République Algérienne Démocratique Et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida 1
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études
Pour l'obtention du diplôme de Master

Domaine : MI

Filière : Informatique

Spécialité : Système Informatique et Réseaux

Performances des détections DDos en fonction des traitements appliqués au Dataset.

Réalisé par :
Hattou Feriel
Mekki Nadjet

Promoteur :
Mr. Mohamed Benyahia

Jury :
Mme Ghebghoub
Mr Chettat

Juin 2025

Remerciements

Ce travail n'aurait pu voir le jour sans l'appui et l'engagement de plusieurs personnes à qui nous tenons à exprimer notre plus profonde gratitude.

Nous remercions tout d'abord Dieu le Tout-Puissant, pour la force, la patience et la persévérance qu'Il nous a accordées tout au long de cette aventure.

Nos remerciements les plus sincères vont à notre promoteur, Monsieur Mohamed Benyahia, pour sa disponibilité, ses conseils éclairés et son accompagnement rigoureux. Son expertise et son implication ont joué un rôle central dans la réalisation de ce mémoire.

Nous adressons également notre reconnaissance aux membres du jury, qui ont accepté d'évaluer notre travail. Nous leur sommes reconnaissants pour le temps qu'ils y consacrent ainsi que pour les remarques et suggestions qu'ils ne manqueront pas d'apporter.

Enfin, nous souhaitons remercier toutes celles et ceux qui, de près ou de loin, nous ont soutenus et encouragés au fil de ce projet. Leur aide précieuse et leur présence bienveillante ont été d'un grand soutien.

Dédicaces

Je dédie ce travail à mon père, dont la sagesse, les encouragements et les sacrifices discrets ont toujours été une source de force et d'inspiration pour moi.

À ma mère, pour son amour inépuisable, ses prières silencieuses, sa tendresse et son soutien constant, sans lesquels je n'aurais jamais pu avancer avec autant de sérénité.

À mon mari, pour sa patience, sa compréhension et sa présence rassurante durant les moments les plus exigeants de ce parcours. Ton soutien a été un pilier essentiel dans cette aventure.

À ma chère binôme et amie Nadjet, avec qui j'ai partagé chaque étape de ce travail avec complicité, sérieux et détermination.

Et enfin, je dédie ce travail à tous mes enseignants, qui ont su, par leur engagement, leur disponibilité et la qualité de leur enseignement, éveiller en moi la curiosité, la rigueur et l'envie d'apprendre. Leur accompagnement a joué un rôle fondamental dans la construction de mon parcours académique et personnel.

Feriel

Hamdoulilah pour la force, la patience et la persévérance que Dieu m'a accordées tout au long de ce parcours.

Je dédie ce travail, avec tout mon amour et ma profonde gratitude :

À ma chère maman Zahia, pour son amour inconditionnel, ses prières silencieuses et sa tendresse qui m'ont portée chaque jour.

À mon père Mohamed, pour sa sagesse, ses conseils et son soutien constant.

À ma sœur Zoubida et à son fils Adem, source de joie et de lumière dans ma vie.

À mes frères Ismail et Islam, pour leur présence, leur humour et leur appui indéfectible.

À mes amies Imène, Meriem et Ikram, pour leur amitié fidèle, leur écoute et leur réconfort tout au long de cette aventure.

Un grand merci à mon binôme Fériel pour son sérieux, sa collaboration et sa persévérance.

Enfin, gratitude à toutes celles et ceux qui, de près ou de loin, m'ont soutenue, encouragée ou inspirée.

À vous tous, un immense merci du fond du cœur.

Nadjet

Résumé

Dans un contexte marqué par la multiplication des attaques par déni de service distribué (DDoS), la détection automatisée de ces menaces est devenue un enjeu majeur pour la cybersécurité. Ce mémoire explore l'impact des différentes étapes de prétraitement des données sur les performances de plusieurs modèles d'apprentissage automatique dans la détection des attaques DDoS. En s'appuyant sur le jeu de données CIC-DDoS2019, quatre versions du dataset ont été construites en appliquant divers niveaux de nettoyage, normalisation, réduction de dimension et équilibrage des classes. Cinq algorithmes classiques (Random Forest, SVM, k-NN, Naïve Bayes, XGBoost) ont été entraînés et évalués selon plusieurs métriques de performance (précision, rappel, F1-score, AUC-ROC, temps d'exécution, taux d'erreur). Les résultats mettent en évidence l'importance cruciale du prétraitement dans l'amélioration de la détection et la généralisation des modèles, tout en offrant une analyse critique de leurs comportements face aux différentes configurations de données. Ce travail vise ainsi à orienter les futures recherches vers des stratégies de traitement plus efficaces et adaptées aux systèmes de détection d'intrusions en environnement réel.

Mots-clés : attaques DDoS, détection d'intrusion, apprentissage automatique, prétraitement des données, classification, jeu de données CIC-DDoS2019, performance des modèles.

Abstract

In a context where Distributed Denial of Service (DDoS) attacks are increasingly frequent and sophisticated, the automated detection of such threats has become a major challenge in cybersecurity. This thesis investigates the impact of different data preprocessing strategies on the performance of several machine learning models for DDoS attack detection. Using the CIC-DDoS2019 dataset, four dataset versions were created by applying various levels of cleaning, normalization, dimensionality reduction, and class balancing. Five classical algorithms (Random Forest, SVM, k-NN, Naïve Bayes, XGBoost) were trained and evaluated using multiple performance metrics (accuracy, recall, F1-score, AUC-ROC, execution time, and error rates). The results highlight the critical role of data preprocessing in improving both detection efficiency and model generalization. This work aims to guide future research towards more effective and tailored data preparation strategies for real-world intrusion detection systems.

Keywords: DDoS attacks, intrusion detection, machine learning, data preprocessing, classification, CIC-DDoS2019 dataset, model performance.

ملخص

في ظل تزايد وتطور هجمات حجب الخدمة الموزعة (DDoS)، أصبحت عملية اكتشاف هذه التهديدات تمثل تحدياً رئيسياً في مجال الأمن السيبراني. يهدف هذا العمل إلى دراسة تأثير استراتيجيات المعالجة المسبقة للبيانات على أداء نماذج تعلم الآلة في كشف هجمات DDoS. تم استخدام مجموعة البيانات CIC-DDoS2019، حيث أُنشئت أربع نسخ مختلفة من البيانات من خلال تطبيق مستويات متعددة من التنظيف، والتطبيع، وتقليل الأبعاد، وموازنة الفئات. تم تدريب خمسة خوارزميات تقليدية (Random Forest، SVM، k-NN، Naïve Bayes، XGBoost) وتقييمها باستخدام مجموعة متنوعة من مؤشرات الأداء مثل الدقة، الاسترجاع، F1-score، AUC-ROC، زمن التنفيذ، ونسب الخطأ. أظهرت النتائج أهمية مرحلة المعالجة المسبقة في تحسين فعالية الكشف وقدرة النماذج على التعميم. يهدف هذا البحث إلى توجيه الدراسات المستقبلية نحو استراتيجيات تحضير بيانات أكثر كفاءة وملاءمة لتطبيقات كشف التسلل في البيئات الواقعية.

الكلمات المفتاحية : هجمات حجب الخدمة الموزعة (DDoS)، كشف التسلل، تعلم الآلة، المعالجة المسبقة للبيانات، التصنيف، مجموعة بيانات CIC-DDoS2019، أداء النماذج.

Table des matières

Liste des Figures

Liste des Équations

Liste des Tableaux

Liste des Codes

Liste des Abréviations

ABOD: Angle-Based Outlier Degree
ACC: Accuracy
ACP/PCA : Analyse en Composantes Principales
ADA/ADASYN: ADAptive SYNthetic Sampling
ACK: Acknowledgment
BGWO: Binary Grey Wolf Optimizer
CIC-DDoS2019: Canadian Ins titute for Cybersecurity Distributed Denial of Service 2019
CNN: Convolutional Neural Network
CSV: Comma-Separated Values
CSE-CIC-IDS2018: Communications Security Establishment - Canadian Institute for Cybersecurity - Intrusion Detection System 2018
DBSCAN: Density-Based Spatial Clustering of Applications with Noise
DDoS: Distributed Denial of Service
DL: Deep Learning
DNS: Domain Name System
DOS: Denial of Service
DDoS: Distributed denial of service
DrDos: Distributed Reflective Denial of Service
FNR: False Negative Rate
FP: False Positive
FPR: False Positive Rate
FAR: False Acceptance Rate
GAN: Generative Adversarial Network
GBDT: Gradient Boosting Decision Trees
GNN: Graph Neural Network
GPU: Graphics Processing Unit
GUI: Graphical User Interface
HLS: Hybrid Learning System
HTTP: HyperText Transfer Protocol
IA: Intelligence Artificielle
ICMP: Internet Control Message Protocol
IDS: Intrusion Detection System
IHT: Instance Hardness Threshold
INF: infinity
IP: Internet Protocol
IPS: Intrusion Prevention System
IoV: Internet of Vehicles
KNN: k-Nearest Neighbors
LDA: Linear Discriminant Analysis
LOF: Local Outlier Factor
LSA: Link State Advertisement
MLP: Multi-Layer Perceptron
ML: Machine Learning
MMH: Maximum Margin Hyperplane
NaN: Not a Number
NB: Naïve Bayes
NIDS : Network Intrusion Detection System
NLP: Natural Language Processing
OSFP: Open Shortest Path First
pkl: pickle
Q-Learning: Quality-learning
RAM: Random Access Memory
RF: Random Forest
RFE: Recursive Feature Elimination
ROS: Random Over-Sampling
RNN: Recurrent Neural Network
RTO: Retransmission Timeout
RTT: Round-Trip Time
RUS: Random Under-Sampling
SDN: Software-Defined Networking
SDD: Solid-State Drive
SSS: Small Sample Size
SMTP: Simple Mail Transfer Protocol
SMOTE: Synthetic Minority Over-sampling Technique
SS: Small Sample Size
Std: Standard Deviation
SVM: Support Vector Machine
SYN: Synchronize
TCP: Transmission Control Protocol
TNR: True Negative Rate
TN: True Negative
TPU: Tensor Processing Unit
TP: True Positive
TPR: True Positive Rate
UDP : User Datagram Protocol
XGB/XGBoost : Extreme Gradient Boosting

Introduction Générale

Contexte du travail :

Dans un monde de plus en plus interconnecté, les services en ligne sont devenus essentiels au fonctionnement des entreprises, des institutions et de la société en général. Cette dépendance accrue s'accompagne d'un risque croissant : les attaques par déni de service distribué (DDoS). Ces attaques visent à rendre indisponibles des ressources critiques en saturant les réseaux de requêtes malveillantes. Leur fréquence et leur sophistication ne cessent d'augmenter, rendant leur détection toujours plus complexe.

Face à cette menace, les approches basées sur l'apprentissage automatique (machine learning) ont gagné en popularité, car elles permettent de repérer des schémas de trafic anormal de manière plus dynamique et automatisée. Cependant, la réussite de ces techniques repose sur la qualité et la préparation des données utilisées pour entraîner les modèles. C'est dans ce cadre que s'inscrit notre travail : à partir du jeu de données CIC-DDoS2019, nous explorons l'impact des différentes stratégies de prétraitement sur la performance des modèles de classification. L'objectif est d'apporter une meilleure compréhension de cette étape cruciale, souvent négligée mais déterminante pour une détection efficace des attaques DDoS.

Problématique :

La détection des attaques DDoS représente un véritable enjeu dans le domaine de la cybersécurité, à cause de l'accroissement continue de leur fréquence mais aussi de leur complexité. Pour y faire face, plusieurs outils et techniques issues de l'apprentissage automatique sont utilisés pour analyser le trafic réseau et détecter des comportements malveillants. Mais l'efficacité de ces modèles dépend largement des traitements appliqués aux ensembles de données qui leur sont fournies. Des étapes comme le nettoyage, la normalisation, l'équilibrage des classes ou la réduction de la dimensionnalité peuvent avoir un impact significatif et même décisif sur les résultats des modèles. Il est donc important d'évaluer de manière systématique l'adéquation entre les traitements mis en œuvre et les performances de détection.

À cet égard, une question principale se pose : comment les différentes stratégies de prétraitement influencent-elles les performances des modèles de machine learning dans la détection des attaques DDoS ?

Motivation :

Ce travail s'est développé à partir d'un constat relativement simple : le jeu de données CIC-DDoS2019 est largement utilisé pour la détection des attaques DDoS, mais sa bonne exploitation demande un fort travail de préparation. En effet, ces données comportent un déséquilibre, du bruit, des redondances, voire des valeurs manquantes, qui peuvent affecter les résultats des

modèles de machine learning. Or, il existe très peu d'études qui évaluent en profondeur l'impact de chacune des étapes de prétraitement sur la performance de ces modèles. Or, notre motivation est ainsi de tester différents traitements des données et de voir l'impact qu'ils ont sur le résultat des classifications afin d'orienter au mieux les futurs travaux de détection DDoS.

Organisation du mémoire :

Ce travail est structuré en cinq chapitres. Nous présentons ci-après l'objectif de chacun, un aperçu de leur contenu ainsi que les points clés à retenir.

- **Chapitre 1 :** Ce chapitre présente les attaques DDoS, leurs impacts, et l'importance de leur détection, tout en explorant le jeu de données CIC-DDoS2019 et les travaux antérieurs dans ce domaine. Il met en lumière les défis de classification du trafic et les approches de détection existantes.
- **Chapitre 2 :** Ce chapitre traite des étapes clés du prétraitement des données dans le cadre de la détection des attaques DDoS, telles que le nettoyage, la normalisation, la réduction de dimension et l'équilibrage des classes. Il souligne l'importance de cette phase pour améliorer la qualité des données et optimiser la performance des modèles de détection.
- **Chapitre 3 :** Ce chapitre présente les fondements du Machine Learning dans le contexte de la détection des attaques DDoS, en mettant l'accent sur cinq algorithmes classiques : Random Forest, SVM, k-NN, Naïve Bayes et XGBoost. Il vise à expliquer leur fonctionnement et leur utilité pour renforcer la cybersécurité.
- **Chapitre 4 :** Ce chapitre a présenté la préparation des données, l'entraînement et la prédiction de plusieurs modèles sur le jeu CICDDoS2019, selon quatre niveaux de prétraitement. Il établit un cadre expérimental rigoureux dont les résultats seront analysés dans le chapitre suivant.
- **Chapitre 5 :** Ce chapitre présente les résultats des prédictions des modèles entraînés précédemment, en les analysant à travers divers indicateurs de performance. Il offre une évaluation critique et comparative des algorithmes testés, tout en les confrontant à des travaux antérieurs pour en dégager des perspectives d'amélioration

Chapitre 1 : Introduction aux Attaques DDoS et à leur Détection

1.1. Introduction

Le présent chapitre traite des attaques DDoS, menace majeure pour la disponibilité des services en ligne. Après une définition et une présentation de leurs variantes et impacts, nous aborderons les défis de leur détection, notamment la classification du trafic réseau. Le jeu de données CIC-DDoS2019 utilisé sera ensuite présenté, ainsi qu'une synthèse des travaux antérieurs comparant jeux de données, modèles et prétraitements.

1.2. Définition et Impacts des Attaques DDoS

1.2.1. Qu'est-ce qu'une attaque DDoS ?

Une attaque par déni de service distribué (DDoS) a pour but d'occasionner une indisponibilité d'un service, d'un serveur ou d'un réseau en les inondant de trafic provenant d'un ensemble d'appareils infectés, appelé botnet. En envoyant simultanément un très grand nombre de requêtes au serveur de la cible, les attaquants consomment les ressources de cette cible, causant un ralentissement voire une interruption du service. [1]

1.2.2. Mécanismes de base des attaques par déni de service distribué

Comprendre les mécanismes de base des attaques DDoS est essentiel pour développer des stratégies de défense efficaces.

- **Attaques par inondation :** Les attaquants envoient de grandes quantités de trafic, telles que des demandes de connexion ou des paquets mal formés, dans le but d'épuiser la bande passante ou la puissance de traitement de la cible, provoquant ainsi un déni de service aux utilisateurs légitimes [1], [2].
- **Épuisement des ressources** : en surchargeant les ressources de la cible, telles que le processeur et la mémoire, les attaquants peuvent ralentir ou bloquer le système, le rendant ainsi inaccessible aux utilisateurs [3], [4].
- **Botnets :** Les attaques DDoS utilisent souvent un réseau d'ordinateurs infectés (botnets) pour lancer des attaques coordonnées, ce qui rend difficile la traçabilité de la source de l'attaque [3], [5].

1.2.3. Conséquences sur les réseaux et les services en ligne

Les attaques par déni de service distribué représentent un risque majeur pour la disponibilité et la fiabilité des réseaux et des services en ligne, provoquant des répercussions opérationnelles et financières graves [6]. Les conséquences majeures des attaques DDoS sont explicitées dans les parties qui suivent.

- **Impact économique :**
    - ***Perte de revenus :*** les entreprises peuvent subir des pertes financières importantes lors d'une attaque ; par exemple, les plateformes de commerce électronique peuvent perdre des milliers de dollars par minute en raison de l'indisponibilité du service [6].

- ***Coûts opérationnels augmentés :*** les organisations sont souvent confrontées à des coûts plus élevés liés aux efforts d'atténuation et à la reprise après les attaques [6].
- ***Atteinte à la réputation :*** les attaques fréquentes peuvent nuire à l'image d'une entreprise, entraîner une perte de confiance des clients et avoir des répercussions commerciales à long terme [6].

- **Interruption opérationnelle :**
  - ***Interruption des services :*** les attaques DDoS peuvent rendre les sites Web et les services en ligne inaccessibles, perturbant ainsi le fonctionnement normal et l'accès des utilisateurs [7], [8].
  - ***Épuisement des ressources :*** les attaques peuvent épuiser les ressources du réseau et empêcher l'accès au trafic légitime, ce qui complique la prestation des services [8], [2].

1.2.4. Exemples d'attaques DDoS récentes et leur impact économique

- **Services Cloud :** il a été démontré que les attaques DDoS dans les environnements cloud affectaient non seulement les services ciblés, mais également les serveurs virtuels co-hébergés et les fournisseurs de services cloud, entraînant une dégradation des performances et des pertes commerciales [9].
- **Echanges de cryptomonnaies :** Une étude sur la plateforme Bitfinex montre que, sur 17 attaques DDoS, 13 n'ont entraîné qu'une interruption temporaire, les volumes de trading étant rétablis le jour même. En revanche, deux attaques plus sévères ont provoqué des perturbations jusqu'à cinq jours, soulignant la vulnérabilité face à des offensives soutenues [10].

1.3. Typologies des Attaques DDoS

Les attaques par déni de service distribué (DDoS) se classent selon leur mécanisme d'attaque et leur cible privilégiée. Voici les principales catégories :

1.3.1. Attaques volumétriques

Ces attaques visent à consommer la bande passante de la cible, en utilisant souvent des méthodes telles que les inondations UDP et les attaques par amplification, qui exploitent le comportement de réponse de certains protocoles [8], [11].

- **ICMP Flood (Ping Flood) :** Un flood ICMP est une attaque qui consiste à saturer un système avec des requêtes écho ICMP (ping), l'amenant à épuiser ses ressources à répondre à toutes ces requêtes, ce qui dégrade le trafic légitime. Pour se protéger, les administrateurs peuvent définir un seuil de trafic ICMP atteignant lequel la protection est activée. Les attaques peuvent également être atténuées par filtres puis limitation des paquets ICMP, même si cela peut nuire à certaines opérations de diagnostic réseau [15].
- **UDP Flood :** envoie massivement des paquets vers des ports aléatoires, saturant les ressources réseau. Contre-mesures : filtrage, limitation du débit et détection d'anomalies[13].
- **DNS Amplification :** envoie des requêtes DNS falsifiées via UDP pour submerger la cible avec des réponses amplifiées. Prévention par restriction de récursivité, mises à jour et filtrage [14].

1.3.2. Attaques par épuisement des ressources

- **SYN Flood :** cette attaque exploite le 3-way handshake du protocole TCP en envoyant des paquets SYN avec des IP usurpées, saturant la table de connexions du serveur avec des connexions semi-ouvertes, empêchant les connexions légitimes [18]. Des contre-mesures incluent cookies SYN, backlog étendu et détection via IDS/IPS [19].
- **OSPF Flood :** cette attaque injecte de faux LSA dans le protocole OSPF pour perturber les tables de routage, causant détours, boucles ou redirections vers des nœuds compromis. Difficile à détecter, le trafic malveillant imite celui d'OSPF légitime.[17]
- **ACK Flood :** il s'agit d'une attaque DDoS par laquelle un serveur est inondé par des paquets ACK usurpés. Le serveur se surconsomme pour traiter des connexions inexistantes. On le détecte par un pic anormalement élevé de paquets ACK. La limitation de débit et le filtrage de type pare-feu ou IDS/IPS permettent d'en atténuer les effets. [18]

1.3.3. Attaques applicatives

- **Slowloris :** est une attaque vise à saturer le serveur Web par le biais d'un grand nombre de connexions HTTP ouvertes avec des requêtes incomplètes, envoyées très lentement. L'attaquant maintient les connexions ouvertes via l'en-tête Keep-Alive, ce qui empeche le traitement de nouvelles requêtes par le serveur [19]. Slowloris est discrète car elle consomme peu de bande passante et imite un trafic légitime. [20]
- **HTTP Flood (GET/POST Flood) :** est une attaque DDoS apparemment courante visant la couche applicative, consistant à simuler des requêtes HTTP GET ou POST légitimes dans l'objectif de surcharger un serveur web, ce qui rend sa détection plus complexe [21].
    - ***Attaque par requêtes HTTP GET :*** consiste à envoyer de nombreuses requêtes GET incomplètes ou lentes et à garder les connexions ouvertes afin de saturer le serveur[24].
    - ***Attaque par requêtes HTTP POST :*** envoie lent de requêtes POST pour bloquer les connexions et rendre le service indisponible, souvent via des botnets [21].
- **SMTP Flood :** cette attaque vise à saturer un serveur de messagerie via des requêtes SMTP malveillantes. Elle peut être détectée par l'analyse de l'entropie du RTT/RTO et du trafic réseau. Les approches en machine learning atteignent environ 75 % de détection. La défense repose sur des méthodes multicouches, listes noires ou contrôleurs SDN avec règles OpenFlow [25].

1.4. Importance de la Détection et Défis de Classification

- **Importance de la détection des attaques DDoS :**
    - ***Continuité des services :*** les attaques DDoS peuvent perturber les services, entraînant des pertes financières importantes et une atteinte à la réputation des organisations [23].
    - ***Renforcement de la sécurité :*** des mécanismes de détection efficaces améliorent la sécurité globale du réseau, permettant de réagir rapidement aux menaces [24].
    - ***Intégrité des données :*** La préservation de l'intégrité des données et des services est un enjeu majeur dans les systèmes cyberphysiques, justifiant la mise en place de protections contre les attaques DDoS [25].

- **Difficultés liées à la classification du trafic réseau :** Le volume élevé, la vitesse du trafic et l'évolution des attaques rendent la distinction entre trafic légitime et malveillant complexe. Le chiffrement complique l'analyse des paquets, nécessitant des approches basées sur les métadonnées. L'ambiguïté de certains comportements et le manque de données étiquetées constituent également des défis majeurs [26].

1.5. Présentation du Dataset utilisé en recherche

- **CIC-DDoS2019 Caractéristiques et avantages :** Produit par l'Institut canadien pour la cybersécurité, ce jeu de données est vaste. Avec plus de 50 millions de lignes et 88 caractéristiques, il englobe tout un éventail d'attaques DDoS et des échanges bénins [30]. Il se distingue par sa richesse, sa diversité d'attaques (TCP/UDP) et son réalisme, idéal pour l'entraînement de modèles de détection. La qualité d'étiquetage aide à la bonne analyse du trafic, donc à l'efficacité des algorithmes d'apprentissage [30].
- **Critères de sélection d'un dataset pour la recherche en DDoS :**
  Il est essentiel de choisir un bon dataset pour faire de la recherche sur la détection d'attaques DDoS. Celui-ci doit être riche et varié, intégrant du trafic légitime et des attaques différentes. Il doit également simuler un environnement réseau réaliste (protocoles, topologies, volumes), être régulièrement mis à jour avec les menaces, garantir la confidentialité par l'anonymisation, et être de bonne qualité, avec un bon étiquetage, des caractéristiques variées et des données publiques [28].

1.6. Analyse des Travaux Existants et Discussion

1.6.1. Travail de 'Stefanos Kiourkoulis, 2020'

Dans le cadre de son mémoire de master réalisé en 2020, Stefanos Kiourkoulis présente une évaluation de l'efficacité de quatre ensembles de données ouvertes pour la détection DDoS via six algorithmes d'apprentissage supervisé, à savoir k-NN, SVM, Naïve Bayes, arbre de décision, forêt aléatoire et régression logistique. Il est observé que l'ensemble de données CSE-CIC-IDS2018 obtient une performance supérieure avec 99 % de précision et 99 % de F-mesure alors que le modèle Naïve Bayes n'obtient de bons résultats que sur l'ensemble de données CICDDoS2019 avec 45 % de précision. Son étude montre que la qualité des jeux de données contribue à la détection des intrusions [29].

1.6.2. Travail de 'Fethi Amira, Litim Yousra, 2021'

Au sein de cette thèse de master, élaboré et soutenu en 2021, A. Fethi et Y. Litim abordent la problématique portant sur la vulnérabilité du cloud face aux attaques DDoS, et proposent une détection basée sur des algorithmes d'apprentissage automatique (arbre de décision, KNN, régression logistique, Naïve Bayes). Basée sur les données CIC-DDoS2019, l'étude montre des taux de précision élevés, jusqu'à 99 %, notamment pour KNN et SVM, soulignant l'importance de mécanismes de détection intelligents dans le cloud [30].

1.6.3. Travail de 'Abdinasir HIRSI, 2024'

Dans la présente étude est proposée une méthode pour la détection dans des réseaux définis par logiciel (SDN), orientée plus particulièrement sur les attaques DDoS. Ce travail réalisé par HIRSI

en 2024 repose sur l'utilisation de techniques d'apprentissage automatique supervisé et un ensemble de données généré via Mininet, qui recèle plus d'informations que des jeux de données existants en proposant des topologies réalistes et des scénarios d'attaques récentes. Le modèle de forêt aléatoire a permis d'atteindre un taux de 98,97 % de réussite, avec un faible taux de fausses alertes (FAR) de 0,023, surpassant les indications de référence CICDDoS2019. Ces résultats illustrent du reste l'impact croissant de la sélection des caractéristiques et de l'évaluation des performances au sein des stratégies de détection DDoS dans les environnements SDN [26].

1.6.4. Travail de 'Debashis Kar Suvra, 2025'

L'étude propose un dispositif de détection DDoS en temps réel basé sur le machine learning, appliqué aux données CIC-DDoS2019. Les algorithmes RF, AdaBoost et XGBoost ont montré de hautes performances, RF atteignant 99,88 % de précision. Le prétraitement, incluant standardisation et PCA, joue un rôle clé. Naive Bayes s'est révélé moins efficace. L'implémentation comprend un tableau de bord web, avec un taux de détection de 92 %. [31]

1.6.5. Tableau comparatif et discussion

Pour mieux situer notre contribution, nous réaliserons une synthèse comparative des travaux existants sur le choix des datasets, la performance des algorithmes et la manière dont ces derniers sont prétraités. Cela va révéler en particulier les forces, les faiblesses et les voies de progrès possibles de la recherche sur la détection des attaques DDoS.

Critères	Stefanos Kiourkoulis (2020)	Fethi Amira (2021)	Hirsi et al. (2024)	Debashis Kar Suvra (2025)
Année	2020	2021	2024	2025
Dataset	CICDDoS2019, CSE-CIC-IDS2018, NDSEC-1	5% CICDDoS2019	SDN-DDoS, créé à l'aide de l'émulateur Mininet	CICDDoS2019
Algorithmes	KNN, SVM, NB, DT, RF, LR	DT, KNN, SVM, RF, LR, NB	LR, SVM, RF, KNN, XGBoost	LR, KNN, RF, SVM, NB, XGBoost, AdaBoost
Prétraitement	Gestion des NaN, transformation	Encodage, nettoyage, StandardScaler	Chi² pour sélection caractéristiques	Nettoyage, PCA, StandardScaler
Résultats	RF : Préc. 0.99, Rappel 0.99, PredTime 84.2	RF : Préc. 0.92, Rappel 0.99	RF : Préc. 97.62%, Rappel 98.70%	RF : Préc. 98.80%, AUC 0.998, PredTime 9.71
Limites	Analyse limitée aux Datasets uniquement pas d'optimisation des algorithmes	Échantillon limité	Portée des données limitée peut ne pas couvrir tous les scénarios possibles d'attaques DDoS.	PCA a trop réduit les informations (perte des informations)
Contributions	Accent sur les ensembles DDoS	Approche proactive DDoS	Nouveau dataset public	Sélection via PCA

Table 1.1: Comparaison des travaux connexes sur la détection des attaques DDoS (2020-2025)

Les études antérieures ayant trait au sujet portant sur la détection des attaques DDoS montrent des avancées, comme des limites. Si certaines recherches comme celles de Fethi Amira et Litim Yousra (2021) affichent une certaine efficacité des modèles d'apprentissage des machines sur des jeux de données comme CICDDoS2019, on note que la question de l'effet des techniques de préparation sur ces résultats a été peu explorée. L'accent a été mis principalement sur la mise au point des algorithmes, leur capacité à détecter les anomalies, au détriment d'une franche exploration des méthodes de préparation des données qui auraient pu jouer un rôle dans la performance des modèles, ce qui ouvre la réflexion sur l'importance de la phase de prétraitement des données dans le cadre de la détection des attaques DDoS.

En s'appuyant sur les travaux évoqués précédemment, une recherche a la possibilité d'être engagée pour étudier de manière plus systématique l'impact des différentes techniques de prétraitement sur les performances des modèles utilisés sur le dataset CICDDoS2019, afin d'enrichir nos connaissances sur le fonctionnement de la détection des DDoS, tout en proposant des pistes d'améliorations pour les recherches à venir. De fait, les limites soulignées par la littérature actuelle ouvrent la voie à une recherche qui pourrait participer à l'optimisation des détections pour faire face à des menaces toujours plus évoluées et pernicieuses.

1.7. Conclusion

Ce premier chapitre avait pour objectif essentiel de poser les bases fondamentales permettant de comprendre les attaques par déni de service distribué (DDoS) par la présentation de leur nature, de leurs mécanismes, de leurs impacts et des différentes catégories d'attaques existantes. Nous avons également souligné l'importance primordiale de la détection rapide et efficace des attaques en mettant en avant les difficultés de classification du trafic réseau. La présentation du dataset CIC-DDoS2019, présent dans de nombreuses études, a permis d'illustrer l'intérêt d'un jeu de données représentatif et structuré pour l'entraînement de modèles. Enfin, la revue de la littérature a mis en lumière des tendances récentes dans l'utilisation des algorithmes d'apprentissage automatique pour la détection des DDoS, tout en permettant également de mettre à jour certaines limites, dont un essentiel travail de préparation des données parfois inobservé.

Ce panorama global montre la direction de futures recherches qui devraient s'orienter vers des approches plus complètes avec le choix pertinent des datasets, la pertinence des prétraitements, la sélection des caractéristiques et techniques de détection pour contrer des attaques de plus en plus élaborées

Chapitre 2 : Prétraitement des Données et Impact sur la Détection

2.1. Introduction

La qualité des données impacte directement la capacité de détection des attaques DDoS. Le prétraitement est une tâche critique pour la transformation de données brutes en données traitées, équilibrées et exploitables. A ce titre, les données réseau sont généralement bruitées, déséquilibrées ou parfois incomplètes, ce qui contribue à réduire la précision et la robustesse des modèles. Ce chapitre se consacre aux différentes étapes du prétraitement pour la détection des DDoS : le nettoyage des données (valeurs manquantes, doublons), la normalisation, la réduction de dimensionnalité et l'équilibrage des classes. Chacune des étapes mises en œuvre contribue à maximiser la précision, la rapidité d'exécution et la capacité de généralisation des algorithmes de détection des DDoS.

2.2. Définition et importance du prétraitement des données

Le prétraitement des données est une phase essentielle en matière de cybersécurité puisqu'il contribue fortement à la qualité, à la cohérence et à la pertinence des données traitées. En effet la phase de nettoyage, de transformation et de structuration des jeux de données permet de réduire les biais d'analyse et de garantir la fiabilité des analyses [32]. Les data scientists consacrent environ 60 % de leur temps, aux tâches de prétraitement des données. Cela souligne l'importance de cette étape comme l'illustre la figure ci-dessous, cette répartition du temps met en évidence le rôle crucial du prétraitement.

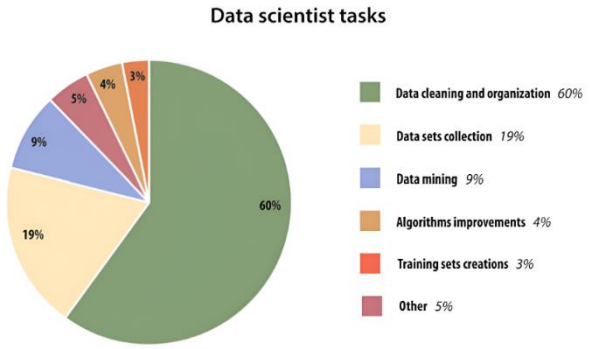

Figure 2.1 - Proportion du temps des data scientists consacré par tâche [36]

Ce processus de préparation des données avant leur analyse est notamment un enjeu majeur dans l'exploration de l'utilisation du Web. Il requiert la suppression de tout ce qui est inutilement redondant, le traitement de toute valeur manquante, ainsi que la correction de toute incohérence, le tout afin de rendre opérationnelles les données pour une éventuelle découverte de modèles [33].

Le prétraitement des données est crucial pour la détection DDoS, en raison de la complexité, du déséquilibre et du bruit des données [34]. Il améliore les performances des systèmes NIDS en facilitant l'analyse du trafic, la réduction des faux positifs et la détection des attaques complexes grâce à des techniques avancées comme l'inspection approfondie des paquets [35]. De plus, des

approches comme la théorie des ensembles approximatifs permettent de traiter la redondance et de pondérer les attributs pour renforcer l'efficacité des systèmes de détection [36].

2.3. Nettoyage des Données

2.3.1. Définition du nettoyage des données

Le nettoyage de données est le sous-processus le plus essentiel dans le processus de prétraitement des données, c'est l'identification et la correction des erreurs ou des incohérences présentes dans les ensembles de données. Il est vrai que le nettoyage est souvent considéré le processus le plus fastidieux, et pourtant, il est celui qui est le plus pertinent [32].

2.3.2. Problèmes de qualité des données :

La qualité des données est très importante pour réussir les taches de groupement ou de classification. Différentes sortes de problèmes peuvent faire du tort à cette qualité :

- **Le Bruit :** Le bruit est considéré comme des fautes dans les données ou des informations sans rapport ou qui ne veulent rien dire, dénuées de sens. Les ensembles de données avec trop de bruit peuvent avoir un impact désastreux sur la réussite d'une tâche de classement, par exemple en baissant la capacité à prédire un classificateur [37]. Le bruit se décline en une multitude de sous-types de problèmes : « valeurs manquantes, valeurs aberrantes, incohérence (étiquettes incorrectes), redondance (instances dupliquées) »
- **Valeurs manquantes :** désigne le fait qu'aucune valeur soit relevée pour un attribut, ce qui peut survenir notamment en raison d'éventuelles défaillances survenues durant certains moments d'observation du dispositif de mesure, exemples de ces défaillances pouvant être les défauts de transfert de certaines données de mesure vers un système d'information, les défauts de capteur et les lacunes des enquêtes etc…[37]
- **Valeurs aberrantes (Outliers) :** constatées lorsqu'une observation se trouve éloignée des autres observations. Une valeur aberrante peut être une valeur incohérente ou une classe anormale de la variable mesurée. [37]
- **Incohérence (Étiquettes incorrectes) :** on parle de cas de duplication mais portant une étiquette de classe différente. [37]
- **Redondance :** liée aux tâches de classification, elle fait référence à des enregistrements identiques. Il s'agit de l'existence de plusieurs enregistrements identiques dans un ensemble de données.[37]

2.3.3. Taxonomie des Tâches de Nettoyage des Données

Pour parvenir à une amélioration de la qualité des données, plusieurs tâches constituant la procédure de nettoyage sont définies. Ces dernières constituent la taxonomie illustrée dans la Figure 2.2.

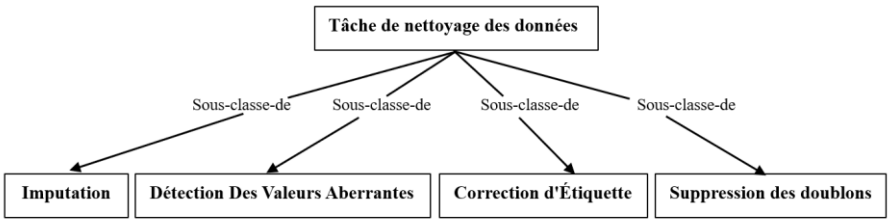

Figure 2.2 - Taxonomie des Tâches de Nettoyage des Données [37]

Chaque sous-classe du nettoyage des données utilise des techniques spécifiques pour résoudre les problèmes associés, comme détaillé ci-après [37].

- **Imputation :** elle consiste à remplacer les données manquantes par des valeurs synthétiques. Différentes techniques sont utilisées pour la mise en imputation des valeurs manquantes [37]:
  - ***Suppression :*** on élimine toutes les instances possédant des valeurs manquantes.
  - ***Hot deck :*** on remplace les données manquantes par des valeurs tirées du même jeu de données.
  - ***Imputation par des attributs manquants :*** on calcule la valeur avec des mesures de tendance centrale (médiane, mode, moyenne…). La valeur obtenue est utilisée pour remplir les données manquantes.
  - ***Imputation par des attributs non manquants :*** on construit un modèle de classification ou de régression à partir des données dont on dispose pour remplir les données manquantes.
- **Détection des valeurs aberrantes :** cherche à désigner des candidats pour les valeurs aberrantes à partir d'algorithmes adaptés aux espaces de haute dimension, comme l'Angle-Based Outlier Degree (ABOD) ou des algorithmes fondés sur la densité comme le Local Outlier Factor (LOF), le clustering spatial sur la densité d'applications avec bruit (DBSCAN), etc. [37]
- **Correction des étiquettes :** Cette tâche consiste à détecter des instances qui partagent les mêmes valeurs. Lorsque les classes sont différentes, alors l'étiquette est corrigée ou l'instance est supprimée [37]. Les approches proposées sont :
  - Des méthodes basées sur un seuil.
  - Des méthodes basées sur des algorithmes de classification.
- **Identification et suppression des doublons :** supprime les enregistrements dupliqués de l'ensemble de données [37].

2.3.4. Avantages du nettoyage des données

Les méthodes utilisées peuvent changer, mais leur effet est toujours le même : une meilleure qualité de données et des aperçus plus sûrs. Les bienfaits d'un nettoyage approfondi des données sont nombreux. Des données précises mènent à des aperçus et résultats plus sûrs et à un choix réfléchi. Des données cohérentes garantissent que les analyses ne sont pas déformées par des différences de forme ou d'unité. Des jeux de données complets permettent des analyses complètes sans les contraintes mises par des informations manquantes. Une bonne gestion des valeurs

étranges ou aberrantes évite des résultats faux et assure une forte performance des modèles. Le nettoyage des données garantit la sincérité de ces derniers, ce qui a la fin rend plus forte toute analyse ou projet d'étude d'apprentissage automatique à venir [32].

2.4. Transformation des Données

La transformation des données vise à adapter leur format ou leurs valeurs pour les rendre exploitables par les algorithmes d'apprentissage automatique. Elle améliore la qualité, révèle les tendances et accélère l'analyse, renforçant ainsi la performance des modèles [32].

2.4.1. Les techniques de transformation des données

- **La mise à l'échelle :** ajuste la plage des variables quantitatives. La normalisation Min-Max ramène les valeurs dans [0,1], tandis que la standardisation (Z-score) centre les données sur 0 avec un écart-type de 1. Cela évite qu'une variable à grande amplitude domine l'entraînement du modèle [32].
- **Encodage one-hot :** convertir les caractéristiques catégorielles en variables numériques via la création de colonnes binaires pour chaque modalité (catégorie), évitant ainsi d'introduire un ordre artificiel pouvant fausser les algorithmes de machine learning [32].
- **Discrétisation (binning) :** convertir les caractéristiques numériques continues en caractéristiques catégorielles discrètes en les regroupant dans des intervalles, ce qui peut améliorer les performances d'algorithmes comme les arbres de décision [32].
- **Création/ingénierie de caractéristiques :** consiste à générer de nouvelles variables à partir de celles existantes (opérations mathématiques, agrégations, connaissances métier) afin d'améliorer la précision du modèle [32].

2.5.Réduction de la Dimensionnalité

La réduction de la dimensionnalité consiste à transformer des données initialement de grande dimension en données de dimension inférieure tout en gardant l'essentiel de l'information. L'objectif est de complexifier moins les données sans perdre d'aspect significatif, ce qui améliore les résultats du modèle et accélère les temps de calcul [32].

2.5.1. Techniques de Réduction de Dimensionnalité

- **Analyse en Composantes Principales (ACP) :** consiste à créer de nouvelles variables qui sont des combinaisons linéaires des variables initiales. Celles-ci sont ordonnées par la quantité de variance qu'elles expliquent, une première composante qui explique le plus de variance, la seconde qui explique la seconde plus grande, etc. En ne conservant que les premières composantes qui expliquent une part significative de la variance, on obtient une dimensionnalité réduite [32].
- **Analyse discriminante linéaire (LDA) :** La LDA a pour objectif de fournir une combinaison linéaire de caractéristiques qui se doit de maximiser le ratio de la variance inter à la variance intra dans l'espace des classes. En d'autres termes, la LDA va veiller à transformer les caractéristiques afin de garantir une séparation maximale des classes dans un espace de dimension inférieure [38].

2.5.2. Avantages et limitations de la réduction de dimensionnalité

La minimisation de la dimensionnalité peut être bénéfique à la fois pour la qualité de la prédiction et pour la compréhension, en permettant d'éliminer les caractéristiques peu pertinentes ou redondantes, c'est-à-dire qu'une caractéristique non pertinente n'apporte absolument aucune information utile et qu'une caractéristique redondante n'apporte aucune information supplémentaire. Toutefois, rien ne garantit qu'ils fonctionneront mieux après cette réduction de la dimensionnalité. Le choix de la méthode et du nombre de dimensions à garder dépend exclusivement de la nature des données et des objectifs du projet [32].

2.6. Équilibrage des Classes

L'objectif du rééquilibrage des classes consiste à répondre au déséquilibre des données afin que les algorithmes ne favorisent pas les éléments de la classe majoritaire face à ceux du reste. Parmi les méthodes récurrentes, on trouve le sous-échantillonnage (RUS) et le sur-échantillonnage (ROS), qui sont souvent utilisés pour améliorer les performances de de classification [39].

2.6.1. Méthodes de Sous-échantillonnage

Le sous-échantillonnage consiste à supprimer des instances de la classe majoritaire. Voici quelques techniques spécifiques :

- **Règle du Plus Proche Voisin (NN) :** Cette approche classe les données en tenant compte des similarités entre un point de données et son voisin le plus proche. Elle a un taux d'erreur moins volumineux que les autres règles de décision. [39]
- **Sous-échantillonnage par Clustering :** Basée sur des algorithmes de clustering comme K-means qui donnent de bonnes performances avec des données déséquilibrées.[39]
- **Seuil de Dureté des Instances (IHT) :** Réduit la taille de la classe majoritaire en supprimant les données ayant un seuil de dureté élevé, qui est la probabilité de mauvaise classification [39].

2.6.2. Méthodes de Sur-échantillonnage

Le sur-échantillonnage consiste à ajouter de nouvelles instances à la classe minoritaire. Les techniques spécifiques incluent :

- **Sur-échantillonnage par Bootstrap :** Répéter itérativement les instances d'un échantillon choisi, les instances pouvant être remplacées et choisies plusieurs fois [39].
- **Technique de Sur-échantillonnage Synthétique des Minorités (SMOTE) :** Augmente le nombre d'instances de la classe minoritaire par création de nouvelles instances « syntactiquement » proches dans l'espace des caractéristiques, entre les plus proches voisins de la classe minoritaire [39].
- **Méthode ADASYN :** Inspirée de SMOTE elle génère des exemples synthétiques pour accroître la taille de la classe minoritaire. La taille de l'exemple est déterminée via un critère lié à une distribution de densité, ce qui permet un contrôle sur le nombre d'échantillons produits [39].

2.6.3. Limitations des méthodes de rééchantillonnage

- **Le sur-apprentissage :** Le sur-échantillonnage est un phénomène qui se manifeste lorsque le modèle en apprentissage mémorise les données d'entraînement y compris le bruit présent

dans les données, au lieu d'en apprendre les motifs généraux, ce qui nuit à sa capacité à généraliser. Pour y remédier, on peut envisager la validation croisée, la simplification du modèle, l'augmentation des données ou encore la validation sur d'autres ensembles [32].

- **Le sous-apprentissage :** La première situation de sous-échantillonnage apparaît lorsque le modèle est trop simple pour traiter la complexité des données. Il en résulte de faibles performances aussi bien pendant l'entraînement que le test. Les solutions incluent un entraînement plus long, une augmentation des caractéristiques, un modèle plus complexe et une régularisation moins forte [32].

La figure 2.3 illustre l'impact du type d'apprentissage du modèle (bon ajustement, sous ajustement ou surajustement) sur l'évolution de la performance des modèles.

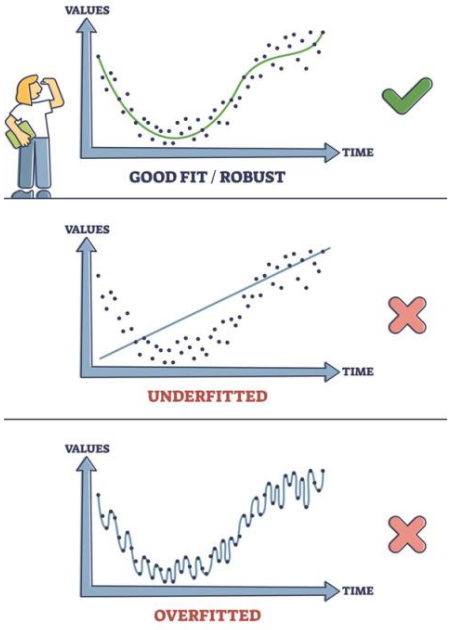

Figure 2.3 - Comparaison des types d'ajustement : Bon, Sous-ajusté et Surajusté [32]

2.7. Conclusion

S'agissant des diverses techniques de prétraitement mises à l'épreuve dans le cadre de ce chapitre, il apparaît que cette étape constitue un axe fondamental dans l'élaboration de systèmes de détection robustes et efficaces, en premier lieu face aux attaques DDoS ; l'étape de nettoyage, de transformation des données, de réduction de dimensionnalité, de rééquilibrage des classes n'est pas qu'un simple ajustement technique, mais se révèle être un axe stratégique crucial pour favoriser tout au long de la chaîne de traitement de données la qualité de celles-ci et par voie de conséquence, la performance des modèles d'apprentissage automatique.

Pour conclure, ce chapitre rappelle que le prétraitement des données représente, à la fois une étape préliminaire, mais aussi un processus itératif et adaptatif, qu'il s'agit indispensable au préalable de bien connaître à la fois les données et les objectifs de l'analyse, pour qu'un traitement des données efficaces soit proposé. S'approprier ces techniques de prétraitement revient ainsi à donner au professionnel qui les maîtrise, les chances exploitant au mieux les ressources offertes principalement par les données dans le domaine de la cybersécurité et au-delà.

Chapitre 3 : Concepts fondamentaux en Machine Learning et Modèles de Détection des Attaques DDoS

3.1. Introduction

En prenant en considération le paysage numérique actuel, les attaques par déni de service distribué (DDoS) constituent une menace qui accentue la disponibilité des services en ligne. Dans ce contexte, le Machine Learning (ML) se présente comme une technologie à fort potentiel pour améliorer la détection et la mitigation de telles attaques. Ce chapitre aborde les concepts de base du ML, avec un objectif de contextualisation et d'ancrage de ce type de traitement dans le paradigme de la détection des attaques DDoS, afin d'outiller le lecteur sur les principes de fonctionnement de ces algorithmes (identification, classification, traitement), notre focalisation se porte sur cinq algorithmes traditionnels : Random Forest, SVM, k-NN, Naïve Bayes et XGBoost pour mieux en cerner le rôle dans l'établissement d'une cybersécurité plus robuste et adaptative.

3.2. Concepts clés en Machine Learning et Intelligence Artificielle

3.2.1. L'intelligence artificielle

C'est la capacité d'une machine à imiter l'intelligence humaine ce qui implique la mise en œuvre d'algorithmes et de techniques d'apprentissage automatique. L'IA vise à permettre à une machine d'effectuer des tâches nécessitant normalement des capacités cognitives, ce qui permet ainsi d'augmenter l'efficacité et la fiabilité dans la réalisation de tâches difficiles [40].

3.2.2. L'apprentissage automatique (Machine Learning)

Le machine learning est un sous-domaine de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir des données, sans programmation explicite et en utilisant des modèles statistiques qui vont lui permettre de détecter des régularités, de prendre des décisions, ou de faire des prédictions. Grâce à l'accumulation de données, ces algorithmes sont capables d'améliorer leurs performances, ce qui les rend d'un point de vue fonctionnel adaptables. Le machine learning trouve aussi bien des applications dans la reconnaissance d'image ou de voix, que pour le traitement du langage, des recommandations ou l'analyse prédictive [41].

- **Étapes de l'apprentissage automatique :** Les étapes de l'apprentissage automatique sont représentées dans la figure 3.1, mettant en évidence le processus allant de la collecte de données à la mise en production du modèle.

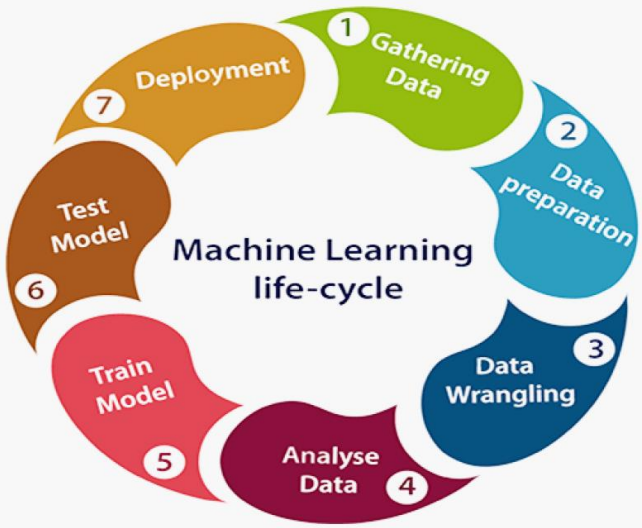

Figure 3.1 - Cycle de vie de l'apprentissage automatique [41]

- ***Définition du problème :*** Il s'agit de définir le problème à traiter et de l'identification de la variable à prédire.
- ***Collecte des données :*** rassemblement des données nécessaires à la résolution du problème.
- ***Exploration des données :*** compréhension des données, y compris leur nettoyage et transformation si besoin.
- ***Sélection du modèle :*** choix du modèle adapté à la problématique et aux données.
- ***Entraînement du modèle :*** Entrainer le modèle sur les données.
- ***Évaluation du modèle :*** Évaluer le modèle en utilisant des indicateurs pertinents pour en évaluer les performances.
- ***Déploiement du modèle :*** Déployer le modèle en environnement de production, avec éventuelle intégration à d'autres systèmes [41].

3.2.3. Types d'apprentissage automatique

- **Apprentissage supervisé :** L'apprentissage supervisé est un entraînement d'un modèle sur des jeux de données étiquetées afin de prédire une variable cible à partir de relations entre les entrées. Ce type d'approche est usité dans les domaines très variés que sont la classification d'images, la détection de fraudes, le diagnostic médical, l'analyse sentimentale ou encore les systèmes de recommandation. Les principales méthodes utilisées sont : la régression (valeurs continues), la classification (attribution à des classes), la régression logistique (décision binaire), les arbres de décision et les forêts aléatoires (décisions séquentielle), les SVM (séparation par hyperplans), Naive Bayes (classement probabiliste), et bien sûr les réseaux neuronaux qui permettent de modéliser des relations complexes [41].
- **Apprentissage non supervisé :** En s'appuyant sur des données non étiquetées, donc sans variable cible connue, le modèle délaisse l'accompagnement d'un expert pour déceler par

lui-même des structures ou motifs cachés dans ces données. On le retrouve dans la segmentation de clients, la détection d'anomalie ou de fraude, le traitement du texte (NLP) ou des images, ou encore dans les systèmes de recommandation. Au rang de ses méthodes principales, on peut mentionner le clustering ou regroupement de données similaires, la réduction de dimensionnalité, la détection d'anomalies, les règles d'association, les auto encodeurs ou encore les GAN (génération de données réalistes) [41].

- **Apprentissage par renforcement :** L'apprentissage par renforcement est une stratégie d'apprentissage par laquelle un agent est entraîné à interagir avec un environnement à l'aide de récompenses et de pénalités selon les actions menées, dans le but d'optimiser ses choix sur le long terme. Ce type d'apprentissage est utilisé dans des domaines variés allant des jeux (échecs, Go) à la robotique (navigation, manipulation) en passant par les véhicules autonomes, le trading algorithmique, la santé (recommandation de traitements) ainsi que la logistique et le marketing. Les deux grandes approches existantes sont les méthodes value-based, comme le Q-Learning, qui évalue les états afin de maximiser la récompense, et les méthodes policy-based (REINFORCE) qui apprennent directement une politique d'actions optimales. Ces deux approches sont souvent complémentaires selon que l'on recherche plus de stabilité ou de flexibilité [41].

3.3. Présentation des modèles de détection des attaques DDoS utilisés

3.3.1. Random Forest

La forêt aléatoire (Random Forest) est une technique d'apprentissage à base d'ensembles, issu de la combinaison de plusieurs arbres de décision considérés comme des classifieurs réalisés à partir d'échantillons aléatoires de l'ensemble d'apprentissage. Chaque arbre vote pour une classe et la prédiction finale est obtenue par vote majoritaire. Cette méthode améliore la capacité de généralisation et contribue à réduire le surapprentissage. Ce modèle est bien adapté à la classification, comme le cas qui pourrait consister à distinguer un chien d'un chat à l'aide d'apprentissages supervisés sur des images de chiens et de chats étiquetés. On pourra comme même dire qu'elle fonctionne aussi sur de grands jeux de données bruités. Sa robustesse a même été décrite comme étant appropriée à plusieurs études de détection d'attaques DDoS dans lesquelles elle a été testée [42].

3.3.2. Machine à Vecteurs de Support (Support Vector Machine SVM)

Les SVM (machines à vecteurs de support) constituent un algorithme d'apprentissage supervisé performant pour des tâches de classification et de régression. Ils visent à trouver un hyperplan optimal qui sépare les classes avec la plus grande marge afin d'améliorer la généralisation du modèle même dans des espaces de grande dimension [26]. Il s'agit d'un processus d'apprentissage fondé sur la recherche d'un hyperplan à marge maximale (MMH), qui sera déterminé à partir des vecteurs de support, c'est-à-dire les points les plus proches du plan de séparation [42]. Cet hyperplan sépare les différentes classes dans l'espace des caractéristiques, comme illustré dans la Figure 3.2.

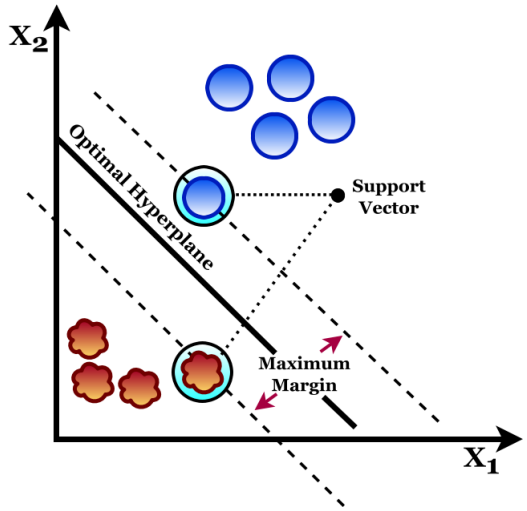

Figure 3.2 - Visualisation de la Classification SVM [26]

En cas de problèmes non linéaires, SVM utilise des astuces de noyau (« kernel tricks »), qui permettent de projeter les données dans des espaces de plus grande dimension pour rendre la séparation linéaire [29]. Les SVM sont très largement utilisés dans des problèmes de classification avec du bruit et des surajustements mais aussi quelques fois dans des problèmes de régression, en raison de la robustesse démontrée de l'algorithme [26].

3.3.3. k-Nearest Neighbors KNN

L'algorithme d'apprentissage supervisé K-Nearest Neighbor (KNN) est un algorithme de classification simple et non paramétrique qui recherche les K voisins les plus proches (à l'aide de la distance euclidienne en général) dans l'ensemble d'apprentissage puis attribue une étiquette à un échantillon, celui du vote de majorité entre voisins dans le cas de cette évaluation de classes [43]. KNN est un algorithme qui apprend à partir de données étiquetées et prédit de nouvelles données en votant selon la majorité des classes présentes parmi les voisins. Le choix du nombre de voisins K est un choix critique car un K trop petit peut être sensible au bruit et un K trop grand peut dégrader la séparation entre les classes [42].

3.3.4. Naïve Bayes

Naïve Bayes (NB) est un modèle d'apprentissage supervisé reposant sur le théorème de Bayes, visant à prédire l'appartenance d'un objectif à une catégorie à partir de probabilités estimées à partir de données passées. Ce classifieur s'appuie sur l'hypothèse dite « naïve » qui fait que chaque attribut d'un exemple est supposé indépendant des autres attributs d'un même exemple. Même si cette hypothèse est souvent optimiste pour la réalité d'observation, l'algorithme a montré une robustesse relative et de bonnes performances, réussissant dans plusieurs cas de dépendance partielle des attributs. Ce modèle est caractérisé par sa simplicité d'utilisation, sa rapidité d'exécution et sa faiblesse d'exigence en taille d'échantillon, représentant un modèle intéressant en situations de ressources limitées. Le modèle dispose de plusieurs variantes comme Naïve

Bayes gaussien pour des données à attributs continus ou multinomial pour des attributs discrets. Toutefois, sa sensibilité aux hypothèses d'indépendance fait qu'il reste peu efficace dans des situations beaucoup plus complexes, il peut être également relativement vulnérable à des attaques comme l'empoisonnement bayésien, qui peut tromper les filtres anti-spams en leur fournissant un bruit adverse. L'algorithme est couramment utilisé pour réaliser des tâches de traitement du langage naturel comme la classification automatique des textes, la détection des spams, la prédiction à des systèmes de diagnostic médical [44].

- **Théorème de Bayes et formulation mathématique :** L'équation du théorème de Bayes est la suivante [44] :

$$P(C|F) = \frac{P(C) * P(F|C)}{P(F)}$$

Équation 3.1 - Théorème de Bayes [44]

Où :
• F : Données dont la classe est inconnue
• C : Hypothèse que les données appartiennent à une classe spécifique
• P(C | F) : Probabilité a posteriori de l'hypothèse C étant donné F
• P(C) : Probabilité a priori de l'hypothèse C
• P(F | C) : Probabilité de F étant donné C
• P(F) : Probabilité de F

3.3.5. Extreme Gradient Boosting XGBoost

L'algorithme XGBoost - pour Extreme Gradient Boosting - est un algorithme d'apprentissage supervisé reposant sur l'optimisation du Gradient Boosting Decision Tree (GBDT) dans le but d'être rapide, efficace et performant [45], par l'assemblage d'un très grand nombre d'arbres de décision faibles construits de manière séquentielle, un nouvel arbre étant ajusté à chaque fois aux erreurs (résidus) des arbres précédents afin de réduire de manière incrémentale l'erreur de prédiction globale [46].

Un des nombreux atouts de XGBoost c'est sa capacité à traiter des ensembles de données massifs, souvent déséquilibrés, avec un bon niveau de précision, sa rapidité, sa résistance au sur-apprentissage et sa solidité en font une méthode de choix pour des cas d'application comme la détection d'attaques DDoS sur de gros volumes de flux réseau [26].

Si l'on revient sur les paramètres, XGBoost présente une large souplesse grâce à de multiples options au paramétrage. On peut notamment prendre en compte les paramètres de la catégorie Booster qui conditionnent la nature de l'algorithme de boosting mobilisé. Une optimisation de ces paramètres, généralement effectuée via une recherche en grille (Grid Search) couplée à une validation croisée (k-fold), permet d'adapter finement le modèle aux spécificités des jeux de données considérés [45].

Pour finir, la figure 3.3 illustre l'évolution des méthodes ensemblistes depuis le Bagging (utilisé par l'algorithme Random Forest) jusqu'au Boosting, en passant par le Gradient Boosting, pour en arriver à XGBoost. Ce modèle constitue une réelle avancée en matière de performance dans la résolution de problèmes de classification et de régression dans des situations complexes.

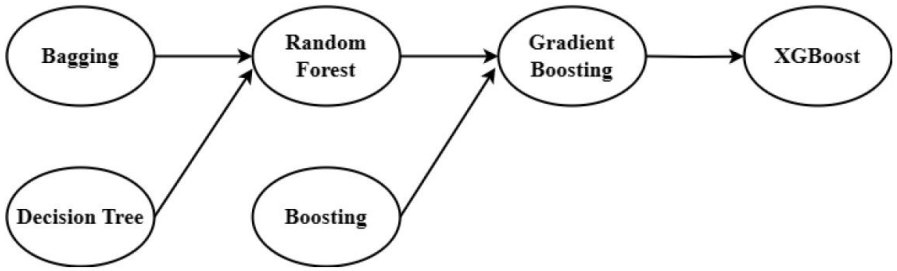

Figure 3.3 - L'évolution de l'algorithme XGBoost [45]

3.4. Conclusion

Le présent chapitre a permis de mettre en perspective les concepts basiques du Machine Learning et de l'Intelligence Artificielle (IA), et leur adéquation à la détection d'attaques DDoS. En premier lieu nous avons considéré les fondements de l'IA, du Machine Learning, des principaux types d'apprentissages pour l'IA, à savoir, supervisé, non supervisé et par renforcement. Nous avons ensuite présenté cinq modèles de détection d'attaques DDoS : Random Forest, SVM, k-NN, Naïve Bayes et XGBoost, ces modèles étant examinés en termes de mécanismes, atouts et limites respectives.

Ces notions sont cruciales car elles permettent de passer de la donnée brute à l'information utile pour la sécurité d'un réseau. On peut citer l'apprentissage supervisé capable de classifier le trafic réseau en fonction de modèles préalablement appris et l'apprentissage non supervisé qui permet de détecter des comportements anormaux en absence d'étiquetage des données.

Au final, ce chapitre a permis de définir les points de repère théoriques et moyens pratiques nécessaires pour comprendre et mettre en œuvre les techniques de Machine Learning dans la lutte contre les attaques par DDoS, mais aussi de montrer la diversité des algorithmes, qui offre plusieurs solutions possibles à ses enjeux sécuritaires du réseau, donc tout l'intérêt de faire des choix éclairés au bon emploi selon chaque contexte.

Chapitre 4 : Modélisation et prétraitement des données pour l'évaluation des performances

4.1.Introduction

Dans ce chapitre, l'idée est de décrire les stratégies de prétraitement des données et la modélisation itérative mis en œuvre dans le cadre de l'évaluation des performances des modèles de détection DDoS. Cette démarche vise à analyser l'influence des différentes stratégies de prétraitement sur les résultats finaux, en adoptant une approche progressive et structurée.

Quatre configurations de prétraitement vont être mises en œuvre (sans prétraitement : approche brute, prétraitement standard : nettoyage classique, prétraitement avancé : nettoyage spécial, prétraitement approfondi) permettant d'évaluer la répercussion de ce traitement sur la performance des modèles de détection DDoS (Random Forest, XGBoost, Naive Bayes, KNN et SVM). En effet, sur chaque configuration les algorithmes vont être entraînés et alors évalués, ce qui donne lieu à une comparaison d'objectivité suffisante pour s'affranchir de biais et à une visibilité suffisante concernant leurs performances. Les mesures d'évaluation considérées sont l'exactitude, la précision, le rappel, la F1-mesure et l'aire sous la surface de la ROC (AUC), les courbes d'apprentissage et les matrices de confusion, permettent de mesurer de manière objective l'efficacité de ces stratégies, tout en préparant une analyse comparative détaillée dans le chapitre suivant.

4.2. Présentation des métriques d'évaluation de performances

L'évaluation des performances des algorithmes d'apprentissage automatique nécessite des indicateurs pertinents, en particulier pour la tâche de classification. Dans cette étude, plusieurs mesures ont été considérées : précision, rappel, F1-Score, AUC-ROC… Toutefois, sur des jeux de données déséquilibrés, ces indicateurs doivent être employés avec prudence.

4.2.1. La matrice de confusion

La matrice de confusion est une métrique fondamentale et très générale pour les tâches de classification. Elle présente un décompte de toutes les prédictions faites par le modèle comparé aux étiquettes des classes réelles. La matrice de confusion est composée de quatre éléments de base [47] :

- Vrai Positif (TP) : Correspond aux instances correctement prédites comme positives.
- Vrai Négatif (TN) : Désigne les instances correctement prédites comme négatives.
- Faux Positif (FP) : Signifie les instances incorrectement prédites comme positives. La valeur réelle est négative, mais le modèle l'a prédite comme positive.
- Faux Négatif (FN) : Représente les instances incorrectement prédites comme négatives. La valeur réelle est positive, mais le modèle l'a prédite comme négative.

4.2.2. Exactitude ou Accuracy

L'exactitude (Accuracy ou ACC) désigne la part des instances correctement classées, elle donne une impression sur la performance globale du modèle. Un ACC élevé est signe que le modèle fait peu d'erreurs, ce qui est crucial pour distinguer les attaques DDoS des flux légitimes [30]. Son équation mathématique est celle qui suive :

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

Équation 4.1 - Formule de l'Exactitude [42]

4.2.3. Rappel ou Recall

Le rappel est la mesure de la capacité d'un modèle à identifier tous les vrais positifs. Le rappel est d'une grande importance dans les scénarios où les faux négatifs peuvent avoir des conséquences catastrophiques, comme des attaques DDoS non détectées [33]. Pour calculer le rappel, l'équation suivante est utilisée :

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

Équation 1.2 - Formule du Rappel [42]

4.2.4. Précision

L'exactitude ne suffit en aucun cas à évaluer un modèle d'apprentissage. La précision, qui mesure le taux de vrais positifs, est une métrique importante car un fort taux de faux positifs peut avoir des effets néfastes [33]. La précision est calculée comme suit :

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

Équation 4.3 - Formule de la Précision [42]

4.2.5. F1-Score

La F-mesure est une mesure qui combine précision et rappel pour évaluer la qualité d'un modèle, un score élevé indique un bon compromis entre la détection des menaces et la réduction des fausses alertes [33]. Le F1-Score est calculé à l'aide de la formule suivante :

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Équation 4.4 - Formule de F1-Score [42]

4.2.6. AUC-ROC

La surface sous la courbe ROC (AUC) est obtenue à partir de la courbe ROC TPR en fonction du FPR, elle permet de mesurer la capacité à bien distinguer les classes, une AUC élevée traduisant une meilleure performance du modèle [47]. Mathématiquement, l'AUC est calculée comme indiqué dans l'équation suivante :

$$AUC_{ROC} = \int_0^1 \frac{TP}{TP + FN} d \frac{FP}{TN + FP}$$

Équation 4.5 - Formule de AUC-ROC [48]

- **Taux de vrais positifs TPR :** quantifie la proportion des occurrences positives que le modèle identifie correctement comme positives [23]. Son équation est la suivante :

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

Équation 4.6 - Taux de vrais positifs [23]

- **Taux de faux positifs FPR :** quantifie la proportion des occurrences négatives que le modèle identifie à tort comme positives [23]. Son équation est la suivante :

$$FPR = \frac{FP}{FP + TN}$$

Équation 4.7 - Taux de faux positifs [23]

- **Taux de faux négatifs FNR :** est le pourcentage de cas positifs identifiés à tort comme négatifs [23]. Il est calculé à l'aide de l'équation suivante :

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

Équation 4.8 - Taux de faux négatifs [23]

- **Taux de vrais négatifs TNR :** représentant le pourcentage de cas négatifs correctement prédits comme négatifs [23]. Le TNR est calculé à l'aide de la formule suivante :

$$TNR = \frac{TN}{TN + FP}$$

Équation 4.9 - Taux de vrais négatifs [23]

4.2.7. Courbe d'apprentissage

Une courbe d'apprentissage vise à montrer comment le modèle évolue en fonction de la taille des données d'apprentissage. Elle contient habituellement le score d'apprentissage et le score de validation. Sa lecture permet de déceler le sur-apprentissage, le sous-apprentissage, et le cas échéant d'adapter le modèle [58].

4.2.8. Temps d'apprentissage et de prédiction

- Le temps d'apprentissage ou d'entrainement : cela n'est pas directement lié à la classification, mais décrit plutôt le temps d'entraînement pris par un modèle. Cette métrique donne une indication de l'efficacité du modèle.
- Le temps de prédiction : est le temps requis pour qu'un modèle d'apprentissage automatique génère des prédictions sur de nouvelles données après avoir été entraîné.

4.2.9. Matrices de corrélation

La matrice de corrélation est construite en utilisant les coefficients de corrélation de Pearson entre les lignes normalisées d'une matrice. Chaque élément représente le coefficient de corrélation entre les composants à travers toutes les fenêtres analysées. Les éléments diagonaux sont toujours égaux à 1, indiquant une corrélation parfaite, tandis que les éléments hors diagonale montrent le degré de corrélation linéaire entre différents composants, variant de -1 à 1. [50]

4.3. Méthodologie expérimentale

Dans le contexte de ce travail, nous portons notre attention sur l'impact du prétraitement des données sur la performance des modèles de détection des attaques DDoS, et tentons d'évaluer cet impact de manière rigoureuse, en mettant en œuvre une approche expérimentale comparative fondée sur l'application de différentes stratégies de prétraitement, sur un même jeu de données, puis sur une modélisation identique à chaque fois. Le jeu de données retenu en entier est le CIC-

DDoS2019 qui est réputé pour la variété des scénarios d'attaques DDoS réalistes qu'il met à disposition.

Quatre configurations de prétraitement seront testées afin de mesurer l'impact de chacun d'entre eux sur la performance des modèles de détection :

- **Sans prétraitement (approche brute) :** Dans cette première approche, les données sont exploitées sous leur forme la plus brute. Aucun nettoyage, aucune normalisation ou rééchantillonnage n'est réalisé, excepté les ajustements minimaux nécessaires pour qu'elles puissent être prises en compte par les modèles d'apprentissage automatique.
- **Prétraitement standard (nettoyage de base) :** Cette deuxième approche consiste à réaliser un nettoyage basique du jeu de données en réalisant :
  - Le traitement des valeurs manquantes ou aberrantes
  - La normalisation des attributs numériques afin d'assurer l'homogénéité des échelles
  - L'équilibrage des différentes classes par la méthode de suréchantillonnage SMOTE, visant à corriger le déséquilibre observé entre les classes d'instances considérées comme attaques et celles considérées comme traffic légitime.
- **Prétraitement avancé (nettoyage spécialisé) :** Cette configuration reprend le chemin de la méthode précédente (nettoyage basique, normalisation), mais y rajoute une phase de sélection des variables, réalisée selon un critère linéaire statistique. L'équilibrage des classes est réalisé ici avec la méthode ADASYN (Adaptive Synthetic Sampling), qui génère de manière adaptative selon la densité locale des exemples synthétiques.
- **Prétraitement approfondie :** Cette dernière approche est une extension de la précédente, en concentrant sur les variables restantes en inspectant s'il reste encore des variables similaires, donc on a ajouté une autre étape de sélection de caractéristiques basé sur l'angle de la droite de régression formé par les paires de variables. Ensuite on prend le même chemin d'équilibrage de classe comme l'approche précédente à l'aide d'ADASYN.

Pour chacune de ces configurations, les mêmes algorithmes de détection seront entraînés et évalués (Random Forest, XGBoost, Naive Bayes, KNN et SVM) afin de permettre une comparaison objective des performances. Les métriques d'évaluation comprendront notamment l'exactitude, la précision, le rappel, la F1-mesure et l'aire sous la courbe ROC (AUC), ainsi que les courbes d'apprentissage, les matrices de confusion, afin de capturer tous les aspects de la performance des modèles.

4.3.1. Environnement et outils de développement

Dans le cadre de la mise en place de notre étude expérimentale, nous avons déterminé l'environnement de développement qui prend en charge d'une part le traitement de données, l'entraînement de modèles d'apprentissage automatique et l'évaluation de leurs performances.

La taille considérable du jeu de données utilisé, CIC-DDoS2019, qui contient environ 50 millions de lignes réparties sur 88 colonnes. Cette volumétrie a imposé des exigences élevées en termes de capacité de traitement et de mémoire vive.

- **Environnement matériel :** Dans un premier temps, une tentative de traitement local a été envisagée sur une machine disposant des caractéristiques suivantes :
  - Processeur : Intel Core i5 11ème génération
  - Mémoire vive (RAM) : 24 Go
  - Stockage : SSD 1.40 To
  - Système d'exploitation : Windows 11

  Toutefois, cette configuration s'est rapidement révélée insuffisante, entraînant des blocages et ralentissements importants lors des opérations de prétraitement et de chargement des données.

  Pour surmonter cette limitation, nous avons migré notre environnement vers Google Colaboratory Pro+, qui offre un accès temporaire à une infrastructure cloud avec une mémoire vive pouvant aller jusqu'à 334 Go. Cet environnement nous a permis de manipuler le jeu de données dans sa globalité, sans réduction préalable, et d'entraîner des modèles sur des sous-ensembles de grande taille.

  Cependant, malgré cette importante capacité mémoire, des problèmes de surcharge sont survenus à plusieurs reprises. Cela a entraîné des interruptions de session du runtime Colab. Ces contraintes nous ont parfois obligés à adapter nos scripts, à fragmenter le traitement pour réduire la pression mémoire
- **Environnement logiciel :**
  - **Python :** langage de programmation largement présent dans les domaines de la science des données et de l'apprentissage machine, en raison de sa grande simplicité et de son écosystème très riche [60].
  - **Jupyter Notebook :** environnement interactif sur le web, permettant l'exploration, l'analyse de données et le prototypage des modèles [62].
  - **Google Colaboratory Pro+ :** version améliorée de Colab donnant accès prioritaire à des ressources GPU/TPU, à des sessions longues, et à une RAM allant jusqu'à 334 Go, adaptée aux tâches lourdes de deep learning [62].
  - **Google Cloud Storage :** solution de stockage cloud utilisée ici pour l'hébergement du dataset volumineux que Google Drive ne pouvait supporter [53].
- **Bibliothèques utilisées :**
  - **Pandas :** un outil essentiel pour la manipulation de données en tant que relationnelles et étiquetées sous Python.[54]
  - **Numpy :** bibliothèque Python pour calculer sur des tableaux multi-dimensionnels et pour le calcul rapide des fonctions mathématiques. [65].
  - **Matplotlib :** bibliothèque Python de visualisation de données qui crée des graphiques de manière personnalisée par une interface de script (pyplot) ou orientée objet [68].
  - **Seaborn :** est une extension de Matplotlib pour la création de graphiques statistiques lisibles, s'intégrant à pandas pour la faciliter l'analyse exploratoire. [57].

4.3.2. Analyse exploratoire des données CICDDoS2019

Dans un premier temps, l'ensemble de données CICDDoS2019 a été téléchargé à partir du site officiel. Ce jeu de données est divisé dans 11 fichiers CSV contenant chacun un type d'attaque DDoS. Dans le but d'utiliser l'ensemble du jeu de données, nous avons choisi de rassembler les 11 fichiers dans un fichier CSV. Cette option est légitimée par l'homogénéité des fichiers qui contiennent tous 88 colonnes identiques en nom et dans l'ordre. La taille considérable des fichiers a nécessité l'usage de Windows PowerShell, outil véritablement adapté à un gros volume de données, pour réaliser la concaténation.

Suite à la concaténation, un fichier CSV unique nommé ddos.csv a été créé, ayant une taille de 22 Go. Pour pouvoir le lire dans Google Colab, il avait été transféré dans Google Cloud Storage, puisque Google Drive ne parvenait pas à gérer un fichier d'une telle taille. Le fichier a été stocké dans un bucket à accès public afin d'en faciliter l'accès. C'est ensuite l'environnement Google Colab Pro+, disposant de TPU et 334 Go de mémoire vive, qui a permis de charger le fichier et de créer le DataFrame sans saturer la mémoire.

Le code suivant est celui qui a servi à l'exploration initiale des données :

- Code permettant de faire apparaître le nombre de lignes et de colonnes dans « df », cette action permet par ailleurs de constater que le jeu de données contient plus de 50 millions de lignes et de 88 colonnes.

```
df.shape
```

```
(50063112, 88)
```

Code 1 – Affichage de la forme de « df »

- Par la suite, la colonne Label a également été explorée afin de repérer les différents types d'attaques DDoS présentes dans le jeu de données, ainsi que le volume de trafic légitime. Le code utilisé à cet effet est reproduit ci-dessous.

```
print(df["Label"].value_counts())
```

```
Label
TFTP             20082580
DrDoS_SNMP        5159870
DrDoS_DNS         5071011
DrDoS_MSSQL       4522492
DrDoS_NetBIOS     4093279
DrDoS_UDP         3134645
DrDoS_SSDP        2610611
DrDoS_LDAP        2179930
Syn               1582289
DrDoS_NTP         1202642
UDP-lag            366461
BENIGN              56863
WebDDoS               439
Name: count, dtype: int64
```

Code 2 - Exploration de la colonne Label

L'analyse a permis de dénombrer 12 familles d'attaques DDoS, à savoir les TFTP, DrDoS_SNMP, DrDoS_DNS, DrDoS_MSSQL, DrDoS_NetBIOS, DrDoS_UDP, DrDoS_SSDP, DrDoS_LDAP, Syn, DrDoS_NTP, UDP-lag, WebDDoS. De l'autre côté, le dataset comprend 56 863 instances de traffic légitime, qui portent le label BENIGN.

4.4. Conception et modélisation de la version 1

4.4.1. Prétraitement minimal des données

Comme nous l'avons déjà précisé plus haut, la volonté dans cette version a consisté à entraîner directement les modèles sur les données brutes, sans aucune transformation complexe. Cependant, des limitations techniques se sont présentées, liées à la présence de valeurs infinies (+inf, -inf) non supportées en entrée par certains algorithmes d'apprentissage comme Random Forest.

De ce fait, un prétraitement minimal a été impérativement nécessaire, les valeurs infinies ayant été converties en NaN, il ne s'agit pas d'une opération analytique, mais d'une nécessité pour faire ensuite passer les données vers la phase d'entraînement.

```
df.replace([np.inf, -np.inf], np.nan, inplace=True)
```

Code 3 - Conversion des inf en NaN

Etant donné que l'objectif principal est d'évaluer la performance de détection des attaques DDoS, sans distinction entre les différents types d'attaques, les labels ont été encodés sous forme binaire, toutes les différentes formes d'attaques DDoS sont regroupées sous la valeur 1, tandis que le trafic légitime est codé par 0.

```
df["Label"] = df["Label"].apply(lambda x: 0 if str(x).strip().upper() == "BENIGN" else 1)
```

Code 4 - Encodage binaire

Nous réalisons ensuite une analyse de la colonne Label afin de vérifier comment cela a été fait

```
print(df["Label"].value_counts())
```

```
Label
1    50006249
0       56863
Name: count, dtype: int64
```

Code 5 - Analyse de la colonne Label

Puis, les colonnes non numériques ont été encodées de manière minimaliste en remplaçant les chaînes de caractères par des entiers, sans cohérence sémantique mais simplement pour permettre d'utiliser ces colonnes avec la majorité des algorithmes d'apprentissage automatique, notamment de scikit-learn, qui ne peuvent pas traiter des données textuelles directement.

```
for col in df.select_dtypes(include=['object', 'category']).columns:
    if col != "Label":  # On ne réencode pas 'Label' ici, déjà fait
        df[col] = LabelEncoder().fit_transform(df[col].astype(str))
```

Code 6 - Encodage des chaînes de caractères vers des entiers

Étant donné que l'objectif est de maintenir les données dans un état aussi brut que possible, aucune imputation classique ne sera menée à bien mais les valeurs manquantes seront remplacées par une valeur arbitraire qui signalera la présence d'anomalies et non une valeur imputée.

```
df.fillna(-999999, inplace=True)
```

Code 7 - Remplacement des valeurs manquantes par une valeur arbitraire

Séparation des variables explicatives et de la variable cible : Les variables explicatives (X) ont été isolées de la variable cible (y) pour permettre la préparation des données à l'entraînement des modèles.

```
X = df.drop("Label", axis=1)
y = df["Label"]
```

Code 8 - Séparation des variables et la cible

Division en ensembles d'entraînement et de test : Les données ont ensuite été séparées en ensembles d'entraînement et de test dans des proportions définies pour évaluer la performance des modèles sur des données non vues.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=42,
                                                    stratify=y)
```

Code 9 - Séparation entrainement et test

Vérification de la taille des ensembles : La forme (nombre de lignes et de colonnes) des ensembles d'entraînement et de test a été analysée pour vérifier que la proportion de répartition des données était bien respectée.

```
X_test.shape
```

```
(15018934, 87)
```

```
X_train.shape
```

```
(35044178, 87)
```

Code 10 - Forme des ensembles d'entrainement et de test

4.4.2. Entrainement des modèles

- **Random Forest, XGBoost et Naive Bayes :** Initialisation des modèles suivie de ses entraînements avec mesure du temps d'exécution.

```
start_train = time.time()
model = RandomForestClassifier(
    n_estimators=10,
    max_depth=10,
    max_features='sqrt',
    min_samples_split=10,
    min_samples_leaf=5,
    n_jobs=-1,
    random_state=42,
    class_weight='balanced'
)
model.fit(X_train, y_train)
end_train = time.time()
```

```
start_train = time.time()
model = XGBClassifier(
    n_estimators=10,
    max_depth=10,
    learning_rate=0.1,
    min_child_weight=1,
    subsample=0.8,
    colsample_bytree=0.8,
    n_jobs=-1,
    random_state=42,
    scale_pos_weight=1
)
model.fit(X_train, y_train)
end_train = time.time()
```

```
start_train = time.time()
model = GaussianNB()
model.fit(X_train, y_train)
end_train = time.time()
```

Code 11 - Initialisation et entrainement RF, XGBoost, NB

- **KNN et SVM :** Comme les algorithmes k-Nearest Neighbors (k-NN) et Support Vector Machine (SVM), qui sont tous deux particulièrement coûteux en temps de calcul lorsqu'ils sont appliqués à de très larges volumes de données, nous avons réduit la taille des ensembles d'entraînement et de test.

  Concrètement, nous avons sous-échantillonné en préservant la répartition des classes, en considérant 1 000 000 d'échantillons pour l'entraînement et 500 000 pour le test. Ces deux ensembles ont été utilisés pour entraîner les deux modèles dans un but de faisabilité d'entraînement et d'inférence tout en préservant la représentativité des données.

  - Sélection stratifiée de 1M d'échantillons pour l'ensemble d'entraînement et 500 000 échantillons pour l'ensemble de test :

```python
X_train_small, _, y_train_small, _ = train_test_split(
    X_train, y_train,
    train_size=1_000_000,
    stratify=y_train,
    random_state=42
)
```

```python
X_test_small, _, y_test_small, _ = train_test_split(
    X_test, y_test,
    train_size=500_000,
    stratify=y_test,
    random_state=42
)
```

Code 12 - Sélection d'échantillons

- Initialisation des modèles des modèles KNN et SVM puis entraînement avec mesure du temps d'exécution :

```python
start_train = time.time()
model = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    n_jobs=-1
)
model.fit(X_train_small, y_train_small)
end_train = time.time()
```

```python
start_train = time.time()
base_svc = LinearSVC(
    C=1.0,
    class_weight='balanced',
    dual=False,
    max_iter=1000,
    random_state=42,
    verbose=1
)
model = CalibratedClassifierCV(base_svc, n_jobs=-1, cv=3)
model.fit(X_train_small, y_train_small)
end_train = time.time()
```

Code 13 – Initialisation et entrainement KNN et SVM

- **Réalisation des prédictions des modèles avec mesure du temps**

Pour les modèles Random Forest, XGBoost et Naive Bayes, le même code de prédiction a été utilisé, appliqué sur l'ensemble de test complet « X_test ».

```python
start_pred = time.time()
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)[:, 1]
end_pred = time.time()
```

Code 14 - Prédiction RF, XGBoost, NB

Pour les modèles KNN et SVM, le même code de prédiction a été utilisé, appliqué sur l'ensemble de test réduit « X_test_small » préalablement défini.

```python
start_pred = time.time()
y_pred = model.predict(X_test_small)
y_pred_prob = model.predict_proba(X_test_small)[:, 1]
end_pred = time.time()
```

Code 15 - Prédiction KNN et SVM

4.5. Conception et modélisation de la version 2

4.5.1. Nettoyage des données :

Dans cette seconde version, le fichier ddos.csv, qui contient les données brutes, a fait l'objet d'un nettoyage élémentaire. L'objectif est d'améliorer la qualité du jeu de données tout en préservant au mieux les données de trafic légitime (BENIGN) qui sont sous-représentées.

- **Gestion des valeurs manquantes :**
  - ***Lignes avec valeurs manquantes :*** Pour commencer, il a fallu identifier les lignes où il y a des valeurs manquantes.

```
nb_lignes_manquantes = df.isnull().any(axis=1).sum()
print(f"Nombre total de lignes avec des valeurs manquantes : {nb_lignes_manquantes}")
```

```
Nombre total de lignes avec des valeurs manquantes : 249024
```

Code 16 - Lignes avec valeurs manquantes

Dans le but d'éviter trop de perte de lignes considérées comme bénignes, une condition a été posée avant de supprimer des lignes : seules les lignes comportant des valeurs manquantes et n'étant pas labellisées BENIGN seront supprimées.

```
mask_a_supprimer = df.isnull().any(axis=1) & (df['Label'] != 'BENIGN')
:
data = df[~mask_a_supprimer]
```

Code 17 - Suppression avec condition

Enfin, nous avons procédé à une vérification de la taille du dataframe.

```
print(f"Lignes supprimées : {mask_a_supprimer.sum()}")
print(f"Taille du DataFrame après suppression : {len(data)}")
```

```
Lignes supprimées : 248911
Taille du DataFrame après suppression : 49814201
```

Code 18 – Vérification de la taille du dataframe après suppression

- ***Colonnes avec valeurs manquantes :*** À ce stade, les seules lignes où sont recensées les valeurs manquantes demeurent celles labellisées comme BENIGN. L'intégrité de ces données légitimes est essentielle, et donc la logique a été de ne pas supprimer les lignes concernées mais de traiter le problème en supprimant les colonnes contenant des valeurs manquantes. Nous avons donc calculé le nombre de valeurs manquantes pour chaque colonne du Dataframe et avons filtré, pour garder celles qui en contenaient au moins une. Afin d'avoir un ensemble de données propre et d'atteindre l'objectif d'une absence de valeur manquante, les colonnes identifiées ont été supprimées tout en préservant la quantité de trafic bénin pour l'analyse.

```
missing_values = data.isnull().sum()
missing_percent = (missing_values / len(data)) * 100

missing_data = pd.DataFrame({
    'Valeurs manquantes': missing_values,
    'Pourcentage (%)': missing_percent
}).loc[missing_values > 0]

missing_data.sort_values(by='Valeurs manquantes', ascending=False)
```

```
[18]:
```

	Valeurs manquantes	Pourcentage (%)
Flow Bytes/s	113	0.000227

Code 19 - Affichage des colonnes avec valeurs manquantes

Une seule colonne contenant 113 valeurs manquantes a été identifiée et supprimée.

```
colonnes_a_supprimer = missing_data.index.tolist()

data = data.drop(columns=colonnes_a_supprimer)
```

Code 20 - Suppression des colonnes avec des valeurs manquantes

Une vérification a suivi pour confirmer la suppression.

```
print(f"Colonnes supprimées : {colonnes_a_supprimer}")
print(f"Nouvelles dimensions du DataFrame : {data.shape}")
```

```
Colonnes supprimées : ['Flow Bytes/s']
Nouvelles dimensions du DataFrame : (49814201, 87)
```

Code 21 - Vérification de la taille de dataframe

- **Gestion des valeurs infinies :**
  - ***Lignes avec des valeurs infinies :*** Une vérification a également été faite pour identifier les lignes qui contiennent des valeurs NaN ou infinies.

```
print(f"Lignes avec NaN : {data.isna().any(axis=1).sum()}")
print(f"Lignes avec infinis : {(data == np.inf).any(axis=1).sum() + (data == -np.inf).any(axis=1).sum()}")
```

```
Lignes avec NaN : 0
Lignes avec infinis : 1114325
```

Code 22 - Lignes avec des valeurs infinies

Après traitement des valeurs NaN, les valeurs infinies (inf+, inf-) ont été à leur tour l'objet d'attention. Un masque a été formé sur les lignes concernées comme suit :

```
mask_problemes = (data == np.inf).any(axis=1) | (data == -np.inf).any(axis=1)
mask_a_supprimer = mask_problemes & (data['Label'] != 'BENIGN')
```

Code 23 - Condition avant suppression des lignes avec des inf

Ces lignes concernées ont ensuite été supprimées sous la même condition que précédemment, ne pas supprimer les lignes labellisées BENIGN.

```
data = data[~mask_a_supprimer]
```

Code 24 - Suppression des lignes avec des inf

Vérification de la taille du dataframe après cette suppression :

```
print(f"Lignes supprimées : {mask_a_supprimer.sum()}")
print(f"Taille du DataFrame après nettoyage : {len(data)}")
```

```
Lignes supprimées : 1113887
Taille du DataFrame après nettoyage : 48700314
```

Code 25 - Vérification de la taille du dataframe

  - ***Colonnes avec des valeurs infinies*** : En ce qui concerne le reste des valeurs infinies, on est dans l'optique de maintenir ces données qui n'apparaissent que dans les lignes labellisées BENIGN, tout en respectant la spécificité des algorithmes d'apprentissage, on a choisi de purger le Dataframe de toutes les colonnes dans lesquelles on a détecté la présence de ces valeurs. Nous avons donc implémenté dans un code permettant de détecter les colonnes concernées.

```
has_inf = (data == np.inf).any() | (data == -np.inf).any()

problem_cols = data.columns[has_inf]

print("Colonnes avec infinis :", problem_cols.tolist())
```

```
Colonnes avec infinis : ['Flow Packets/s']
```

Code 26 - Détection des colonnes avec infinis

Identification des noms des colonnes contenant des valeurs infinies.

```
colonnes_a_supprimer = problem_cols.index.tolist()
```

Code 27 - Colonnes à supprimer

Suppression de ces colonnes du Dataframe original

```
data= data.drop(columns=colonnes_a_supprimer)
# 3. Vérification
print("Colonnes supprimées :", colonnes_a_supprimer)
print("Nouvelles dimensions du DataFrame :", data.shape)
```

```
Colonnes supprimées : ['Flow Packets/s']
Nouvelles dimensions du DataFrame : (48700314, 86)
```

Code 28 - Suppression des colonnes avec inf

Vérification finale de l'absence de valeurs infinies dans l'ensemble des données.

```
#vérification finale
print(f"Lignes avec NaN : {data.isna().any(axis=1).sum()}")
print(f"Lignes avec infinis : {(data == np.inf).any(axis=1).sum() + (data == -np.inf).any(axis=1).sum()}")
```

```
Lignes avec NaN : 0
Lignes avec infinis : 0
```

Code 29 - Vérification finale

- **Gestion des colonnes vides :** Détection des colonnes vides, c'est-à-dire celles dont toutes les valeurs sont égales à zéro.

```
colonnes_zero = [col for col in data.columns if (data[col] == 0).all()]

# Afficher les colonnes concernées
print("Colonnes avec uniquement des 0 / 0.0 :", colonnes_zero)
```

```
Colonnes avec uniquement des 0 / 0.0 : ['Bwd PSH Flags', 'Fwd URG Flag
s', 'Bwd URG Flags', 'FIN Flag Count', 'PSH Flag Count', 'ECE Flag Coun
t', 'Fwd Avg Bytes/Bulk', 'Fwd Avg Packets/Bulk', 'Fwd Avg Bulk Rate',
'Bwd Avg Bytes/Bulk', 'Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate']
```

Code 30 - Détection des colonnes vides

Suppression de ces colonnes :

```
data.drop(columns=colonnes_zero, inplace=True)
```

Code 31 - Suppression des colonnes vides

Vérification de la nouvelle taille du Dataframe

```
print("Nouvelles dimensions du DataFrame :", data.shape)
```

```
Nouvelles dimensions du DataFrame : (48700314, 74)
```

Code 32 - Vérification

- **Gestion des colonnes quasi-vides :** Les colonnes dans lesquelles plus de 90 % des valeurs sont nulles ou égales à zéro ont été identifiées comme non informatives et à supprimer.

```
seuil = 0.9
colonnes_zero_90 = [
    col for col in data.columns
    if (data[col] == 0).mean() >= seuil
]
print(f"Colonnes avec ≥{seuil:.0%} de zéros :", colonnes_zero_90)
```

```
Colonnes avec ≥90% de zéros : ['Total Backward Packets', 'Total Length
of Bwd Packets', 'Fwd Packet Length Std', 'Bwd Packet Length Max', 'Bwd
Packet Length Min', 'Bwd Packet Length Mean', 'Bwd Packet Length Std',
'Bwd IAT Total', 'Bwd IAT Mean', 'Bwd IAT Std', 'Bwd IAT Max', 'Bwd IAT
Min', 'Fwd PSH Flags', 'Bwd Header Length', 'Bwd Packets/s', 'Packet Le
ngth Std', 'Packet Length Variance', 'SYN Flag Count', 'RST Flag Coun
t', 'ACK Flag Count', 'URG Flag Count', 'CWE Flag Count', 'Down/Up Rati
o', 'Avg Bwd Segment Size', 'Subflow Bwd Packets', 'Subflow Bwd Bytes',
'Active Mean', 'Active Std', 'Active Max', 'Active Min', 'Idle Mean',
'Idle Std', 'Idle Max', 'Idle Min', 'SimillarHTTP']
```

Code 33 - Détection des colonnes quasi-vides

Suppression de ces colonnes :

```
data.drop(columns=colonnes_zero_90, inplace=True)
```

Code 34 - Suppression des colonnes quasi-vides

- **Gestion des colonnes quasi-constantes :** Les colonnes avec une faible variance (p.ex. : plus de 99% d'une même valeur), ayant été considérées comme quasi-constantes, n'apportent pas d'informations utiles à l'apprentissage.

  Identification des colonnes quasi-constantes :

```
print(f"Colonnes quasi-constantes (≥{seuil:.0%} de valeurs identiques):")
print(pd.DataFrame({
    'Colonne': quasi_constantes,
    'Valeur dominante': [data[col].mode()[0] for col in quasi_constantes],
    'Fréquence': [data[col].value_counts(normalize=True).max() for col in quasi_constantes]
}))
```

```
Colonnes quasi-constantes (≥90% de valeurs identiques):
                  Colonne Valeur dominante  Fréquence
0              Source IP       172.16.0.5   0.998508
1         Destination IP     192.168.50.1   0.998510
2                Protocol               17   0.961944
3   Init_Win_bytes_forward               -1   0.961996
4  Init_Win_bytes_backward               -1   0.993542
5                 Inbound                1   0.998708
```

Code 35 - Détection des colonnes quasi-constantes

Suppression des colonnes quasi-constantes :

```
data = data.drop(columns=quasi_constantes)
```

Code 36 - Suppression des colonnes quasi-constantes

Vérification de la nouvelle taille du Dataframe :

```
print(f"\nDimensions: {data.shape}")
```

```
Dimensions: (48700314, 31)
```

Code 37 - Vérification

- **Gestion des colonnes redondantes :** Identification des colonnes redondantes :

```
cols = data.columns
duplicate_cols = []

for i in range(len(cols)):
    for j in range(i+1, len(cols)):
        if data[cols[i]].equals(data[cols[j]]):
            duplicate_cols.append(cols[j])
```

Code 38 - Détection des colonnes redondantes

Suppression de colonnes dupliquées pour ne conserver qu'une seule occurrence de chaque colonne redondante.

```
data = data.drop(columns=duplicate_cols)
```

Code 39 - Suppression des colonnes redondantes

Affichage des colonnes supprimées et vérification de la nouvelle taille du Dataframe :

```
print(f"Colonnes redondantes détectées : {duplicate_cols}")
print(f"Shape avant/apres : {data.shape}")
```

```
Colonnes redondantes détectées : ['Subflow Fwd Packets', 'Fwd Header Le
ngth.1']
Shape avant/apres : (48700314, 37)
```

Code 40 - Vérification

- **Métadonnées techniques :** Il a été observé que certaines colonnes sont constituées de métadonnées techniques (ex. : index, timestamps, identifiants techniques) qui n'ont pas une grande valeur dans la détection d'attaques. Ces colonnes ont été retirées du jeu de données.

```
data.drop(columns=['Unnamed: 0', 'Flow ID', 'Timestamp'], inplace=True)
```

Code 41 - Suppression des métadonnées

Vérification finale de la taille du Dataframe :

```
data.shape
```

```
[55]:

(48700314, 28)
```

Code 42 - Taille du dataframe

4.5.2. Encodage binaire pour la colonne label :

Afin de simplifier le problème en une tâche de classification binaire, la colonne Label a été modifiée comme suit : Les lignes avec le label BENIGN ont été encodées avec la valeur 0, et les autres labels (correspondant à une attaque DDoS) ont été encodés avec la valeur 1.

```
from sklearn.preprocessing import LabelEncoder
data['Label'] = LabelEncoder().fit_transform(data['Label'])

data['Label'] = np.where(data['Label'] == 0, 0, 1)
```

Code 43 - Encodage binaire

Affichage des valeurs uniques présentes dans la colonne Label et leur répartition :

```
print(data["Label"].value_counts())
```

```
Label
1    48643451
0       56863
Name: count, dtype: int64
```

Code 44 - Exploration de la colonne Label

Enfin, l'ensemble de données nettoyé et encodé et qui contient 28 caractéristiques, a été enregistré dans un fichier « data.csv », en vue d'une utilisation ultérieure pour l'entraînement et l'évaluation des modèles.

```
chemin = "/content/data.csv"
data.to_csv(chemin, index=False)
print(f"Fichier sauvegardé : {chemin}")
```

```
Fichier sauvegardé : /content/data.csv
```

Code 45 - Sauvegarde du dataframe dans data.csv

4.5.3. Partition des données (train/test split) :

Une séparation des données a été réalisée, en affectant 70 % des échantillons pour l'ensemble d'entraînement et 30 % pour l'ensemble de test, tout en respectant la proportion des classes Attack et BENIGN au sein de chacun des ensembles, afin de préserver la représentativité des données pour l'apprentissage supervisé.

```
X = data.drop('Label', axis=1)
y = data['Label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)
```

Code 46 - Séparation train/test

4.5.4. Normalisation :

Dans une première phase, un contrôle a été mis en œuvre afin de repérer les colonnes dans lesquelles la valeur maximale au sein de l'ensemble de test excède celle rencontrée au sein de l'ensemble d'entraînement.

```
test_higher = X_test.max() > X_train.max()
print("Colonnes avec max(test) > max(train):\n", test_higher[test_higher].index.tolist())
```

```
Colonnes avec max(test) > max(train):
 ['Fwd Packet Length Min', 'Flow IAT Std']
```

Code 47 - Colonnes où le test dépasse le train

Les noms des colonnes concernées ont été consignés dans une liste dédiée :

```
problem_cols = ['Fwd Packet Length Min', 'Flow IAT Std']
```

Code 48 - Liste des colonnes problématiques

Puis, les minimums et maximums globaux ont été calculés à partir des ensembles d'entraînement et de test combinés dans le but d'obtenir des bornes cohérentes pour l'ensemble des données.

```
global_min = np.minimum(X_train.min(), X_test.min())
global_max = np.maximum(X_train.max(), X_test.max())
```

Code 49 - Calcul des min max globaux

On fait une initialisation standard pour l'objet MinMaxScaler :

```
scaler = MinMaxScaler()
scaler.fit(X_train)
```

Code 50 - Initialisation du normaliseur Min-Max

Ensuite forcer les min/max globaux pour les colonnes problématiques :

```
scaler.data_min_[X_train.columns.get_indexer(problem_cols)] = global_min[problem_cols].values
scaler.data_max_[X_train.columns.get_indexer(problem_cols)] = global_max[problem_cols].values
```

Code 51 - Forcement des min/max pour les colonnes problématiques

Enfin, on fait la transformation sur train et test :

```
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

Code 52 - Transformation sur train et test

Sauvegarde du normaliseur Min-Max utilisé pour cette version :

```
scaler_path = 'new_scaler_v2.pkl'
dump(scaler, scaler_path)
```

Code 53 - Sauvegarde du normaliseur

4.5.5. Equilibrage de classe en utilisant SMOTE :

Un découpage additionnel du jeu de données d'entraînement a été réalisé pour la création d'un sous-ensemble permettant l'optimisation du seuil de décision. Ce partitionnement a donc permis de définir deux ensembles X_train_base, contenant 27 272 175 données avec 27 caractéristiques, et X_val, avec 6 818 044 données également avec 27 caractéristiques.

```
X_train_base, X_val, y_train_base, y_val = train_test_split(
    X_train_normalized, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train
)
```

Code 54 - Séparation pour optimisation du seuil

L'algorithme SMOTE a ensuite été appliqué sur l'ensemble X_train_base pour corriger le déséquilibre initial entre les classes.

```
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_base, y_train_base)
```

Code 55 - Application de l'algorithme SMOTE

Vérification de la taille de l'ensemble de données d'entrainement après suréchantillonnage :

```
X_train_resampled.shape
```

```
[25]:

(54480664, 27)
```

Code 56 - Vérification après suréchantillonnage

Vérification de la colonne label avant et après SMOTE : on observe une augmentation significative du nombre d'échantillons appartenant à la classe minoritaire (label 0), confirmant ainsi l'efficacité du suréchantillonnage.

```
Distribution avant SMOTE :
Label
1    27240332
0       31843
Name: count, dtype: int64

Distribution après SMOTE :
Label
1    27240332
0    27240332
Name: count, dtype: int64
```

Figure 4.1 - Distribution de la colonne label avant et après SMOTE

Visualisation graphique de la distribution des classes après application de SMOTE :

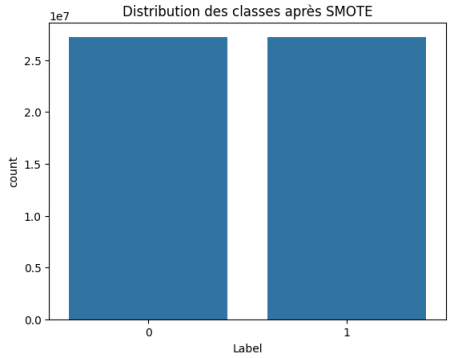

Figure 4.2 - Distribution des classes après SMOTE

4.6. Modélisation de la version 2

4.6.1. Entrainement des modèles :

- **Random Forest, XGBoost et Naïve Bayes :** Initialisation des modèles suivie de ses entraînements avec mesure du temps d'exécution.

```
model = RandomForestClassifier(
    max_depth=5,
    n_estimators=8,
    min_samples_split=50,
    min_samples_leaf=5,
    max_features='sqrt',
    random_state=42,
    n_jobs=-1,
    verbose=1
)
start_train = time.time()
model.fit(X_train_resampled, y_train_resampled)
end_train = time.time()
```

```
model = XGBClassifier(
    max_depth=5,
    n_estimators=8,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    min_child_weight=5,
    gamma=0,
    reg_alpha=0,
    reg_lambda=1,
    random_state=42,
    n_jobs=-1,
    verbosity=1,
)
start_train = time.time()
model.fit(X_train_resampled, y_train_resampled)
end_train = time.time()
```

```
model = GaussianNB()
start_train = time.time()
model.fit(X_train_resampled, y_train_resampled)
end_train = time.time()
```

Code 57 - Initialisation et entrainement RF, XGBoost, NB

- **KNN et SVM :** nous avons sous-échantillonné en préservant la répartition des classes, en considérant 1M d'échantillons pour l'entraînement et 500 000 pour le test et 250 000 échantillons pour la validation. Ces ensembles ont été utilisés pour entraîner les deux modèles dans un but de faisabilité d'entraînement et d'inférence tout en préservant la représentativité des données.
  - Sélection stratifiée d'échantillons pour l'ensemble d'entraînement, de test et de validation :

```
X_train_res_small, _, y_train_res_small, _ = train_test_split(
    X_train_resampled, y_train_resampled,
    train_size=1_000_000,
    stratify=y_train_resampled,
    random_state=42
)
```

```
X_test_normalized_small, _, y_test_small, _ = train_test_split(
    X_test_normalized, y_test,
    train_size=500_000,
    stratify=y_test,
    random_state=42
)
```

```
X_val_small, _, y_val_small, _ = train_test_split(
    X_val, y_val,
    train_size=250_000,
    stratify=y_val,
    random_state=42
)
```

Code 58 - Sélection d'échantillons pour entrainement, test et validation

  - Initialisation des modèles des modèles KNN et SVM puis entraînement avec mesure du temps d'exécution :

```
model = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    n_jobs=-1
)
start_train = time.time()
model.fit(X_train_res_small, y_train_res_small)
end_train = time.time()
```

```
model = LinearSVC(
    C=1.0,
    class_weight='balanced',
    random_state=42,
    max_iter=1000,
    dual=False,
    verbose=1
)
start_train = time.time()
model.fit(X_train_res_small, y_train_res_small)
end_train = time.time()
```

Code 59 - Initialisation et entrainement KNN et SVM

- **Optimisation du seuil de décision :** pour la classification binaire en maximisant le score F1 sur l'ensemble de validation non rééquilibré.
  - Pour Random Forest, XGBoost et Naive Bayes le code suivant a été utilisé :

```
y_proba_val = model.predict_proba(X_val)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_val, y_proba_val, pos_label=1)
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-9)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]
```

Code 60 - Optimisation du seuil pour RF, XGBoost et NB

  - Pour KNN le code suivant a été utilisé :

```
y_proba_val = model.predict_proba(X_val_small)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_val_small, y_proba_val, pos_label=1)
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-9)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]
```

Code 61 - Optimisation du seuil pour KNN

  - Pour SVM le code suivant a été utilisé car LinearSVC n'a pas predict_proba par défaut, on utilise decision_function

```
decision_scores_val = model.decision_function(X_val_small)
precision, recall, thresholds = precision_recall_curve(y_val_small, decision_scores_val, pos_label=1)
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-9)
optimal_idx = np.argmax(f1_scores)
optimal_threshold = thresholds[optimal_idx]
```

Code 62 - Optimisation du seuil pour SVM

- **Réalisation des prédictions des modèles avec mesure du temps**

  Pour les modèles Random Forest, XGBoost et Naive Bayes, le même code de prédiction a été utilisé, appliqué sur l'ensemble de test complet « X_test_normalized ».

```
start_pred = time.time()
y_pred_prob = model.predict_proba(X_test_normalized)[:, 1]
y_pred = (y_pred_prob > optimal_threshold).astype(int)
end_pred = time.time()
```

Code 63 - Prédiction pour RF, XGBoost, NB

  Pour le modèle KNN, le code de prédiction suivant a été utilisé, appliqué sur l'ensemble de test réduit « X_test_normalized_small » préalablement défini.

```
start_pred = time.time()
y_pred_prob = model.predict_proba(X_test_normalized_small)[:, 1]
y_pred = (y_pred_prob > optimal_threshold).astype(int)
end_pred = time.time()
```

Code 64 - Prédiction pour KNN

  Pour SVM le code de prédiction suivant a été utilisé, appliqué sur l'ensemble de test réduit « X_test_normalized_small » préalablement défini.

```
start_pred = time.time()
decision_scores_test = model.decision_function(X_test_normalized_small)
y_pred = (decision_scores_test > optimal_threshold).astype(int)
end_pred = time.time()
```

Code 65 - Prédiction pour SVM

4.7. Conception de la version 3

4.7.1. Réduction de dimensionnalité par sélection de caractéristiques

Dans la version actuelle, nous repartons du fichier « data.csv » qui a été obtenu après nettoyage et encodage de la version 2. L'objectif est de réduire la dimensionnalité du jeu de données, en identifiant et en supprimant les caractéristiques linéairement dépendantes a 100%.

Pour cela nous allons sélectionner un échantillon aléatoire de 50 000 lignes, qui sera uniquement utilisé pour visualiser les relations linéaires entre toutes les paires de variables numériques. Le jeu de données est composé de 28 colonnes, ce qui implique 378 paires uniques (calculées avec la formule n(n−1)/2).

Une boucle produite pour chaque paire de colonnes : un graphe hexbin montrant la densité des points (sous forme d'hexagones colorés), une courbe de régression linéaire via regplot, et le coefficient de corrélation de Pearson, mis en légende dans le titre du graphe.

- **Détection des paires fortement corrélées :** Un script permet l'automatisation de la détection des colonnes, dans les paires ayant une corrélation linéaire quasi-parfaite (supérieure ou égale à 0,9999). Chaque paire de colonnes détectée est intégrée à une liste, nommée perfect_pairs.

```
perfect_pairs = []
for col1, col2 in combinations(sample_data.columns, 2):
    corr = np.corrcoef(sample_data[col1], sample_data[col2])[0, 1]
    if abs(corr) >= 0.9999 or round(corr, 2) == 1.00:
        perfect_pairs.append((col1, col2, corr))
```

Code 66 - Détection des paires fortement corrélées

Sur le total des 378 graphiques produits, 32 montrent que des paires sont en relation linéaire parfaite. La figure 4.3 qui suit est, justement, l'un de ceux-là, représentant les valeurs de deux variables très corrélées.

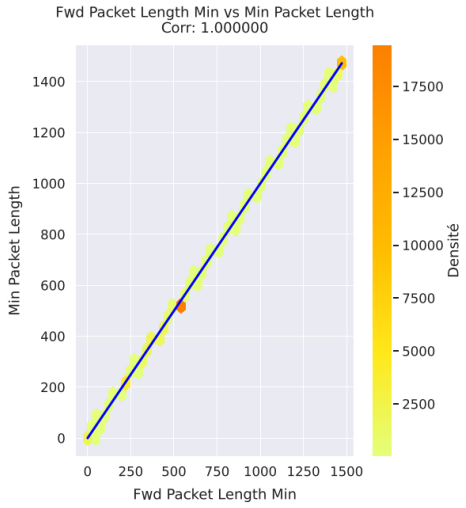

Figure 4.3 – Relation linéaire entre deux colonnes fortement corrélées

- **Comptage des occurrences des colonnes corrélées :** calculer combien de fois chaque colonne figure dans une relation de corrélation parfaite dans la liste perfect_pairs

```
column_counts = Counter()
for col1, col2, _ in perfect_pairs:
    column_counts[col1] += 1
    column_counts[col2] += 1
```

Code 67 - Calcul des occurrences des colonnes

- **Visualisation des fréquences :** Un histogramme est ensuite produit pour visualiser sous forme graphique la fréquence d'apparition des différentes colonnes dans les relations de redondance, qui permet rapidement de localiser visuellement la colonne la plus « centrale ».

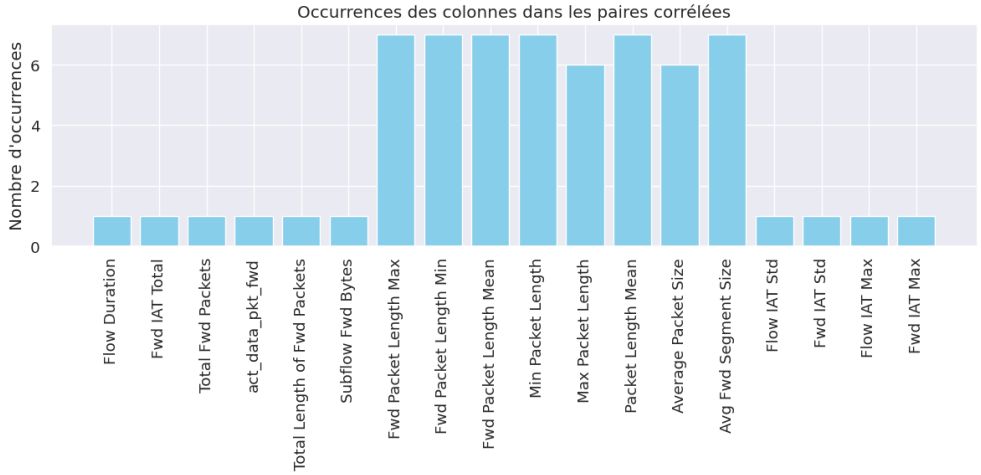

Figure 4.4 - Histogramme des occurrences des colonnes dans les paires corrélées

- **Sélection de la colonne à conserver :** La colonne apparaissant le plus fréquemment dans les paires est considérée comme la plus représentative. Elle sera conservée.

```
keep_column = max(column_counts, key=column_counts.get)
print(f"Colonne à conserver : {keep_column} (apparaît {column_counts[keep_column]} fois)")
```

Code 68 - Identification de la colonne à conserver

- **Identification des colonnes à supprimer :** toutes les colonnes corrélées à la colonne conservée sont listées pour suppression, afin de réduire le nombre de dimensions tout en conservant de l'information.

```
columns_to_drop = set()
for col1, col2, _ in perfect_pairs:
    if col1 == keep_column:
        columns_to_drop.add(col2)
    elif col2 == keep_column:
        columns_to_drop.add(col1)

print("Colonnes à supprimer :", columns_to_drop)
```

Code 69 - Identification des colonnes à supprimer

Dans notre cas la colonne à conserver est « Fwd Packet Length Max » avec un nombre d'occurrence égale à 7, On suppose ici que la colonne dominante contient déjà toute l'information de ses colonnes partenaires, donc on peut les supprimer, donc les colonnes à supprimer sont « Average Packet Size, Avg Fwd Segment Size, Fwd Packet Length Min, Min Packet Length, Fwd Packet Length Mean, Packet Length Mean, Max Packet Length »

```
Colonne à conserver : Fwd Packet Length Max (apparaît 7 fois)
Colonnes à supprimer : {'Average Packet Size', 'Avg Fwd Segment Size', 'Fwd Packet Length Mi
n', 'Min Packet Length', 'Fwd Packet Length Mean', 'Packet Length Mean', 'Max Packet Lengt
h'}
```

Figure 4.5 - Liste des colonnes à conserver et à supprimer

Suppression de ces colonnes :

```
cleaned_data = data.drop(columns=columns_to_drop)
```

Code 70 - Suppression des colonnes à supprimer

- **Gestion des colonnes avec nombre d'occurrence = 1 :** Apres la suppression des colonnes déjà identifiées, il nous reste que les colonnes avec le même nombre d'occurrence qui est égal à 1 donc on a décidé de garder pour ces paires les colonnes de l'axe X et supprimer celle de l'axe Y, simplifiant ainsi la structure des données sans perte d'information.

```
columns_to_drop.update({
    'Fwd IAT Total',
    'act_data_pkt_fwd',
    'Subflow Fwd Bytes',
    'Fwd IAT Std',
    'Fwd IAT Max'
})

# Nettoyage final
final_data = data.drop(columns=columns_to_drop)
```

Code 71 - Suppression des colonnes avec nombre d'occurrence =1

- **Vérification :** Après la sélection des caractéristiques, une vérification de la dimension du dataframe a été effectuée afin de confirmer la réduction du nombre de colonnes.

```
final_data.shape
```

```
(48700314, 16)
```

Code 72 - Vérification de la forme du dataframe

Enregistrement du dataframe final dans un fichier CSV nommé « data_cleaned.csv » :

```
data.to_csv('data_cleaned.csv', index=False)
```

Code 73 - Enregistrement du dataframe dans un CSV

- **Dataframe final obtenu pour la version 3 :**

1	Source Port	int64
2	Destination Port	int64
3	Flow Duration	int64
4	Total Fwd Packets	int64
5	Total Length of Fwd Packets	float64
6	Fwd Packet Length Max	float64
7	Flow IAT Mean	float64
8	Flow IAT Std	float64
9	Flow IAT Max	float64
10	Flow IAT Min	float64
11	Fwd IAT Mean	float64
12	Fwd IAT Min	float64
13	Fwd Header Length	int64
14	Fwd Packets/s	float64
15	min_seg_size_forward	int64
16	Label	int64

Table 4.1 – Dataframe obtenu par la version 3

4.7.2. Partition des données (train/test split)

Les données ont subi une séparation, avec 70 % des échantillons affectés au jeu d'entraînement et 30% au jeu de test, tout en respectant la proportion des classes Attack et BENIGN dans chacun des jeux, afin de garantir la représentativité.

```
X = data.drop('Label', axis=1)
y = data['Label']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)
```

Code 74 - Séparation des données d'entrainement et de test

4.7.3. Normalisation

Dans cette première étape, un dispositif de vérification a été constitué de sorte à repérer les colonnes dans lesquelles le maximum repéré dans l'ensemble de test dépasse celui trouvé dans l'ensemble d'entraînement.

```
test_higher = X_test.max() > X_train.max()
print("Colonnes avec max(test) > max(train):\n", test_higher[test_higher].index.tolist())
```

```
Colonnes avec max(test) > max(train):
 ['Flow IAT Std']
```

Code 75 - Identification des colonnes avec max-test supérieur à max-train

Les noms des colonnes concernées ont été consignés dans une liste dédiée : dans notre cas on a trouvé une seule colonne « Flow IAT Std »

```
problem_cols = ['Flow IAT Std']
```

Code 76 - Déclaration de la liste des colonnes problématiques

Ensuite les minimums et maximums globaux ont été calculés sur les ensembles d'entraînement et de test combinés afin de fournir des bornes cohérentes à l'ensemble des données.

```
global_min = np.minimum(X_train.min(), X_test.min())
global_max = np.maximum(X_train.max(), X_test.max())
```

Code 77 - Calcul des min-max globaux

Puis on instancie un MinMaxScaler standard et on l'entraîne sur l'ensemble d'entrainement uniquement :

```
scaler = MinMaxScaler()
scaler.fit(X_train)
```

Code 78 - Initialisation de MinMaxScaler

Ensuite forcer les min/max globaux pour les colonnes problématiques : On modifie directement les attributs internes du scaler (data_min_, data_max_) pour les colonnes problématiques. Cela garantit que la normalisation appliquera les mêmes bornes [min, max] à X_train et X_test pour ces colonnes.

```
scaler.data_min_[X_train.columns.get_indexer(problem_cols)] = global_min[problem_cols].values
scaler.data_max_[X_train.columns.get_indexer(problem_cols)] = global_max[problem_cols].values
```

Code 79 - Forcement des min-max globaux sur les colonnes problématiques

Enfin, on fait la transformation sur train et test : on obtient X_train_normalized et X_test_normalized qui sont dans le même espace, sans risque de valeurs hors limites.

```
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

Code 80 - Transformation des données de test et d'entrainement

Sauvegarde du normaliseur Min-Max utilisé pour cette version :

```
scaler_path = 'v3_scaler_final.pkl'
dump(scaler, scaler_path)
```

Code 81 - Sauvegarde du normaliseur Min-Max

4.7.4. Equilibrage de classes avec ADASYN

- **Application de l'algorithme ADASYN :** cet algorithme est appliqué aux données d'entraînement normalisées « X_train_normalized » et aux étiquettes associées « y_train », permettant de générer un nouvel ensemble de données équilibré, où les classes minoritaires sont suréchantillonnées via la création d'exemples synthétiques.

```
adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train_normalized, y_train)
```

Code 82 - Application de l'algorithme ADASYN

- **Comparaison des proportions des labels avant et après suréchantillonnage :** L'utilisation d'ADASYN a équilibré la distribution des classes, augmentant significativement le nombre d'exemples pour la classe minoritaire tout en maintenant le nombre d'exemples de la classe majoritaire.

```
Distribution avant adasyn :
Label
1    34050415
0       39804
Name: count, dtype: int64
```

```
Distribution après adasyn :
Label
0    34050729
1    34050415
Name: count, dtype: int64
```

Figure 4.6 – Distribution dans la colonne Label avant et après ADASYN

- **Visualisation graphique de la distribution des classes après application d'ADASYN :**

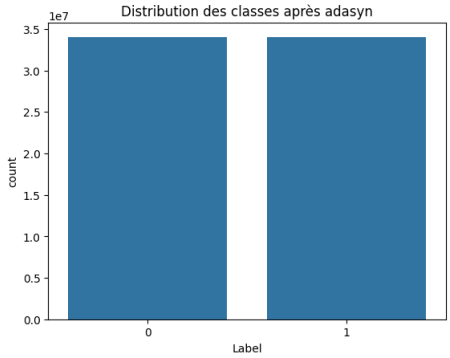

Figure 4.7 – Distribution des classes après ADASYN

- **Vérification de la taille de l'ensemble d'entrainement suréchantillonnée :**

```
X_train_adasyn.shape
```

```
(68101144, 15)
```

```
y_train_adasyn.shape
```

```
(68101144,)
```

Code 83 - Vérification de la taille après suréchantillonnage

4.8. Modélisation de la version 3

4.8.1. Entrainement des modèles :

- **Random Forest et Naive Bayes :** Initialisation des modèles avec des paramètres optimisés pour vitesse/performance, suivie de ses entraînements avec mesure du temps d'exécution.

```
model = RandomForestClassifier(
    n_estimators=8,
    max_depth=15,
    min_samples_split=50,
    min_samples_leaf=5,
    n_jobs=-1,
    random_state=42,
    class_weight='balanced',
    verbose=1
)
start_train = time.time()
model.fit(X_train_adasyn, y_train_adasyn)
end_train = time.time()
```

```
model = GaussianNB(
    priors=None,
    var_smoothing=1e-9
)
start_train = time.time()
model.fit(X_train_adasyn, y_train_adasyn)
end_train = time.time()
```

Code 84 - Initialisation et entrainement RF, NB

- **XGBoost :** Initialisation du modèle suivie de son entraînement avec mesure du temps d'exécution.

```
model = XGBClassifier(
    n_estimators=8,
    max_depth=15,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic',
    n_jobs=-1,
    random_state=42,
    scale_pos_weight=len(y_train_adasyn[y_train_adasyn==0])/len(y_train_adasyn[y_train_adasyn==1]),
    verbosity=1,
    gamma=0,
    eval_metric='logloss'
)
start_train = time.time()
model.fit(X_train_adasyn, y_train_adasyn,
          eval_set=[(X_test_normalized, y_test)],
          verbose=True)
end_train = time.time()
```

Code 85 - Initialisation et entrainement XGBoost

- **KNN et SVM :** Étant donné que le coût des algorithmes KNN et SVM augmente avec la taille des volumes de données, nous avons réduit la taille des ensembles d'apprentissage et de test. Pour ce faire, nous avons procédé à un sous-échantillonnage stratifié afin de préserver la distribution des classes : 1 000 000 d'échantillons pour l'entraînement et 500 000 pour le test, utilisés pour les deux modèles.

```
X_train_adasyn_small, _, y_train_adasyn_small, _ = train_test_split(
    X_train_adasyn, y_train_adasyn,
    train_size=1_000_000,
    stratify=y_train_adasyn,
    random_state=42
)
```

```
X_test_normalized_small, _, y_test_small, _ = train_test_split(
    X_test_normalized, y_test,
    train_size=500_000,
    stratify=y_test,
    random_state=42
)
```

Code 86 - Sélection d'échantillons pour entrainement et test

- Initialisation des modèles des modèles KNN et SVM puis entraînement avec mesure du temps d'exécution :

```
model = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    n_jobs=-1
)
start_train = time.time()
model.fit(X_train_adasyn_small, y_train_adasyn_small)
end_train = time.time()
```

```
model = LinearSVC(
    C=1.0,
    class_weight='balanced',
    random_state=42,
    max_iter=1000,
    dual=False,
    verbose=1
)
start_train = time.time()
model.fit(X_train_adasyn_small, y_train_adasyn_small)
end_train = time.time()
```

Code 87 - Initialisation et entrainement pour KNN et SVM

- **Réalisation des prédictions des modèles avec mesure du temps**

  Pour les modèles Random Forest, XGBoost et Naive Bayes, le même code de prédiction a été utilisé, appliqué sur l'ensemble de test complet « X_test_normalized».

```
start_pred = time.time()
y_pred = model.predict(X_test_normalized)
y_pred_prob = model.predict_proba(X_test_normalized)[:, 1]
end_pred = time.time()
```

Code 88 - Prédiction pour RF, XGBoost et NB

- Pour le modèle KNN, le code de prédiction suivant a été utilisé, appliqué sur l'ensemble de test réduit « X_test_normalized_small » préalablement défini.

```
start_pred = time.time()
y_pred = model.predict(X_test_normalized_small)
y_pred_prob = model.predict_proba(X_test_normalized_small)[:, 1]
end_pred = time.time()
```

Code 89 - Prédiction pour KNN

- Pour SVM le code de prédiction suivant a été utilisé, appliqué sur l'ensemble de test réduit « X_test_normalized_small » préalablement défini.

```
start_pred = time.time()
y_pred = model.predict(X_test_normalized_small)
decision_scores = model.decision_function(X_test_normalized_small)
from scipy.special import expit
y_pred_prob = expit(decision_scores)
end_pred = time.time()
```

Code 90 - Prédiction pour SVM

4.9. Conception de la version 4

4.9.1. Sélection des caractéristiques par pente relative

Dans le cadre d'un perfectionnement continu de notre processus de sélection de caractéristiques, cette version représente une nouvelle étape par rapport à la version 3. En effet, on prend à présent comme point de départ le fichier « data_cleaned.csv », produit suite à la version 3, et nous y appliquons une méthode heuristique de réduction de dimension fondée sur l'analyse des angles des droites de régression issues des paires de variables. L'objectif est de faire le tri entre les variables redondantes ou dépendantes pour conserver les caractéristiques les plus informatives.

La stratégie mise en œuvre consiste dans le fait de réaliser une régression linéaire pour chacune des combinaisons de variables présente dans le jeu de données considéré. Sur la base de la pente de chacune des droites de régression ainsi obtenues, nous calculons l'angle qu'elle forme avec l'axe des abscisses au moyen de la fonction arc tangente, et cette valeur sert un critère d'exclusion :

Si l'angle est inférieur à 45°, c'est que la variable en abscisse (X) est mieux expliquée par celle en ordonnée (Y). La variable Y est donc supprimée.

À l'inverse, si l'angle est supérieur ou égal à 45°, c'est la variable X qui est supprimée.

Le jeu de données comporte 16 variables, ce qui génère 120 paires uniques (selon la formule combinatoire $n(n-1)/2$).

Tout d'abord, nous avons élaboré l'ensemble des graphes de dispersion concernant toutes les paires de variables de manière à pouvoir visualiser leur relation linéaire, mais nous ne retiendrons pour analyser que les paires offrant une relation linéaire marquée.

La figure suivante montre le code utilisé pour générer les graphes :

```
sns.regplot(data=data, x=col1, y=col2, ax=ax,
            scatter=False, ci=None,
            line_kws={'color': 'blue', 'linewidth': 2})
```

Code 91 - Traçage des graphes pour les paires via regplot

Nous avons filtré parmi les 120 graphes que les graphes qui montre une distribution linéaire, comme le montre la figure suivante :

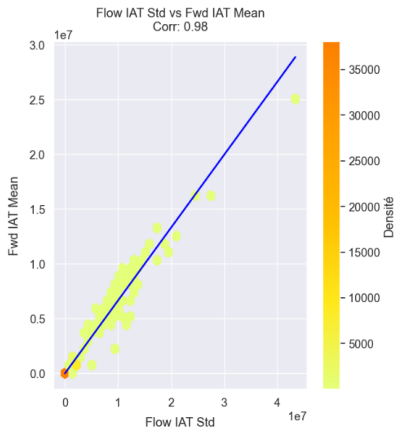

Figure 4.8 - Visualisation de la distribution des données pour une paire de colonne

Ces paires particulièrement étudiées sont les suivantes :

Flow Duration
Total Fwd Packets
Total Length of Fwd Packets
Flow IAT Mean
Flow IAT Std
Flow IAT Max
Flow IAT Min
Fwd IAT Mean
Fwd IAT Min

Table 4.2 - Colonnes linéairement dépendantes à étudier

Pour faire une analyse des droites de régression plus claire pour ces paires nous allons donc calculer les régressions linéaires pour estimer Y en fonction de X, donc on a appliqué un modèle de régression linéaire.

Les métriques utilisées sont : la pente qui repressente la variation de Y en fonction de X, le coefficient de détermination R^2 qui mesure la proportion de la variance expliquée par le modèle, et au final l'indicateur le plus important est l'angle en degrés qui est l'angle entre la droite de régression et l'axe des abscisses calculé à partir de la pente. Le code utilisé est le suivant :

```
for col_x, col_y in combinations(columns, 2):
    x = data[col_x].values.reshape(-1, 1)
    y = data[col_y].values

    model = LinearRegression()
    model.fit(x, y)

    slope = model.coef_[0]
    r_squared = model.score(x, y)
    angle_rad = math.atan(slope)
    angle_deg = math.degrees(angle_rad)

    results.append({
        "X": col_x,
        "Y": col_y,
        "Slope": slope,
        "Angle (deg)": angle_deg,
        "R²": r_squared
    })
```

Code 92 - Exploration des Relations Linéaires : Pentes et Coefficient de Détermination

Ainsi on n'a pas suffi par les calculs numériques seulement on a encore tracé des graphiques pour visualiser de plus près la relation entre X et Y et la droite formée. Et comme résultat voici les graphiques pertinents :

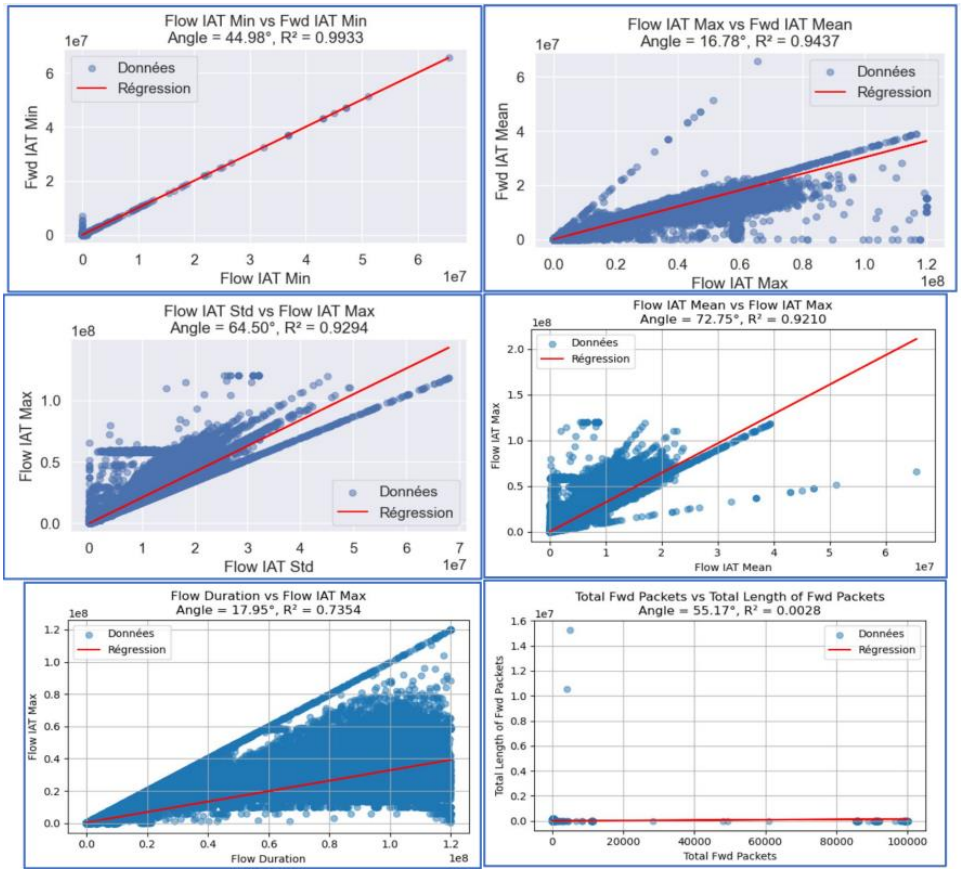

Figure 4.9 - Visualisation des relations entre les intervalles d'arrivée et les paquets dans les flux de données

L'analyse de l'angle des droites de régression pour ces paires nous a permis de détecter des redondances, conduisant à l'élimination de variables jugées dépendantes, tandis que d'autres, jugées indépendantes, ont été retenues pour les analyses ultérieures. Le tableau suivant montre les variables éliminées et celles conservées :

Variables éliminées	Variables retenues
Fwd IAT Min	Source Port
Fwd IAT Mean	Destination Port
Flow IAT Std	Flow Duration
Flow IAT Mean	Total Fwd Packets
Flow IAT Max	Fwd Packet Length Max
Total Length of Fwd Packets	Flow IAT Min
	Fwd Header Length
	Fwd Packets/s
	min_seg_size_forward
	Label

Table 4.3 - Variables Éliminées et Retenues

Cette sélection vise à améliorer la performance des modèles de classification tout en réduisant le risque de surapprentissage dû à la multi colinéarité entre variables.

4.9.2. Partition des données (train/test split)

Les données ont été issues d'une séparation en deux jeux de données, est attribué 70 % à l'entraînement et 30 % au test, en préservant les proportions des classes Attack, et BENIGN dans les deux jeux de données, pour garantir leur représentativité.

```
X = data.drop('Label', axis=1)
y = data['Label']
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)
```

Code 93 - Séparation en ensembles d'entraînement et de test

4.9.3. Normalisation

Nous avons instancié un MinMaxScaler classique pour l'entraînement sur l'ensemble des données d'entraînement uniquement. Ensuite, on effectue la transformation sur train et test : nous obtenons « X_train_normalized » et « X_test_normalized » qui se situent dans le même espace.

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

Code 94 - Transformation des Données : Normalisation des Caractéristiques

Sauvegarde du normaliseur Min-Max utilisé pour cette version :

```
scaler_path = 'v4_scaler_final.pkl'
dump(scaler, scaler_path)
```

Code 95 - Sauvegarde du MinMaxScaler dans un Fichier PKL

4.9.4. Equilibrage de classes avec ADASYN

- **Application de l'algorithme ADASYN :** cet algorithme est appliqué aux données d'apprentissage «X_train_normalized » normalisées et aux étiquettes « y_train » correspondantes, afin de donner lieu à un nouvel ensemble de données équilibré par suréchantillonnage des classes minoritaires via la création d'exemples synthétiques.

```
adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train_normalized, y_train)
```

Code 96 - Équilibrage des Données avec ADASYN pour l'Ensemble d'Entraînement

- **Comparaison des proportions des labels avant et après suréchantillonnage :** L'outil ADASYN a corrigé la distribution des classes, en accroissant le nombre d'exemples pour la classe minoritaire, tout en préservant le nombre d'exemples pour la classe majoritaire.

```
Distribution avant adasyn :
Label
1    34050415
0       39804
Name: count, dtype: int64
```

```
Distribution après adasyn :
Label
0    34050653
1    34050415
Name: count, dtype: int64
```

Figure 4.10 - Distribution des Labels avant et après l'utilisation de ADASYN

- **Visualisation graphique de la distribution des classes après application d'ADASYN :**

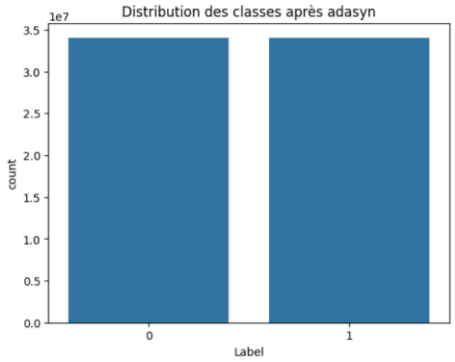

Figure 4.11 - Distribution des Classes Post-ADASYN

- **Vérification de la taille de l'ensemble d'entrainement suréchantillonnée :**

```
X_train_adasyn.shape
[21]:
(68101068, 9)
```

```
y_train_adasyn.shape
[22]:
(68101068,)
```

Code 97 - Dimensions des ensembles de données X et y après ADASYN

4.10. Modélisation de la version 4

4.10.1. Entrainement des modèles

Concernant la phase d'initialisation, d'entraînement, de prédiction, et de sauvegarde des modèles, les mêmes procédures et scripts que ceux utilisés dans la version 3 sont réemployés pour l'ensemble des algorithmes de détection considérés, à savoir : « Random Forest, XGBoost, Naïve Bayes, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) ».

4.11. Conclusion

Ce chapitre a décrit la phase de préparation des données, d'entraînement et de prédiction de plusieurs modèles de classification appliqués au jeu de données CICDDoS2019. En étudiant quatre versions successives du jeu de données intégrant différents niveaux de prétraitement et d'équilibrage, il a servi à mettre en place un cadre d'expérimentation rigoureux pour évaluer les performances des modèles. L'analyse des résultats de ces prédictions sera exposée dans le chapitre suivant.

Chapitre 5 : Résultats expérimentaux et discussion

5.1.Introduction

Ce chapitre propose une analyse approfondie des performances des différents modèles de classification appliqués à notre problématique. Nous évaluons ces modèles selon plusieurs axes : les métriques numériques classiques (précision, rappel, F1-score), l'AUC-ROC, le temps d'exécution, ainsi que les taux d'erreur (FPR, FNR), les courbes d'apprentissage sont analysées pour chaque modèle (Random Forest, XGBoost, Naive Bayes, KNN, SVM) afin de rendre compte de la dynamique d'entraînement de chacun des algorithmes, les matrices de confusion et de corrélation viennent compléter l'interprétation des résultats. Enfin, en comparaison avec d'autres études, des perspectives de contribution et d'amélioration viennent s'inscrire dans cette analyse. L'objectif est de donner une vision homogène et critique de la performance des différents modèles testés dans tout contexte.

5.2. Analyse comparative des métriques numériques

5.2.1. Métriques de performances générales

Le tableau suivant présente les résultats de tous les modèles selon les indicateurs « Accuracy, Précision, Rappel, F1-Score » :

Algo	Ver	Acc	Précision Global	Précision Benign	Précision Attaque	Recall Global	Recall Benign	Recall Attaque	F1-Score Global	F1-Score Benign	F1-Score Attaque
RF	V1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	V2	0.9997	0.9272	0.8544	0.9999	0.9687	0.9376	0.9998	0.9470	0.8941	0.9999
	V3	0.9999	0.9617	0.9234	1.00	0.9992	0.9985	0.9999	0.9797	0.9595	1.00
	V4	0.9999	0.9680	0.9360	1.00	0.9995	0.9990	0.9999	0.9832	0.9665	1.00
XGB	V1	1.00	0.9998	0.9996	1.00	0.9996	0.9993	1.00	0.9997	0.9995	1.00
	V2	0.9998	0.9389	0.8778	1.00	0.9957	0.9916	0.9998	0.9656	0.9312	0.9999
	V3	0.9998	0.9401	0.8803	1.00	0.9995	0.9991	0.9998	0.9679	0.9359	0.9999
	V4	0.9998	0.9305	0.8609	1.00	0.9993	0.9988	0.9998	0.9623	0.9248	0.9999
NB	V1	0.9947	0.5184	0.0378	0.9990	0.5720	0.1482	0.9957	0.5288	0.0602	0.9974
	V2	0.9980	0.5310	0.0630	0.9989	0.5246	0.0500	0.9991	0.5274	0.0558	0.9990
	V3	0.9925	0.5109	0.0228	0.9990	0.5612	0.1290	0.9935	0.5175	0.0387	0.9962
	V4	0.4086	0.5009	0.0019	0.9999	0.6911	0.9743	0.4080	0.2917	0.0038	0.5795
KNN	V1	0.9999	0.9874	0.9749	0.9999	0.9780	0.9560	1.00	0.9826	0.9653	1.00
	V2	0.9998	0.9152	0.8305	1.00	0.9990	0.9983	0.9998	0.9533	0.9067	0.9999
	V3	0.9989	0.7620	0.5239	1.00	0.9960	0.9932	0.9989	0.8427	0.6860	0.9995
	V4	0.9989	0.7564	0.5128	1.0000	0.9952	0.9914	0.9989	0.8377	0.6760	0.9994
SVM	V1	0.9989	0.8359	0.6727	0.9990	0.5651	0.1303	0.9999	0.6089	0.2183	0.9995
	V2	0.9989	0.9813	0.9636	0.9989	0.5454	0.0908	1.00	0.5827	0.1659	0.9995
	V3	0.8187	0.5027	0.0057	0.9998	0.8511	0.8836	0.8187	0.4557	0.0113	0.9002
	V4	0.8026	0.5026	0.0053	0.9999	0.8499	0.8973	0.8024	0.4504	0.0105	0.8903

Table 5.1 - Évaluation des Algorithmes de Classification : Précision, Rappel et F1-Score

Analyse approfondie du tableau :

- **Random Forest :**
  - ***Version 1 :*** toutes les métriques = 1.00.
    Conclusion : Les performances parfaites suggèrent un risque de surapprentissage, car les données sont brutes et peuvent contenir des fuites d'informations. Bien que les résultats soient excellents, le modèle pourrait ne pas généraliser correctement sur de nouvelles données.
  - ***Version 2 :***
    Accuracy : 0.9997, légère baisse par rapport à V1.
    Précision Global : 0.9272, indique une performance légèrement moins bonne.
    Précision Benign : 0.8544, une baisse significative, ce qui pourrait indiquer des difficultés à identifier certains exemples bénins.
    Recall Global : 0.9687, bon mais moins que V1.
    Recall Benign : 0.9376, moins bon que V1. Tandis que pour Attaque est presque parfait
    F1-Score Global : 0.9470, toujours bon mais en baisse.
    F1-Score Benign : 0.8941, bonne performance mais inférieure à V1.
    Conclusion : Bonne performance globale, mais la précision sur la classe benign reste préoccupante.
  - ***Version 3 :***
    Accuracy : 0.9999, très proche de la perfection.
    Précision Global : 0.9617, meilleure que V2.
    Précision Benign : 0.9234, amélioration par rapport à V2.
    Recall Global : 0.9992, excellent rappel.
    Recall Benign : 0.9985, bon mais pas parfait.
    F1-Score Global : 0.9797, bon compromis entre précision et rappel.
    F1-Score Benign : 0.9595, solide performance pour la classe benign.
    Conclusion : Version équilibrée et robuste, avec meilleure reconnaissance des bénins que V2, confirmant l'effet positif de la réduction des redondances.
  - ***Version 4 :***
    Accuracy : 0.9999, stable par rapport à V3.
    Précision Global : 0.9680, très bonne, légèrement supérieure à celle de la V3.
    Précision Benign : 0.9360, amélioration par rapport à V3.
    Recall Global : 0.9995, excellent rappel, supérieur au V3.
    Recall Benign : 0.9990, très bon, amélioration par rapport à V3.
    F1-Score Global : 0.9832, amélioré, très bon équilibre.
    F1-Score Benign : 0.9665, excellente performance pour la classe benign.
    Conclusion : Version robuste avec des performances élevées, recommandée pour une utilisation.
- **XGBoost :**
  - ***Version 1 :***

Accuracy : 1.00, parfait mais soulève des préoccupations similaires à RF V1.
Précision Global : 0.9998, très proche de la perfection.
Précision Benign : 0.9996, excellente performance.
Recall Global : 0.9996, presque aucune erreur de classification.
Recall Benign : 0.9993, tous les exemples bénins sont correctement identifiés.
F1-Score Global : 0.9997, très bon équilibre.
F1-Score Benign : 0.9995, excellent pour la classe benign.
Conclusion : Comme RF, XGB montre des performances parfaites, ce qui soulève également des préoccupations concernant le surapprentissage. Résultat surestimé à cause du dataset biaisé.

- ***Version 2 :***

  Accuracy : 0.9998, très élevée mais légèrement inférieur à V1.
  Précision Global : 0.9389, bonne mais moins bonne que V1.
  Précision Benign : 0.8778, relativement faible.
  Recall Global : 0.9957, bon rappel 99.57% des échantillons positifs réels identifiés.
  Recall Benign : 0.9916, légère baisse par rapport à V1.
  F1-Score Global : 0.9656, bon compromis.
  F1-Score Benign : 0.9312, bonne performance pour la classe benign.
  Conclusion : Excellente performance, mais la précision sur BENIGN pourrait être améliorée.

- ***Version 3 :***

  Accuracy : 0.9998, stable par rapport à V2.
  Précision Global : 0.9401, amélioration significative par rapport à V2.
  Précision Benign : 0.8803, meilleure que dans V2 mais reste inférieure à Attaque 1.00.
  Recall Global: 0.9995, excellent rappel.
  Recall Benign : 0.9991, amélioration significative.
  F1-Score Global : 0.9679, amélioration et bon équilibre.
  F1-Score Benign : 0.9359, amélioré légèrement par rapport à V2, solide performance.
  Conclusion : Très bon modèle, détection d'attaques optimale et meilleure reconnaissance des bénins qu'en V2.

- ***Version 4 :***

  Accuracy : 0.9998, élevée et stable.
  Précision Global : 0.9305, légère baisse.
  Précision Benign : 0.8609, encore faible.
  Recall Global : 0.9993, excellent rappel, inférieure à V3 mais supérieure à V2.
  Recall Benign : 0.9988, faible par rapport à V3 mais toujours mieux que V2.
  F1-Score Global : 0.9623, moins équilibrée que V2 et V3.
  F1-Score Benign : 0.9248, moindre parmi toutes les versions.
  Conclusion : Modèle performant, mais la précision benign continue de montrer une légère baisse, indiquant une possible sur-adaptation.

- **Naive Bayes :**
  - ***Version 1 :***
    Accuracy : 0.9947, élevée mais peut être trompeur.
    Précision Global : 0.5184, très faible moins de la moitié des prédictions sont correctes.
    Précision Benign : 0.0378, extrêmement basse.
    Recall Global : 0.5720, meilleur que la précision mais reste faible.
    Recall Benign : 0.1482, très faible, tandis que pour "Attaque" est très élevé (0.9957).
    F1-Score Global : 0.5288, faible performance.
    F1-Score Benign : 0.0602, très insatisfaisant.
    Conclusion : Catastrophique sur la classe BENIGN, ne distingue presque pas les bons paquets. Cela montre que NB ne s'adapte pas bien aux données brutes.
  - ***Version 2 :***
    Accuracy : 0.9980, bonne amélioration par rapport à V1.
    Précision Global : 0.5310, supérieure à V1 mais toujours faible.
    Précision Benign : 0.0630, meilleure que V1 mais reste très faible.
    Recall Global : 0.5246, faible, légèrement inférieur à V1.
    Recall Benign : 0.0500, encore insuffisant.
    F1-Score Global : 0.5274, très insatisfaisant.
    F1-Score Benign : 0.0558, très faible.
    Conclusion : Toujours axé sur la détection, avec un léger gain sur les bénins qu'en V1.
  - ***Version 3 :***
    Accuracy : 0.9925, inférieure à celles des versions 1 et 2 mais acceptable.
    Précision Global : 0.5109, très faible, inférieure à V1 et V2.
    Précision Benign : 0.0228, catastrophique.
    Recall Global : 0.5612, légèrement mieux que V2 mais reste faible.
    Recall Benign : 0.1290, mieux que V2 mais reste très faible.
    F1-Score Global : 0.5175, très insatisfaisant, inférieur à V2 et V1.
    F1-Score Benign : 0.0387, très faible, inférieur à V2 et V1.
    Conclusion : Mauvais sur tous les plans. Ne devrait pas être retenu.
  - ***Version 4 :***
    Accuracy : 0.4086, très faible.
    Précision Global : 0.5009, légèrement inférieure aux autres, très faible.
    Précision Benign : 0.0019, presque nul.
    Recall Global : 0.6911, plus élevé mais peu fiable.
    Recall Benign : 0.9743, très bon.
    F1-Score Global : 0.2917, très insuffisant.
    F1-Score Benign : 0.0038, très faible.
    Conclusion : Catastrophique, cette version est la moins performante.
- **KNN :**
  - ***Version 1 :***

Accuracy : 0.9999, très bon mais avec un dataset biaisé.
Précision Global : 0.9874, excellente performance.
Précision Benign : 0.9749, très bon.
Recall Global : 0.9780, bon rappel.
Recall Benign : 0.9560, bon.
F1-Score Global : 0.9826, excellent équilibre.
F1-Score Benign : 0.9653, excellente performance pour la classe benign.
Conclusion : Bons résultats, mais potentiellement inutilisable en production à cause du temps d'entraînement.

- ***Version 2 :***
  Accuracy : 0.9998, très élevée, légèrement inférieure à V1.
  Précision Global : 0.9152, légère baisse par rapport à V1 mais solide précision.
  Précision Benign : 0.8305, bonne mais inférieur à V1.
  Recall Global : 0.9990, excellent mieux qu'en V1.
  Recall Benign : 0.9983, très élevé.
  F1-Score Global : 0.9533, inférieur à V1 mais bon compromis.
  F1-Score Benign : 0.9067, solide performance, inférieur à V1.
  Conclusion : Excellente détection des attaques, mais limite sur les bénins.
- ***Version 3 :***
  Accuracy : 0.9989, légère baisse mais reste très élevée.
  Précision Global : 0.7620, baisse significative.
  Précision Benign : 0.5239, relativement faible.
  Recall Global : 0.9960, très élevé, moins qu'en V2 mais mieux qu'en V1.
  Recall Benign : 0.9932, élevé mais légèrement inférieur à V2.
  F1-Score Global : 0.8427, baisse significative.
  F1-Score Benign : 0.6860, faible.
  Conclusion : Perte importante de performance avec moins de caractéristiques. Bon sur les attaques, mais nette dégradation sur les bénins par rapport aux versions précédentes.
- ***Version 4 :***
  Accuracy : 0.9989, identique à V3.
  Précision Global : 0.7564, légère baisse.
  Précision Benign : 0.5128, faiblesse significative.
  Recall Global : 0.9952, bon mais légèrement moins qu'en V3.
  Recall Benign : 0.9914, élevé mais légèrement inférieur à V3.
  F1-Score Global : 0.8377, encore une baisse.
  F1-Score Benign : 0.6760, plus faible qu'en V3.
  Conclusion : Performances similaire à V3 avec un léger recul sur les deux classes.

- **SVM** :
  - ***Version 1 :***
    Accuracy : 0.9989, très élevée.

Précision Global : 0.8359, acceptable.
Précision Benign : 0.6727, faible.
Recall Global : 0.5651, moins bon.
Recall Benign : 0.1303, très bas.
F1-Score Global : 0.6089, faible.
F1-Score Benign : 0.2183, très faible.
Conclusion : Bonne détection des attaques, mais forte difficulté sur les bénins, ne supporte pas bien les données déséquilibrées. Résultats insatisfaisants.

- ***Version 2 :***
  Accuracy : 0.9989, stable.
  Précision Global : 0.9813, très bonne, supérieure à V1.
  Précision Benign : 0.9636, amélioration significative.
  Recall Global : 0.5454, faible avec une légère baisse qu'en V1.
  Recall Benign : 0.0908, faible, inférieur qu'en V1.
  F1-Score Global : 0.5827, insuffisant pire qu'en V1.
  F1-Score Benign : 0.1659, très faible, inférieur à V1.
  Conclusion : Mieux que V1 sur la précision Benign, mais le rappel reste mauvais.
- ***Version 3 :***
  Accuracy : 0.8187, chute notable donc performance moins solide.
  Précision Global : 0.5027, très faible.
  Précision Benign : 0.0057, catastrophique.
  Recall Global : 0.8511, élevé.
  Recall Benign : 0.8836, amélioration significative par rapport aux autres versions.
  F1-Score Global : 0.4557, très faible, en baisse.
  F1-Score Benign : 0.0113, catastrophique.
  Conclusion : Modèle déséquilibré, rappel fort, précision faible sur les bénins.
- ***Version 4 :***
  Accuracy : 0.8026, légèrement inférieure à V3.
  Précision Global : 0.5026, très faible et similaire à V3.
  Précision Benign : 0.0053, pire que V3, presque nul.
  Recall Global : 0.8499, élevé par rapport à V1 et V2, légèrement inférieur à V3.
  Recall Benign : 0.8973, le meilleur parmi les autres.
  F1-Score Global : 0.4504, très insuffisant.
  F1-Score Benign : 0.0105, catastrophique, inférieur à V3.
  Conclusion : Proche de V3, légère baisse globale, mais rappel toujours excellent sur les deux classes.

- **Conclusion finale :** Les V1 de Random Forest et XGBoost affichent des performances idéales, suggérant un surapprentissage et une fuite d'information. Les versions suivantes montrent une évolution progressive vers une meilleure robustesse et un meilleur équilibre entre précision et rappel. La version 4 de Random Forest et V3 de XGBoost se démarquent

comme les plus équilibrées et fiables, avec d'excellentes reconnaissance des deux classes. KNN V2 offre des résultats corrects, mais couteux en temps de calcul. En revanche, Naive Bayes et SVM montrent des faiblesses majeures, surtout sur la classe Benign, avec des précisions extrêmement basses dans la plupart des versions. La V2 de NB et SVM se révèle être la meilleure option comparativement aux autres. Ces deux modèles ne semblent pas adaptés à ce type de données. Globalement, Random Forest V4 est donc recommandé comme meilleur choix, suivi de près par XGBoost V3, tandis que les autres modèles nécessitent des ajustements importants avant d'être utilisés dans un cadre opérationnel.

5.2.2. AUC-ROC et Temps d'exécution

Les figures suivantes montrent les indicateurs AUC-ROC, Temps d'entrainement et Temps de prédiction pour tous les modèles entrainés.

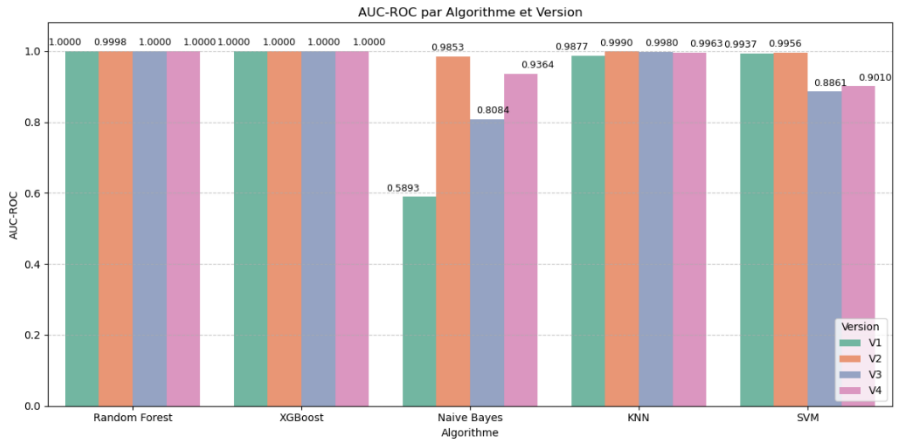

Figure 5.1 - Visualisation des AUC-ROC des algorithmes de classification selon les versions

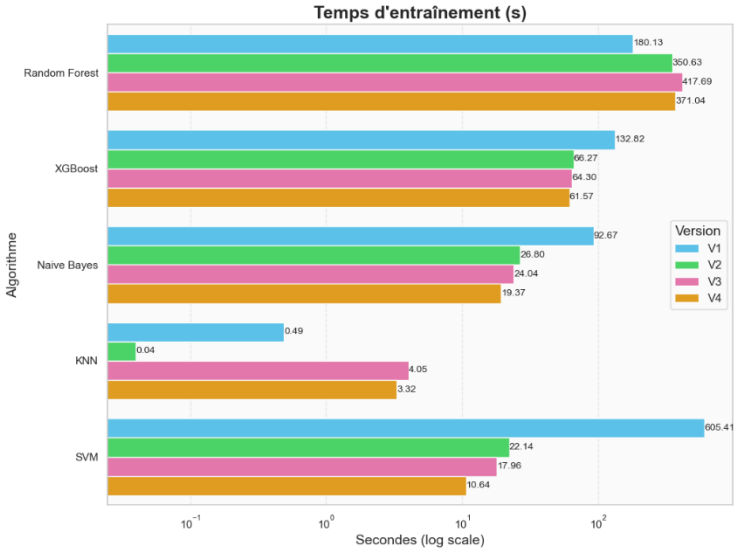

Figure 5.2 - Temps d'Entraînement des Algorithmes de Classification par Version

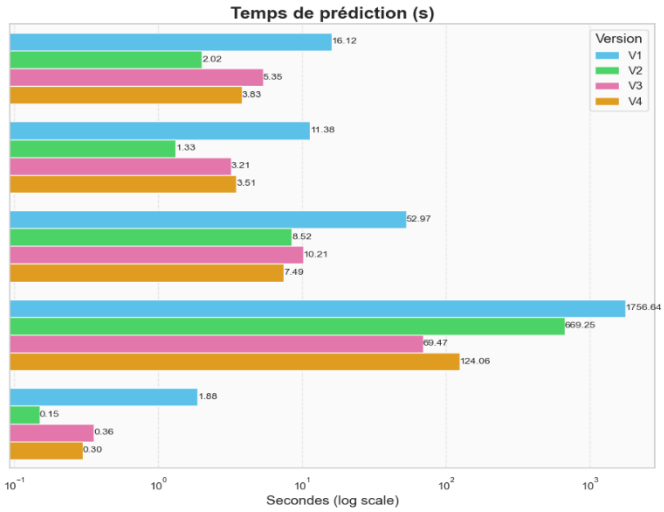

Figure 5.3 - Temps de prédiction des algorithmes de classification par version

- **Analyse RF :** Toutes les versions montrent une excellente capacité de discrimination (AUC-ROC proche de 1). V2 est la plus rapide en termes de prédiction, mais V3 a le temps d'entraînement le plus long, ce qui pourrait limiter son utilisation dans certaines cas.
- **Analyse XGB :** Toutes les versions montrent une très haute performance AUC-ROC proche de 1. Contrairement à RF, le temps d'entraînement diminue avec les versions, avec V4 étant la plus rapide (61.57 s) tout en maintenant un AUC élevé. Cela en fait une version particulièrement efficace.
- **Analyse NB :** Les performances varient fortement selon les versions : V1 a un AUC très bas (0.58), mais les versions suivantes s'améliorent significativement, avec un pic à 0.985 en V2. Le temps d'entraînement diminue constamment, tout comme le temps de prédiction, rendant les versions V2 à V4 plus attractives malgré une légère baisse de précision en V3.
- **Analyse KNN :** V1 s'entraine très vite 0,49s mais prédit très lentement 1756s, limitant son usage. V2 est la plus rapide en entrainement 0.04s mais un peu lente en prédiction 669s avec un excellent AUC 0.999. V3 réduit ce temps à 69s pour un AUC de 0.9979, meilleur équilibre observé. V4 est plus lente que V3 124s malgré un bon AUC 0.996.
- **Analyse SVM :** V1 est très lente à s'entraîner 605s, mais rapide en prédiction 1.88s avec un bon AUC 0.993. À partir de V2, le temps d'entraînement diminue drastiquement 22s, avec même une légère amélioration de précision AUC = 0.9956. Cependant, les versions V3 et V4 sacrifient trop de performance AUC < 0.9 pour gagner en vitesse, faisant de V2 la version la plus équilibrée et la plus fiable du modèle.
- **Conclusion Générale par Algorithme :**
  - ***RF :*** V3 et V4 offrent le meilleur compromis entre AUC très élevé et prédiction rapide, malgré un entraînement plus long.
  - ***XGBoost :*** Toutes les versions sont performantes, mais V4 se démarque par sa rapidité d'entraînement et une précision presque parfaite.

- o ***Naive Bayes :*** Seule V2 montre des performances satisfaisantes en précision AUC=0.985 et en vitesse, les autres versions étant peu fiables ou instables.
- o ***KNN :*** V3 est la version la plus équilibrée avec un bon AUC 0.9979, un temps d'entraînement raisonnable 4s et une prédiction rapide 69s.
- o ***SVM :*** V2 est clairement la meilleure version, combinant faible temps d'entraînement 22s, prédiction ultra-rapide 0.15s et bon AUC 0.9956.

En résumé, pour des applications nécessitant une bonne précision et un temps de réponse rapide, XGBoost et Random Forest sont les meilleurs choix.

5.2.3. Taux d'erreur FPR et FNR

Les figures suivantes montrent les taux de faux positifs et faux négatifs pour tous les modèles entrainés.

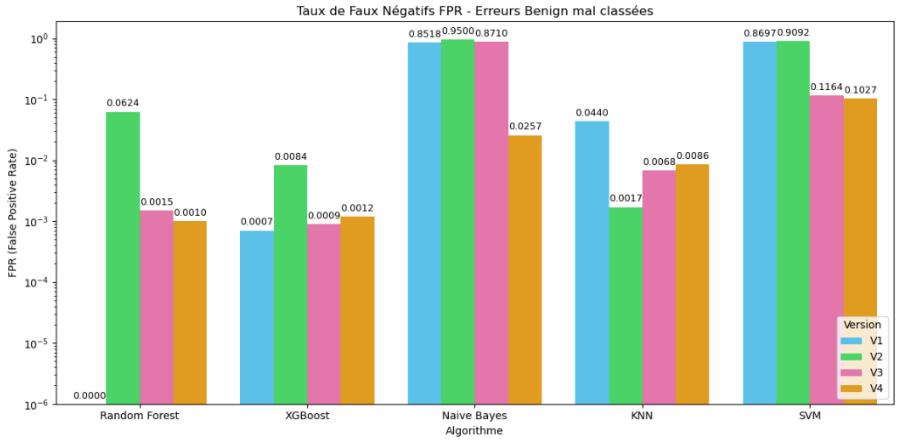

Figure 5.4 - Taux de Faux Positifs (FPR) des Algorithmes

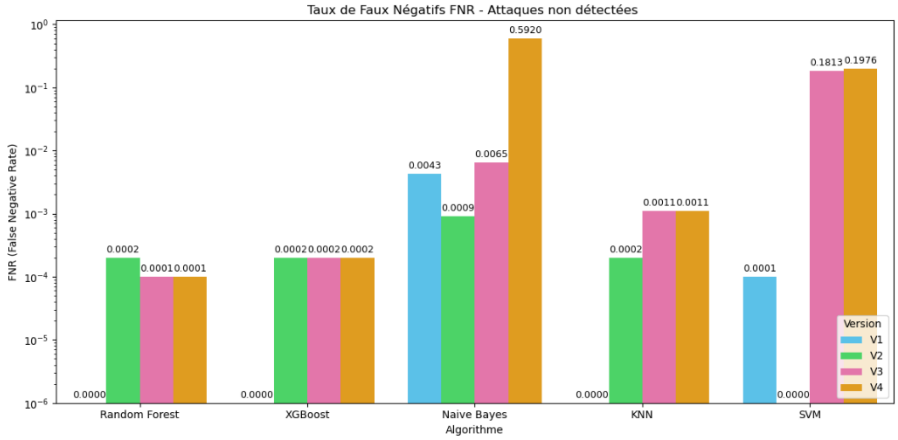

Figure 5.5 - Taux de Faux Négatifs (FNR) des Algorithmes

- **Random Forest (RF) :** Random Forest affiche d'excellentes performances globales. V1 et V4 se distinguent par des FPR et FNR quasi nuls, signe d'un bon équilibre. V2 à un FPR plus élevé (6,24 %) mais conserve un FNR très bas, assurant une détection fiable malgré plus de faux positifs.

- **XGBoost (XGB) :** XGBoost montre une grande stabilité et robustesse. Toutes les versions ont un FPR < 1 % et un FNR quasi nul, garantissant une détection fiable. V1 se démarque avec un FPR de 0,07 %, idéal si l'on veut minimiser les faux positifs. Les versions V2 à V4 restent très performantes, avec peu de variations, preuve d'une bonne généralisation.
- **Naive Bayes (NB) :** Naive Bayes montre des performances très variables selon les versions. V1 à V3 présentent un FPR très élevé (>85 %), rendant le modèle peu fiable. V4 améliore nettement le FPR (2,57 %) mais souffre d'un FNR très élevé (59,2 %), indiquant une mauvaise détection des cas positifs. Le modèle peine donc à équilibrer précision et sensibilité sans ajustement ou transformation des données.
- **K-Nearest Neighbors (KNN) :** KNN offre une performance stable, avec des FPR bas et des FNR très faibles. V2 se démarque (FPR 0,17 %, FNR 0,02 %) comme la meilleure version. V3 et V4 dégradent légèrement le FPR (~0,7 %) mais restent solides. Globalement fiable, KNN reste toutefois derrière Random Forest et XGBoost.
- **Support Vector Machine (SVM) :** SVM montre des résultats instables selon les versions. V1 et V2 ont un FPR très élevé (>86 %), générant trop de faux positifs. V3 et V4 améliorent ce point (FPR ~11 %) mais au prix d'un FNR élevé (~19 %). Le modèle peine à équilibrer précision et rappel, suggérant un besoin d'optimisation avant usage.

5.3. Analyse des Courbes d'apprentissage

5.3.1. Random Forest

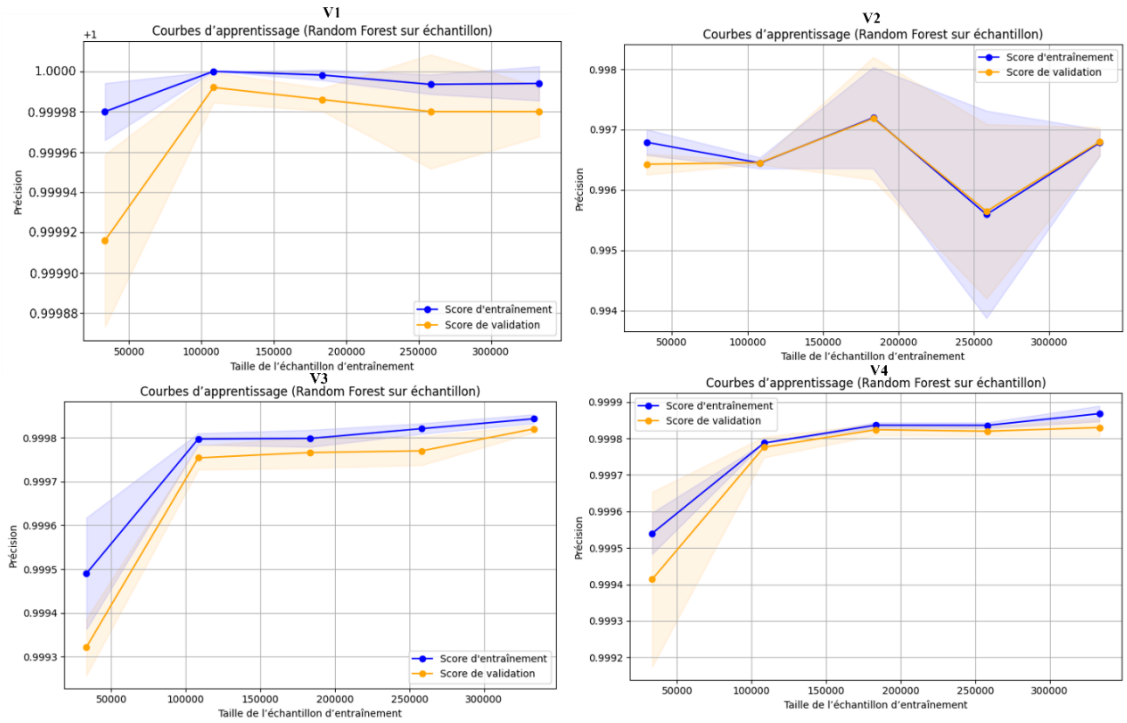

Figure 5.6 - Courbes d'apprentissage RF pour toutes les versions

- **Version 1 :**
  - Scores très proches de 1.0 sur les données d'entrainement et de validation.

- Fuite de données probable due au score parfait signe des attributs révélant la cible.
- Le score d'entrainement reste pratiquement constant à 1.0, tandis que la validation stagne plus bas ce qui peut indiquer que le modèle est trop ajusté aux données d'entrainement donc un surapprentissage faible même si cet ajustement est acceptable pour RF.
- Ecart constant entre les deux courbes, bien que faible, il ne diminue pas de manière continue avec l'augmentation de la taille du jeu d'entrainement.
- Conclusion : Léger surapprentissage persistant, le modèle n'a pas réussi à réduire complètement l'écart entre les deux courbes, malgré une convergence progressive, et il n'a pas parfaitement généralisé aux nouvelles données.

- **Version 2 :**
  - Les scores varient fortement, pic à 150k et chute à 250k, traduisant une instabilité d'apprentissage.
  - Les variations sont probablement dues au bruit dans les données.
  - Surapprentissage temporaire entre 100k et 150k léger écart entre entrainement et validation.
  - Conclusion : Modèle instable, peu robuste face aux variations des données avec un surapprentissage léger et temporaire.
- **Version 3 :**
  - Courbes d'entrainement et de validation augmentent ensemble jusqu'à ~0.9999, avec un écart faible qui diminue avec plus de données.
  - Aucune divergence notable entre les deux courbes donc pas de surapprentissage.
  - Scores élevés donc pas de sousapprentissage.
  - Conclusion : le modèle généralise bien et atteint une performance stable et optimale.
- **Version 4 :**
  - Scores élevés et proches, entrainement ~0.9999 et validation ~0.9998 avec un écart stable et minime moins de 0.0001.
  - Les courbes se stabilisent vers 200k échantillons, indiquant une complexité adaptée. Ajouter plus de données aurait un impact limité, le modèle a atteint ses limites de performances.
  - Les scores élevés excluent tout sousapprentissage.
  - Aucun surapprentissage, les deux courbes évoluent ensemble sans divergence.
  - Conclusion : Apprentissage optimal, modèle exceptionnellement performant et stable.

5.3.2. XGBoost

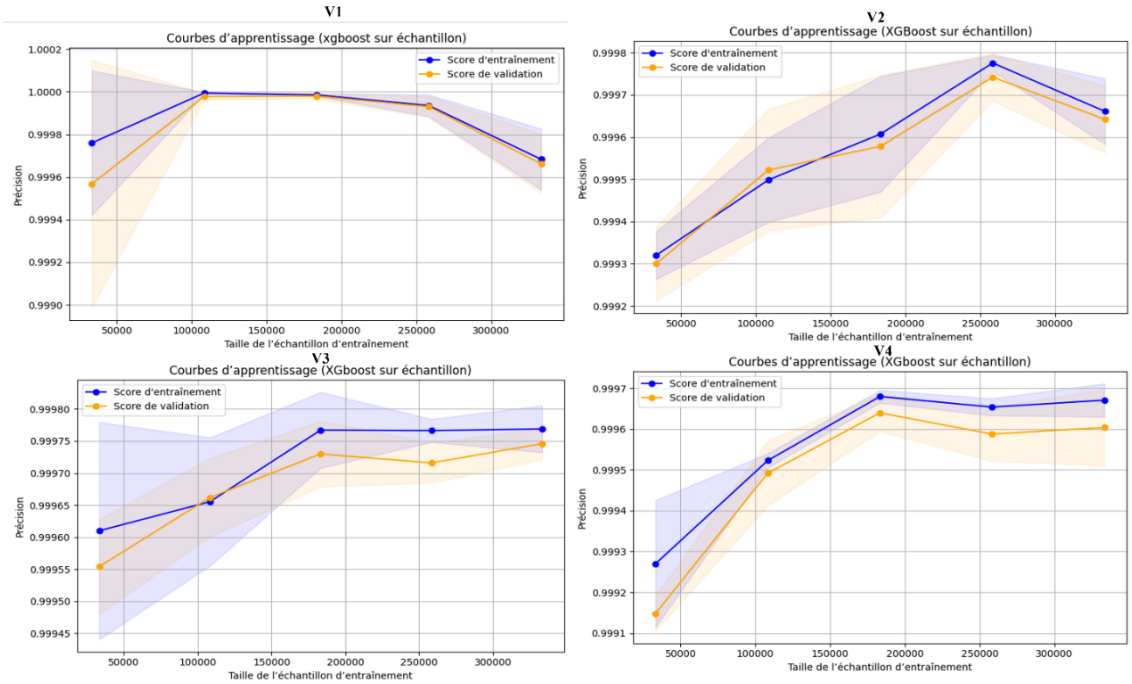

Figure 5.7 - Courbes d'apprentissage XGBoost pour toutes les versions

- **Version 1 :**
  - Score d'entrainement =1.0 dès 100k échantillons, signe de mémorisation parfaite des données donc un surapprentissage.
  - Diminution des scores après 200k échantillons de 1 vers ~0.9996, signe que le modèle perd en généralisation malgré plus de données.
  - Le modèle devient instable sur les nouveaux échantillons.
  - Conclusion : Surapprentissage clair, mémorisation excessive, perte progressive de généralisation et instabilité.
- **Version 2 :**
  - Scores élevés et parallèles dès 50k échantillons donc pas de sousapprentissage.
  - Convergence vers 0.9997-0.9998 après 200k échantillons, signe d'apprentissage maitrisé.
  - Ecart très faible entre entrainement et validation pas de surapprentissage
  - La baisse légère après 250k due à une stabilisation naturelle du modèle.
  - Conclusion : apprentissage normal et efficace avec une excellente généralisation.
- **Version 3 :**
  - Scores élevés, signe d'une bonne performance.
  - L'écart entre les deux courbes reste très faible excluant tout sur/sousapprentissage.
  - Les intervalles de confiance sont étroits, montrant une performance constante et fiable.
  - Conclusion : Le modèle est bien entraîné, stable et généralise correctement.
- **Version 4 :**
  - Scores élevés, plateau atteint après 200k échantillons indiquant un modèle mature.

- Ecart très faible entre les courbes, sans divergence, donc pas de surapprentissage.
- Progression régulière des deux courbes signe d'absence de sousapprentissage.
- Conclusion : Bon apprentissage, avec une généralisation efficace et une stabilité robuste et des performances proches du maximum atteignable.

5.3.3. Naive Bayes

- **Version 1 :**
  - Les courbes chutent fortement entre 50k et 200k échantillons, ensuite se stabilisent ~0.994 sans réelle amélioration.
  - L'écart entre les courbes reste faible, ce qui exclut un surapprentissage.
  - Scores finaux faibles, et l'ajout de données n'améliore pas les performances.
  - Conclusion : Sousapprentissage, le modèle est trop simple pour la complexité croissante des données.
- **Version 2 :**
  - Scores en croissance rapide puis se stabilisent autour de 0.97 dès 100k échantillons.
  - Ecart minime entre les courbes signe d'une bonne généralisation.
  - Les courbes sont lisses avec des intervalles de confiance étroits, résultats cohérents.
  - Conclusion : Bon apprentissage, modèle efficace et bonne généralisation.

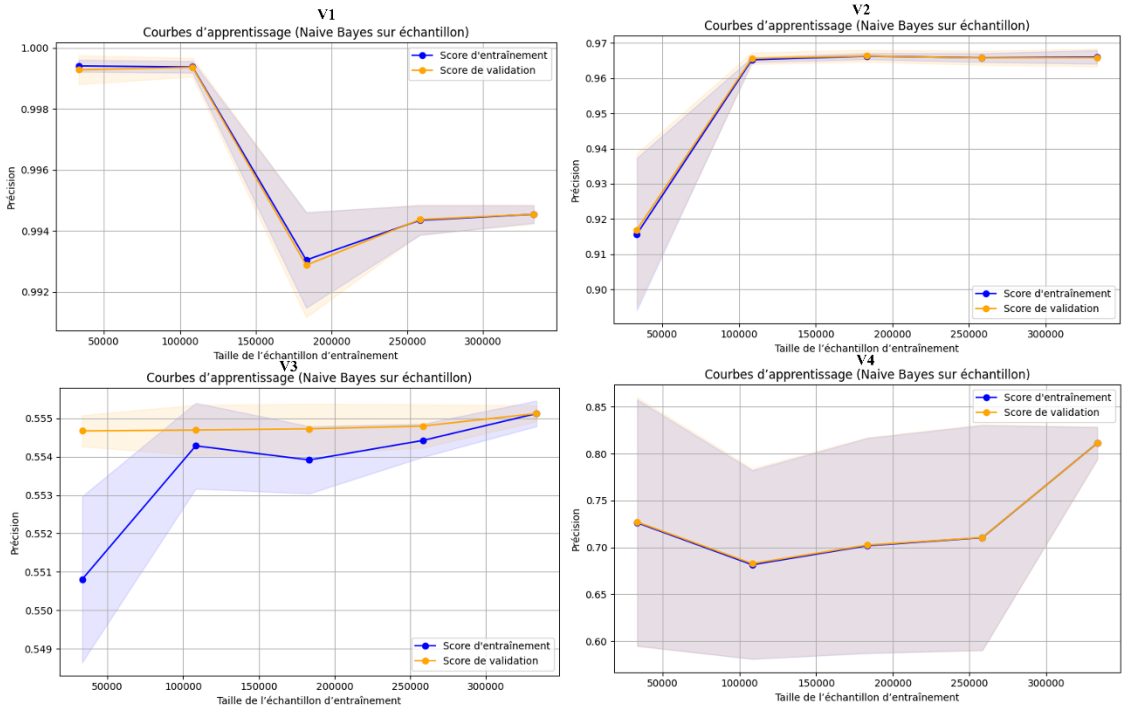

Figure 5.8 - Courbes d'apprentissage NB pour toutes les versions

- **Version 3 :**
  - Scores faibles ~0.55, même avec 300k échantillons les scores restent très bas indiquant que le modèle n'apprend pas efficacement.
  - Plateau rapide, stagnation dès 100k échantillons.

- Conclusion : Sousapprentissage, le modèle étant trop simple pour capturer la complexité des données.

- **Version 4 :**
  - Scores reste autour de 0.70-0.72 puis progressent ~0.80 à 300k indiquant un apprentissage progressif.
  - Pas de surapprentissage, les courbes restent proches sans écart important.
  - Pas de sousapprentissage, les scores ne stagnent pas à des niveaux bas, ils s'améliorent avec la taille de l'échantillon, signe d'une capacité d'apprentissage suffisante.
  - Conclusion : apprentissage modéré, avec une bonne généralisation et une convergence progressive, malgré des performances modérées dues aux limites structurelles du NB.

5.3.4. KNN

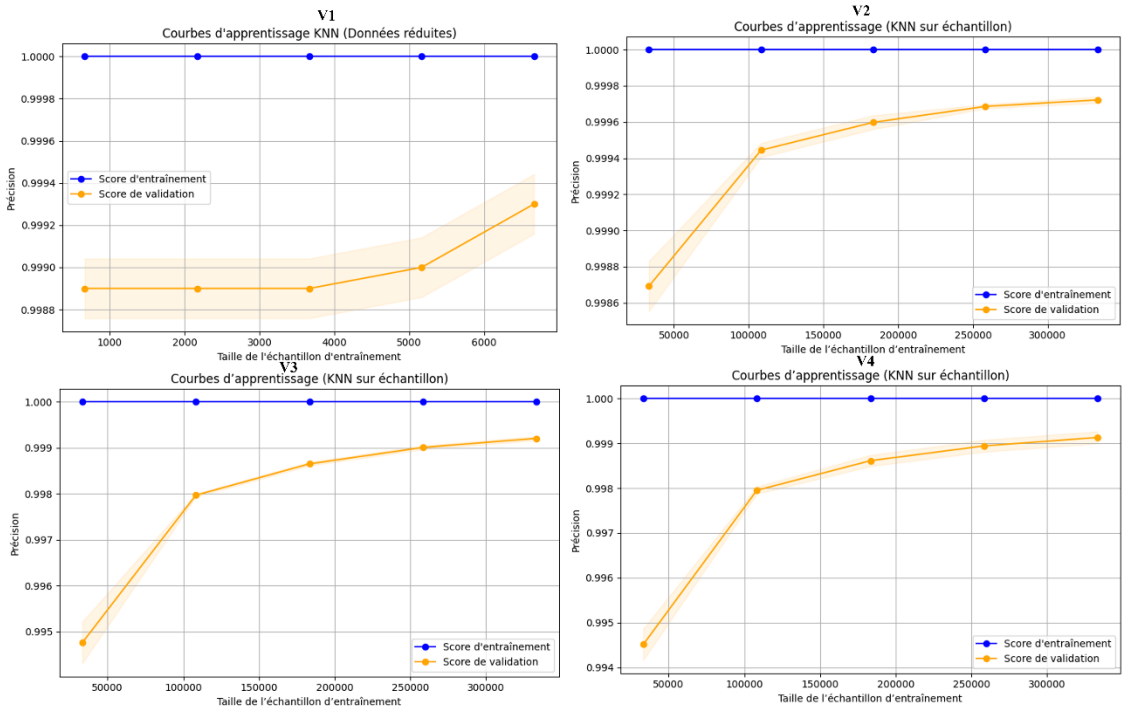

Figure 5.9 - Courbes d'apprentissage KNN pour toutes les versions

- **Version 1 :**
  - Score d'entrainement parfait 1.0 dès les premiers échantillons, signe qu'il mémorise les données.
  - Score de validation plus faible montrant une mauvaise généralisation.
  - Ecart stable et significatif entre les deux scores est un signe clair de surapprentissage.
  - Conclusion : Surapprentissage fort.

- **Version 2 :**
  - Score d'entrainement constant à 1.0, signe de mémorisation parfaite.
  - Score de validation augmente de ~0,9986 à ~0,9998 donc meilleure généralisation.
  - L'écart diminue avec plus de données, signe d'amélioration de la généralisation.
  - Conclusion : Surapprentissage léger, corrigé par l'augmentation de l'échantillon.

- **Version 3 :**
  - Score d'entrainement reste à 1.0 indiquant une mémorisation complète.
  - La validation commence autour de 0.995 puis augmente jusqu'à ~0.999 avec plus de données et reste légèrement inférieure à l'entrainement.
  - Conclusion : Surapprentissage léger qui diminue avec plus de données.
- **Version 4 :**
  - Score d'entrainement constant à 1, signe d'une mémorisation parfaite des données.
  - Score de validation plus faible au départ 0.994, l'écart diminue avec plus de données.
  - L'amélioration continue du score de validation ~0.999, indice que le modèle généralise de mieux en mieux.
  - Conclusion : Surapprentissage léger au début mais bien compensé par l'augmentation des données. Bonne généralisation globale.

5.3.5. SVM

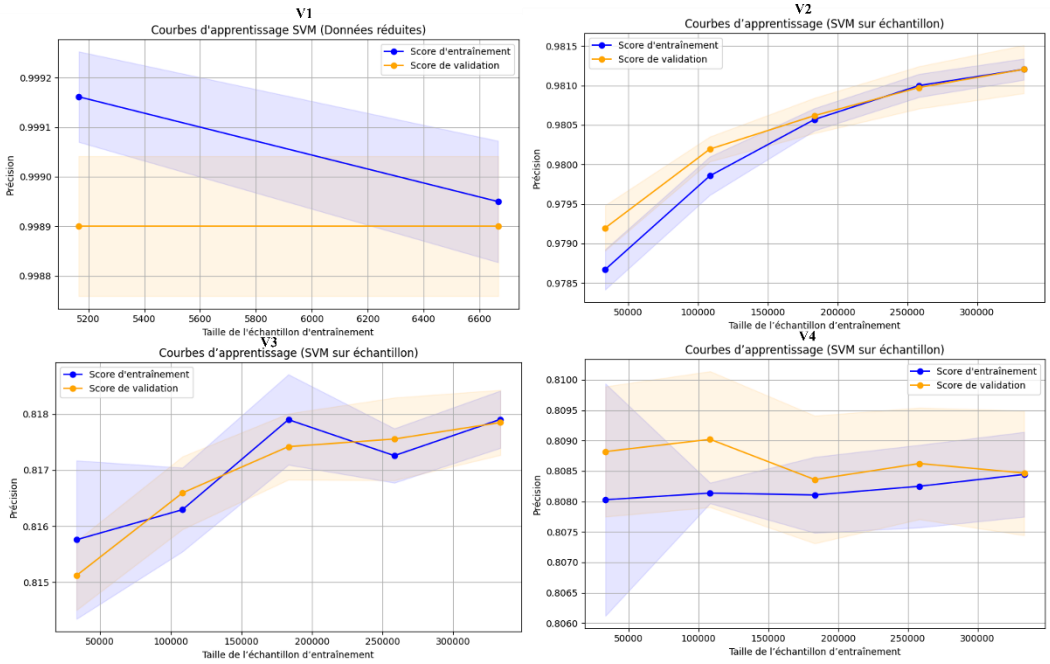

Figure 5.10 - Courbes d'apprentissage SVM pour toutes les versions

- **Version 1 :**
  - Score d'entrainement en baisse ~0.9992 vers ~0.9989 et score de validation stable à 0.9989 et ne progresse pas.
  - Ecart faible entre les deux courbes et en diminution, donc pas de surapprentissage.
  - Le modèle n'exploite pas mieux les données malgré l'ajout d'échantillons.
  - Conclusion : Sousapprentissage.
- **Version 2 :**
  - Les deux scores progressent avec plus de données, pas de sousapprentissge.
  - Ecart très faible entre les deux courbes, pas de surapprentissage.
  - Conclusion : Apprentissage stable et efficace, le modèle généralise bien.

- **Version 3 :**
  - Score d'entrainement ~0.818 et validation ~0.817 restent bas et très proches, sans écart significatif, donc pas de surapprentissage.
  - L'ajout de données n'améliore plus vraiment les performances après 200k échantillons.
  - Conclusion : Sousapprentissage, le modèle est trop simple pour capter la complexité des données.
- **Version 4 :**
  - Le score d'entrainement reste autour de 0.808, sans amélioration malgré plus de 300k échantillons.
  - Après une légère baisse, le score de validation se stabilise autour de 0.8085, sans progression notable.
  - L'écart entre les courbes est faible mais les deux scores plafonnent à un niveau insuffisant. Pas de généralisation.
  - Conclusion : le modèle est en sousapprentissage.

5.3.6. Comparaison des façons d'apprentissage

Le tableau suivant montre la façon d'apprentissage pour chaque modèle :

Modèle	V1	V2	V3	V4
RF	Surapprentissage léger	Instable léger surapprentissage	Bon apprentissage	Apprentissage optimal
XGB	Surapprentissage clair	Bon apprentissage	Bon apprentissage	Bon apprentissage
NB	Sousapprentissage	Bon apprentissage	Sousapprentissage sévère	Apprentissage modéré
KNN	Surapprentissage fort	Surapprentissage léger	Surapprentissage léger	Léger surapprentissage
SVM	Sousapprentissage	Bon apprentissage	Sousapprentissage	Sousapprentissage

Table 5.2 - Tableau comparatif selon la façon d'apprentissage

- **Comparaison des versions par algorithme :** Le tableau ci-dessous présente pour chaque algorithme la version ayant donné les meilleurs résultats en termes de stabilité, de précision et de capacité de généralisation.

Algorithme	Meilleure version
Random Forest	Version 4
XGBoost	Version 2
Naive Bayes	Version 2
K-Nearest Neighbors	Version 3 et 4
SVM	Version 2

Table 5.3 - Meilleure versions par algorithme en termes d'apprentissage

5.4. Analyse des Matrices de confusions

5.4.1. Comparaison par algorithme

- **Random Forest :**

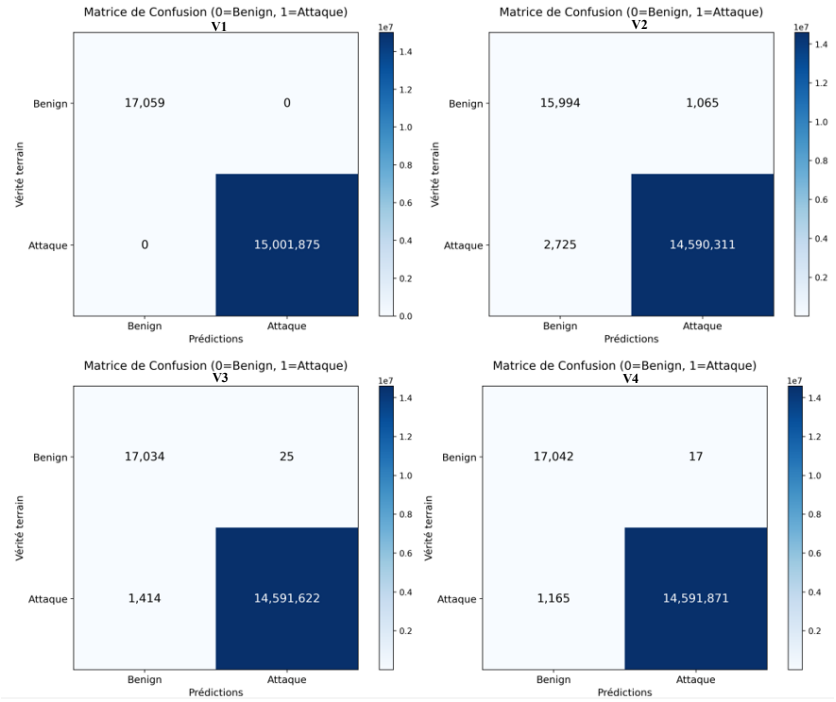

Figure 5.11 - Matrices de confusion RF

Version	TP	FP	FN	TN	Observations
V1	15 001 875	0	0	17 059	Parfaite prédiction (aucune erreur), irréaliste.
V2	14 590 311	1 065	2 725	15 994	Moyenne performance, beaucoup d'erreurs FP et FN.
V3	14 591 622	25	1414	17 034	Moins d'erreurs que V2, bon compromis entre détection et contrôle d'erreurs.
V4	14 591 871	17	1165	17042	Modèle optimal meilleur que V3, équilibre entre détection et minimisation des erreurs.

Table 5.4 - Comparaison des Performances des Modèles Random Forest

Conclusion : la version 1 est théoriquement parfaite mais elle semble irréaliste en pratique du coup la version 4 paraît être le meilleur choix réaliste car elle arrive à un bon nombre de vrais positifs, tout en gardant un nombre de faux positifs et de faux négatifs faible, performance globale correcte et équilibrée, pas de sur-apprentissage.

- **XGBoost :**

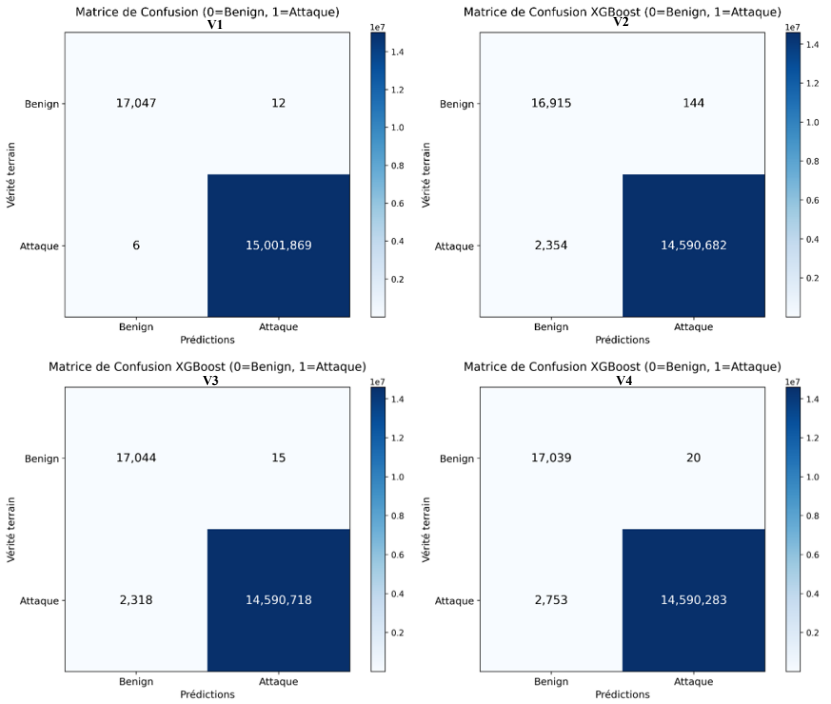

Figure 5.12 - Matrices de confusion XGBoost

Version	TP	FP	FN	TN	Observations
V1	15 001 869	12	6	17 047	Meilleur équilibre, détection maximale d'attaques, erreurs minimales.
V2	14 590 682	144	2 354	16 915	Bonne performance, équilibre entre détection TP élevé et gestion des cas bénins TN élevé, mais avec plus de fausses alertes.
V3	14 590 718	15	2 318	17 044	Similaire à V2 mais avec moins de faux positifs.
V4	14 590 283	20	2753	17 039	Faible performance, trop de FN malgré un contrôle des FP indique un biais vers les cas négatifs.

Table 5.5 - Analyse des versions pour XGBoost

Conclusion : Le modèle V3 est clairement le plus pertinent car il distingue très peu de faux positifs (FP) tout en préservant pérenne un nombre de vrais positifs (TP) approchant celui du modèle V2.

- **Naive Bayes :**

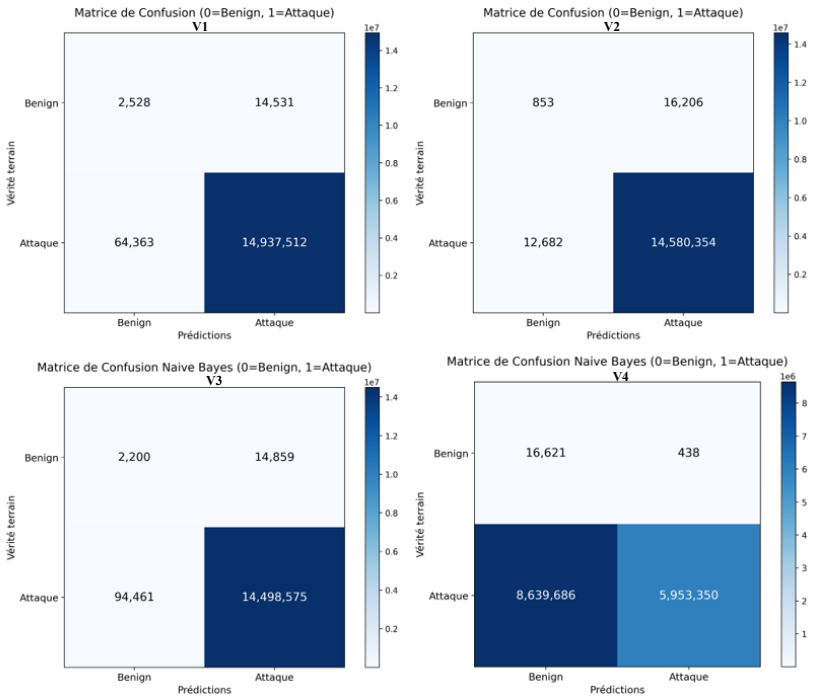

Figure 5.13 - Matrices de confusion Naive Bayes

Version	TP	FP	FN	TN	Observations
V1	14 937 512	14 531	64 363	2 528	Sousapprentissage évident, trop d'erreurs. Performance globale bonne.
V2	14 580 354	16 206	12 682	853	Meilleur compromis, faibles FN et détection robuste malgré un léger excès de FP.
V3	14 498 575	14 859	94 461	2 200	Plus de faux négatifs, moindre qualité globale.
V4	5 953 350	438	8,639,686	16 621	Réduction drastique des vrais positifs, et FN trop élevés. Problème de classification probable, avec un biais fort vers les bénins.

Table 5.6 - Analyse des versions pour NB

Conclusion : Le modèle V2 offre la meilleure performance globale, avec un excellent équilibre entre précision et rappel, sans signe de sous- ou sur-apprentissage.

- **KNN :**

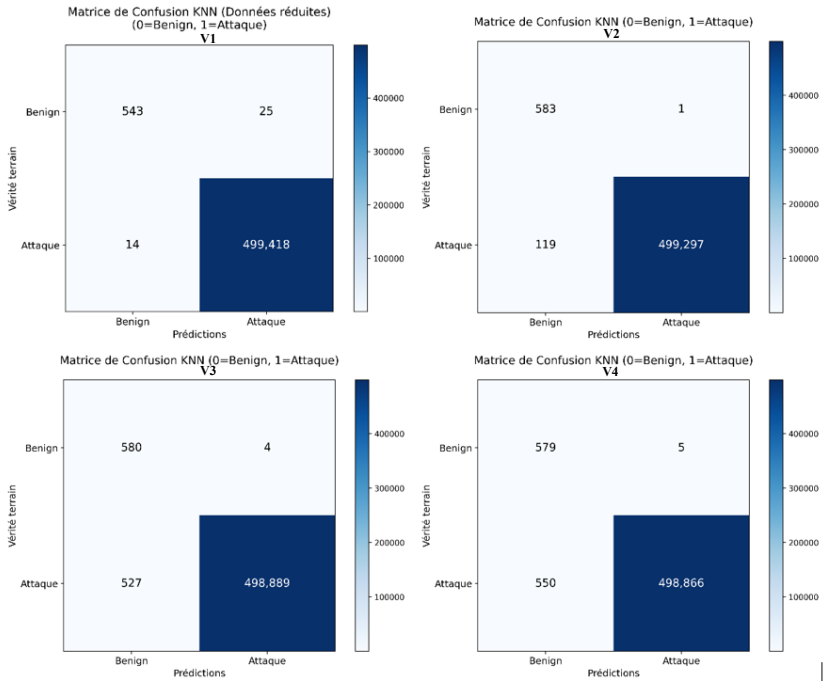

Figure 5.14 - Matrices de confusion KNN

Version	TP	FP	FN	TN	Observations
V1	499 418	25	14	543	Equilibre optimal entre sensibilité et spécificité.
V2	499 297	1	119	583	Détection agressive des positifs, au détriment des négatifs.
V3	498 889	4	527	580	Moyenne performance, peu de faux positifs, mais FN élevé.
V4	498 866	5	550	579	Faible performance, malgré un FP et TN raisonnable.

Table 5.7 - Analyse des versions pour KNN

Conclusion : Le modèle V1 est le meilleur, avec la meilleure performance globale : très peu d'erreurs et un excellent équilibre entre précision et rappel.

- **SVM :**

Version	TP	FP	FN	TN	Observations
V1	499 396	494	36	74	Modèle équilibré avec peu d'erreurs.
V2	499 414	531	2	53	Presque aucun cas positif manqué, mais plus de faux positifs.
V3	408 858	68	90 558	516	Bonne précision mais manque de nombreux positifs.
V4	400 755	60	98 661	524	Nombre élevé de faux négatifs indique un sousapprentissage. Le modèle ne parvient pas à identifier correctement les cas positifs.

Table 5.8 - Analyse des versions pour SVM

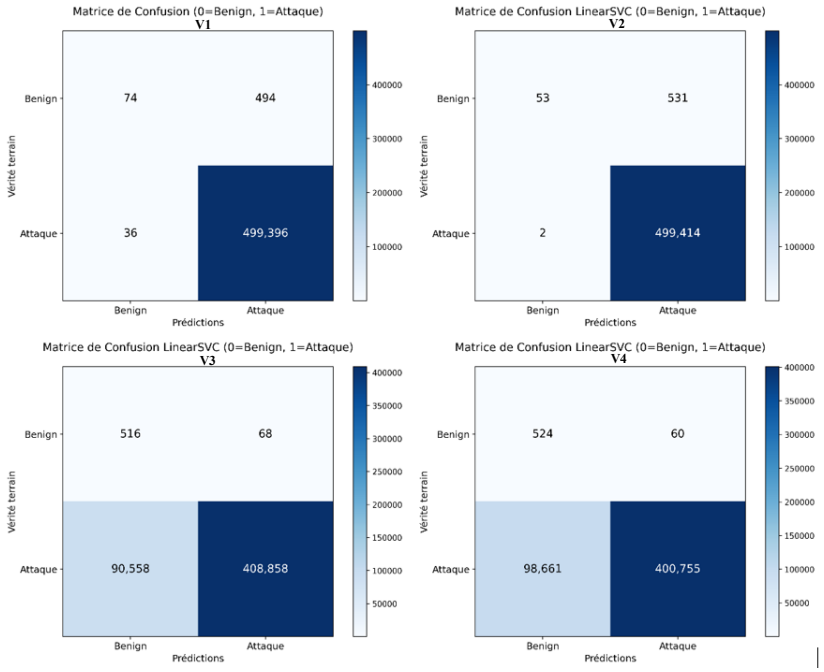

Figure 5.15 - Matrices de confusion SVM

Conclusion : Le modèle V2 est le meilleur, avec une sensibilité quasi parfaite et une excellente précision. Malgré un léger excès de faux positifs, il surpasse les autres versions, surtout celles en net sous-apprentissage (V3, V4).

5.4.2. Comparaison par version

- **Version 1 :**

Modèle	TP	FP	FN	TN	Analyse synthétique
RF	15 001 875	0	0	17 059	Parfait (aucune erreur)
XGBoost	15 001 869	12	6	17 047	Excellente performance
NB	14 937 512	14 531	64 363	2 528	Trop d'erreurs sur les deux classes
KNN	499 418	25	14	543	Acceptable mais insuffisant
SVM	499 396	494	36	74	Acceptable mais trop de FP

Table 5.9 - Performances comparées des modèles de la V1

Meilleur modèle V1 : Random Forest, suivi de XGBoost et KNN

- **Version 2 :**

Modèle	TP	FP	FN	TN	Analyse synthétique
RF	14 590 311	1 065	2 725	15 994	Très bon équilibre
XGBoost	14 590 682	144	2 354	16 915	Optimal, haute précision et bon rappel
NB	14 580 354	16 206	12 682	853	Moyen, trop de FP
KNN	499 297	1	119	583	Précis mais faible couverture globale
SVM	499 414	531	2	53	Très sensible mais trop de FP

Table 5.10 - Performances comparées des modèles de la V2

Meilleur modèle V2 : XGBoost (plus équilibré) suivie de RF.

- **Version 3 :**

Modèle	TP	FP	FN	TN	Analyse synthétique
RF	14 591 622	25	1414	17 034	Excellent, hautement précis et sensible
XGBoost	14 590 718	15	2 318	17 044	Légèrement plus conservateur que RF
NB	14 498 575	14 859	94 461	2 200	A éviter, trop de faux positifs
KNN	498 889	4	527	580	Très précis mais faible couverture
SVM	408 858	68	90 558	516	Échec global, déséquilibré et peu fiable

Table 5.11 - Performances comparées des modèles de la V3

Meilleur modèle V3 : Random Forest, suivi de XGBoost.

- **Version 4 :**

Modèle	TP	FP	FN	TN	Analyse synthétique
RF	14 591 871	17	1165	17042	Performances quasi-parfaites.
XGBoost	14 590 283	20	2753	17 039	Bonne performance, mais le nombre de FN indiquant des cas manqués.
NB	5 953 350	438	8,639,686	16 621	Mauvaise performance avec un très grand nombre de FN, indiquant une incapacité à détecter les classes positives.
KNN	498 866	5	550	579	Bonne précision, mais trop peu de TP
SVM	400 755	60	98 661	524	Faible performance, trop de FP et FN

Table 5.12 - Performances comparées des modèles de la V4

Meilleur modèle V4 : Random Forest, suivi de XGBoost

- **Résumé - Meilleur modèle par version :**

Version	Meilleur modèle global	Remarques
V1	**Random Forest**	Zéro erreur, parfait
V2	**XGBoost**	XGBoost : bon équilibre
V3	**Random Forest**	Très bonne détection sur les deux classes
V4	**Random Forest**	Excellente performance avec un très bon équilibre entre TP et FN

Table 5.13 - Meilleur algorithme pour chaque version

5.5. Analyse des Matrices de corrélations

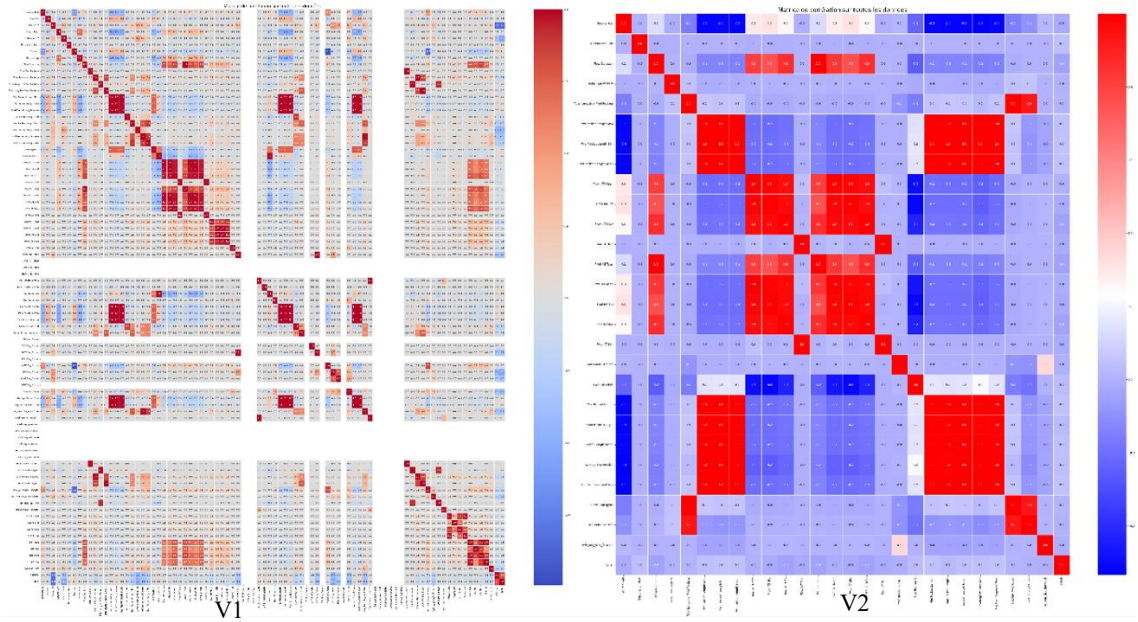

Figure 5.16 - Matrice de corrélation V1 et V2

- **Version 1 : Données brutes non nettoyées**

  Cette version comporte 88 caractéristiques très corrélées entre elles, certaines avec des relations de corrélation qui atteignent la perfection (corrélation = 1). Ce très fort caractère redondant des variables pénalise leur interprétation et la modélisation. Les données brutes sont bruyantes et contiennent des valeurs aberrantes, susceptibles de promouvoir des résultats erronés et de faire passer les fuites de données qui représentent le champ cible.

- **Version 2 : Données nettoyées avec nettoyage basique**

  À l'issue de ce premier nettoyage simple, il apparaît que la taille de la matrice est passée à 28 caractéristiques, et que les corrélations sont moins extrêmes, témoignant d'une réduction du bruit et des valeurs aberrantes. Les relations entre certaines variables sont encore significatives, mais moins redondantes que précédemment.

  Ce niveau de nettoyage facilite l'interprétation des relations, en dégageant éventuellement d'importantes redondances, il serait cependant intéressant d'approfondir davantage la sélection de caractéristiques pour gagner en performance dans les modèles.

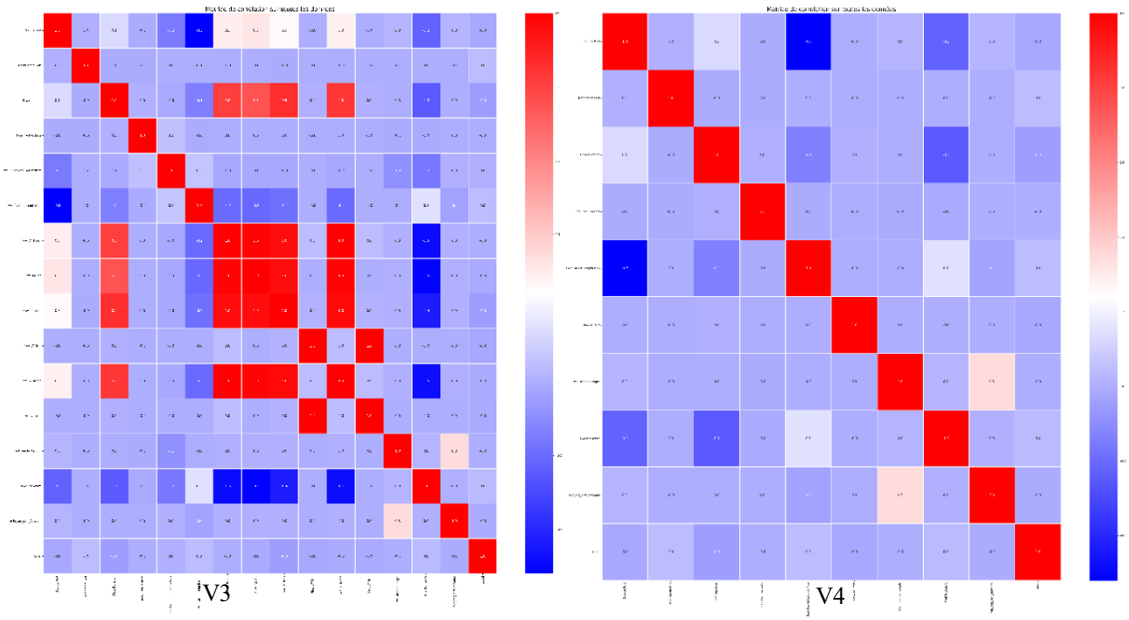

Figure 5.17 - Matrice de corrélation V3 et V4

- **Version 3 : Elimination des variables corrélées à 1**

  Dans cette version, la matrice est de dimension 16*16, une des deux variables ayant une corrélation parfaite (1) a été supprimée. On réduit donc la redondance et on ne garde qu'une seule variable représentative d'une relation donnée. Si la réduction de la redondance est un premier pas dans le bon sens, il reste à examiner les corrélations restantes pour veiller à ce qu'il n'y ait plus d'autres variables fortement corrélées pouvant encore occasionner bruit ou multi colinéarité dans le modèle.

- **Version 4 : Elimination approfondie des variables corrélées**

  L'objectif de cette édition consiste à supprimer systématiquement les variables dont la corrélation est plus grande que 0.70, pour ce faire, nous retenons la variable ayant la pente de régression la plus porteuse, c'est-à-dire plus ou moins grande que 45 degrés.

  Nous obtenons alors une matrice de dimension 10 * 10, afin de bénéficier d'une multi colinéarité minimisée et d'une variabilité maximale. En enlevant les variables corrélées, nous construisons ainsi un modèle plus robuste et moins à même de surajuster, et par conséquent plus explicite. Chaque variable retenue a davantage d'impact sur le modèle lui-même.

  L'élimination systématique des variables corrélées donne comme résultat un ensemble de départ plus net, moins corrélé et mieux équilibré, il apparaît alors favorable au développement de modèles en machine Learning.

5.6. Comparaison avec d'autres études

Le tableau suivant comporte les résultats de quelques versions de modèles de notre étude comparés avec les résultats de deux études précédentes sur le même jeu de données. Les modèles RF, NB, KNN et SVM de l'étude [29] et le modèle XGBoost de l'étude [31].

		Résultats de notre étude				Résultats de [29], [31]			
Ver	Model	Accuracy	Précision	Recall	F1-Score	Accuracy	Précision	Recall	F1-Score
V4	RF	0.9999	1.00	0.9995	1.00	0.99	0.99	0.99	0.99
V3	XGB	0.9998	1.00	0.9998	0.9999	0.98	1.00	1.00	1.00
V2	NB	0.9980	0.9989	0.9991	0.9990	0.45	0.66	0.54	0.38
V2	KNN	0.9998	1.00	0.9998	0.9999	0.98	0.99	0.99	0.99
V2	SVM	0.9989	0.9989	1.00	0.9995	0.86	0.86	0.87	0.85

Table 5.14 – Comparaison des performances des modèles avec des études précédentes

Les résultats de notre étude sont impressionnants et surpassent les études de référence. En effet, le Random Forest (RF) est en tête avec une exactitude incroyable de 0.9999, suivi de près par le XGBoost (XGB) qui affiche tout de même 0.9998, révélant ainsi leur grande efficacité. Le Naive Bayes (NB), en dépit de ses critiques passées, grimpe à 0.9980, passant ainsi au-dessus des seulement 0.45 dans l'étude précédente. Les K-Nearest Neighbors (KNN) et les Support Vector Machine (SVM) se montrent également compétents en proposant des précisions et recalls presque parfaits. Ce travail souligne non seulement le progrès des algorithmes, mais encore plus l'importance de la sélection des bonnes versions pour optimiser les performances en classification. Globalement, ces résultats traduisent une avancée importante de la capacité des modèles à bien classer les données et ouvrent des perspectives intéressantes vers les applications de demain.

5.7. Optimisation et Perspectives

Dans les sections précédentes, nous avons analysé les performances de cinq modèles de machine learning : Random Forest, SVM, Naive Bayes, XGBoost et k-NN, appliqués à quatre versions du jeu de données CIC-DDoS2019, prétraitées pour estimer l'impact de ces opérations sur l'exactitude des modèles.

Cette étude a fait ressortir les forces et faiblesses des algorithmes susmentionnés. D'une part, Random Forest et XGBoost se sont avérés relativement robustes, d'autre part, Naive Bayes et k-NN ont montré une vulnérabilité face à la variabilité des données.

Dans cette section, nous proposons des approches diverses pour améliorer les performances de détection, en intégrant des techniques avancées de feature engineering, d'optimisation des hyperparamètres et l'intérêt des modèles hybrides combinant deep learning et machine learning, capables de détecter des caractéristiques complexes tout en garantissant une classification rapide.

Ces pistes ont pour but de renforcer la robustesse et la généralisation des systèmes de détection pour les rendre compatibles aux environnements réels, en particulier dans le champ des attaques DDoS de plus en plus évoluées.

5.7.1. Amélioration des modèles

- **Ingénierie des caractéristiques :** est l'ensemble d'outils techniques qui s'appliquent sur les données après la collecte et le nettoyage des données pour améliorer la qualité des variables qui seront fournies aux algorithmes d'apprentissage automatique. Elle consiste à transformer, à créer ou à combiner les caractéristiques existantes pour obtenir davantage d'information

utile à la modélisation. Elle peut aller de l'application de méthodes statistiques à des transformations ou à l'utilisation de méthodes de réduction de dimension. [58]

Il s'agit cependant de ne pas mélanger l'ingénierie des caractéristiques et la sélection de caractéristiques qui vise à retirer les variables redondantes ou peu pertinentes, mais aussi faire la différence avec l'extraction de caractéristiques qui vise à créer de nouvelles variables synthétiques telles que l'analyse en composantes principales (ACP). Dans tous les cas, les frontières sont parfois floues car des techniques avancées de nettoyage peuvent être considérées comme de l'ingénierie des caractéristiques.[58]

- ***Technique proposée « Sélection de caractéristiques basée sur le Binary Grey Wolf Optimizer BGWO » :***

  L'algorithme BGWO est un algorithme d'optimisation inspiré par le mode de chasse du loup gris. C'est une méthode heuristique, qui se veut efficace dans le cas d'un espace de solutions pour trouver un sous-ensemble de caractéristiques optimal. L'objectif est de réduire la taille de l'ensemble de données en se débarrassant des caractéristiques superflues artéfactuelles ou redondantes tout en améliorant les performances des modèles de classification. Cette méthode se révèle particulièrement appropriée aux cas d'utilisation des réseaux définis par logiciel (SDN), où la quantité importante ainsi que la rapidité des données requièrent une grande efficacité.
- ***Etude de cas :*** l'étude [59] appliquée au jeu CSE-CIC-IDS2018 combine prétraitement, sélection de caractéristiques par BGWO (pour réduire le jeu initial à un sous-ensemble optimal de caractéristiques) et entraînement de modèles (RF, SVM, k-NN, DT, XGBoost). Random Forest se distingue par les meilleurs scores (précision, rappel, F1, AUC). L'optimisation des caractéristiques améliore à la fois les performances et le temps de calcul, essentiel en temps réel.

- **Optimisation des hyperparamètres :** Elle améliore les performances des modèles en ajustant des valeurs définies avant l'entraînement, influençant directement des métriques clés comme la précision ou l'AUC. Ce processus est souvent traité comme une optimisation en boîte noire [64].
  - ***Technique proposée « Sélection par RFE suivie d'une recherche par grille » :*** La méthode proposée repose sur deux étapes principales :

    Tout d'abord la sélection des caractéristiques : le Recursive Feature Elimination (RFE) élimine progressivement les attributs les moins pertinents du jeu CIC-DDoS2019 en fonction de la performance du modèle. Cela réduit la dimension, élimine le bruit et conserve les informations utiles à la classification.

    Le second pallier consiste en l'optimisation des hyperparamètres du Random Forest via la recherche par grille (Grid Search) pour optimiser ses hyperparamètres. Elle permet de tester toutes les combinaisons des paramètres d'un espace donné pour déterminer laquelle optimise les performances du modèle.

    Les hyperparamètres concernés : « n_estimators » nombre total d'arbres dans la forêt, « max_depth » profondeur maximale de chaque arbre, « min_samples_split » nombre

minimum d'échantillons pour diviser un nœud, et « max_features » nombre de caractéristiques à prendre en compte à chaque division.

L'objectif final étant d'améliorer la précision, le f1-score et la courbe AUC tout en limitant le sur-apprentissage. [61]

- ***Etude de cas :*** Dans l'étude [61], cette approche est appliquée au jeu CIC-DDoS2019, riche en types d'attaques et réaliste. Après réduction via RFE, le Random Forest est affiné par Grid Search. Les résultats montrent une excellente performance, validée par une étude parue en 2025 dans Scientific Reports, qui atteint jusqu'à 99,99 % de précision grâce à cette double optimisation.

5.7.2. Approches hybrides combinaison de Machine Learning et Deep Learning

Le Système d'Apprentissage Hybride (HLS) combine des algorithmes classiques de machine learning (ML) avec des techniques avancées de deep learning (DL), afin d'optimiser la performance et l'interprétabilité des modèles. Cette approche exploite les atouts des deux paradigmes pour traiter des problèmes complexes aux données hétérogènes, là où ni le ML ni le DL seuls ne suffisent [66].

Les modèles classiques de machine learning sont généralement plus rapides à entraîner et donc plus faciles à interpréter, ce qui les rend les mieux adaptés aux cas où la capacité d'expliquer le modèle est fondamentale. En revanche, les modèles d'apprentissage profond, grâce à leurs nombreuses couches et à leur architecture complexe, sont très adaptés pour apprendre des représentations hiérarchiques de caractéristiques, notamment lorsqu'ils sont appliqués à des données non structurées comme des images, du texte ou des séries temporelles [63].

- **Technique proposée « Extraction de caractéristiques par CNN, classification par RF et MLP » :** L'étude [64] propose un modèle hybride où un CNN extrait automatiquement des caractéristiques du trafic brut, évitant l'ingénierie manuelle. Ces représentations sont ensuite classées via Random Forest, robuste au surapprentissage, et MLP, adapté aux relations non linéaires.
- **Etude de cas** : Testée sur CIC-DDoS2019, l'étude [64] montrent que la combinaison CNN + RF/MLP atteint une haute précision. Elle exploite la capacité du deep learning à détecter des motifs complexes et la rapidité des classifieurs traditionnels, offrant une solution prometteuse pour la sécurité réseau.

5.7.3. Limites des modèles traditionnels

Les modèles classiques comme Random Forest, SVM ou k-NN sont efficaces contre les attaques DDoS, mais limités face à des environnements dynamiques, des volumes massifs de données ou des attaques complexes. Leur faiblesse réside dans la mauvaise représentation des relations entre entités réseau, réduisant leur performance en cas de trafic interconnecté [65].

Pour dépasser ces limites, des approches basées sur les graphes émergent. Elles modélisent les hôtes et flux réseau sous forme de nœuds et d'arêtes, permettant une meilleure compréhension des relations entre entités. Les Graph Neural Networks (GNN) se révèlent efficaces pour détecter

des comportements anormaux, notamment lors d'attaques ciblées ou distribuées où ces relations sont cruciales [65].

Sur un autre plan, l'intelligence artificielle fédérée permet d'entraîner les modèles localement, sans transfert de données vers un serveur central. Elle préserve la confidentialité, réduit les coûts de communication et s'adapte bien aux environnements distribués sensibles [66].

5.7.4. Perspective pour une détection temps réel et intégration dans des environnements modernes

Dans une vision d'avenir, la détection des attaques DDoS doit évoluer vers une intégration en temps réel dans des infrastructures intelligentes, autonomes et contextuelles. Cela inclut l'usage de modèles de machine learning dans les architectures cloud/SDN et les environnements ultradynamiques comme l'Internet des Véhicules (IoV). [67]

- **Intégration des modèles de détection dans les environnements Cloud et SDN :** L'alliance du SDN (centralisé et dynamique) avec le cloud permet l'intégration de systèmes de détection intelligents, assurant surveillance continue, analyse comportementale via machine learning et réponse automatisée aux attaques [67].
  L'étude [67] propose un modèle hybride (supervisé/non supervisé) déployé dans un contrôleur SDN en cloud. Il assure une surveillance en temps réel, atteint 93,3 % de précision et adapte automatiquement ses règles de filtrage.
- **Détection en temps réel des attaques DDoS dans l'Internet des Véhicules (IoV) :** l'Internet des Véhicules nécessite une détection rapide, car toute coupure de communication peut être critique. D'où l'importance de systèmes embarqués réactifs, capables d'analyser localement le trafic réseau. Les auteurs de l'étude [68] proposent un modèle combinant RNN et mécanisme d'attention pour détecter les attaques DDoS en IoV. Exploitant des caractéristiques comme le volume, la fréquence et la durée, il identifie les anomalies en temps réel avec 96,63 % de précision, prouvant sa viabilité pour une intégration embarquée.

5.8. Conclusion

Les résultats de l'analyse effectuée dans ce chapitre permettent de valider que les modèles d'ensemble comme Random Forest et XGBoost sont ceux offrant les meilleures performances de façon globale, notamment en matière de généralisation, notamment en raison d'un bon compromis entre précision et rappel, mais que des modèles moins élaborés comme Naive Bayes, ou KNN, peuvent s'avérer efficaces dans des contextes particuliers. Par ailleurs, les courbes d'apprentissage et les matrices de confusion présentées ont permis de faire ressortir certains cas de sousapprentissage et de sur-apprentissage, ce qui montre l'intérêt de procéder à un minutieux réglage des hyperparamètres des modèles. Pour finir, les prospectives ouvertes par les approches hybrides ou temps réel laissent présager des systèmes plus performants et adaptatifs, susceptibles d'être mis en œuvre dans des environnements modernes et exigeants.

Conclusion générale

La détection des attaques DDoS au sein des systèmes d'information constitue, à l'heure actuelle, un enjeu critique pour la cybersécurité, tant ces attaques se sont multipliées, diversifiées et sophistiquées. Dans ce mémoire, nous avons étudié la contribution des techniques de prétraitement des données à la performance des modèles de classification mis en œuvre dans le cadre de la détection des attaques DDoS, en nous basant sur le jeu de données riche et complexe CIC-DDoS2019.

De manière structurante et expérimentale, nous avons conçu quatre versions de notre dataset, chacune enrichie d'une combinaison croissante de techniques, à savoir nettoyage, normalisation, équilibrage (SMOTE, ADASYN) et sélection de caractéristiques. Nous avons entraîné et évalué cinq algorithmes classiques (Random Forest, XGBoost, Naïve Bayes, KNN, SVM) sur des métriques rigoureuses (accuracy, recall, F1-score, AUC-ROC, temps d'exécution) et sur des analyses visuelles (courbe d'apprentissage, matrices de confusion et de corrélations).

Les résultats de l'expérimentation mettent effectivement en évidence l'impact décisif des prétraitements sur la qualité de la détection. Dans la plupart des cas, l'ajout progressif de traitements améliore les résultats des modèles, en particulier des modèles d'ensemble (Random Forest et XGBoost) qui s'avèrent généralement plus robustes et efficaces que les algorithmes plus simples, comme Naïve Bayes ou KNN, très sensibles à la structure des données et aux déséquilibres. Toutefois, certaines méthodes d'équilibrage ou de sélection de caractéristiques permettent de réduire le surapprentissage ou d'accélérer les temps de calcul sans nuire à la précision.

Ainsi ce travail met en évidence l'importance primordiale d'un prétraitement adéquat et maîtrisé au sein des systèmes de détection d'intrusions par IA. En effet, l'efficacité d'un modèle ne repose pas uniquement sur sa complexité ou ses hyperparamètres, mais surtout sur la qualité des données d'entrée.

Une des principales limites a été la gestion d'importants volumes de données (près de 50 millions de lignes). Ainsi, malgré l'utilisation d'un environnement Colab Pro + avec TPU et 334 Go de RAM, des problèmes de dépassement de mémoire ont été régulièrement rencontrés dans l'exploitation complète du dataset. Ce qui a nécessité de nombreuses adaptations comme du traitement par lots, de la réduction de la dimension ou de l'échantillonnage dans certains cas.

Dans cette optique, ce travail ouvre donc plusieurs nouvelles perspectives d'amélioration : d'une part, l'exploration de modèles hybrides incluant Machine Learning et Deep Learning, d'autre part, l'adaptation du modèle à des applications d'environnement temps réel, et encore l'intégration dans des systèmes de détection actifs et distribués. De fait, les défenses face aux attaques DDoS ne peuvent se gagner qu'à la condition d'une compréhension fine, dynamique et rigoureuse des données elles-mêmes.

Bibliographie

[1] R. Tandon, « A Survey of Distributed Denial of Service Attacks and Defenses », 4 août 2020, *arXiv*: arXiv:2008.01345. doi: 10.48550/arXiv.2008.01345.

[2] C. Sandeep et M. Sadanandam, « A Comprehensive Survey Report on Distributed Denial of Service(DDoS) Attacks and Possible Defense Mechanisms », *Int. J. Res.*, vol. 04, nº 13, 2017, [En ligne]. Disponible sur: https://journals.pen2print.org/index.php/ijr/article/download/9413/9081

[3] S. Kumar, Dr. N. P. Singh, et Dr. N. Kumar, « Literature Review of Distributed Denial of Service (DDoS) Attacks, its Detection Techniques and Prevention Mechanisms », *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, nº 9, p. 1681-1685, sept. 2022, doi: 10.22214/ijraset.2022.46882.

[4] M. K. Rafsanjani et N. Kazeminejad, « Distributed denial of service attacks and detection mechanisms », *J. Comput. Methods Sci. Eng.*, vol. 14, nº 6, p. 329-345, févr. 2015, doi: 10.3233/JCM-140510.

[5] V. Parekh et M. Saravanan, « An Empirical Overview on DDoS: Taxonomy, Attacks, Tools and Attack Detection Mechanism », in *Proceedings of International Conference on Recent Trends in Computing*, vol. 600, R. P. Mahapatra, S. K. Peddoju, S. Roy, et P. Parwekar, Éd., in Lecture Notes in Networks and Systems, vol. 600. , Singapore: Springer Nature Singapore, 2023, p. 151-161. doi: 10.1007/978-981-19-8825-7_14.

[6] M. Eklund et P. Ståhlberg, « Distributed denial of service attacks Protection, Mitigation, and Economic Consequences », Bachelor's Thesis, KTH Royal Institute of Technology School of Information and Communication Technology (ICT) Department of Communication Systems, SE-100 44 Stockholm, Sweden, 2015. [En ligne]. Disponible sur: http://www.diva-portal.org/smash/get/diva2:842434/FULLTEXT01

[7] Rutvij Upadhyay et Dr. Shivam Upadhyay, « A Review Paper on Detection and Mitigation of DDoS Attacks », *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 11, nº 1, p. 704-708, janv. 2025, doi: 10.32628/CSEIT25111280.

[8] L. Poonia et S. Tinker, « A Comprehensive Analysis of the Types, Impacts, Prevention, and Mitigation of DDoS Attacks », *Recent Pat. Eng.*, vol. 19, oct. 2024, doi: 10.2174/0118722121322166240828112546.

[9] G. Somani, M. S. Gaur, et D. Sanghi, « DDoS/EDoS attack in cloud: affecting everyone out there! », in *Proceedings of the 8th International Conference on Security of Information and Networks*, Sochi Russia: ACM, sept. 2015, p. 169-176. doi: 10.1145/2799979.2800005.

[10] A. Abhishta, R. Joosten, S. Dragomiretskiy, et L. J. M. Nieuwenhuis, « Impact of Successful DDoS Attacks on a Major Crypto-Currency Exchange », in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Pavia, Italy: IEEE, févr. 2019, p. 379-384. doi: 10.1109/EMPDP.2019.8671642.

[11] K. Dizdar, Z. Moric, V. Dakic, et M. Basic, « Denial of Service Attacks Using the Example of Croatian Hosters », in *DAAAM Proceedings*, 1re éd., vol. 1, B. Katalinic, Éd., DAAAM International Vienna, 2023, p. 0088-0095. doi: 10.2507/34th.daaam.proceedings.013.

[12] M. Bogdanoski et A. Risteski, « Wireless Network Behavior under ICMP Ping Flood DoS Attack and Mitigation Techniques », *Int J Commun Netw. Inf Secur*, vol. 3, nº 1, p. 17-24, nov. 2011.

[13] T. KIATTIKUL, « PERFORMANCE ANALYSIS OF DEFENSE MECHANISMS AGAINST UDP FLOOD ATTACKS », M.S, Unitec Institute of Technology, Auckland, New Zealand, 2014. [En ligne]. Disponible sur: https://www.researchbank.ac.nz/server/api/core/bitstreams/0205e0d7-9907-4c78-a888-c4449a121182/content

[14] Sanjay, B. Rajendran, et P. Shetty D., « DNS Amplification & DNS Tunneling Attacks Simulation, Detection and Mitigation Approaches », in *2020 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, India: IEEE, févr. 2020, p. 230-236. doi: 10.1109/ICICT48043.2020.9112413.

[15] A. S. Dhaliwal, « Detection and mitigation of SYN and HTTP flood DDoS attacks in software defined networks », Master of Applied Science, Toronto Metropolitan University, Toronto, Ontario, Canada, 2021. doi: 10.32920/ryerson.14647329.v1.

[16] M. Dimolianis, A. Pavlidis, et V. Maglaris, « SYN Flood Attack Detection and Mitigation using Machine Learning Traffic Classification and Programmable Data Plane Filtering », in *2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, France: IEEE, mars 2021, p. 126-133. doi: 10.1109/ICIN51074.2021.9385540.

[17] K. S. D. Vairale, « An Overview of OSPF and its Attack », *Int. J. Comput. Appl.*, vol. 0975, nº 8887, p. 15-18, nov. 2015.

[18] M. A. Al-Naeem, « Prediction of Re-Occurrences of Spoofed ACK Packets Sent to Deflate a Target Wireless Sensor Network Node by DDOS », *IEEE Access*, vol. 9, p. 87070-87078, 2021, doi: 10.1109/ACCESS.2021.3089683.

[19] D. Tymoshchuk et V. Yatskiv, « SLOWLORIS DDOS DETECTION AND PREVENTION IN REAL-TIME », in *THEORETICAL AND EMPIRICAL SCIENTIFIC RESEARCH: CONCEPT AND TRENDS*, European Scientific Platform, août 2024. doi: 10.36074/logos-16.08.2024.036.

[20] V. Rios, P. Inacio, D. Magoni, et M. Freire, « Detection of Slowloris Attacks using Machine Learning Algorithms », in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, Avila Spain: ACM, avr. 2024, p. 1321-1330. doi: 10.1145/3605098.3635919.

[21] C. Kemp, C. Calvert, T. M. Khoshgoftaar, et J. L. Leevy, « An approach to application-layer DoS detection », *J. Big Data*, vol. 10, nº 1, p. 22, févr. 2023, doi: 10.1186/s40537-023-00699-3.

[22] Z. B. A. A. MOHD, « Study on Mitigation of SMTP Flood Attack using Software Defined Network », 20 mars 2018, *Kyushu University*. doi: 10.15017/1931935.

[23] H. M. Alqahtani et M. Abdullah, « A Review on DDoS Attacks Classifying and Detection by ML/DL Models », *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, nº 2, janv. 2024, doi: 10.14569/IJACSA.2024.0150283.

[24] K. S. S. Reddy, V. Krishna, M. Prabhakar, P. Srilatha, K. G. Gupta, et R. A. Kumar, « Machine Learning Based Classification Model for Network Traffic Anomaly Detection », *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 11, nº 7s, p. 563-576, juill. 2023, doi: 10.17762/ijritcc.v11i7s.7048.

[25] R. M. A. Haseeb-ur-rehman *et al.*, « High-Speed Network DDoS Attack Detection: A Survey », *Sensors*, vol. 23, nº 15, p. 6850, août 2023, doi: 10.3390/s23156850.

[26] A. Hirsi, L. Audah, A. Salh, M. A. Alhartomi, et S. Ahmed, « Detecting DDoS Threats Using Supervised Machine Learning for Traffic Classification in Software Defined Networking », *IEEE Access*, vol. 12, p. 166675-166702, 2024, doi: 10.1109/ACCESS.2024.3486034.

[27] H. Amaad et H. Mughal, « Experimenting Ensemble Machine Learning for DDoS Classification: Timely Detection of DDoS Using Large Scale Dataset », in *2023 4th International Conference on Advancements in Computational Sciences (ICACS)*, Lahore, Pakistan: IEEE, févr. 2023, p. 1-7. doi: 10.1109/ICACS55311.2023.10089656.

[28] H. Dhadhal et P. Kotak, « Leveraging datasets for effective mitigation of DDoS attacks in software-defined networking: significance and challenges », *Radioelectron. Comput. Syst.*, vol. 2024, nº 2, p. 136-146, avr. 2024, doi: 10.32620/reks.2024.2.11.

[29] S. Kiourkoulis, « DDoS datasets: Use of machine learning to analyse intrusion detection performance », Master Thesis Project, Luleå University of Technology, Luleå, Sweden, 2020. [En ligne]. Disponible sur: https://www.researchgate.net/publication/342047060_DDoS_datasets_Use_of_machine_learning_to_analyse_intrusion_detection_performance

[30] A. Fethi et Y. Litim, « Détection des attaques DDOS dans le Cloud Computing », Master Thesis Project, Université Abd Elhafid Boussouf Mila, Mila, Algerie, 2021. [En ligne]. Disponible sur: https://dspace.centre-univ-mila.dz/jspui/bitstream/123456789/1396/1/Détection%20des%20attaques%20DDOS%20dans%20le%20Cloud.pdf

[31] D. K. Suvra, « An Efficient Real Time DDoS Detection Model Using Machine Learning Algorithms », 24 janvier 2025, *arXiv*: arXiv:2501.14311. doi: 10.48550/arXiv.2501.14311.

[32] C. El Morr, M. Jammal, H. Ali-Hassan, et W. El-Hallak, « Data Preprocessing », in *Machine Learning for Practical Decision Making*, vol. 334, in International Series in Operations Research & Management Science, vol. 334. , Cham: Springer International Publishing, 2022, p. 117-163. doi: 10.1007/978-3-031-16990-8_4.

[33] M. Ali Mohammed, R. A. Hamid, et R. Razzaq AbdulHussein, « Data Collection and Preprocessing in Web Usage Mining: Implementation and Analysis », *Iraqi J. Comput. Inform.*, vol. 50, nº 2, p. 54-74, nov. 2024, doi: 10.25195/ijci.v50i2.486.

[34] Gauri Dhongade, Dr. Omprakash Chandrakar, et Dr. Rajeshree Khande, « Enhancing Cyber Security : A Study of Data Preprocessing Techniques for Cyber Security Datasets », *Int. J. Sci. Res. Sci. Technol.*, vol. 11, nº 5, p. 71-75, sept. 2024, doi: 10.32628/IJSRST2411427.

[35] J. J. Davis et A. J. Clark, « Data preprocessing for anomaly based network intrusion detection: A review », *Comput. Secur.*, vol. 30, nº 6-7, p. 353-375, sept. 2011, doi: 10.1016/j.cose.2011.05.008.

[36] X. Cui, G. Yin, et X. Teng, « Research on the Data Pre-Processing in the Network Abnormal Intrusion Detection », *Open Autom. Control Syst. J.*, vol. 6, nº 1, p. 1228-1232, déc. 2014, doi: 10.2174/1874444301406011228.

[37] D. C. Corrales, A. Ledezma, et J. C. Corrales, « From Theory to Practice: A Data Quality Framework for Classification Tasks », *Symmetry*, vol. 10, nº 7, p. 248, juill. 2018, doi: 10.3390/sym10070248.

[38] S. Nanga *et al.*, « Review of Dimension Reduction Methods », *J. Data Anal. Inf. Process.*, vol. 09, nº 03, p. 189-231, 2021, doi: 10.4236/jdaip.2021.93013.

[39] E. Jafarigol et T. Trafalis, « A Review of Machine Learning Techniques in Imbalanced Data and Future Trends », 2023, *arXiv*. doi: 10.48550/ARXIV.2310.07917.

[40] F. Morandín-Ahuerma, « What is Artificial Intelligence? », *Int. J. Res. Publ. Rev.*, vol. 03, nº 12, p. 1947-1951, 2022, doi: 10.55248/gengpi.2022.31261.

[41] V. Jain et S. K. Tiwari, « OVERVIEW: MACHINE LEARNING », in *Machine Learning An Art of Computer Thinking*, Mrs. S. Mudgil, Dr. S. J. Choudhary, et Prof. S. K. Tiwari, Éd., Iterative International Publishers, Selfypage Developers Pvt Ltd, 2024, p. 130-144. doi: 10.58532/nbennurch183.

[42] E. Altulaihan, M. A. Almaiah, et A. Aljughaiman, « Anomaly Detection IDS for Detecting DoS Attacks in IoT Networks Based on Machine Learning Algorithms », *Sensors*, vol. 24, nº 2, p. 713, janv. 2024, doi: 10.3390/s24020713.

[43] R. Mohammadi, C. Lal, et M. Conti, « HTTPScout: A Machine Learning based Countermeasure for HTTP Flood Attacks in SDN », *Int. J. Inf. Secur.*, vol. 22, nº 2, p. 367-379, avr. 2023, doi: 10.1007/s10207-022-00641-3.

[44] P. A. W. Purnama et N. Pohan, « Naive Bayes Algorithm Classification for Predicting Graduation Rate », *J. Teknol. DAN OPEN SOURCE*, vol. 7, nº 2, p. 138-144, déc. 2024, doi: 10.36378/jtos.v7i2.3866.

[45] D. Arifah, T. H. Saragih, D. Kartini, M. Muliadi, et M. I. Mazdadi, « Application of SMOTE to Handle Imbalance Class in Deposit Classification Using the Extreme Gradient Boosting Algorithm », *J. Ilm. Tek. Elektro Komput. Dan Inform.*, vol. 9, nº 2, p. 396-410, juin 2023, doi: 10.26555/jiteki.v9i2.26155.

[46] G. Zhou, J. Gao, D. Zuo, J. Li, et R. Li, « MSXFGP: combining improved sparrow search algorithm with XGBoost for enhanced genomic prediction », *BMC Bioinformatics*, vol. 24, nº 1, p. 384, oct. 2023, doi: 10.1186/s12859-023-05514-7.

[47] M. M. Şimşek et E. Atılgan, « DoS and DDoS Attacks on Internet of Things and Their Detection by Machine Learning Algorithms », *DÜMF Mühendis. Derg.*, avr. 2024, doi: 10.24012/dumf.1421337.

[48] Y. Wei, J. Jang-Jaccard, F. Sabrina, A. Singh, W. Xu, et S. Camtepe, « AE-MLP: A Hybrid Deep Learning Approach for DDoS Detection and Classification », *IEEE Access*, vol. 9, p. 146810-146821, 2021, doi: 10.1109/ACCESS.2021.3123791.

[49] D. Hoiem, T. Gupta, Z. Li, et M. M. Shlapentokh-Rothman, « Learning Curves for Analysis of Deep Networks », 5 avril 2021, *arXiv*: arXiv:2010.11029. doi: 10.48550/arXiv.2010.11029.
[50] S. Mendez-Moreno, « Syntropy in complex systems: A complement to Shannon's Entropy », 18 mars 2024, *arXiv*: arXiv:2403.13022. doi: 10.48550/arXiv.2403.13022.
[51] S. Raschka, J. Patterson, et C. Nolet, « Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence », 31 mars 2020, *arXiv*: arXiv:2002.04803. doi: 10.48550/arXiv.2002.04803.
[52] Dr. S. R. Sukhdeve et S. S. Sukhdeve, « Google Colaboratory », in *Google Cloud Platform for Data Science*, Berkeley, CA: Apress, 2023, p. 11-34. doi: 10.1007/978-1-4842-9688-2_2.
[53] S. R. Sukhdeve et S. S. Sukhdeve, « Data Analytics and Storage », in *Google Cloud Platform for Data Science*, Berkeley, CA: Apress, 2023, p. 161-187. doi: 10.1007/978-1-4842-9688-2_6.
[54] « Package overview — pandas 2.2.3 documentation ». Consulté le: 31 mai 2025. [En ligne]. Disponible sur: https://pandas.pydata.org/docs/getting_started/overview.html
[55] « What is NumPy? — NumPy v2.2 Manual ». Consulté le: 31 mai 2025. [En ligne]. Disponible sur: https://numpy.org/doc/stable/user/whatisnumpy.html
[56] D. M. McGreggor, *Mastering matplotlib: a practical guide that takes you beyond the basics of matplotlib and gives solutions to plot complex data*, 1st ed. in Community Experience Distilled. Place of publication not identified: Packt Publishing, 2015.
[57] « An introduction to seaborn — seaborn 0.13.2 documentation ». Consulté le: 31 mai 2025. [En ligne]. Disponible sur: https://seaborn.pydata.org/tutorial/introduction.html
[58] T. Verdonck, B. Baesens, M. Óskarsdóttir, et S. Vanden Broucke, « Special issue on feature engineering editorial », *Mach. Learn.*, vol. 113, nº 7, p. 3917-3928, juill. 2024, doi: 10.1007/s10994-021-06042-2.
[59] Z. Liu, Y. Wang, F. Feng, Y. Liu, Z. Li, et Y. Shan, « A DDoS Detection Method Based on Feature Engineering and Machine Learning in Software-Defined Networks », *Sensors*, vol. 23, nº 13, p. 6176, juill. 2023, doi: 10.3390/s23136176.
[60] R. G. Mantovani *et al.*, « Better Trees: An empirical study on hyperparameter tuning of classification decision tree induction algorithms », 2018, doi: 10.48550/ARXIV.1812.02207.
[61] M. S. Sawah, H. Elmannai, A. A. El-Bary, Kh. Lotfy, et O. E. Sheta, « Distributed denial of service (DDoS) classification based on random forest model with backward elimination algorithm and grid search algorithm », *Sci. Rep.*, vol. 15, nº 1, p. 19063, mai 2025, doi: 10.1038/s41598-025-03868-x.
[62] Vedika Bengani, « Hybrid Learning Systems: Integrating Traditional Machine Learning with Deep learning Techniques. », 2024, doi: 10.13140/RG.2.2.10461.22244/1.
[63] I. Goodfellow, Y. Bengio, et A. Courville, *Deep learning*. in Adaptive computation and machine learning. Cambridge, Mass: The MIT press, 2016.
[64] N. J. Shohan, G. Tanbhir, F. Elahi, A. Ullah, et Md. N. Sakib, « Enhancing Network Security: A Hybrid Approach for Detection and Mitigation of Distributed Denial-of-Service Attacks Using Machine Learning », 2025, doi: 10.48550/ARXIV.2503.05477.
[65] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, et P. S. Yu, « A Comprehensive Survey on Graph Neural Networks », 2019, doi: 10.48550/ARXIV.1901.00596.
[66] Q. Yang, Y. Liu, T. Chen, et Y. Tong, « Federated Machine Learning: Concept and Applications », 2019, doi: 10.48550/ARXIV.1902.04885.
[67] A. V. Songa et G. R. Karri, « An integrated SDN framework for early detection of DDoS attacks in cloud computing », *J. Cloud Comput.*, vol. 13, nº 1, p. 64, mars 2024, doi: 10.1186/s13677-024-00625-9.
[68] M. Ababsa, S. Ribouh, A. Malki, et L. Khoukhi, « Deep Multimodal Learning for Real-Time DDoS Attacks Detection in Internet of Vehicles », 2025, *arXiv*. doi: 10.48550/ARXIV.2501.15252.