La dernière couche $l = n_{couche}$ contient l'information de sortie dite *output* qu'on note $\{\tilde{y_1}, ..., \tilde{y_{d'}}\}$ avec $d' = n_{n_{couche}}$, et dans un problème d'apprentissage supervisé on va construire les poids $\{(W_{i,j}^1)_{1\leq i\leq n_2, 1\leq j\leq n_1}, ..., (W_{i,j}^l)_{1\leq i\leq n_{l+1}, 1\leq j\leq n_l}, ...\}$ qui vont minimiser l'erreur $\frac{1}{2}\|Y - \tilde{Y}\|^2_{\mathbb{R}^{n\times d'}}$ lorsqu'on dispose d'un jeu de données labellisées $\{x^1, .., x^n\} \subset \mathbb{R}^d, \{y^1, .., y^n\} \subset \mathbb{R}^{d'}$ et où on note $Y = (y_j^i)$, $\tilde{Y} = (\tilde{y_j^i})$.

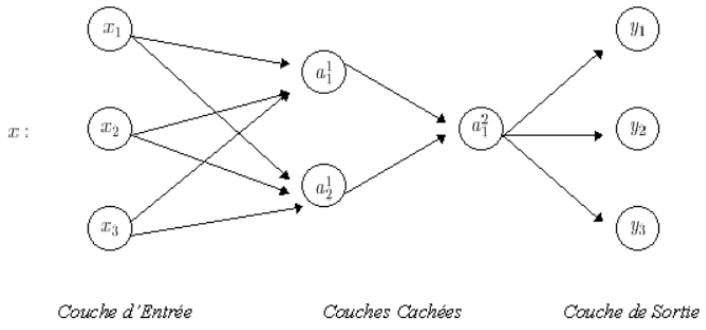


FIGURE 9 – Schéma d'un perceptron multicouche à 4 couches.

Autoencoder : Un autoencoder est un cas particulier de perceptron multicouche composé de deux parties symétriques dont le but est d'approximer la fonction identité. En d'autre terme, pour une donnée d'entrée $x \in \mathbb{R}^d$ on va essayer de récupérer une donnée de sortie $\tilde{x} \approx x$.

La première partie du réseau de neurones, dite partie "codante" peut être représentée par une fonction $\psi : x \in \mathbb{R}^d \longmapsto z \in \mathbb{R}^{d'}$. Si on a $d' < d$ on voit que la partie codante vient compresser la donnée x en une donnée de plus petite dimension z, dans le cas de détection d'*outliers*, on se placera dans cette situation. En pratique l'autoencoder, en compressant les données et en les reconstruisant, apprend une certaine représentation du jeu de données. La deuxième partie du réseau de neurone est la partie "décodante" qui cherche à reconstruire la donnée x à partir de la donnée compressée z, on peut la représenter par une fonction $\phi : z \in \mathbb{R}^d \longmapsto \tilde{x} \in \mathbb{R}^d$, et on cherche à minimiser l'erreur $\|x - \phi \circ \psi(x)\|_{\mathbb{R}^d}$ pour chaque donnée x.

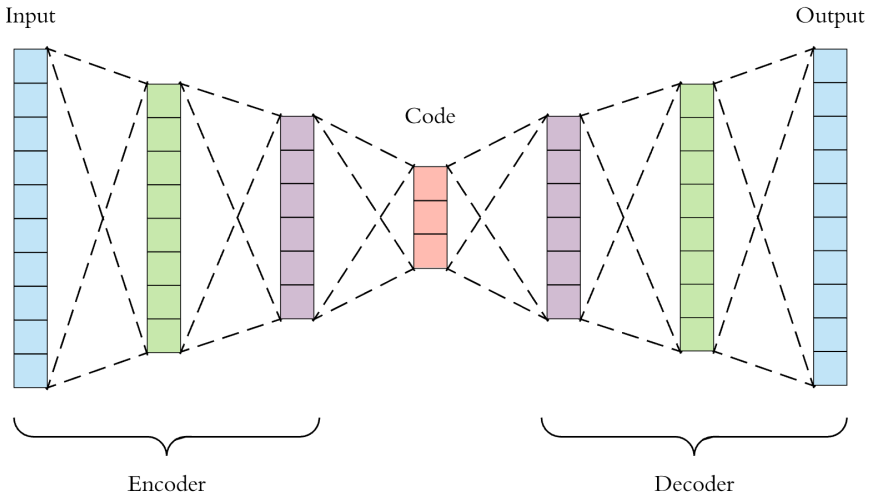


FIGURE 10 – Schéma représentatif d'un autoencoder.

Détection d'outliers : Puisque l'autoencoder crée une représentation réduite des données, il est naturellement adéquat pour la détection de données aberrantes. En effet, les données aberrantes seront plus difficiles à représenter sous forme réduite, et alors on constatera une différence plus grande entre la donnée reconstruite et la donnée initiale. Il est donc naturel de considérer le "score d'*outlier*" correspondant à la norme de la différence entre la donnée initiale et la donnée reconstruite, puis de classer la donnée comme *outlier* à partir d'une valeur seuil de ce score (ou bien de sélectionner un nombre de données ayant les scores les plus élevés lorsqu'on connaît la proportion d'*outliers*).

Limites : L'autoencoder est un système d'apprentissage composé d'un nombre de paramètres assez conséquent (coefficients des matrices de poids de chaque couche). Ainsi si on entraîne l'autoencoder sur un nombre de données assez réduit, on va avoir un phénomène d'*overfitting*, toutes les données et donc a fortiori les *outliers* seront bien représentés dans le modèle réduit, et bien reconstruits, ce qui rendra difficile leur détection. Au contraire, si on se donne un jeu de données trop large, les temps de calculs seront trop importants pour que le modèle soit efficace.

4 Expérimentations et résultats

4.1 Description des jeux de données

Plusieurs jeux de données ont été utilisés pour implémenter, tester et optimiser les différents algorithmes. Il y a trois principaux jeux de données :

— *Synthetic* : jeu de données généré artificiellement, fourni par M. Xu au début du projet pour pouvoir tester l'implémentation des algorithmes de détection d'aberrations.

— *Creditcard* : jeu de données anonymisé correspondant à des transactions par carte bancaire, disponible librement sur *Kaggle*. [1]

— *KDD10* : jeu de données correspondant à 10% du jeu de données proposé par la *KDD Cup 1999*, qui organisait une compétition de détection des connexions Internet frauduleuses dans le jeu de données fourni.

Les caractéristiques de ces 3 jeux de données sont décrites dans les 3 premières lignes du tableau 11. Les lignes suivantes décrivent des sous-jeux de données extraits de *KDD10*.

	Données	Dimensions	Outliers
Synthetic	1000	100	13,6 %
Creditcard	284807	29	0,17 %
KDD10	494021	38	1,77 %
SA10	100655	38	3,36 %
SF10	73237	3	4,50 %
HTTP10	58725	3	3,76 %
SMTP10	9571	3	0,03 %

FIGURE 11 – Description des jeux de données utilisés dans le cadre du projet.

Les constats principaux réalisés en traitant ces jeux de données dans le cadre de la recherche de données aberrantes sont les suivants :

— *Synthetic* ayant été généré artificiellement, l'interprétation des résultats de nos algorithmes sur ce jeu de données est peu pertinente. Ce jeu de données étant très petit et n'ayant pas de signification dans le monde réel, il permet uniquement de vérifier que les algorithmes s'exécutent correctement.

— *SMTP10* présentant un nombre d'outliers très faible (seulement 3), il est peu pertinent de travailler sur un tel jeu de données pour vérifier le bon comportement de nos algorithmes et calibrer les hyperparamètres de ces derniers.

1. Ce jeu de données nous a été indiqué par M. Xu après que ce dernier a essayé d'obtenir des données de la banque pour tester nos algorithmes, mais les démarches pour pouvoir utiliser les données de la banque auraient été trop longues et fastidieuses pour être réalisées dans le cadre du projet.
https ://www.kaggle.com/mlg-ulb/creditcardfraud

4.2 Recherche des hyperparamètres optimaux

Après avoir réalisé des expérimentations sur *Synthetic*, *Creditcard* et sur des sous-jeux de données de *KDD10* pour vérifier la bonne implémentation de nos algorithmes, nous avons travaillé principalement avec le jeu de données *Creditcard* pour réaliser la recherche des hyperparamètres optimaux.

La méthode utilisée pour cette recherche des hyperparamètres optimaux est la méthode *Grid Search*. Cette méthode est très simple, puisqu'elle consiste tout simplement à tester les algorithmes pour différents hyperparamètres, de façon exhaustive, et de sélectionner les paramètres qui donnent les meilleurs résultats selon une fonction de score donnée.

Ce score est mesuré selon la méthode de *validation croisée*. Cette méthode consiste à couper le jeu de données en N ensembles. On définit alors $N-1$ ensembles comme ensembles d'entraînement, et le N-ième ensemble comme ensemble de test, puis on mesure le score selon ce découpage de données. On réalise ce processus N fois en changeant à chaque fois l'ensemble de test, puis on calcule le score associé à ce paramétrage de l'algorithme de détection d'aberrations comme la moyenne des N scores.

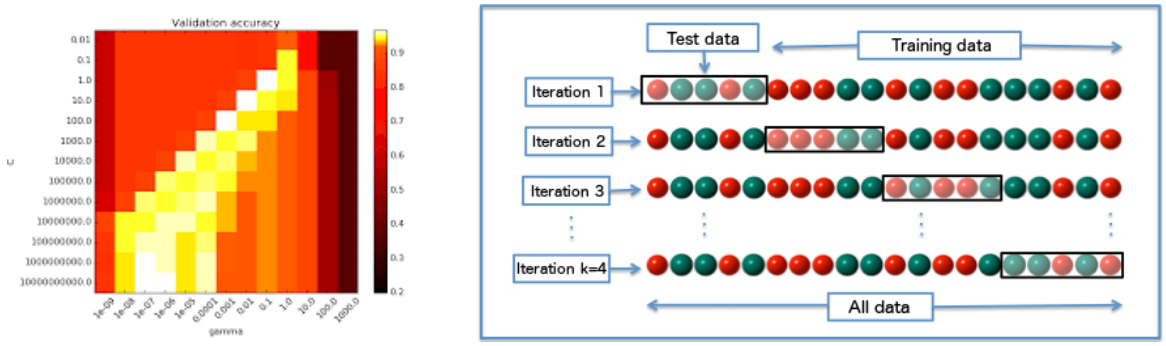


FIGURE 12 – Illustration des méthodes de *Grid Search* (à gauche) et de *validation croisée* (à droite).

Dans le cadre du projet, le score correspond au gain résultant de la détection de fraudes, comme explicité en Section 2. Ainsi, nous faisons varier les hyperparamètres de nos 5 algorithmes par la méthode de *Grid Search*, puis pour chaque ensemble de paramètres nous procédons au calcul du bénéfice, fonction du nombre de fraudes détectées ou non, par validation croisée sur le jeu de données. Les hyperparamètres optimaux sont alors définis comme ceux qui donnent le plus grand bénéfice.

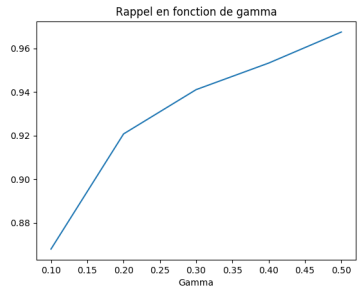
Les résultats de ces processus d'optimisation seront décrits en *Section (4.3)*.

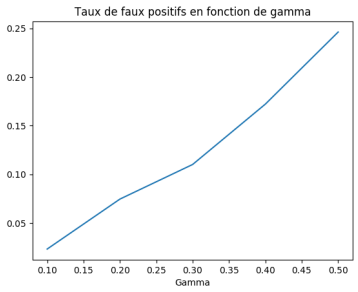
4.3 Description et analyse des résultats

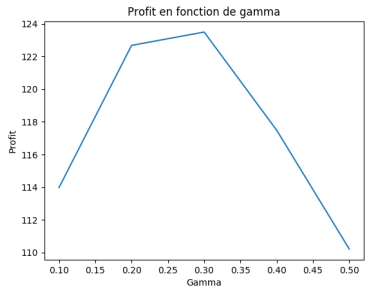
One Class SVM : Cet algorithme nécessite avant tout de choisir un noyau. Après quelques tests, il s'est rapidemment avéré que le noyau gaussien était le noyau qui fournissait de loin les meilleurs résultats. Nous avons donc utilisé ce noyau pour calibrer les paramètres. *One-class SVM* avec un noyau gaussien nécessite la calibration d'essentiellement deux paramètres :

- ν : une borne supérieure de la fraction d'*outliers* dans les données et une borne inférieure de la fraction de vecteurs supports.
- γ : le paramètre du noyau gaussien

Nous avons choisi ν comme étant la fraction d'*outliers* dans le jeu de données *Creditcard*. En effet, la méthode de *Grid Search* nous a permis de nous rendre compte que cette valeur fournissait les meilleurs résultats à la fois en terme de profit π mais aussi en terme de temps de calcul. Il ne restait donc plus qu'à optimiser le paramètre γ par Grid Search. Nous obtenons les résultats suivants :

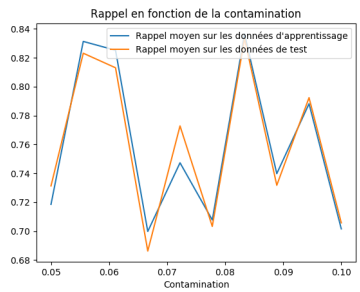


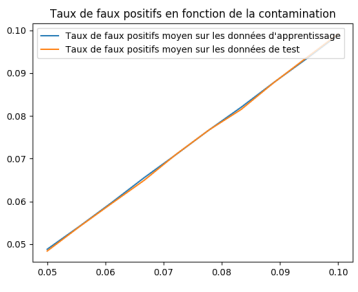


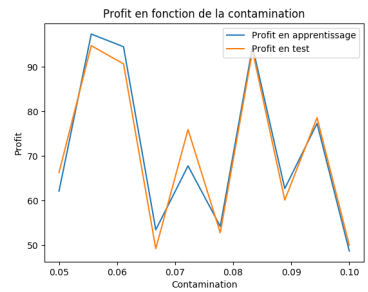


On notera qu'il n'y a pas une courbe d'apprentissage et une courbe de test. En effet, nous avons entraîné notre modèle sur l'ensembles des données normales de *Creditcard* et nous l'avons testé sur le jeu de données tout entier (données normales et données aberrantes) comme cela nous a été conseillé. Compte tenu des très bons résultats obtenus, on peut néanmoins s'interroger sur cette démarche.

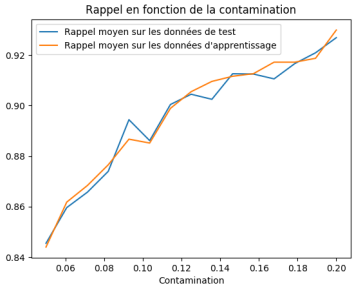
Robust Estimator of Covariance : Pour cet algorithme, nous avons uniquement cherché à optimiser sur la paramètre de contamination qui correspond à la proportion d'*outliers* dans les données. Nous obtenons les résultats suivants :

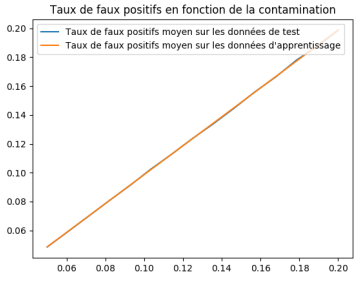


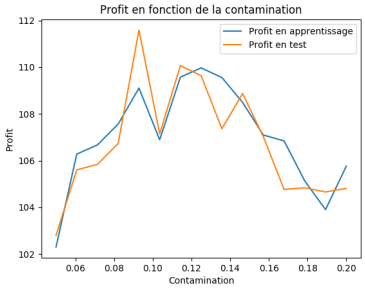




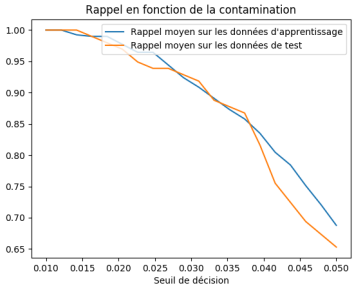
Isolation Forest : Comme pour la méthode précédente, nous avons uniquement optimisé l'*Isolation Forest* sur le paramètre de contamination. Il est en effet apparu qu'il était peu pertinent de chercher à optimiser sur le nombre d'arbres de la forêt ou sur la taille des échantillons utilisés pour la construction des arbres. Nous obtenons les résultats suivants :

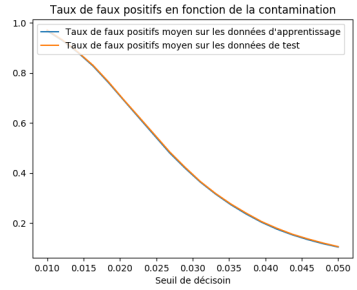


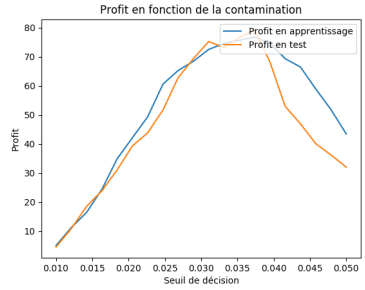




Autoencoder : Nous avons construit un réseau de neurones avec une couche pour l'encodage et une couche pour le décodage. La fonction d'activation pour l'encodage est *ReLu* et la fonction d'activation pour le décodage est la fonction sigmoïde. La dimension de la fonction encodée est plus petite que celle des données d'entrée. Nous n'avons pas eu le temps d'optimiser sur la dimension d'encodage car *Autoencoder* a une longue phase d'apprentissage et nous avons donc pris une dimension égale à 80% de celle des données d'entrée. Nous avons donc uniquement optimisé le seuil de décision, *i.e.* le seuil à partir duquel on considère l'erreur $\|x - \phi \circ \psi(x)\|_{\mathbb{R}^d}$ comme suffisamment élevée pour étiqueter le point comme un *outlier*.







Local Outlier Factor : La méthode *Local Outlier Factor* ayant un temps d'exécution bien plus long que les autres méthodes, nous n'avons pas réussi à réaliser les tracés analogues aux tracés présentés ci-dessus dans le temps imparti au projet.

Analyse globale des résultats : On obtient globalement de très bons résultats avec les 4 algorithmes optimisés. En effet, on arrive à des taux de vrai positifs de plus de 80 % (voire plus de 90% pour *Isolation Forest* et *One-class SVM*) tout en gardant un taux de faux positifs raisonnable (de l'ordre de 10 %) sauf pour *Autoencoder* qui fournit des résultats un peu en dessous des autres. L'algorithme qui fournit les meilleurs résultats est *One-class SVM*.

4.4 Combinaison des algorithmes de détection d'aberrations

Une des techniques les plus utilisées dans les compétitions de *Machine Learning* pour améliorer les performances de classifieurs sur un problème donné est de combiner plusieurs classifieurs. Afin de chercher à améliorer encore nos résultats, nous avons essayé d'appliquer cette approche sur les algorithmes que nous avons étudié. En particulier, nous avons testé deux méthodes.

La première méthode consiste à faire du *rank averaging*. On commence par entraîner les quatre classifieurs sur les données d'apprentissage. Puis, on calcule les scores donnés par chaque classifieur sur les données de test. On fait en sorte, pour chacun des classifieurs, que plus le score d'un point est élevé, plus il est susceptible d'être un *outlier* (en multipliant par exemple les scores par -1). Sur la base de ces scores, on attribue ensuite à chaque point un rang pour chaque classifieur. Le point avec le score le moins élevé a le rang 0 et le point avec le score le plus élevé a le rang $n_{\text{test}} - 1$ où n_{test} désigne le nombre de données de test. On peut ensuite calculer le rang moyen de chacun des points à partir des 4 rangs obtenus. Enfin, on étiquette les points avec le rang moyen le plus haut comme des *outliers*. Le nombre de points est choisi de façon à maximiser la fonction de profit π.

On obtient les résultats suivants :

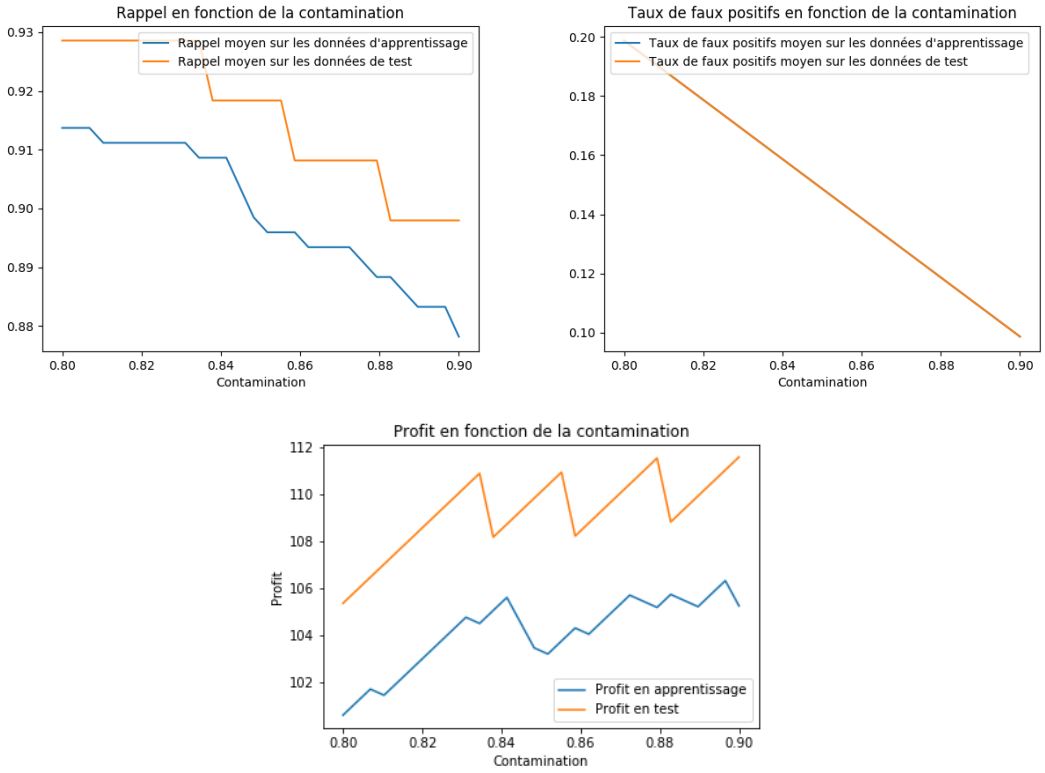


FIGURE 17 – Performances de la combinaison des algorithmes par la méthode *rank averaging*.

Les résultats sont bons (assez étonnamment meilleurs sur les données de test que sur les données d'apprentissage) mais ne permettent pas une amélioration par rapport à *One-class SVM*.

La seconde méthode consiste à faire une régression logistique sur les scores donnés par les différents classifieurs. On commence donc par entraîner les quatre classifieurs sur les données d'apprentissage et on calcule tous les scores (à la fois sur les données d'apprentissage et sur les données de test). On fait ensuite une régression logistique sur la matrice des scores d'apprentissage. On calcule ensuite les probabilités donnés par le prédicteur sur les données de test. Finalement, on cherche le seuil à partir duquel on étiquette une donnée comme aberrante de façon à maximiser le profit π.

On obtient les résultats suivants :

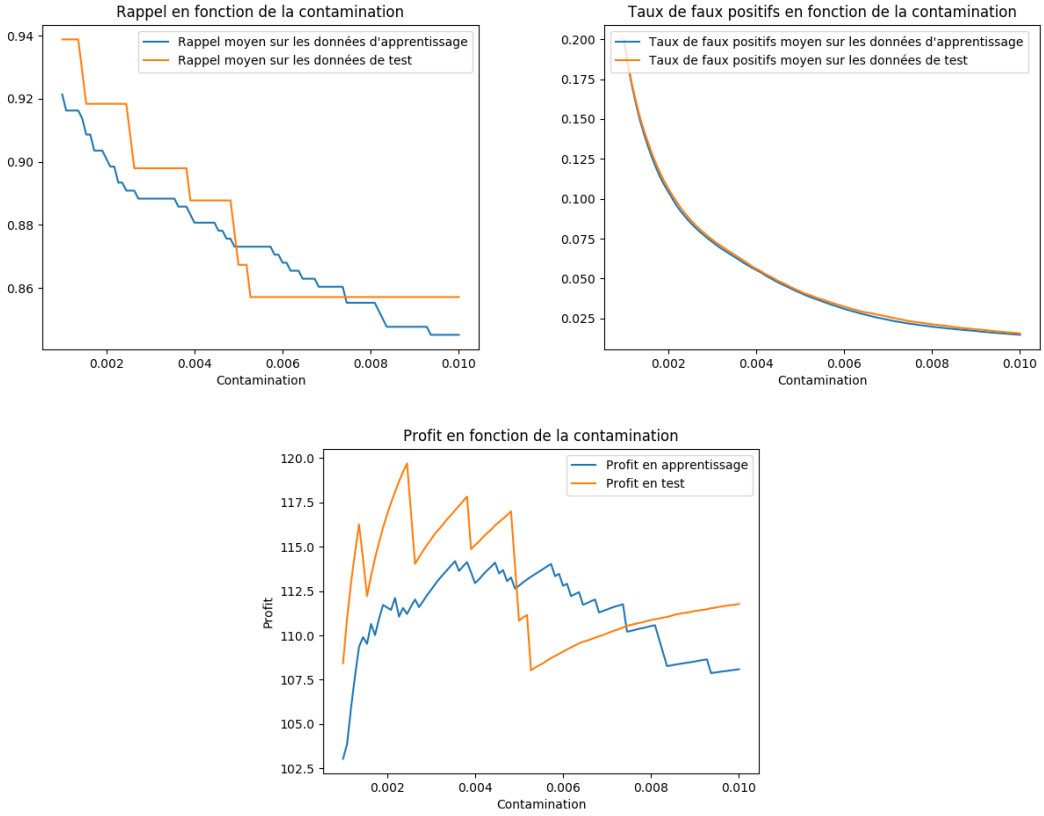


FIGURE 18 – Performances de la combinaison des algorithmes par régression logistique sur les scores.

Les résultats sont meilleurs que pour la méthode précédente mais restent en dessous de *One-class SVM*.

5 Conclusion

5.1 Synthèse des résultats

Après avoir testé les 5 algorithmes sur différents jeux de données et avec différents hyperparamètres, nous avons pu constater par nous-même les différents comportements des algorithmes. Notamment, il s'avère que *Local Outlier Factor* avait une durée d'apprentissage très longue sur les jeux de données de grande taille, et notamment sur *Creditcard*.

Nous nous sommes ainsi concentrés sur l'optimisation des 4 algorithmes restants, à savoir *One-class SVM, Robust Estimator of Covariance, Isolation Forest et Autoencoder*. Parmi ces 4 algorithmes, celui qui semble donner les meilleurs résultats à l'issue de notre projet est *One-class SVM*.

Bien sûr, la modélisation de la fonction de coût est primordiale, et les résultats obtenus sont à relativiser à la lumière de la fonction de coût utilisée, qui donne une grande importance aux taux de vrais positifs et de faux négatifs. En effet, les 4 derniers algorithmes cités précédemment ont tous les 4 des résultats satisfaisants, et leur efficacité par rapport aux autres dépend de la quantité que l'on cherche à optimiser.

5.2 Limites du projet et perspectives d'amélioration

La principale limite du projet réside dans le temps qu'il a été possible de consacrer à l'optimisation des algorithmes. En effet, les jeux de données traités dans le cadre du projet ont été de petits jeux de données, qui pouvaient être traités en temps raisonnable par les algorithmes utilisés (quelques minutes par exécution). Notamment, le jeu de données *KDD10*, qui représentait 10% du jeu de données original de la *KDD Cup 1999*, a rapidement été mis de côté car le temps d'une seule exécution d'un algorithme sur ce jeu de données était déjà bien trop long.

Par conséquent, les algorithmes de détection d'aberrations ayant été implémentés dans le cadre de ce projet, ainsi que les techniques d'optimisation de leurs hyperparamètres, une suite à donner à ce projet serait d'exécuter les techniques d'optimisation sur de plus gros jeux de données et sur des ordinateurs plus puissants, de façon à pouvoir avoir de meilleurs résultats et une détection plus efficace.

Une autre limite rencontrée dans le cadre de ce projet est que nous avons travaillé sur des données anonymisées - il était impossible de travailler sur de vraies données bancaires de la Société Générale dans le temps imparti, pour des raisons administratives. Ainsi, si l'on travaille sur des données dont on connaît la signification, il est plus facile d'améliorer la détection d'aberrations, en utilisant d'autres techniques qui permettraient de diminuer la taille du jeu de données à traiter, et par conséquent d'améliorer les résultats puisque les algorithmes d'optimisation prendraient beaucoup moins de temps.

Références

[1] Ng Breunig, Kriegel and Sander. Lof : identifying density-based local outliers. Proc. ACM SIGMOD, 2000.

[2] Observatoire de la sécurité des moyens de paiement. Rapport annuel de l'observatoire de la sécurité des moyens de paiement. `https://www.banque-france.fr/sites/default/files/medias/documents/osmp2016_web.pdf`, 2016.

[3] Deeplearningbook. Autoencoders. `http://www.deeplearningbook.org/contents/autoencoders.html`, 2001.

[4] Schölkopf Bernhard et al. Estimating the support of a high-dimensional distribution. Neural computation 13.7, 2001.

[5] Fei Tony Liu Kai Ming Ting et Zhi-Hua Zhou. Isolation-based anomaly detection. `https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/tkdd11.pdf`, 2012.

[6] Van Driessen K. Rousseeuw, P.J. A fast algorithm for the minimum covariance determinant estimator. Technometrics 41(3), 212, 1999.