Rapport de projet
Sécu & IA - Détection d'anomalies

Alexandre Lemonnier - Sarah Gutierez - Victor Simonin

Promo 2023

Table des matières

1 Introduction

Dans le cadre du cours intitulé "Sécu & IA" enseigné à l'EPITA lors de la seconde année du cycle ingénieur en majeure SCIA, nous avons pour projet d'analyser des données de cybersécurité répertoriant des attaques sur des flux réseaux. Nous avons donc choisi l'Objectif 1 : "Anomaly detection for tracking attacks" sur la base de données UGR'16. Ce projet a pour but de déterminer les flux qui correspondent à des attaques et quelles sont les données impliquées dans leur détection.

Dans un premier temps, nous avons dû observer ces données pour comprendre les différentes informations dont le jeu de données était composé, et ainsi pouvoir relever les informations manquantes, aberrantes ou inutiles pour la suite du projet. Nous avons ensuite pu nettoyer nos données, tout en appliquant des transformations utiles à l'apprentissage de nos futurs modèles de classification.
Enfin nous avons développé différents modèles de classification et de détection d'anomalies grâce à des algorithmes de Machine Learning supervisé et non-supervisé, pour pouvoir les comparer, réaliser des benchmarks et finalement pour conclure sur les différents évènements de cybersécurité dans la base de données.

2 Les données

Les données sur lesquelles nous avons décidé de travailler sont les données UGR'16. Ces données très complètes s'étalent sur quatre mois de calibration avec uniquement des données de trafic web réel et environ deux mois de données de test contenant du trafic réel et des données synthétiques. Nous avons dû réduire notre analyse à une seule semaine par jeu de données afin de pouvoir travailler convenablement dessus.

Pour nos données d'entraînement nous avons choisi la semaine du 1 juin 2016 au 5 juin 2016 et pour nos données de test nous avons pris la semaine du 17 juillet 2016 au 31 juillet 2016.

L'échantillon de la base de données que nous avons choisi possède plus de 680 millions de lignes et 13 colonnes différentes qui contiennent toutes les informations sur les différents échanges réseaux ayant eu lieu sur la semaine que nous avons choisi. Chaque ligne correspond à un échange, et possède beaucoup d'informations pour décrire ce qu'il s'est passé pendant celui-ci.

2.1 Les champs

Les différents champs de la base de données ne sont pas précisés directement en *header* de celles-ci, mais ils sont pour certains explicités dans le papier correspondant, et pour les autres facile à inférer ou disponibles sur des sources externes.

Les colonnes sont les suivantes :

Nom de la colonne	Type	Informations
Date et temps de fin d'un flux	object	-
Durée du flux	float64	En secondes
IP source	object	IP venant d'un potentiel attaquant
IP de destination	object	IP potentiellement touchée
Port source	int64	Port venant d'un potentiel attaquant

Port de destination	int64	Port potentiellement touché
Protocole utilisé	object	TCP ou UDP
Flag utilisé	object	Indique certaines conditions comme : l'utilisateur source à réussi à se connecter à un shell avec toutes les permissions
Forwarding status	int64	Indique s'il y a eu une redirection ou non
Type de service	int64	Donne des informations sur le type de service utilisé lors de l'échange de données
Nombre de paquets échangés	int64	-
Nombre d'octets échangés	int64	-
Label	object	Possède différentes valeurs permettant d'indiquer si c'est un échange frauduleux. Les valeurs sont : *background, blacklist, anomaly-spam, anomaly-udpscan, anomaly-sshscan, nerisbotnet, scan11*, ou *dos*. Dans notre analyse nous utilisons ce que possède notre base de données c'est-à-dire *background* (pas de soucis) et *blacklist* (attaque).

Cependant, considérer les champs *Port source* et *Port de destination* comme des variables numériques n'est pas correct. En effet, les ports sont représentés par des nombres mais cela ne veut pas dire que l'on peut les comparer tel quels. C'est pourquoi nous avons changé le type de ces deux champs afin de les analyser en tant que variables catégorique et non numérique.

2.2 Pré-traitement

Chaque modification effectuée au cours du pré-traitement des données a été appliquée à la fois aux données d'entraînement, soit la semaine du 1 juin 2016 au 5 juin 2016, et aux données de tests, soit la semaine du 17 juillet 2016 au 31 juillet 2016.

A ce stade de notre analyse, nous avions comme variables numériques :

- Durée du flux
- Forwarding status
- Type de service
- Nombre de paquets échangés
- Nombre d'octets échangés

D'autre part, nous avions comme variables catégoriques :

- Date et temps de fin d'un flux
- IP source
- IP de destination
- Port source
- Port de destination
- Protocole utilisé
- Flag utilisé
- Label

L'un des premiers choix qui a été fait est d'avoir une répartition contrôlée des attaques et du flux normal dans nos données d'entraînement. Nous avons donc décidé de garder 4% des lignes caractérisées comme attaque dans nos modèles et donc 96% des lignes caractérisés en flux *background*.

Cette répartition va forcément impacter la performance et les résultats de nos algorithmes de classification et de détection d'anomalies, mais il nous semblait alors important de fixer ce paramètre pour avoir toujours la même répartition d'attaques quoi qu'il arrive dans nos données d'entraînement.

2.2.1 Nettoyage

Le nettoyage est une partie prépondérante du projet ainsi que de toute analyse de données. En effet, les données récupérées proviennent directement d'un logiciel d'enregistrement des échanges réseaux sont potentiellement erronées. L'objectif est de faire une première analyse de nos données, d'éliminer les éléments inutiles, d'en normaliser certaines afin qu'à terme nos modèles de classification soient en capacité de relever les éléments déterminants d'un échange suspicieux ou non.

Dans un premier temps, nous avons vérifié l'existence de données manquantes dans nos données, ce qui aurait été problématique pour la suite de notre étude. Dans le cas où certaines l'étaient, nous aurions dû faire le choix entre perdre l'intégralité de l'échange ou remplir la donnée manquante en fonction des autres données, soit par rapport à leur moyenne ou leur médiane, soit en entraînant un modèle de régression logistique si les données existantes étaient suffisantes. Par chance, les données choisies ne comportaient pas de données manquantes, nous avons alors continué notre analyse sans modification.

Nous avons ensuite décidé d'étudier les statistiques des valeurs prises par les différents champs afin de relever si certaines étaient à éliminer.

	Duration	Forwarding_Status	Service_Type	Packets	Bytes
count	10000.000000	10000.0	10000.000000	10000.000000	1.000000e+04
mean	1.267133	0.0	1.967500	17.802200	1.305524e+04
std	6.917004	0.0	12.731742	784.207307	9.818564e+05
min	0.000000	0.0	0.000000	1.000000	2.900000e+01
25%	0.000000	0.0	0.000000	1.000000	8.600000e+01
50%	0.148000	0.0	0.000000	3.000000	2.255000e+02
75%	0.789000	0.0	0.000000	7.000000	9.350000e+02
max	261.412000	0.0	208.000000	68650.000000	9.814513e+07

FIGURE 2.1 – Statistiques des variables numériques pour 100 000 échanges des données d'entraînement

En observant les statistiques des variables numériques, nous avons remarqué que la colonne *Forwarding status* comportait uniquement des zéros. Nous avons décidé de retirer cette colonne pour la suite de notre analyse car cette information manquait de pertinence pour la détermination de la nature de l'échange.

On observe également que les colonnes ne comportent pas de valeurs négatives, ce qui aurait été considéré comme des valeurs aberrantes sachant que chaque variable numérique doit, pour ces données, prendre une valeur strictement positive. Cependant, on remarque que pour les paquets et les octets échangés, ainsi que la durée de l'échange qui est liée, l'écart des valeurs prises est très grand, alors que la majorité des échanges semble être destinée à de faibles valeurs. Nous avons décidé de conserver ces données extrêmes pour la suite de notre analyse car elles pouvaient se retrouver dans nos données de test.

	Date	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Label
count	10000	10000	10000	10000	10000	10000	10000	10000
unique	103	1704	2830	4435	4775	4	19	2
top	2016-06-01 00:05:16	42.219.156.211	42.219.156.211	53	53	TCP	.A....	background
freq	1182	1161	940	1947	1447	6415	3859	9600

FIGURE 2.2 – Statistiques des variables catégoriques pour 100 000 échanges des données d'entraînement

Pour les statistiques des variables catégoriques, il est intéressant de remarquer que pour les IPs et les Ports, les données ne sont pas toutes uniques et apparaissent à de multiples reprises. C'est pourquoi nous intégrerons ces données lors de la classification car cela permettra de relever certains patterns dans les échanges réalisés en fonction d'une attaque ou d'un échange normal.

2.2.2 Feature engineering

Pour que notre détection d'anomalie soit plus efficace, nous avons transformé certaines colonnes en une information plus cohérente avec notre étude.

Premièrement, l'information de la date complète d'un flux qui nous semblait alors assez importante pour notre analyse. Nous avons tout d'abord décidé de la décomposer en plusieurs colonnes qui contiennent les années, mois, jours, heures et secondes pour pouvoir les analyser. Malheureusement, nos données d'entraînement ne se déroulent que sur une seule semaine, la majorité de ces colonnes ne nous sont donc pas utiles.

Cependant, nous avons réussi à trouver une information qui nous paraissait importante à étudier et qui est le temps écoulé depuis le début de la journée,

pour déterminer des plages horaires auxquelles les attaques ont plus ou moins lieu. Nous avons ainsi rajouté une colonne, *timeDelta*, à nos jeux de données afin d'avoir le temps depuis le début de la journée (0h00) en seconde, obtenu à partir de la date, et ainsi supprimer les informations de cette dernière.

Pour réduire la dimension de notre jeu de données, nous avons également transformer les colonnes *Duration*, *Packets* et *Bytes* en une seule colonne *transfered_ratio*, qui correspond au nombre d'octets par paquets par minutes. Cela permettait de conserver l'essentiel de l'information tout en réduisant le nombre de variables à entraîner pour notre modèle de classification.

Après cette phase de pré-traitement, nous avions comme variables numériques :

- Type de service
- Nombre de octets par paquets par minute

D'autre part, nous avions comme variables catégoriques :

- IP source
- IP de destination
- Port source
- Port de destination
- Protocole utilisé
- Flag utilisé
- Label

Pour pouvoir entraîner nos algorithmes de classification, nous devons transformer les valeurs des variables catégoriques en valeur numérique, qui ne sera pas interprété d'une façon qu'on puisse la comparer à une autre, par exemple que le port 5000 est supérieur au port 88. Pour cela, avant d'entraîner nos modèles, nous avons appliqué un *One Hot Encoder* sur nos variables catégoriques, permettant de transformer chaque valeur présente dans une variable catégorique en une nouvelle colonne comportant l'information si elle est présente (=1) ou non (=0). Pour le cas de la colonne *Label*, nous avons simplement transformer la colonne en modifiant les échanges normaux (*background*) par un 0 et les attaques par un 1.

2.3 Analyse statistique et visualisation

Au cours du pré-traitement de nos données, nous nous sommes intéressé à la corrélation entre nos différentes variables, afin de déterminer quelles allaient être les données les plus pertinentes dans la détection d'anomalie.

Pour une meilleure compréhension de nos variables, nous avons décidé de les comparer avec la colonne cible *Label* et d'analyser leur relation.

On a alors pu observer plusieurs comportement :

- Le trafic de certains Port source et de certains Port de destination ne sont que des attaques, idem pour les IP de destination et les IP sources. Pour d'autres ce trafic est uniquement normal.

	Source_IP	Label
1292	50.33.218.194	1.0
839	222.11.231.252	1.0
695	213.30.31.179	1.0
836	221.17.125.37	1.0
1293	50.33.218.239	1.0

FIGURE 2.3 – Moyenne des *Label* pour certaines IP sources, ici des IP uniquement attaqué

- Dans notre jeu de donnée, seulement deux protocoles subissent des attaquent, *UDP* et *TCP*.

	Protocol	Label
3	UDP	0.042237
2	TCP	0.039283
0	ESP	0.000000
1	ICMP	0.000000

FIGURE 2.4 – Moyenne des *Label* pour les protocoles

- Les différents *Flag* ou *Service_Type* utilisé lors de la connexion semblent également avoir un impact sur la quantité d'attaques.

	Flag	Label
4	.A...F	0.257732
9	.A.RS.	0.181818
16	.APR.F	0.142857

FIGURE 2.5 – Moyenne des *Label* pour les *Flag*

Suite à ces analyses comparatives en deux dimensions avec le *Label*, nous avons pu réaliser différents graphiques en trois dimensions pour valider ou non certaines hypothèses sur la relation en des données et leur label. On pourrait notamment savoir si lorsque la *Duration* est très grande, alors celle-ci sont le plus souvent des attaques ou si le *timeDelta* a un impact sur le nombre d'attaques.

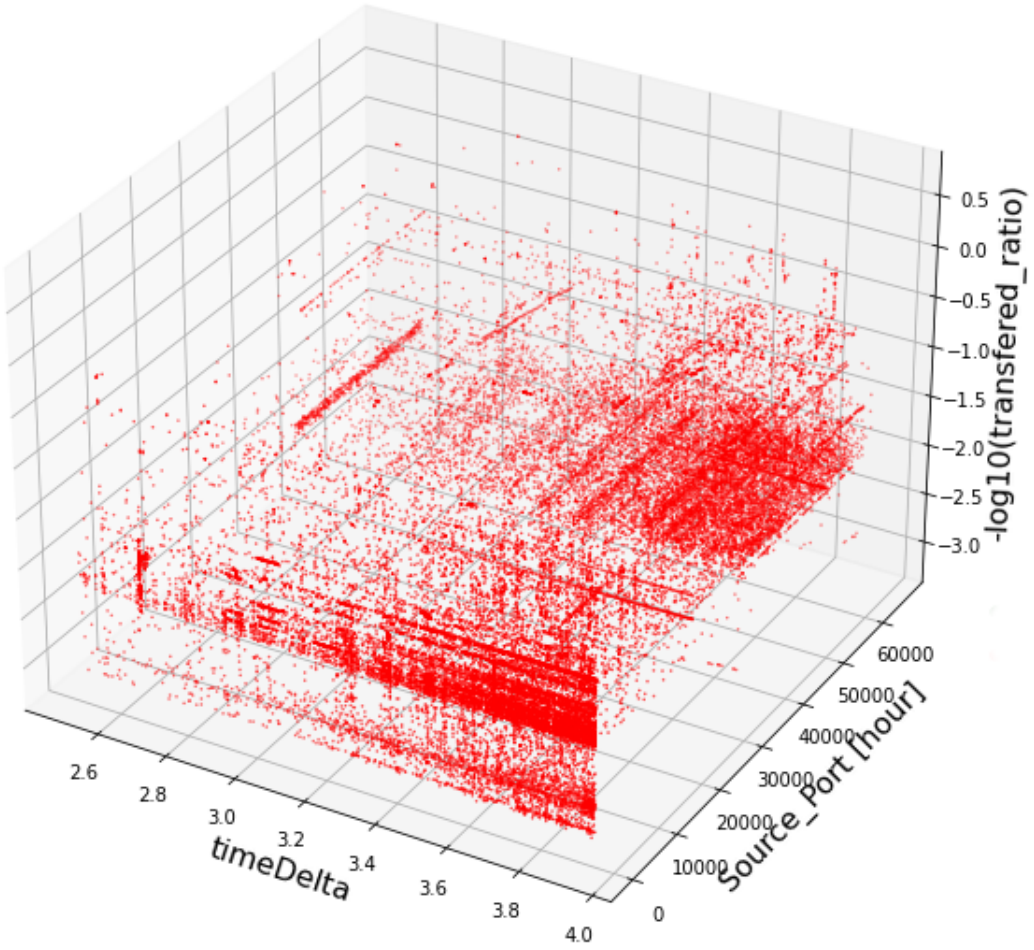

FIGURE 2.6 – Graphique 3D entre notre *timeDelta*, *Source_port* et *transfered_ratio*

On peut voir sur ce premier graphique que les attaques sont assez réparties au niveau du *timeDelta* notamment mais qu'elles sont plutôt groupés pour les deux autres variables. On observe notamment que les attaques sont un peu plus souvent présentes lorsque le *Source_port* est faible ou élevé, ou que les attaques surviennent aussi un peu plus autour d'un certains *transfered_ratio*.

Il est difficile d'avoir des heuristiques nettes a ce stade, mais cela permet d'y voir plus clair et de formuler des hypothèses sur nos données.

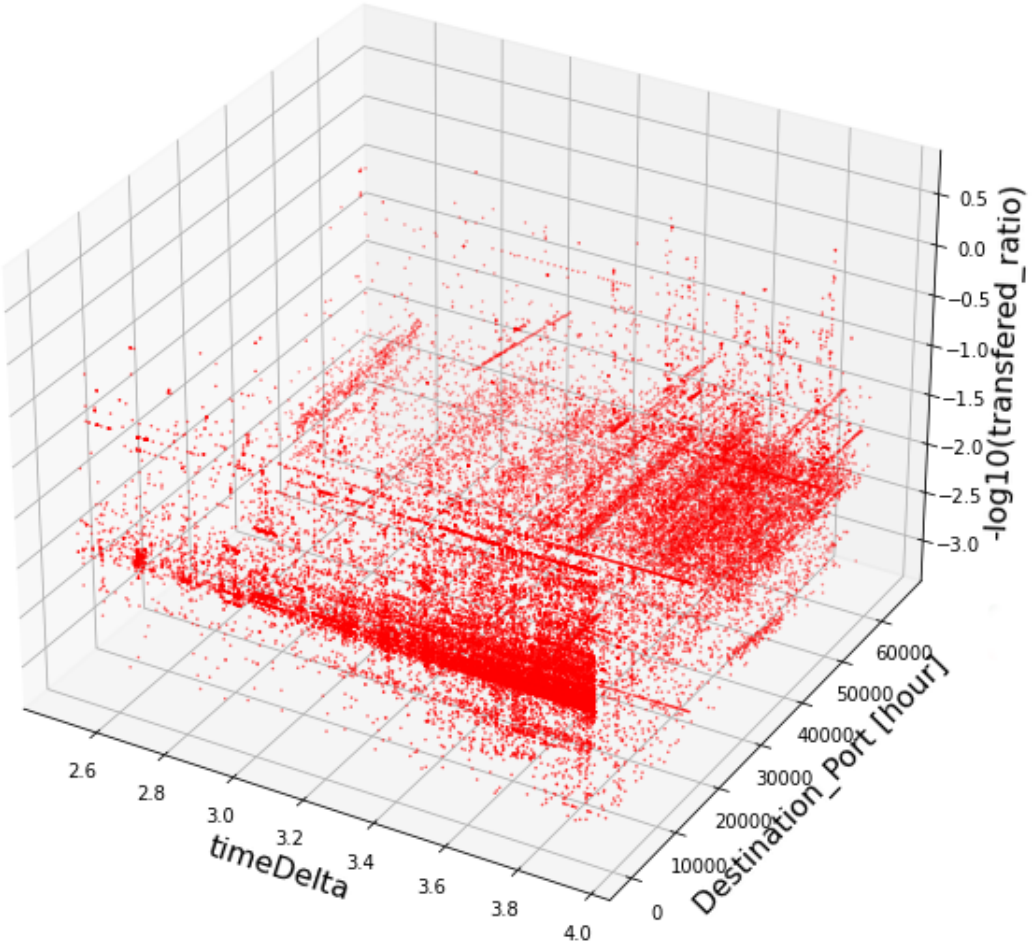

FIGURE 2.7 – Graphique 3D entre notre *timeDelta*, *Destination_port* et *transfered_ratio*

Sur ce second graphique, il est possible de vérifier certaines conclusions faites depuis le premier graphique. On observe la même répartition au niveau du *timeDelta* notamment mais aussi que les attaques sont un peu plus souvent présentes lorsque le *Destination_port* est faible ou élevé, et que les attaques surviennent aussi un peu plus autour d'un certains *transfered_ratio*.

Ensuite nous avons pu réaliser le graphe de corrélation entre nos différentes variables numériques. Celui-ci nous permet d'analyser de façon plus profonde les relations entre toutes nos variables et d'améliorer nos conclusion sur les évènements de cybersécurité dans les données.

Le voici ci-dessous :

	Service_Type	Label	timeDelta	transfered_ratio
Service_Type	1.000000	0.076680	0.080787	-0.047649
Label	0.076680	1.000000	0.850875	0.030599
timeDelta	0.080787	0.850875	1.000000	0.012505
transfered_ratio	-0.047649	0.030599	0.012505	1.000000

FIGURE 2.8 – Graphe de corrélation de nos variables numériques

Il est donc possible de conclure plusieurs éléments après analyse de ce graphe. On peut notamment voir que la variable *timeDelta* semble extrêmement liée à notre variable *Label*, ceci est sûrement du au fait que l'on doit se restreindre à une petite partie du dataset et donc à seulement quelques heures d'analyse de netflow pour pourvoir faire tourner nos algorithmes assez rapidement.

On peut voir également que les autres variables semblent un peu moins liées à notre variable *Label*. En effet pour *Service_Type* et *transfered_ratio*, la corrélation est nettement moins évidente que pour la variable précédente.

Il ne faut pas oublier non plus toutes nos autres variables catégoriques qui restent : *IP source, IP destination, Port source, Port destination, Type de Service, Protocole, Flag*, qui auront aussi une grande importance dans nos modèles.

3 Analyse via Machine Learning

3.1 Analyse supervisée

3.1.1 XGBoost

Pour cette première partie de Machine Learning nous avons décidé de nous pencher principalement sur XGBoost et sur quelques autres algorithmes qui sont exposés par Sci-kit Learn.

XGBoost est l'un des algorithmes présentés dans le cours, c'est un algorithme très optimisé de classification qui utilise notamment les mécaniques de Tree Boosting. Cet algorithme souvent prisé des compétitions de Machine Learning est extrêmement efficace, son principe est de combiner les résultats d'un ensemble de modèles plus simple et plus faibles, ici un ensemble de DecisionTree, afin de fournir une meilleure prédiction.

Nous avons choisi cette algorithme car il présente plusieurs caractéristiques qui nous paraissent intéressante pour notre projet.

Tout d'abord il prend bien en charge les données qui peuvent être déséquilibrées et assez vaste comme les nôtres. C'est aussi un algorithme rapide car il utilise la parallélisation et qui supporte la régularisation (Lasso et Ridge Regression intégrées) pour éviter un overfitting du modèle. Il permet également un réglage et un meilleur équilibrage des poids de la classe positive par rapport aux poids de la classe négative.

3.1.2 Autres algorithmes

Pour pouvoir nous faire une idée précise de la performance de notre algorithme supervisé, nous avons décidé de tester de façon rapide plusieurs autres algorithme de Machine Learning supervisé pour pouvoir les comparer entre eux et les comparer avec XGBoost.

Voici les différentes algorithmes que nous avons testés :

- SVC (Support Vector Classifier) et LinearSVC : Ces deux implémentation sont basées sur la libSVM. Une SVM construit un hyper-plan ou un ensemble d'hyper-plans dans un espace dimensionnel élevé ou infini, qui peut être utilisé pour notamment pour de la classification.

- Logistic Regression : La régression logistique est un modèle statistique permettant d'étudier les relations entre nos variables qualitatives et une variable qualitative comme notre *Label*.

- Random Forest : Une Random Forest est un estimateur qui adapte un certain nombre de classificateurs d'arbre de décision sur divers sous-échantillons de l'ensemble de données et utilise la moyenne pour améliorer la précision de la prédiction et contrôler l'overfitting.

- Perceptron : Le perceptron est un neurone artificiel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé et donc de réaliser une classification.

- SGD : Cet estimateur implémente des modèles linéaires régularisés avec apprentissage de descente de gradient stochastique. Le gradient de la perte est estimé pour chaque échantillon à la fois et le modèle est mis à jour en cours de route avec le taux d'apprentissage.

- Decision Tree : Les arbres de décision sont une méthode d'apprentissage supervisé non paramétrique. L'objectif est de créer un modèle qui prédit la valeur d'une variable cible en apprenant des règles de décision simples déduites des caractéristiques des données.

3.2 Analyse non supervisée

Cette section traite des algorithmes d'analyse de Machine Learning non-supervisé que nous avons implémenté dans notre solution.

3.2.1 IsolationForest

Dans un premier temps, nous avons utilisé l'algorithme de l'IsolationForest. Cet algorithme non supervisé de machine learning permet de détecter des anomalies dans un jeu de données. Il isole les données atypiques, autrement dit celles qui sont trop différentes de la plupart des autres données.

Cet algorithme calcule, pour chaque donnée du jeu, un score d'anomalie, c'est à dire une mesure qui reflète à quel point la donnée en question est atypique. Afin de calculer ce score, l'algorithme isole la donnée en question de manière récursive : il choisit un descripteur et un "seuil de coupure" au hasard, puis il évalue si cela permet d'isoler la donnée en question ; si tel est le cas, l'algorithme s'arrête, sinon il choisit un autre descripteur et un autre point de coupure au hasard, et ainsi de suite jusqu'à ce que la donnée soit isolée du reste.

IsolationForest est intéressant pour nos données pour plusieurs raisons. Il est plus rapide que des méthodes de clustering ou de calcul de distances, il est aussi efficace et conçus pour des problèmes qui comportent beaucoup de dimensions et pour de l'analyse en temps réel de séries chronologiques.

3.2.2 LOF

Dans un second temps nous avons choisit d'étudier les anomalies avec l'algorithme appelé Local Outlier Factor, ou LOF. Tout comme l'IsolationForest il permet d'analyser les données afin de détecter les anomalies présentes. Il isole donc aussi les données qui sortent du lots.

Un point est défini comme un *local outlier* (ou exception locale) à partir de son voisinage. L'algorithme de LOF identifie les outliers en regardant la densité du voisinage de la donnée. La distance entre les voisins et la donnée de départs est ce qui est utilisé pour calculé la densité. Mais cet méthode compare aussi la densité locale d'un objet avec la densité de ses voisins. Les données qui ont la plus petite densité par rapport à ses voisins est considéré comme une exception locale d'après l'algorithme de LOF.

LOF est très utilisé car il a pour avantage d'avoir une approche plus précise de part son approche locale et non globale. Il est capable d'identifier des extrêmes dans un jeu de données qui ne serait pas des *outliers* à d'autres endroits de celui-ci. Dans notre cas c'est très intéressant car nous voulions nous remettre dans le contexte des attaques, c'est-à-dire comparer un échange à ses voisins afin de voir s'il est suspect ou non.
LOF possède aussi des inconvénients. Il peut être compliqué à interpréter car il y a pas de seuil qui définisse ce qui est une exception ou non, tout dépend du voisinage de la donnée, donc du problème.

4 Analyse des résultats

4.1 Données ordonnés ou aléatoire ?

Dans l'optique d'améliorer nos modèles et techniques d'analyse nous avons essayé de trouver un échantillon qui contienne le plus d'informations possible. Le fichier contenant le jeu de données faisant plus de 680 millions de lignes il nous était impossible d'importer celle-ci dans son entièreté. C'est pourquoi nous avons du prendre seulement une partie pour pouvoir l'analyser. Cela a causé beaucoup de réflexion et tests pour savoir ce qui était le mieux pour appliquer nos méthodes d'analyses.

Au début nous avons choisit d'extraire les données dans l'ordre du fichier. Nous extrayions 10 000 lignes. Ce qui est peu en considérant la taille de la base de données et peu représentatif de l'entièreté du fichier. Nous avons donc augmenté la taille de l'échantillon. Nous avons essayé de prendre des lignes de façon aléatoire tout en laissant les lignes dans l'ordre temporel mais les performances de détection était moins bonne, nous sommes donc rester sur l'ordre initial des lignes sans en sauter. Cette méthode est celle qui nous donnait les meilleurs résultats.

Dans notre recherche d'améliorations des performances de nos modèles et de variété des informations analysées, nous avons essayé une autre méthode pour avoir des données plus variées. Le fichier que nous lisons en entrée était rangé de façon aléatoire, en utilisant la commande bash shuf(1) pour faire des permutations de lignes arbitraire.

4.2 Performances

Pour comparer nos algorithmes de modèles de classification supervisé, nous avons comparer leur f1_score afin de voir lequel s'adaptait le mieux aux données.

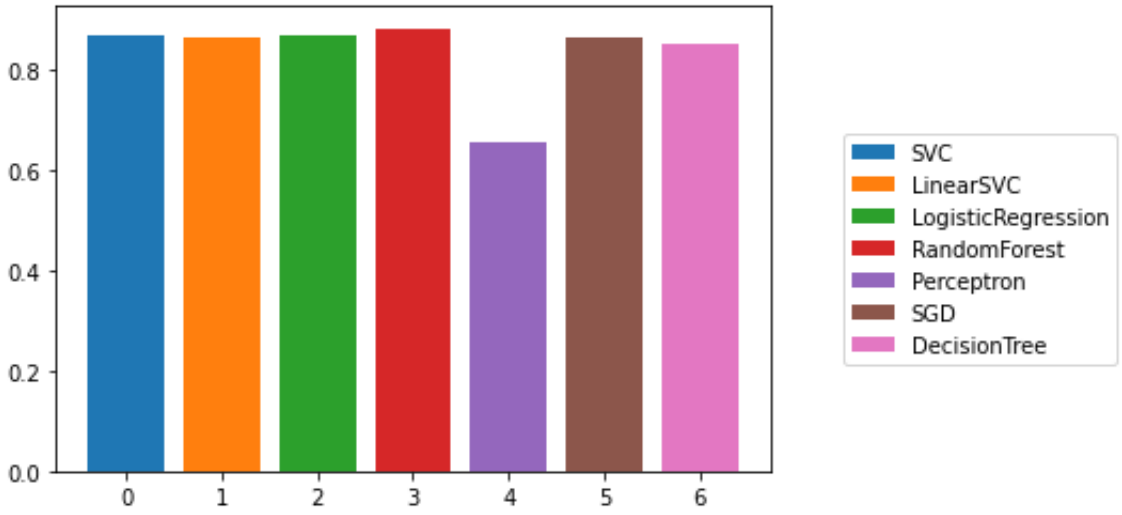

FIGURE 4.1 – F1_score des algorithmes de modèles supervisée

On observe que les modèles avec arbre, notamment le *RandomForest*, sont ceux qui obtiennent les meilleurs résultats.

Pour notre analyse supervisée avec XGBoost, nous avons obtenus sur un échantillon de donnée de 10 000 éléments de test, avec 9000 échanges normaux et 1000 anomalies, la matrice de confusion suivante :

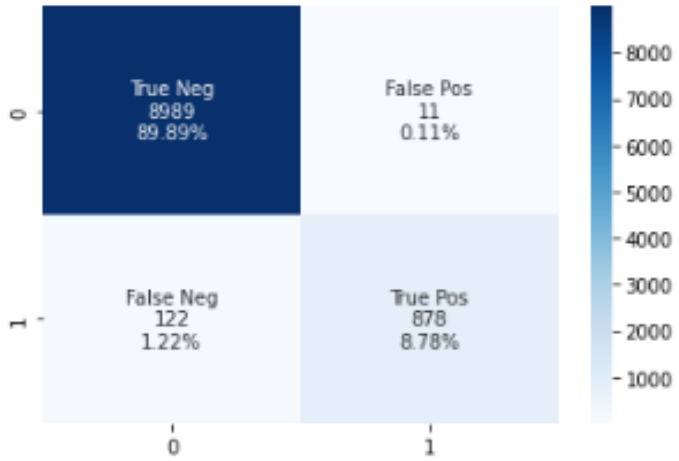

FIGURE 4.2 – Matrice de confusion pour l'analyse supervisée avec l'algorithme *XGBoost*

Les résultats obtenus avec *XGBoost* sont excellent, avec plus de 94% de F1_score obtenus pour un jeu de donnée test avec 1000 anomalies sur 10000 échanges réseau.

En comparaison, les algorithmes d'analyse non-supervisée ont été beaucoup moins performant dans la classification des échanges. En effet, on obtient comme matrice de confusion :

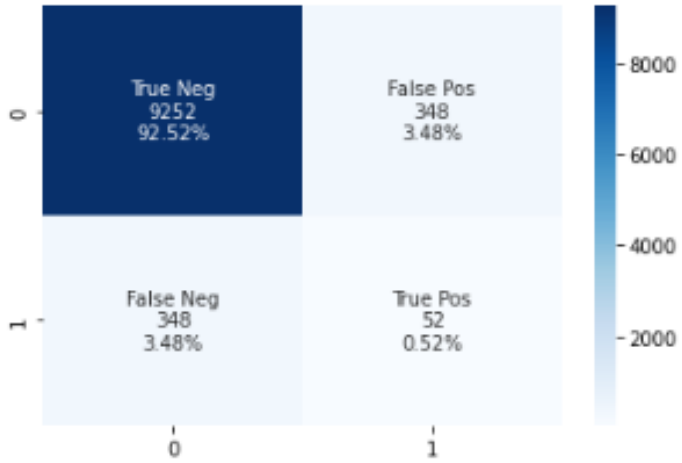

FIGURE 4.3 – Matrice de confusion pour l'analyse non-supervisée pour l'algorithme *Isolation Forest*

On observe que l'algorithme Isolation Forest détecte très difficilement les attaques, ici environ 12% des attaques réelles, ce qui démontre que l'utilisation d'un algorithme non-supervisée n'est pas approprié dans ce type de configuration.

On peut confirmer ces résultats en observant également la matrice de confusion du *Local Outlier Factor*, qui comportent des résultats encore moins bon :

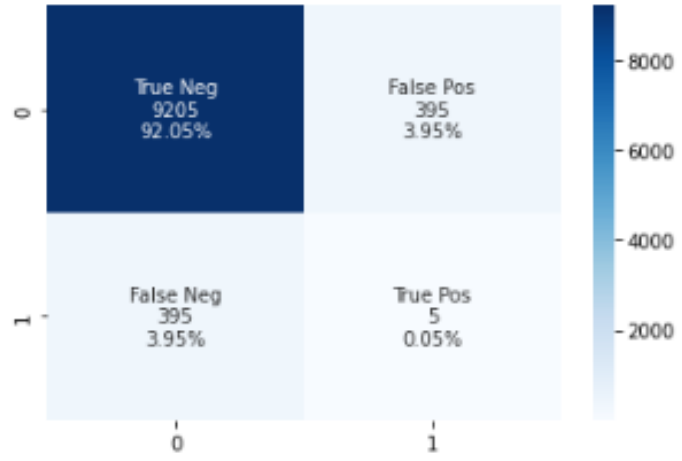

FIGURE 4.4 – Matrice de confusion pour l'analyse non-supervisée pour l'algorithme *Local Outlier Factor*

5 Conclusion

Tout d'abord pour nos données, un des points majeures de notre réflexion a été sur la taille de l'échantillon choisi et sur les données qu'on prenait. Nous avons finalement choisit de travailler sur un jeu de données de 100000 lignes et en ne prenant pas de lignes aléatoire car on perdait le contexte des échanges observés.

Dans un second temps nous avons analyser le jeu de données afin de déterminé les informations qui n'étaient pas nécessaires à la détection d'anomalies et qu'on pouvait enlevé. Nous avons aussi transformé certaines données afin de pouvoir au mieux les utilisés dans nos analyses avec les modèles que nous avons choisit.

Pour réaliser nos analyses nous avons eu plusieurs approches, utiliser des modèles d'analyses supervisées et non supervisées. Et nous avons pu les tester et les comparer afin de déterminer lequel parmi tout ceux testés est le meilleur pour détecter les échanges frauduleux et à quel point ce résultat est fiable.

Avec les résultats que nous avons obtenu on en conclue que le modèle le plus performant pour détecter les anomalies et les échanges frauduleux est XGBoost car lorsqu'on compare les performances et les matrices de confusion, c'est celui qui est le meilleur. Nos analyses nous ont permis d'observer que les algorithmes de classifications sont les plus adaptés dans notre situation.