

PROJET INDUSTRIEL

Détection des outliers atypiques pour la détection des fraudes



Auteur :

AKRIM Anass, Étudiant Ingénieur ICM 2016,
Mines Saint-Etienne

Encadrants :

JUGANARU-MATHIEU Mihaela,
Enseignante-chercheuse, Mines Saint-Etienne
CHOPINEAU Antoine, Senior Manager,
Mazars
MIGUIL Amr, Data Scientist, Mazars

Saint-Étienne, France, Janvier 2019

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 1.1 | Contexte | 3 |
| 1.1.1 | Mazars | 3 |
| 1.1.2 | Fraudes | 3 |
| 1.2 | Problématique | 3 |
| 1.2.1 | Définition d'une anomalie | 4 |
| 1.2.2 | Machine Learning dans la détection d'outliers | 4 |
| 1.3 | Objectifs | 6 |
| 2 | Théorie | 7 |
| 2.1 | Machine Learning | 7 |
| 2.1.1 | K-Means | 7 |
| 2.1.2 | DBSCAN (density-based spatial clustering of applications with noise) | 8 |
| 2.1.3 | Isolation Forest | 10 |
| 2.1.4 | OCSVM (One-Class Support Vector Machine) | 11 |
| 2.2 | Deep Learning | 12 |
| 2.3 | Oversampling ou Data Augmentation | 13 |
| 3 | Expérience | 15 |
| 3.1 | Data Cleaning | 15 |
| 3.1.1 | Données Manquantes | 15 |
| 3.1.2 | Numérisation des données | 16 |
| 3.1.3 | Centrage et réduction des données | 16 |
| 3.2 | Réduction de dimension | 17 |
| 3.3 | Analyse de données | 18 |
| 3.4 | Choix des paramètres, amélioration et implémentation | 20 |
| 3.4.1 | Ensemble Learning ou méthodes ensemblistes | 20 |
| 3.4.2 | Paramétrage et optimisation du DBSCAN | 20 |
| 3.4.3 | Paramétrage et optimisation de l'Isolation Forest | 21 |
| 3.4.4 | Paramétrage et optimisation du OCSVM | 22 |
| 3.4.5 | Paramètres de K-means | 22 |
| 3.5 | Mesure et Évaluation de performance | 23 |
| 3.5.1 | Évaluation de qualité de méthodes supervisées | 24 |
| 3.5.2 | Évaluation de qualité de méthodes non supervisées | 24 |
| 3.6 | Détection d'outliers : méthode proposée | 25 |
| 3.6.1 | Méthode de clustering | 25 |
| 3.6.2 | Méthode de clustering avec un score en sortie : du non-supervisé au supervisé | 26 |
| 4 | Résultats | 28 |
| 4.1 | Test sur un autre jeu de données | 30 |

5 Conclusion

33

1 Introduction

1.1 Contexte

1.1.1 Mazars

Mazars est une entreprise internationale d'origine française, créée en 1940 à Rouen, organisation spécialisée dans l'audit, l'expertise comptable, la fiscalité et le conseil aux entreprises.

L'organisation indépendante Mazars accompagne les entreprises, en qualité d'expert financier, en proposant des outils informatiques d'aide à la décision pour gérer et détecter les anomalies (fraudes plus précisément).



1.1.2 Fraudes

"La fraude est le fait d'agir en utilisant des **moyens déloyaux** afin d'obtenir un avantage indu, un consentement ou dans le but de **contourner les obligations légales ou réglementaires**. Elle peut être accompagnée, ou non, d'un **enrichissement personnel** de son auteur, via des prélèvements de trésorerie et autres détournements d'actifs."

(cf. www.mazars.fr/Accueil/Nos-métiers/Financial-Advisory-Services/Litiges-et-fraudes/Glossaire-Litiges-et-Fraudes/)

Selon la "Global Economic Crime Survey 2018" de PwC, "près des trois quarts des entreprises françaises indiquent avoir été victimes de fraude au cours des deux dernières années", et environ 12% des entreprises françaises interrogées ont indiqué avoir été victimes de fraude avec une perte de plus d'1 million de dollars.

La lutte contre la fraude est devenue un enjeu majeur avec le développement des organisations. En effet, les entreprises évoluent aujourd'hui dans un environnement incertain et peuvent être confrontées à des problématiques différentes :

-Lutte contre les fraudes en assurance

-Lutte contre le détournement d'actifs et blanchiment d'argent, etc...

Par conséquent, il en devient nécessaire de se munir des outils pour faire face à cette menace et trouver une solution pour détecter, gérer les fraudes.

1.2 Problématique

La détection des fraudes se résume souvent à la détection des valeurs aberrantes, dans laquelle un ensemble de données récoltées est scanné pour trouver des anomalies potentielles dans les données. Auparavant, cela était fait par des employés qui vérifiaient toutes les transactions manuellement. Avec l'essor de l'apprentissage automatique, de l'intelligence artificielle, de

l'apprentissage "profond" et d'autres domaines pertinents de la technologie de l'information, il devient possible d'automatiser ce processus et d'économiser une partie du travail intensif consacré à la détection de la fraude, en faisant appel à des techniques spécifiques de Machine Learning et Deep Learning.

Dans ce projet industriel, nous cherchons à mettre en place une méthode robuste capable d'identifier une fraude financière, ou anomalie dans un jeu de données récoltées.

Toutefois, avant d'entamer l'étude, il est nécessaire de se poser quelques questions :

- Une fraude financière est définie comme étant une opération anormale, atypique : une anomalie. Cependant, qu'est-ce qu'une anomalie ?
- Quel serait le rôle de l'apprentissage automatique ou Machine Learning dans la détection de fraudes ?

1.2.1 Définition d'une anomalie

Une anomalie est définie comme étant un "cas isolé", ou déviant. C'est quelque chose présentant des caractéristiques s'éloignant de la norme.

En statistique, un outlier est une observation qui dévie tellement des autres observations qu'elle tend à susciter la suspicion. En d'autres termes, une anomalie est une observation qui diffère d'un schéma global d'un échantillon.

Une observation est qualifiée d'anomalie lorsqu'elle dispose de l'une des caractéristiques suivantes :

- Propriétés statistiques différentes du reste.
- Se trouve dans une région à faible densité.
- Dans une modélisation, celle-ci se trouve à l'écart des autres points.

Nous pouvons trouver des outliers dans deux types de jeu de données :

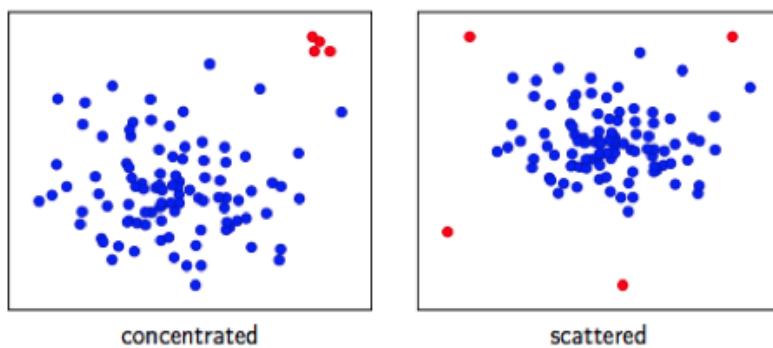


Figure 1: Anomalies plus éloignées (concentrated), anomalies dans des régions à faible densité (scattered).

1.2.2 Machine Learning dans la détection d'outliers

Les techniques d'apprentissage automatique ou Machine Learning font désormais l'objet d'une attention considérable de la part des chercheurs en détection d'anomalies pour remédier aux faiblesses des techniques de détection de fraudes utilisées jusqu'ici. La détection d'anomalies

peut aider efficacement à détecter la fraude, découvrant une activité étrange dans de grands ensembles de données massives complexes.

La détection des valeurs aberrantes ou transactions anormales est le problème de détection dans une base de données, qui sont très différents de ce à quoi ressemble une transaction 'classique' dans l'ensemble de données dont un établissement financier dispose.

Cependant, nous rencontrons souvent 3 difficultés non négligeables dans la détection de fraudes :

- **Jeu de données de taille très importante**, dû au nombre important de transactions par seconde (plus de 200.000 opérations à traiter dans notre cas par exemple), ce qui s'avère complexe si l'on ne dispose pas d'un CPU assez puissant pour traiter nos données.

Cette étude a été effectuée sur Python, Jupyter Lab (cf. www.github.com/jupyterlab/jupyterlab). Il serait peut être plus intéressant d'allier Apache Spark à Python, en réduisant la dimension du data set dont on dispose (source : <http://cedric.cnam.fr/vertigo/Cours/RCP216/tpSparkScala.html>).

- **Les fraudes sont considérées d'événements relativement rares et différentes** (heureusement) et donc peu représentés dans les données d'apprentissage. Nous utilisons donc pour cela une approche dite non-supervisée. Cette fois, il s'agit de repérer, des points anormaux, soit des observations qui s'écartent trop de la moyenne : on parle de détection d'outliers.

- Les données que nous traitons sont non étiquetées : ceci rend difficile de valider la performance de chaque modèle de machine learning utilisé pour la détection de fraudes. Il serait judicieux de mettre place une métrique d'évaluation de la qualité de la méthode non supervisée de partitionnement des données que l'on utilise.

En effet en machine learning, nous avons deux catégories d'algorithmes :

- Supervisés : l'apprentissage est dit supervisé lorsque les données qui entrent sont déjà catégorisées et que les algorithmes doivent s'en servir pour prédire un résultat, en vue de pouvoir le faire plus tard lorsque les données ne seront plus catégorisées. Les algorithmes de cette catégorie sont appelés algorithmes de classification.

- Non supervisé : l'apprentissage est dit non supervisé lorsque les données ne sont pas catégorisées, ce qui rend l'étude beaucoup plus complexe vu que le système va devoir détecter les similarités (caractéristiques en commun par exemple) dans les données qu'il traite et les organiser en fonction de ces dernières. Les algorithmes de cette catégorie sont appelés algorithmes de clustering, ou partitionnement.

En théorie, les méthodes d'apprentissage supervisé nous offrent de meilleures prédictions que les méthodes d'apprentissage non supervisé, vu le manque de données étiquetées (à défaut de pouvoir valider le modèle de prédiction).

Ici dans notre étude de détection de fraudes, nous allons utiliser une approche dite non-supervisée, dans laquelle il s'agit de repérer des outliers.

1.3 Objectifs

Le travail réalisé concrètement et les objectifs fixés sont les suivants :

- Benchmark des algorithmes de Machine Learning existants pour la détection de fraudes
- Établir une méthodologie robuste, tout en calibrant et optimisant les modèles prédictifs utilisés.

En effet, l'objectif de ce projet est de mettre en place une méthode robuste et efficace capable de détecter des fraudes financières, en appliquant des algorithmes de machine learning.

2 Théorie

Notre étude se base sur un jeu de données non étiqueté, une étude non supervisée. Celle-ci sera fondée principalement sur des algorithmes de clustering de Machine Learning, que l'on pourra combiner avec un algorithme de Deep Learning : les réseaux de neurones artificiels.

2.1 Machine Learning

Les techniques de clustering de Machine Learning ne requièrent pas des jeu de données d'entraînement. Elles reposent sur deux hypothèses de base :

- Premièrement, elles présument que la plupart des transactions ne sont pas frauduleuses et que seul un faible pourcentage est anormal.
- Deuxièmement, elles prévoient que le set des transactions frauduleuses est statistiquement différent du set de transactions non frauduleuses.

Selon ces deux hypothèses, les groupes de données de cas semblables qui apparaissent fréquemment sont considérés comme du 'non fraude' et les groupes de données qui sont peu fréquents sont considérés comme 'fraude' ou 'anomalies'.

L'idée est qu'un algorithme de détection d'anomalies non supervisé évalue les données uniquement en fonction des propriétés intrinsèques de l'ensemble de données. Généralement, les distances ou les densités sont utilisées pour estimer ce qui est normal et ce qui est aberrant.

Notre méthode reposera sur trois catégories d'algorithmes de Machine Learning en apprentissage non supervisé :

- Clustering par densité (DBSCAN, OCSVM);
- Clustering par distance (K-means);
- Clustering par 'isolation' (Isolation Forest).

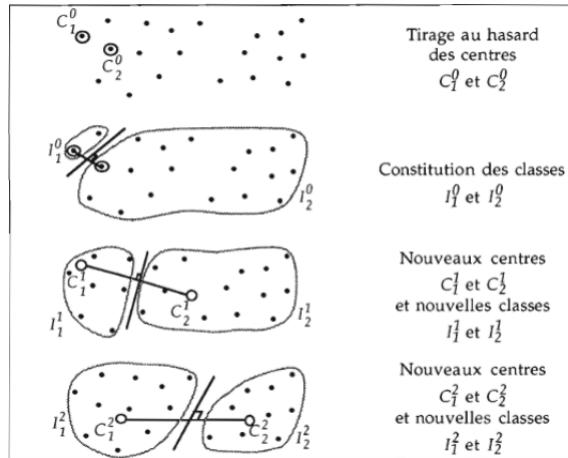
2.1.1 K-Means

K-means est une méthode d'apprentissage non supervisé de partitionnement de données et un problème d'optimisation combinatoire.

Étant donnés des points et un entier k , le problème est de diviser les points en k groupes, souvent appelés clusters. L'algorithme de Clustering K-means se base sur la mesure de la distance (ici distance euclidienne). Une particularité de l'algorithme K-means est que le nombre de clusters est fixé au préalable à son initialisation.

Dans la figure ci-dessous, nous avons fixé le nombre de clusters souhaité $K=2$. Les centres (centroïdes) sont tirés au hasard initialement, et les nouveaux centres obtenus sont les barycentres du cluster correspondant.

Les étapes de l'algorithme sont répétées jusqu'à convergence (en fixant un seuil de convergence ou bien jusqu'à ce que les barycentres soient 'stables').



Lebart et al., 1995 ; page 149.

Figure 2: Regroupement possible à l'aide de l'algorithme K-Means.

Il permet de regrouper en K clusters distincts(K déterminé à l'avance) les observations du data set. Ainsi les données similaires se retrouveront dans un même cluster.

Les avantages et inconvénients de l'algorithme K-Means sont les suivants :

| Avantages | Inconvénients |
|--|---|
| Simple, rapide (complexité en temps assez faible) et efficace | Le résultat dépend du tirage initial (aléatoire) des centres des classes. |
| Applicable à des données de grande taille et à tout type de données (même textuelle) | Le nombre de classes doit être fixé au départ, ce qui le rend très peu compatible avec la détection d'outliers. |

Figure 3: Avantages et inconvénients de l'algorithme K-Means.

2.1.2 DBSCAN (density-based spatial clustering of applications with noise)

DBSCAN (density-based spatial clustering of applications with noise) est un algorithme de clustering relativement récent (Ester et al. 1996), basé sur du clustering par densité, qui est capable de découvrir des clusters de n'importe quelle taille ou de n'importe quelle forme et identifier les valeurs aberrantes avec précision.

Contrairement aux autres méthodes de clustering (K-means notamment) qui assignent chaque point à un cluster, le DBSCAN est capable de reconnaître les outliers et de ne pas les regrouper.

L'algorithme DBSCAN requiert 2 paramètres :

- **Epsilon** : distance maximale entre deux observations (la distance utilisée est la distance euclidienne dans cette étude) pour qu'elles soient considérés comme faisant partie d'un groupe (ou cluster);
- **MinPts** : nombre minimal d'observations pour pouvoir former un cluster.

En effet, au lieu d'exiger un nombre de clusters (comme K-means), DBSCAN est éventuellement en mesure de calculer automatiquement n'importe quel nombre de clusters, basé sur des paramètres donnés.

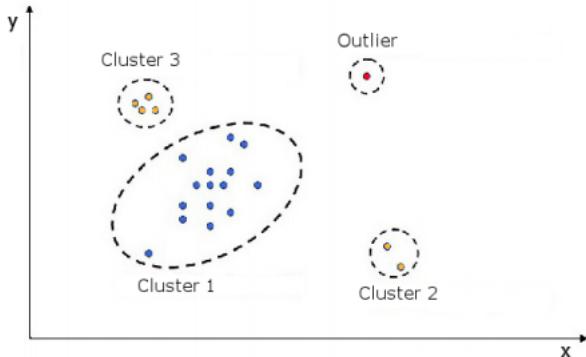


Figure 4: Regroupement possible à l'aide de l'algorithme DBSCAN.

Nous pouvons voir dans la Figure ci-dessus que :

- Selon la valeur du paramètre MinPts, Cluster 2 et Cluster 3 peuvent être divisés et considérés comme anomalies.
- Selon la valeur du paramètre Epsilon, Cluster 1 pourrait être divisé en plusieurs clusters (et outliers) si nous utilisons un epsilon plus petit; ou pourrait être fusionné avec Cluster 3 si nous utilisons un epsilon plus grand.

Cependant, ces paramètres ne sont pas connus d'avance, et ils doivent être choisis avec précaution. Ceci nécessite donc une étude rigoureuse du jeu de données dont ont dispose. Ceci correspond à l'une des difficultés de l'algorithme DBSCAN : simple d'usage, difficile à paramétrier de manière exacte. Nous verrons plus tard une approche mise en place capable de faciliter le choix des paramètres.

DBSCAN est donc un algorithme efficace et simple d'usage, si l'on choisit bien les paramètres. Toutefois celui-ci présente quelques inconvénients et avantages :

| Avantages | Inconvénients |
|---|---|
| Simple et efficace | Choix des paramètres avec précaution |
| Ne nécessite pas qu'on lui précise le nombre de clusters à trouver | Incapable de trouver des clusters lorsque les données sont réparties de manière uniforme dans le plan (toutefois ce n'est pas le cas dans la détection d'anomalies) |
| Capable de gérer les outliers en les éliminant du processus de partitionnement (en fonction des paramètres) | Ordre de complexité en temps élevé ($O(n^2)$) |

Figure 5: Avantages et Inconvénients de DBSCAN

2.1.3 Isolation Forest

Introduit en 2008 par Liu et al., l'algorithme 'Isolation Forest' [1] est basé sur le même principe que celui des 'Random Forests', mais repose sur la constructions d'un ensemble d'arbres complètement aléatoires : isolation tree. Cet algorithme n'utilise ni mesure de distance ni mesure de densité afin de détecter des anomalies : ce qui le rend moins coûteux en terme de calculs.

Chaque arbre est construit de manière aléatoire (tirage de noeud, seuil, répartition aléatoires) jusqu'à l'obtention d'une observation par feuille. Cette approche est qualifiée de partitionnement par 'isolation'.

Nous accordons ensuite un score à chaque observation, chaque arbre, selon une fonction de score préétablie. Au final, nous obtenons un score pour chaque observation après avoir fait la moyenne des scores obtenus. La fonction de score établie par ses auteurs prend en compte la profondeur du noeud : plus la profondeur est faible, plus le score est élevé et donc plus l'observation est considérée comme étant une anomalie.

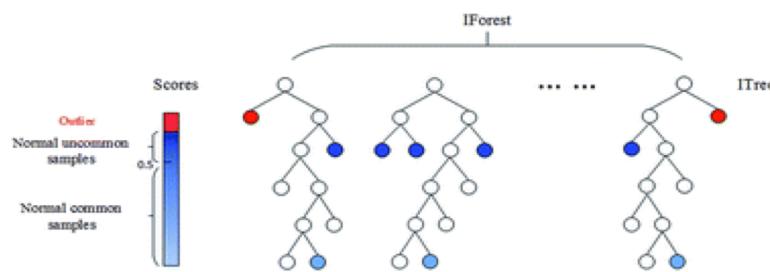


Figure 6: Exemple de partitionnement de l'algorithme Isolation Forest.

L'algorithme requiert deux paramètres :

- "**Sous-échantillonnage**" : Le nombre d'observations à prélever à partir du data set pour former chaque estimateur de base.
- "**Nombre d'arbres**" : Le nombre d'arbres, ou d'estimateurs de base de l'ensemble.

Les études de son auteur ont montré que la précision de détection d'isolation Forest converge rapidement avec un très petit nombre d'arbres (100 serait suffisant); et il suffirait d'un petit sous-échantillonnage pour obtenir une haute précision de détection avec une grande efficacité. Selon lui, empiriquement, le réglage de ce paramètre à 2^8 ou 256 fournit généralement suffisamment de détails pour effectuer la détection d'anomalies dans un large éventail de données. Son étude a montré que la performance de détection du modèle de prédiction serait quasi-optimale à ces valeurs de paramètres, et serait quasi-invariante pour des valeurs de paramètres plus grandes. Les avantages et inconvénients de l'algorithme sont les suivants :

| Avantages | Inconvénients |
|---|--|
| Détecte des anomalies seulement basé sur le concept de « l'isolation », sans employer une mesure de distance ou de densité. | Encore récent, limites de l'algorithme ne sont pas encore identifiées. |
| Adapté pour des data set à grande dimension | Côté aléatoire. |
| Peu de paramètres à choisir : le rend robuste et simple à optimiser | Coûteux en temps si l'algorithme n'est pas correctement optimisé. |
| Inutile de mettre les variables dans la même échelle (aucune mesure de distance/densité) | |

Figure 7: Avantages et inconvénients de l'Isolation Forest.

2.1.4 OCSVM (One-Class Support Vector Machine)

Le OCSVM est une extension de l'algorithme d'origine SVM.

C'est une approche qui cherche à mettre en place une «enveloppe», ou un support des observations jugées « normales », définie par des séparateurs à vaste marge (One Class Classification SVM) (Schölkopf et al. 1999).

Le principe consiste à poser un problème d'optimisation avec pour objectif de séparer les données, en maximisant la marge en fonction des paramètres de notre problème.

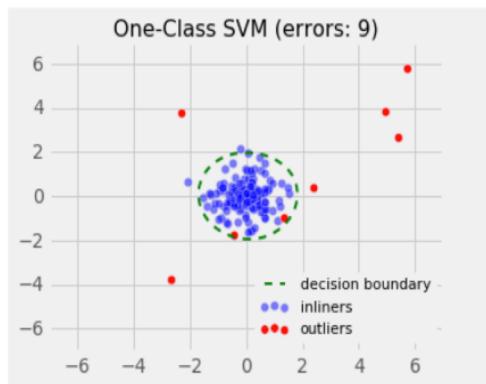


Figure 8: Exemple de partitionnement de l'algorithme OCSVM.

La solution produit une fonction binaire qui vaut +1 dans la plus petite région captant les données, soit les inliers et 1 ailleurs, les outliers.

Le One-Class SVM requiert deux paramètres :

- ν : Soit la limite supérieur du nombre d'outliers attendus.
- γ : en théorie, le coefficient du noyau de la fonction noyau de l'algorithme, qui détermine la frontière de séparation entre les deux classes et donc le nombre d'outliers.

Comme pour le Support Vector Machine classique, cet algorithme est basé sur l'utilisation d'un noyau (linéaire, polynomial, sigmoïde ou RBF) et utilise l'astuce du kernel pour calculer le produit scalaire entre les points de données représentés dans un espace haute dimension pour trouver un hyperplan séparateur.

Les avantages et inconvénients de l'OCSVM sont les suivants :

| Avantages | Inconvénients |
|--|--|
| Permet de séparer les observations : adapté pour la détection d'outliers | Paramètres d'optimisation à choisir avec précaution |
| Basé sur la mesure de densité : utile en cas de répartition 'scattered' | Ne donne pas de score : seulement une classification |
| | Difficile de bien calibrer le modèle en cas de data set de grande dimension : impossible de visualiser les résultats |

Figure 9: Avantages et inconvénients de l'algorithme OCSVM.

2.2 Deep Learning

Ces dernières années, les réseaux neuronaux artificiels profonds, branche du Machine Learning, ont remporté de nombreux concours de reconnaissance de modèles et d'apprentissage automatique.

L'apprentissage profond (aussi appelé apprentissage profond structuré ou apprentissage hiérarchique) fait partie d'une famille plus vaste de méthodes de Machine Learning fondées sur des représentations de données d'apprentissage, par opposition à des algorithmes spécifiques à une tâche. Apprentissage profond ou Deep Learning est un ensemble puissant de techniques d'apprentissage dans les réseaux neuronaux. L'apprentissage profond est l'application de réseaux neuronaux artificiels à des tâches d'apprentissage utilisant des réseaux de couches multiples.

Les réseaux neuronaux, méthodes d'apprentissage supervisé et développées dans les années 1950 peu après l'aube de la recherche sur l'IA, semblaient prometteuses parce qu'elles tentaient de simuler la façon dont le cerveau fonctionnait, bien que sous une forme très simplifiée. Un programme cartographie un ensemble de neurones virtuels et attribue ensuite des valeurs numériques aléatoires, ou « poids », aux connexions entre elles. Ces poids déterminent la façon dont chaque neurone simulé réagit.

Les neurones neuronaux sont généralement organisés en couches. Les couches sont constituées d'un certain nombre de 'nœuds' inter-connectés qui contiennent une 'fonction d'activation'.

Les modèles sont présentés au réseau via la 'couche d'entrée', qui communique avec une ou plusieurs 'couches cachées' où le traitement réel se fait via un système de 'connexions' pondérées. Les couches cachées se connectent ensuite à une 'couche de sortie' où la réponse est l'output, comme indiqué dans le graphique ci-dessous.

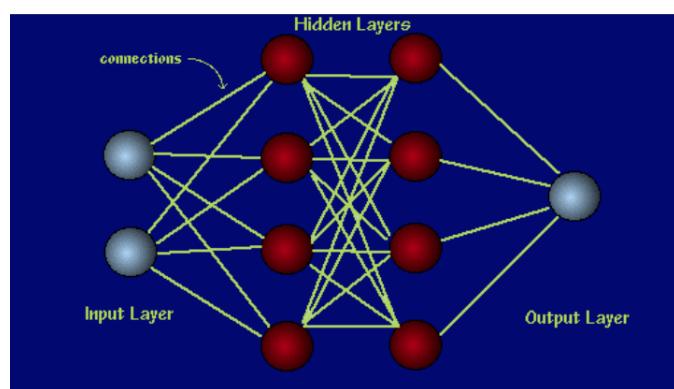
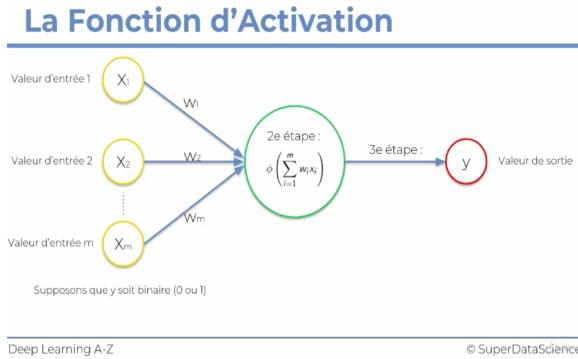


Figure 10: Réseau de neurones artificiels.

La couche cachée est composée de nombreuses cellules, ou neurones, avec un poids associé à chacune. Premièrement, les poids du réseau sont initialisés aléatoirement (avec une valeur proche de 0), puis ils sont ajustés pendant la phase d'entraînement.

À la fin de la phase de formation, selon la fonction d'activation utilisée, nous obtenons un réseau neuronal qui peut prédire avec une bonne approximation la valeur de sortie selon les

valeurs données en entrées.



Deep Learning A-Z © SuperDataScience

Figure 11: Réseau de neurones artificiels avec la fonction d'activation *sigmoïde*.

Ainsi dans notre étude de détection de fraudes, nous pouvons avoir recours aux réseaux de neurones, capables d'apprendre les données entrantes et nous afficher en sortie un score de 'fraude' par la suite pour chaque opération, en choisissant la fonction d'activation *sigmoïde* (ce qui affiche en sortie une valeur entre 0 et 1 selon les variables d'entrée, correspondant à la probabilité que l'observation appartienne à la classe '1', soit la classe 'fraude').

Cette approche pourrait être intéressante pour un client désireux d'avoir une vision globale sur les transactions, et pouvoir attribuer un seuil de score lui-même pour désigner les transactions suspectes.

Cette méthode est bien une méthode supervisée, toutefois nous pouvons la combiner avec une méthode d'apprentissage non supervisée pour l'utiliser : nous pouvons parler de passage du non supervisé au supervisé. Nous verrons par la suite dans la section 3.6.2 en quoi consiste cette méthode proposée.

2.3 Oversampling ou Data Augmentation

Souvent, dans la pratique, des exemples de certaines classes seront sous-représentés dans le jeu de données dont on dispose. C'est le cas, par exemple, lorsque notre classificateur doit faire la distinction entre les transactions de commerce authentiques et frauduleuses : les exemples de transactions authentiques sont beaucoup plus fréquents. Ce manque de données joue un rôle important dans les techniques de Deep Learning : les exemples « frauduleux », qui sont minoritaires, risquent d'être mal classés afin de classer plus d'exemples de la classe majoritaire correctement.

Un moyen bien connu et populaire de traiter les données déséquilibrées est le 'suréchantillonage' ou 'oversampling'. Suréchantillonner signifie créer artificiellement des observations dans notre ensemble de données appartenant à la classe sous représentée dans nos données.

Une technique courante est SMOTE — Synthetic Minority Over-sampling Technique. À un niveau élevé, SMOTE crée des observations synthétiques de la classe minoritaire (dans ce cas, des transactions frauduleuses). SMOTE effectue les étapes suivantes :

- Trouver les voisins les plus proches pour les observations de la classe minoritaire (trouver des observations semblables)
- Choisir au hasard l'un des voisins k-proches et l'utiliser pour créer une nouvelle observation similaire, mais aléatoirement modifiée.

Il existe de nombreuses implémentations SMOTE. Dans notre cas, nous allons tirer parti de la classe SMOTE de la bibliothèque imblearning. La bibliothèque imblearn est une boîte à outils vraiment utile pour traiter les problèmes de données déséquilibrés.

Pour en savoir plus sur la technique SMOTE, vous pouvez consulter cet article.

Dans notre étude, nous allons faire appel à la technique d'oversampling lorsque nous entamerons l'approche supervisée (par réseau de neurones).

3 Expérience

Dans notre étude, nous cherchons à mettre en place une procédure d'amélioration des algorithmes fondamentaux de clustering existants et les comparer avec une méthode que l'on va proposer. Notre expérience est basée sur un jeu de données importé de Kaggle (cf. www.kaggle.com/shubhamjoshi2130of/abstract-data-set-for-credit-card-fraud-detection) avec 3076 observations et 12 variables. Nous utilisons la méthode d'oversampling SMOTE détaillée dans la section 2.3 afin d'étudier un data set plus large (de près de 15000 individus au lieu de 3076, ceci en effectuant un 'oversampling' sur la classe de non fraudeurs).

Notre étude repose sur la procédure suivante :

- Utiliser un ensemble de données étiqueté.
- Retirer les étiquettes.
- Appliquer l'algorithme de clustering en question.
- Évaluer la performance du partitionnement, en le comparant aux données étiquetées.

Avant toute chose, avant de proposer notre méthode, nous allons décrire les étapes préalables de notre étude :

- 1) Data Cleaning
- 2) Réduction de Dimension
- 3) Analyse de données
- 4) Choix des paramètres, amélioration et implémentation des algorithmes
- 5) Définition d'une mesure d'évaluation de performance

Notre étude sera effectuée sur Python.

3.1 Data Cleaning

Avant de pouvoir entamer notre étude, il est important de prendre en compte la procédure de 'data cleaning' suivante, qui consiste en trois étapes :

- 1) Traitement des données manquantes.
- 2) Numériser nos variables
- 3) Centrer et réduire nos données

3.1.1 Données Manquantes

Dans certains cas, les données sont transmises à l'analyste sous la forme d'un ensemble de données avec des caractéristiques déjà définies. Dans certaines bases de données, les valeurs de certaines caractéristiques peuvent être manquantes. Cela arrive souvent lorsque l'ensemble de données a été fabriqué à la main, et la personne qui y travaille a oublié de remplir certaines valeurs ou ne les a pas mesurées du tout.

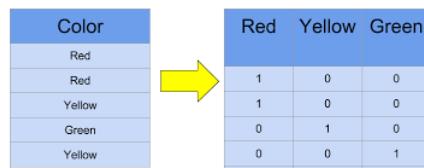
Les approches typiques pour traiter les valeurs manquantes d'une caractéristique sont les suivantes :

- **Suppression des observations à données manquantes** (ce qui peut être fait si l'ensemble de données est assez grand pour que l'on puisse sacrifier quelques exemples d'entraînement);

- **Imputation des données manquantes** en utilisant un algorithme d'apprentissage qui peut traiter les valeurs manquantes (ex: missForest algorithm [2]);

3.1.2 Numérisation des données

Pour que les algorithmes d'apprentissage automatique puissent traiter des variables catégorielles, qui peuvent être sous forme numérique ou sous forme de texte, ils doivent d'abord être transformés en une représentation numérique. Nous pouvons voir sur la figure ci-dessus un data set comprenant des données non numériques mais cardinales.



The diagram illustrates the transformation of a categorical variable into numerical variables. On the left, a table shows a 'Color' column with values 'Red', 'Red', 'Yellow', 'Green', and 'Yellow'. An arrow points from this table to another table on the right. The right table has columns 'Red', 'Yellow', and 'Green'. The first row contains values 1, 0, 0. The second row contains values 1, 0, 0. The third row contains values 0, 1, 0. The fourth row contains values 0, 0, 1. This represents a one-hot encoding where each category is represented by a binary vector of length 3, with a 1 at the position corresponding to the category and 0s elsewhere.

Figure 12: Transformation de variable catégorielle en variables numériques.

Les algorithmes d'apprentissage automatique ne peuvent pas fonctionner directement avec les données catégorielles ou cardinales. Les données catégorielles doivent être converties en chiffres.

Une transformation largement répandue dans l'apprentissage automatique est appelée 'One-Hot Encoding', que l'on peut retrouver dans la bibliothèque de Scikit-Learn (scikit-learn.org).

3.1.3 Centrage et réduction des données

Centrer et réduire les variables est une exigence commune pour la formation de nos méthodes de clustering efficacement puisque elles sont sensibles à la façon dont les vecteurs d'entrée sont ajustées. Dans plusieurs data sets, les variables ont une gamme de valeurs différentes (par exemple : les variables 'montant' et 'montant moyen de transactions par jour' peuvent avoir des échelles de valeurs différents), ce qui va qui se traduire par un temps d'entraînement plus long ainsi que des calculs de distance qui peuvent être biaisés et/ou faussés.

Le principal avantage de la centration-réduction est de rendre comparables des variables qui ne le seraient pas directement parce qu'elles ont des moyennes et ou des variances trop différentes.

Par conséquent, il est utile de centrer et réduire toutes les variables dans une manière dont chaque fonctionnalité est transformée en distribution normale en supprimant la valeur moyenne de chaque fonctionnalité, puis en la divisant par son écart-type.

Ainsi, à chaque valeur X_i nous ferons correspondre une valeur dite centrée-réduite, que nous notons usuellement par la lettre Z . La fonction de centrage-réduction est donnée par

$$Z_i = \frac{X_i - \bar{X}}{s_X}$$

Avec \bar{X} la moyenne et s_x l'écart type de la variable. La distribution ainsi obtenue aura pour moyenne 0 et pour écart-type 1.

3.2 Réduction de dimension

Réduire la dimensionnalité des données, c'est-à-dire le nombre de variables utilisées pour les représenter, permet de :

- Faciliter la visualisation des données ;
- Réduire les coûts de calcul, de stockage et d'acquisition des données ;
- Améliorer l'apprentissage en construisant des modèles moins complexes;
- Éliminer les variables non pertinentes qui pourraient fausser les prédictions.

Mais n'a-t-on pas besoin de toutes nos variables pour détecter des anomalies ?

En effet, certaines variables pourraient fausser et biaiser les résultats obtenus avec des techniques fondées sur la distance ou densité : K-Means, OCSVM, et DBSCAN par exemple. Ainsi, serait-il intéressant de réduire la dimension ?

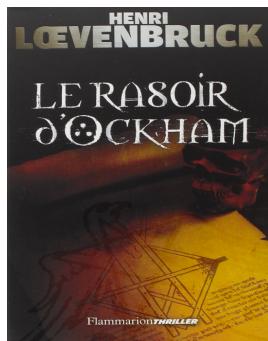


Figure 13: « Pluralitas non est ponenda sine necessitate» ou « Les multiples ne doivent pas être utilisés sans nécessité. » (principe du rasoir d'Ockham).

Un modèle, avec peu de variables, serait donc plus facilement généralisable en terme de robustesse ?

Il serait de ce fait intéressant de réduire la dimension pour les méthodes se basant sur la mesure de distance ou densité (ici OCSVM, DBSCAN, et K-Means), et conserver la totalité des variables pour les autres méthodes qui ne font pas appel à des calculs de distance ou densité, ici la méthode d'isolation notamment : l'isolation Forest.

Nous pouvons réduire la dimension à l'aide de l'ACP (Analyse des composantes principales). L'ACP est un outil intéressant utilisé pour réduire la dimensionnalité d'un jeu de données.

Lorsque l'on traite avec des data sets de grande dimension (plusieurs variables), les données pertinentes peuvent être exprimées en utilisant moins de dimensions.

C'est une approche géométrique et statistique compressant un ensemble de 'N' variables (en grande dimension) et synthétise l'information extraite en un ensemble de 'n' variables (plus petite dimension), avec $n < N$, à l'aide de transformations statistiques linéaires permettant de récupérer le plus d'information possible.

3.3 Analyse de données

Avant toute chose, il est primordial d'analyser notre jeu de données avant de appliquer des méthodes de clustering. Notamment pour des méthodes se basant sur la mesure de la distance ou densité comme DBSCAN ou K-Means.

Nous commençons par étudier la matrice des distances des observations (distance euclidienne), notamment la distance minimale (ou distance au plus proche voisin) et moyenne de chaque point :

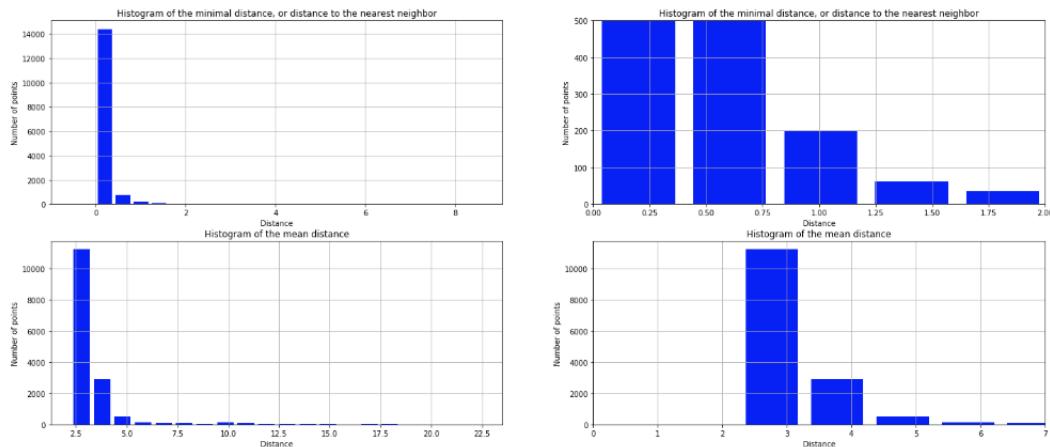


Figure 14: Matrice de distances du set de données de 15448 observations.

Nous remarquons que la majorité des observations se trouvent à une distance d'au plus de 0.75 unités de leur plus proche voisin. Ainsi, nous pouvons supposer que les observations présentant une distance minimale de plus de 0.75 sont soit dans une région de faible densité, soit écartés des autres. 0.75 serait une distance raisonnable pour le paramètre epsilon de DBSCAN.

De plus, nous pouvons voir que la majorité des observations se trouvent à une distance moyenne de moins de 4.5 unités des autres observations. Ainsi, nous pouvons émettre l'hypothèse que les observations qui ont une distance moyenne de plus de 4.5 unités se trouvent soit dans 'écartés' des autres, soit dans une région de faible densité. Nous fixons donc 1198 (nombre d'observations avec une distance moyenne supérieure à 4.5) de manière arbitre le nombre de fraudes prévu. Bien sur cette approche a été effectuée d'une manière assez large et peu stricte, ceci afin de pouvoir identifier le plus de fraudes possibles. Le 'filtrage' sera effectué par la suite.

Nous allons ensuite nous pencher vers le nombre de voisins de chaque observation, dans une boule de rayon $r=0.75$ (soit la distance minimale que l'on a choisis) :

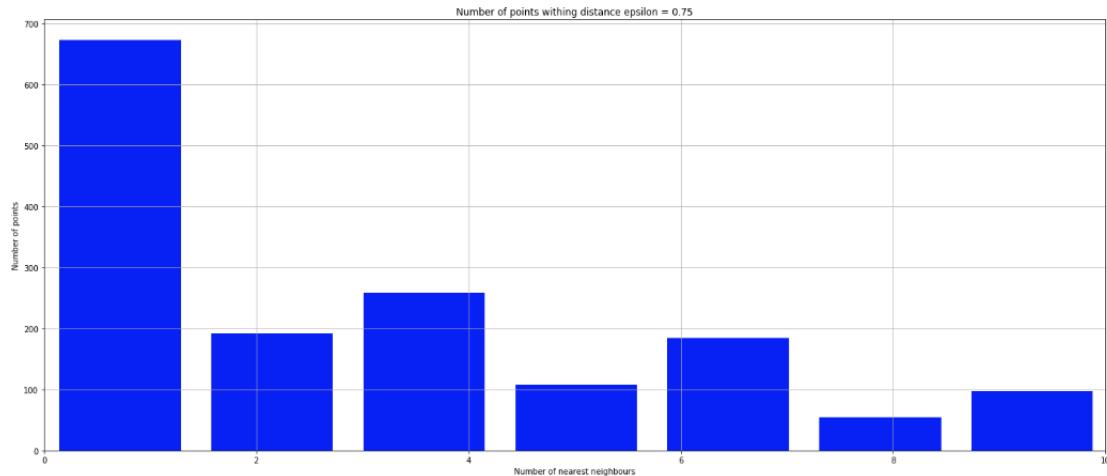


Figure 15: Histogramme du nombre de voisins dans une boule de rayon 0.75 unités.

Nous pouvons voir dans le graphique ci-dessus que près de 1200 (soit moins de 10% des observations, ici le nombre d'outliers prévu) ont très peu de voisins, entre 0 et 5 voisins dans une boule de rayon 0.75 unités.

Ceci signifierait que soit les points sont isolés, soit ils se trouvent dans une région de très faible densité. Ceci pourrait donc dire que la majorité des outliers se trouvent parmi cette proportion d'observations. Et donc que les observations avec plus de 5 voisins dans la boule de rayon $r=0.75$ unités pourraient être non frauduleux.

Ainsi, basé sur les informations extraites de ces histogrammes, nous allons fixer les paramètres de DBSCAN tel que **epsilon = 0.75**, **minPts = 5** et vérifier notre approche. **Le nombre d'outliers prévu reste fixé à 1198 outliers.**

Nota Bene :

L'histogramme ci-dessous nous montre bien que la majorité des outliers font bien partie de la portion d'observations jugées comme étant des anomalies dans notre approche précédente (moins de 5 voisins) :

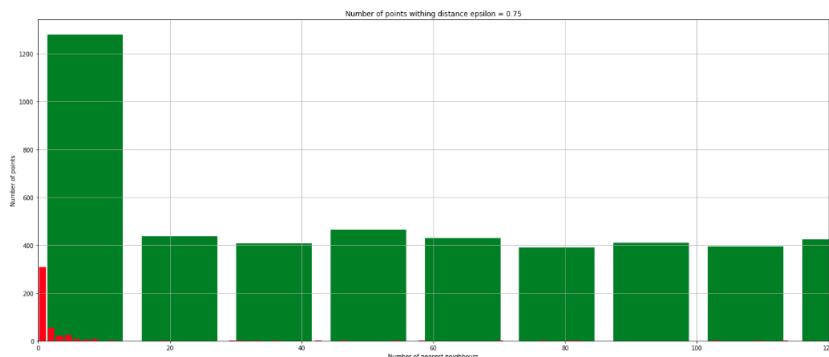


Figure 16: Les barres rouges représentent les 'outliers', tandis que les barres vertes représentent les 'inliers'.

3.4 Choix des paramètres, amélioration et implémentation

Suite à cette analyse de données, nous pouvons fixer les paramètres de la sorte : **epsilon = 0.75**, **minPts = 5** et **le nombre d'outliers prévu reste fixé à 1198 outliers**. Les algorithmes de clustering utilisés se trouvent sur le package de *sklearn* (scikit-learn.org). Nous cherchons par la suite à optimiser et/ou améliorer les algorithmes étudiés.

3.4.1 Ensemble Learning ou méthodes ensemblistes

Les algorithmes de clustering fondamentaux que nous avons examinés ont leurs limites. En raison de leur simplicité, parfois ils ne peuvent pas produire un modèle assez précis pour notre problème de détection de fraudes. Une autre approche pour stimuler la performance des algorithmes d'apprentissage simples est l'apprentissage 'ensembliste'.

L'apprentissage ensembliste est un paradigme d'apprentissage qui, au lieu d'essayer d'optimiser un modèle afin d'obtenir un modèle 'superprécis', met l'accent sur la formation d'un grand nombre de modèles de faible précision, puis la combinaison des prédictions données par ces modèles de faible précision pour obtenir un métamodèle de haute précision.

Pour obtenir la prédition pour l'entrée x , les prévisions de chaque modèle faible sont combinées en utilisant une sorte de vote pondéré. La forme spécifique de pondération des votes dépend de l'algorithme, mais, indépendamment de l'algorithme, l'idée est la même : si la majorité des modèles 'faibles' prédit que la transaction x est 'fraude', alors nous assignons le label 'fraude' à x .

Nous montrerons par la suite dans la section 3.6 comment nous allons utiliser cette approche dans notre méthode proposée.

3.4.2 Paramétrage et optimisation du DBSCAN

Nous avons fixé les paramètres suivants dans notre étude précédente :

- **Epsilon** = 0.75 unités. La distance maximale entre deux voisins pour qu'ils soient considérés comme dans le même cluster. Si la distance à son plus proche voisin est supérieure à epsilon , soit à 0.75 unités, alors il est considéré comme outlier.

- **MinPts** = 5 voisins. Soit chaque point doit avoir au moins 5 points dans son Epsilon-voisinage (de 0.75 unités) pour former un cluster.

Nous considérons le cluster le plus important comme étant frauduleux, les autres clusters sont regroupés et jugés 'outliers'.

Vérification de la sensibilité de l'algorithme en fonction du paramètres MinPts entre 1 et 50 (Epsilon fixé à 0.75 Unités) :

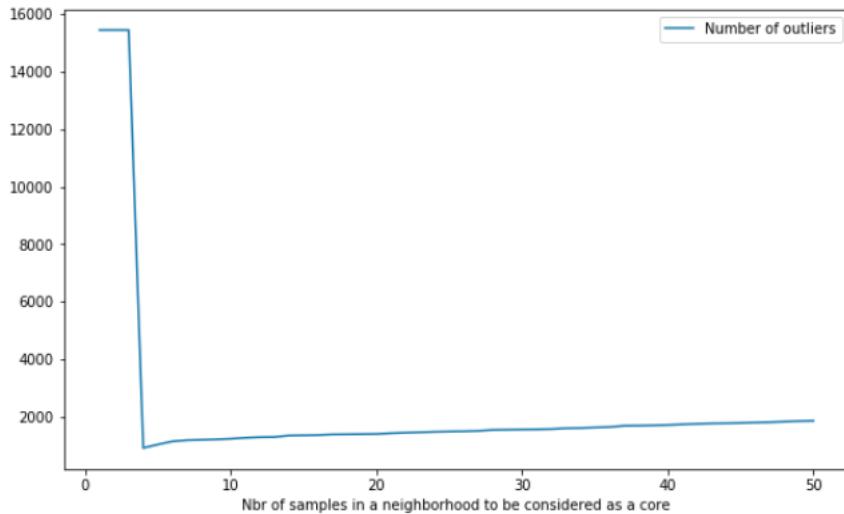


Figure 17: Nombre d’outliers détectés en fonction de la valeur du paramètre MinPts.

Nous pouvons donc voir qu’à partir de $\text{MinPts}=5$, le nombre d’outliers détectés tourne autour de 2000. Toutefois pour $\text{MinPts} = 5$, le nombre d’outliers détectés se rapproche du nombre d’outliers prévu (soit 1198), ce qui nous rassure dans notre choix de paramètres.

3.4.3 Paramétrage et optimisation de l’Isolation Forest

Nous savons que l’algorithme repose sur deux paramètres : le ‘sous-échantillonnage’ et le ‘nombre d’arbres’.

Nous avons choisi de garder toutes les variables dont on dispose (donc pas de sous-échantillonnage). Toutefois, concernant le nombre d’arbres, et sachant que l’algorithme dispose d’un côté aléatoire, nous avons décidé de réitérer le modèle plusieurs fois et avoir recours à de l’apprentissage ensembliste.

En effet, comme vu précédemment dans la section 3.4.1 cet approche nous permettra de mettre l’accent sur la formation d’un grand nombre de modèles d’isolation forest de ‘faible précision’ (ici $K=5$ modèles avec 100 arbres, K modèles avec 150 arbres, etc... jusqu’à $K=5$ modèles avec 300 arbres) puis la combinaison des prédictions données par ces modèles de faible précision pour obtenir un métamodèle de haute précision. En effet chaque modèle nous permet d’obtenir un score pour chaque observation. Nous récupérons ainsi les scores pour chaque individu pour en faire la moyenne : nous établissons ensuite un seuil à partir duquel les observations concernées sont jugées fraudes.

Dans notre étude, nous utilisons la fonction `score` mise en place sur l’algorithme de *scikit-learn*, selon laquelle les observations avec un score négatif (et donc inférieur à 0) sont jugées outliers. Après avoir fait la moyenne des scores obtenus, chaque observation avec un score inférieur à 0 est classé dans la catégorie ‘fraude’.

L’algorithme disponible sur *scikit-learn* requiert aussi un autre paramètre important : *contamination*, soit la proportion d’outliers dans le dataset que l’on cherche à obtenir. On le fixe selon le nombre d’outliers prévu (ici 1198).

3.4.4 Paramétrage et optimisation du OCSVM

Nous avons vu dans la section 2.1.1 que les paramètres les plus significatifs sont ν et γ . Nous fixons ν égal au nombre d'outliers prévu (ici 1198), et faisons varier γ :

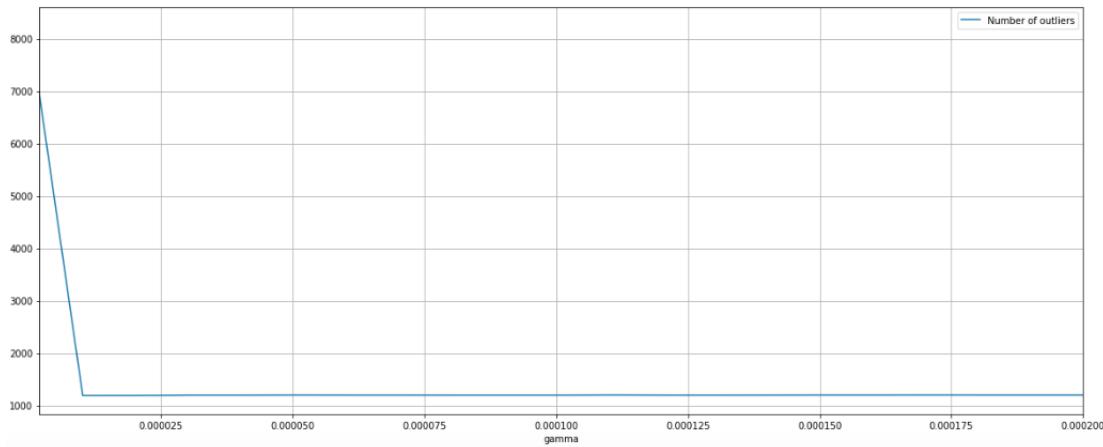


Figure 18: Nombre d'outliers détectés en fonction de la valeur du paramètre γ .

Nous cherchons un γ le plus petit possible, mais répondant au nombre d'outliers attendu. Nous fixons donc $\gamma = 0.00002$, car nous pouvons voir que cette valeur est assez petite et qu'à partir de cette valeur le nombre d'outliers est fixé à la valeur de ν , soit 1198.

3.4.5 Paramètres de K-means

Le principe de l'algorithme K-Means repose en grande partie sur un seul paramètre : le nombre de clusters K . L'algorithme dispose aussi d'un côté aléatoire, car le résultat dépend du tirage initial (aléatoire) des centres des classes. Nous avons décidé de réitérer le modèle plusieurs fois et avoir recours à de l'apprentissage ensembliste.

En effet, comme vu précédemment dans la section 3.4.1 cet approche nous permettra de mettre l'accent sur la formation d'un grand nombre de modèles K-Means de 'faible précision' (ici modèle de $K=2$ clusters répété nb fois, modèle de $K=3$ clusters répété nb fois etc... jusqu'à modèle de $K=N$ clusters répété nb fois) puis la combinaison des prédictions données par ces modèles de faible précision pour obtenir un méta-modèle de haute précision.

Nous récupérons ainsi les scores pour chaque individu de chaque modèle (ici $nb*(N - K + 1)$ modèles) pour en faire la moyenne : nous établissons ensuite de manière arbitraire un seuil à partir duquel les observations concernées sont jugées fraudes. En effet, plus l'observation i est apparue dans la classe 'fraude' les prédictions, plus son score final est élevé.

Plus précisément, l'algorithme modifié des K-Means suit la procédure suivante :

-Soit $K = i$, avec $i = 2 \dots N$.

-Première itération : les clusters d'une taille assez importante sont considérés comme 'non fraude', tandis que les autres clusters sont fusionnés pour former la classe 'fraude'. Pour ce faire nous fixons arbitrairement un seuil tol , tel que si la taille d'un cluster est supérieure à tol ,

alors celui-ci est considéré comme 'non fraude'. Nous fixons $tol = \text{nombre d'outliers prévu} = 1198$. En effet, nous avons prévu un nombre d'outliers d'au plus 1198, ainsi nous ne pouvons nous permettre de désigner un cluster de taille plus grande que 1198 comme étant outlier.

-Nous réitérons ce modèle nb fois. Nous attribuons un score à chaque point en fonction de son apparition dans la classe d'outliers. Nous fixons un seuil de 0.5, tel que si l'observation apparaît en tant qu'outlier plus de 50% des itérations, alors celui-ci sera considéré outlier. Nous choisissons un nombre d'itérations nb assez grand, ici $nb=50$ afin de 'gommer' le côté aléatoire de l'algorithme K-Means.

-Nous répétons cette procédure $N - 2 + 1$ fois, $K = 2K = N$. Ainsi, pour chaque modèle i , chaque observation a un score de 1 dans le modèle si elle est considérée comme une fraude, score de 0 sinon. Nous effectuons ensuite la moyenne des scores d'apparition, et nous assignons ensuite les observations à leur cluster de la sorte : si l'observation apparaît dans plus de 50% des modèles en tant que fraude, alors celui-ci sera affecté à la classe 'fraude'.

Il est vrai que le seuil de formation des clusters semble trop important (ici 1198), et nous pourrions donc obtenir une prédition faussée. Ceci seulement si le modèle n'est pas assez bien calibré (le nombre maximal de clusters N à utiliser dans notre algorithme).

En effet, il est nécessaire d'étudier la variation du nombre d'outliers détectés par notre algorithme en fonction du nombre maximal de clusters. Dans cette étude, nous avons remarqué qu'à partir de $K=10$ clusters, le modèle commence à détecter plus de 1200 outliers, ce qui ne correspond pas au nombre d'outliers prévu. Voici pourquoi nous fixons le nombre maximal de clusters à 10. Chaque modèle $K = i$ avec $i \leq 10$ se chargera d'isoler les groupes éloignés du reste des observations, ce qui pourrait constituer des outliers. Cet algorithme de K-Means s'avère bien plus robuste que les autres méthodes utilisées auparavant (plus précis), seulement il faut bien le calibrer. Vous pouvez très choisir un seuil de formation des clusters plus petit pour avoir une étude plus restreinte, ou plus grand pour pouvoir identifier le plus d'outliers possibles.

3.5 Mesure et Évaluation de performance

L'évaluation des modèles (y compris l'évaluation des modèles d'apprentissage supervisés et non supervisés) est le processus qui consiste à mesurer objectivement la mesure dans laquelle les modèles de Machine Learning effectuent les tâches précises pour lesquelles ils ont été conçus, comme la prévision du prix des actions ou la détection de fraudes.

Comme chaque modèle d'apprentissage automatique est unique, les méthodes d'évaluation optimales varient selon que le modèle en question est « supervisé » ou « non supervisé ». Les modèles d'apprentissage automatique supervisés produisent des prédictions ou des classifications spécifiques basées sur des données de formation étiquetées, tandis que les modèles d'apprentissage automatique non supervisés cherchent à regrouper ou à trouver des modèles dans des données non étiquetées.

Nous pouvons distinguer deux catégories de métrique de validation :

3.5.1 Évaluation de qualité de méthodes supervisées

Nous pouvons énumérer

Dans cette étude, nous utilisons trois indices de performance efficaces d'une classification que l'on utilisera dans cette étude :

- **La matrice de confusion** : est un outil servant à mesurer la qualité d'un système de classification, en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité.

| | | Classe réelle | |
|-------------------|---|------------------------------------|------------------------------------|
| | | - | + |
| Classe prédictive | - | True Negatives (vrais négatifs) | False Negatives (faux négatifs) |
| | + | False Positives (faux positifs) | True Positives (vrais positifs) |

Figure 19: Matrice de confusion.

- **Le rappel** (ou "recall" en anglais) : le taux de vrais positifs, c'est à dire la proportion de positifs que l'on a correctement identifiés. Ici : C'est la capacité de notre modèle à détecter toutes les fraudes.

La formule du 'recall' est la suivante :

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

- **La précision**, c'est-à-dire la proportion de prédictions correctes parmi les points que l'on prédits positifs. Ici : la capacité de notre modèle à ne déclencher d'alarme que pour une vraie fraude.

La formule de la précision est la suivante :

$$Precision = \frac{TP}{TP + FP}$$

3.5.2 Évaluation de qualité de méthodes non supervisées

En général, l'évaluation des algorithmes de clustering en apprentissage non supervisé est difficile parce qu'il n'y a pas de mesure bien définie pour le partitionnement.

Une façon courante d'évaluer les algorithmes de clustering est la suivante :

- Utiliser un ensemble de données étiqueté.
 - Retirer les étiquettes.
 - Appliquer l'algorithme de clustering en question.
 - Évaluer la performance du partitionnement, en le comparant aux données étiquetées.
- Ce n'est pas très satisfaisant, mais c'est la meilleure approche actuellement utilisée.

Toutefois, il existe des mesures de qualité robustes de modèles non supervisés que l'on pourrait utiliser. Notamment l'indice "Silhouette", qui mesure à quel point un cluster est distinct ou bien séparé des autres clusters.

Plus précisément, la ‘silhouette’ est une mesure de la façon dont un ‘point’ est similaire à son propre cluster (cohésion) par rapport à d’autres clusters (séparation).

La formule de l’indice ‘Silhouette’ est la suivante :

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

avec :

- $s(i)$ Coefficient ‘silhouette’ de l’individu i
- $b(i)$ La distance moyenne entre un individu tous les autres points d’un même cluster/classe.
- $a(i)$ La distance moyenne entre un individu tous les autres points d’un même cluster/classe.

La silhouette varie de -1 à +1, où une valeur élevée indique que l’objet est bien adapté à sa propre classe/cluster et mal adapté à la classe voisine. Si score proche de 0 : Le point n’est pas très bien classé, pourrait appartenir tant à l’une que l’autre classe.

Si la plupart des objets ont une valeur élevée, alors le partitionnement a été bien réalisé. Un score élevé indique que le modèle comporte des clusters bien définis.

Si de nombreux points ont une valeur faible ou négative, alors le partitionnement peut avoir trop ou trop peu de clusters.

Ainsi, nous pouvons dire que l’indice ‘Silhouette’ joue un rôle non négligeable dans la ‘précision’ d’un algorithme : plus le score est élevé (proche de 1), mieux le partitionnement est effectué et donc plus la valeur de l’indice de précision est grande. Nous utiliserons cette mesure par la suite pour optimiser notre algorithme.

3.6 Détection d’outliers : méthode proposée

La méthode proposée repose sur le principe de l’apprentissage ensembliste.

Nous considérons plusieurs classifieurs basés sur des approches de clustering différents (distance, densité, isolation), pour ensuite combiner l’information que chacun porte afin d’obtenir un super-classifieur plus performant. Nous moyennons ainsi les résultats des classifieurs, ce qui diminue la sensibilité aux données sans augmenter son biais puisque le ‘super-classifieur’ produit devrait disposer de l’information de chaque classifieur seul.

3.6.1 Méthode de clustering

La méthode proposée repose sur le principe suivant:

Après plusieurs tests, les méthodes améliorées d’Isolation Forest, DBSCAN, OCSVM et K-means sont jugées efficaces et robustes. En effet, elles sont capables d’identifier une grosse partie des fraudes. Toutefois, elles présentent une précision assez faible.

Il faut noter que ces méthodes se basent sur des approches de clustering différentes (densité, distance et isolation). Il serait de ce fait intéressant de combiner les résultats obtenus afin d’obtenir un résultat plus robuste en mettant en place la méthode suivante :

1) Détection :

Application des quatre algorithmes de machine learning améliorés et optimisés.

2) Combinaison ou Apprentissage Ensembliste :

Nous combinons ensuite les résultats de chaque algorithme : si une observation est identifiée dans la classe "fraude" dans l'une des 4 techniques, alors celle-ci sera identifiée comme fraude.

Nous obtenons alors un nombre considérable d'observations jugées frauduleuses, et parmi celles-ci : une grosse partie des fraudes répertoriées, avec beaucoup d'imprécis et d'erreur. En effet, cette méthode ensembliste permet de combiner les avantages, mais aussi les inconvénients de chaque algorithme.

3) Filtre :

Nous cherchons désormais à ne garder que les avantages de chaque algorithme utilisé, soit détecter le plus de fraudes tout en diminuant le plus d'erreurs possibles. Nous cherchons ainsi à mettre en place un compromis détection-précision, et pour ce faire, nous allons appliquer la méthode d'évaluation silhouette pour filtrer notre clustering obtenue.

En effet, nous avons vu dans la section 3.5.2 que l'application de l'indice "Silhouette" nous permet d'obtenir un score pour chaque observation : - Si son score est proche de -1, alors celui-ci serait mal classé et devrait appartenir à l'autre classe. - Si son score est proche de 0, alors celui-ci pourrait appartenir tant à l'une que dans l'autre classe.

Alors nous allons fixer un seuil *tol* tel que si le score de l'observation est inférieur au seuil, alors celui-ci va être transféré à l'autre classe.

Nous obtenons au final une classification filtrée, plus précise. Nous présentons donc une méthode robuste et insensible à la variation des données, car les algorithmes utilisés présentent tous des approches de clustering différentes.

3.6.2 Méthode de clustering avec un score en sortie : du non-supervisé au supervisé

Cette méthode dérive de la précédente, à quelques détails près : celle-ci nous permet d'avoir un score de fraude pour chaque observation en sortie.

Les étapes de la méthode sont les suivantes :

1)Détection : Application des quatre algorithmes de machine learning améliorés et optimisés.

2)Combinaison ou Apprentissage Ensembliste V2 :

Même principe que la deuxième étape de la méthode précédente, seulement ici nous attribuons un "score" à chaque observation entre 0 et 1 de la manière suivante :

- Si une observation n'a pas été identifiée comme 'fraude' par les algorithmes utilisés, alors nous lui attribuons un score de 0.
- Si une observation a été identifiée comme 'fraude' par 1 seul algorithme utilisé, alors nous lui attribuons un score de 0.25
- Si une observation a été identifiée comme 'fraude' par 2 algorithmes utilisés, alors nous lui attribuons un score de 0.5
- Si une observation a été identifiée comme 'fraude' par 3 algorithmes utilisés, alors nous lui attribuons un score de 0.75
- Si une observation a été identifiée comme 'fraude' par les 4 algorithmes utilisés, alors nous lui attribuons un score de 1.

3) Rééquilibrage : oversampling :

Nous envisageons à appliquer l'algorithme de réseaux de neurones dans cette méthode. Toutefois, nous remarquons que notre data set n'est pas équilibré (le cluster des observations avec un score de 0 présente une taille dans l'ordre de 12000 opérations, soit près de 75% de notre data set). En effet, les classes minoritaires risquent d'être mal classés afin de classer plus d'exemples de la classe majoritaire correctement (classe '0'). De plus, il serait complexe de mettre en place un jeu de données de validation afin d'entraîner le réseau de neurones (par séparation en train data et test data). En effet, si l'on choisit 0.7 comme proportion de données d'entraînement et 0.3 celle des données de validation, on pourrait très bien tomber sur un train data composé uniquement des observations de classe '0', ce qui serait inutile dans notre étude par réseau de neurones.

Il serait donc intéressant de rééquilibrer le jeu de données en attribuant la même taille à chaque classe, en faisant appel à la méthode SMOTE d'oversampling.

4) Du non supervisé au supervisé :

Après avoir appliqué la méthode SMOTE d'oversampling, nous obtenons ainsi un jeu de données et un vecteur "cible" de taille beaucoup plus importante (près de 50000 observations) et équilibrés. Nous pouvons les utiliser afin d'entraîner un réseau de neurones dont la fonction d'activation est la fonction sigmoïde : celle-ci nous permettra d'obtenir en sortie un score entre 0 et 1 pour chaque observation, soit la probabilité d'appartenir à la classe '1' ou 'Fraude'.

Après avoir utilisé ce data set oversampled afin d'entraîner le réseau de neurones, nous pouvons désormais utiliser ce modèle afin de prédire le score de chaque individu dans le data set précédent non équilibré (celui obtenu à la fin de l'étape 2). Cette approche nous permet d'obtenir en sortie un score et non une classification, ce qui laisse la possibilité de fixer un seuil de score pour identifier les transactions suspectes.

4 Résultats

Le jeu de données utilisé est composé de 15448 observations et 448 fraudes, soit 2.9%, avec 11 variables. Nous avons cherché à réduire la dimensionnalité en ne gardant que 7 variables (pas appliqué dans l'étude de l'Isolation Forest), soit la conservation de 95% de l'information.

Dans cette étude, nous fixons les seuils suivants :

- Le seuil du score de silhouette fixé à -0.3. Soit si une observation a un score inférieur à -0.3, alors on l'assigne à l'autre classe.
- Le seuil du score issu de l'application des réseaux de neurones à 0.5. Soit si l'observation a un score supérieur à 0.5, alors on l'assigne à la classe 'fraude'.

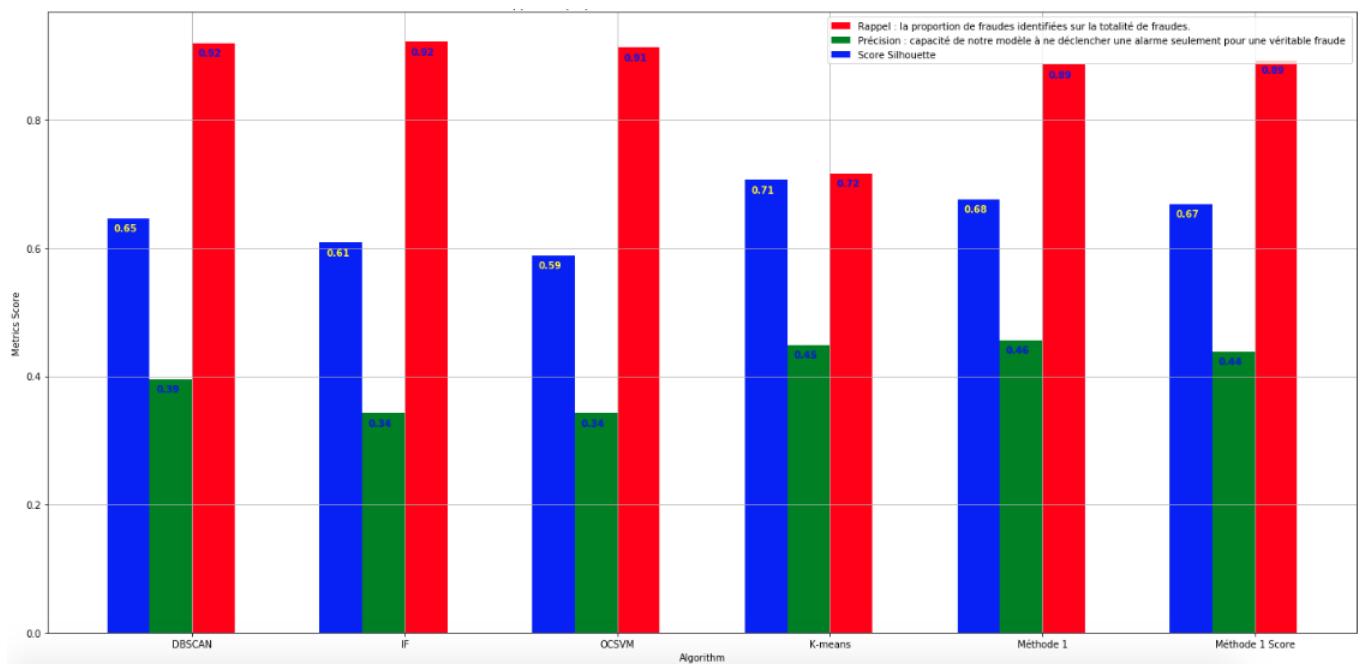


Figure 20: Indicateurs de performance des algorithmes : précision en vert, recall en rouge et indice de silhouette en bleu.

Nous remarquons que :

- Les algorithmes améliorés ainsi que la méthode proposée offrent de très bons scores au niveau de l'identification des fraudes (indice rappel). Le DBSCAN et l'Isolation Forest présentent un score de 92%, ce qui signifie que les deux méthodes ont réussi à identifier 92 des fraudes. L'OCSVM a identifié 91% des fraudes, et le K-means 72%. Notre méthode proposée quant à elle a réussi à identifier 89% des fraudes, ce qui reste un très bon score. En effet, si l'on dispose de 500 fraudes, 1% du score de rappel représenterait 5 fraudes. C'est-à-dire que le DBSCAN et l'Isolation Forest arrivent à identifier 460 fraudes, 15 fraudes de plus que la méthode proposée.

- Concernant la précision, la méthode proposée (sans score) détient le meilleur score de précision avec 46% de précision, contre 34% pour l'OCSVM, 39% pour le DBSCAN et 34%

pour l'Isolation Forest.

Ceci veut donc dire que l'avantage des algorithmes améliorés est qu'ils arrivent à identifier une grosse partie des fraudes, mais avec beaucoup d'imprécis que le modèle proposé (car en effet, en fonction du nombre de fraudes, 10% de différence en précision peut dépasser 500 fausses alarmes en plus). De ce fait il est assez compliqué de mettre en place un compromis recall-précision.

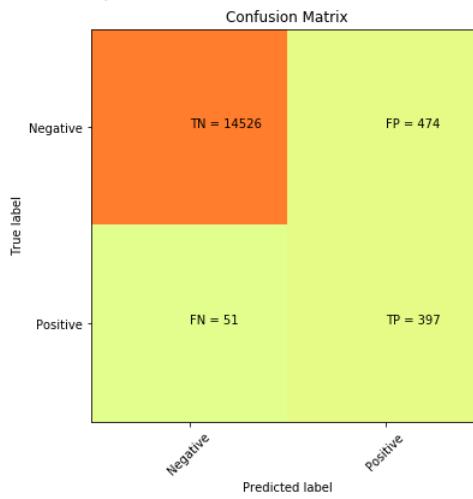


Figure 21: Matrice de Confusion de la méthode 1 après avoir fixé le seuil de silhouette à -0.3.

Le modèle prédictif a bien identifié 397 fraudeurs, et s'est trompé sur 474 individus non fraudeurs.

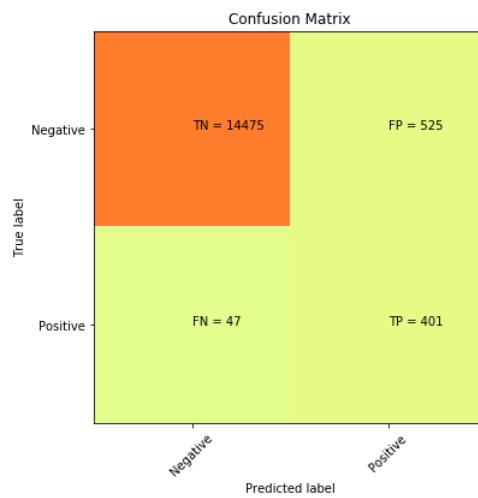


Figure 22: Matrice de Confusion de la méthode 1 avec score, après avoir fixé le seuil de score à 0.5.

Le modèle prédictif a bien identifié 401 fraudeurs, et s'est trompé sur 525 individus non fraudeurs.

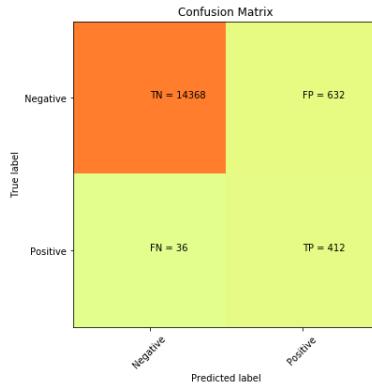


Figure 23: Matrice de Confusion du DBSCAN.

Le modèle prédictif a bien identifié 412 fraudeurs, et s'est trompé sur 632 individus non fraudeurs.

Ceci montre bien que 3% de différence de score 'rappel' entre le DBSCAN et la méthode 1 représente 15 fraudes non identifiées, mais seulement 7% d'erreur de précision représentent près de 150 fausses alarmes.

4.1 Test sur un autre jeu de données

Dans cette étude, nous cherchons à appliquer notre approche sur un autre jeu de données afin de vérifier la robustesse de notre approche. Ce jeu de données est importé de Kaggle sur www.kaggle.com/mlg-ulb/creditcardfraud/home, de 284 807 transactions (seulement en deux jours de transactions, ce qui décrit la fréquence journalière des opérations) et 492 fraudes, soit 0.172% des opérations. Nous ne retiendrons que 15000 individus non fraudeurs (par tirage aléatoire) et 492 fraudeurs dans notre étude. Le jeu de données est composé de 30 variables. En utilisant la même approche (sauf celle de la réduction de dimension, car le data set semble avoir été déjà traité), nous obtenons les résultats suivants :

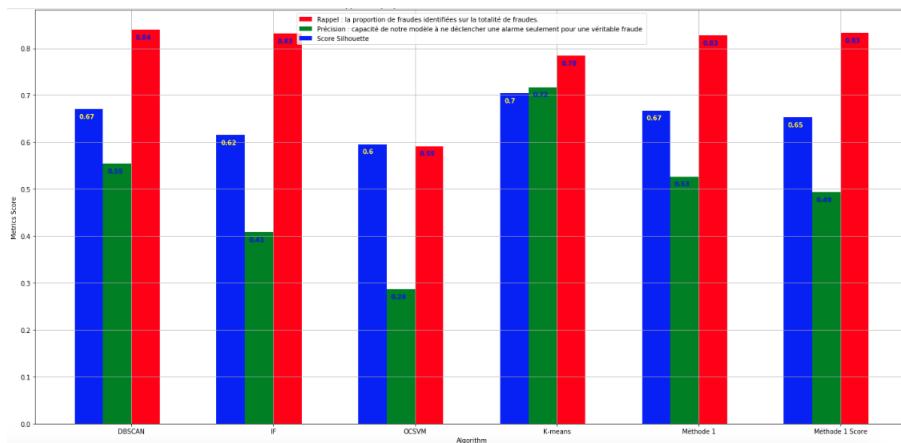


Figure 24: Indicateurs de performance des algorithmes : précision en vert, recall en rouge et indice de silhouette en bleu.

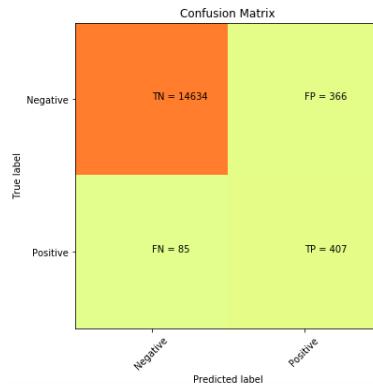


Figure 25: Matrice de Confusion de la méthode 1 après avoir fixé le seuil de silhouette à -0.3.

Le modèle prédictif a bien identifié 407 fraudeurs, et s'est trompé sur 366 individus non fraudeurs.

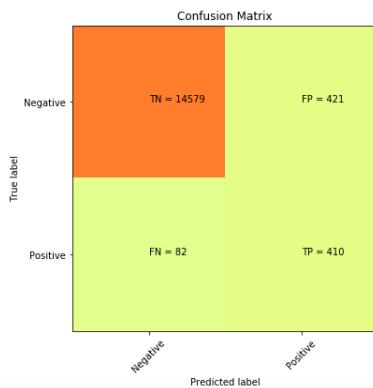


Figure 26: Matrice de Confusion de la méthode 1 avec score, après avoir fixé le seuil de score à 0.5.

Le modèle prédictif a bien identifié 410 fraudeurs, et s'est trompé sur 421 individus non fraudeurs.

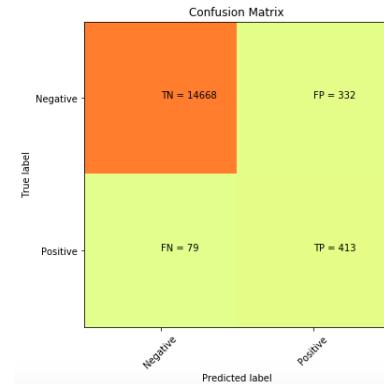


Figure 27: Matrice de Confusion du DBSCAN.

Le modèle prédictif a bien identifié 413 fraudeurs, et s'est trompé sur 332 individus non fraudeurs.

Ceci montre bien que notre modèle est autant efficace pour un autre jeu de données (83% de fraudes identifiées et 53% de précision pour le sans score, 49% pour la méthode avec score), de même pour l'algorithme DBSCAN avec 84% de fraudes identifiées et 55% de précision. Toutefois, nous remarquons que l'OCSVM n'est pas aussi performant qu'il l'était pour notre premier jeu de données. Ceci nous montre une sensibilité face à la variation des données. Cependant nous soulignons la robustesse de la méthode proposée (ainsi que celle du DBSCAN et Isolation Forest).

5 Conclusion

Suite à notre étude, nous pouvons donc émettre les conclusions suivantes :

- L'indice 'Silhouette' est un bon indicateur de la précision et permet de 'filtrer' nos résultats.

- L'analyse de données mise en place semble être efficace pour les méthodes DBSCAN, OCSVM, et Isolation Forest. Il serait intéressant d'approfondir cette étude et établir un choix de paramètres plus strict et précis afin de privilégier la précision. Il serait aussi intéressant d'améliorer le paramétrage de l'algorithme modifié de K-Means, car il est très précis (un peu trop), mais n'identifie pas assez de fraudes (seulement 73% pour notre étude).

- Il est intéressant de réduire la dimension pour les méthodes basées sur la densité ou distance : DBSCAN, OCSVM et K-Means. C'est de même une bonne idée de garder toutes les variables pour l'Isolation Forest.

Ainsi nous avons pu montrer que cette approche par méthode ensembliste a réussi à réunir les avantages de chaque algorithme de clustering utilisé. Filtrer les résultats obtenus à l'aide de l'indice de "Silhouette" nous a permis d'identifier près de 90% des fraudes, même s'il présente une précision de moins de 50%. Toutefois cette précision reste négligeable : en consultant les matrices de confusion, nous remarquons que ces 50% représentent pas plus de 500 opérations, soit près 0.3% du data set, ce qui reste négligeable.

La méthode nous donnant un score pourrait donner de meilleurs résultats si les paramètres du réseau de neurones sont bien calibrés et si le seuil de score est bien choisi. Cette méthode mise en place, robuste et assez efficace, pourrait tout de même être améliorée si l'on fixe bien les paramètres.

De plus, nous soulignons la robustesse de la méthode utilisée en combinant les trois catégories d'algorithmes de clustering. Peut être devrions nous garder seulement le DBSCAN, Isolation Forest et K-Means pour former un méta-modèle à haute précision et robuste face à la variation des données.

References

- [1] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” 2008.
- [2] J. S. Daniel, “Using the missforest package,” 2011.
- [3] A. Sharma and P. K. Panigrahi, “A review of financial accounting fraud detection based on data mining techniques,” 2012.
- [4] W. Jarrod, B. Maumita, and I. Rafiqul, “Intelligent financial fraud detection practices: An investigation,” 2017.
- [5] L. Daijiang and Z. Qingsheng, “Automatic pam clustering algorithm for outlier detection,” May 2012.
- [6] T. M. Thang and J. Kim, “The anomaly detection by using dbSCAN clustering with multiple parameters,” 2011.
- [7] M. Ji and H.-J. Xing, “Adaptive-weighted one-class support vector machine for outlier detection,” 2017.
- [8] A. Agrawal and S. Kumar, “A novel approach for credit card fraud detection,” 2015.
- [9] L. Jin-Miao and J. Tian, “A hybrid semi-supervised approach for financial fraud detection,” 2017.
- [10] C. Nitesh V., B. Kevin W., O. H. Lawrence, and K. W. Philip, “Smote: Synthetic minority over-sampling technique,” 2002.
- [11] “Scikit learn.” www.scikit-learn.org.