N^0 104

UNIVERSITE D'ABOMEY-CALAVI

Ecole Doctorale des Sciences de l'Ingénieur (ED-SDI)

Master de Recherche en Télécommunications et Réseaux Informatiques

---

Rapport de stage

Thème :

Classification du Trafic Internet grâce aux Méthodes d'Apprentissage Automatique

Présenté par :

Erick Adjamonsi ADJE

Encadré par

Dr Ing. Vinasétan Ratheil HOUNDJI

Maître-Assistant des Universités du CAMES

Enseignant-Chercheur à IFRI/UAC

Sous la direction de :

Lt-Col. (Dr Ing.) Michel DOSSOU

Maître de Conférences des Universités du CAMES

Enseignant-Chercheur à EPAC/UAC

Laboratoire d'Électrotechnique, de Télécommunications et d'Informatique Appliquée (LETIA)

2020 - 2021

Table des matières

Dédicace

A mon père Alexis A. ADJE

Pour avoir su m'inculquer les valeurs morales et le goût du travail bien fait.

A ma mère Alimatou SOUMANOU

Tu as toujours été là pour moi aussi bien dans les moments difficiles que dans les moments heureux. Chère maman, les mots me manquent pour t'exprimer toute ma gratitude.

A ma sœur Christelle et mes frères Franck, Michaël

Je salue ici la tendre complicité qui a toujours existé entre nous et qui nous a permis de surmonter tant d'obstacles. Que le Seigneur nous unisse davantage afin de relever ensemble d'autres défis qui nous attendent.

Remerciements

Le couronnement de ce travail n'a été possible que grâce au concours direct ou indirect de certaines personnes. J'exprime toute ma reconnaissance à toutes et à tous. Je tiens sincèrement à remercier :

- Tous les enseignants de l'ED-SDI pour avoir accepté de partager une partie de leurs connaissances avec moi.
- Dr. (MA) Max Fréjus O. SANYA, coordonnateur adjoint du Master Télécommunications et Réseaux Informatiques de l'ED-SDI et toute son administration, pour m'avoir permis de suivre cette formation de qualité.
- Lt-Col Dr. (MC) Michel DOSSOU qui, malgré ses nombreuses occupations, a su se rendre disponible pour superviser ce travail ;
- Dr. (MA) Ratheil V. HOUNDJI, pour son encadrement et pour m'avoir accompagné tout au long de ce travail.
- Tous mes camarades étudiants de promotion pour leur sens de partage.
- Tous ceux qui ont participé de près et de loin à la réalisation de ce travail.

Table des figures

Liste des tableaux

Liste des sigles et abréviations

ACK : Acknowledgement

ACM : Association for Computing Machinery

DB : Database (Base de Données)

FAI : Fournisseur d'Accès Internet

FTP : File Transfer Protocol

HTTP : HyperText Transfer Protocol

HTTPS : HyperText Transfer Protocol Secure

IP : Internet Protocol

K-NN : K-Nearest Neighbors

MineNet : Mining Network

P2P : Peer to Peer

POP3 : Post Office Protocol 3

SCR : Somme des Carrés des Résidus

SIGCOMM : Special Interest Group on Data Communications

SIGMETRICS : Association for Computing Machinery's Special Interest Group on Measurement and Evaluation

SMTP : Simple Mail Transfer Protocol

SSH : Secure Shell

SSL : Secure Socket Layer

SVM : Support Vector Machine

TCP : Transmission Control Protocol

Telnet : Terminal Network

UDP : User Datagram Protocol

VoIP : Voice over Internet Protocol

XGBoost : eXtreme Gradient Boosting.

Résumé. L'étude de ce travail consiste à exploter des algorithmes d'apprentissage automatique interprétables tels que : l'arbre de décision, la forêt aléatoire et XGBoost pour proposer des modèles performants destinés à la classification du trafic selon le service qui le génère. L'ensemble de données utilisé pour la mise en place des modèles contient 377 526 trafics. Chaque trafic est décrit par 248 caractéristiques qui représentent principalement des informations telles que les numéros de ports, des statistiques sur les paquets échangés, etc. Un autre ensemble de données constitué de 19 626 trafics est utilisé pour évaluer les modèles. L'étude consiste à construire des modèles puis à analyser l'importance des caractéristiques ayant servi à la construction afin de supprimer les caractéristiques superflues sur plusieurs itérations pour une amélioration des performances. À l'issue de l'étude, on constate que l'arbre de décision est le plus efficace pour détecter les trafics de type attaque. La forêt aléatoire quant à elle donne de très bons résultats avec le moins de caractéristiques. Enfin XGBoost arrive à s'adapter aux déficits de données et donne de très bons résultats malgré les données manquantes. Un des meilleurs modèles construit à partir de 12 caractéristiques, a permis d'atteindre une exactitude globale de 98,40 % sur l'ensemble des données destiné à l'évaluation. Également, ce travail a permis de dégager un ensemble de 14 caractéristiques importantes dans la classification du trafic internet qui se sont démarquées tout au long de l'étude. Parmi ces 14 caractéristiques, deux d'entre elles se sont revélées incontournables. Il s'agit du numéro de port mis en jeu au niveau du serveur et de la taille minimale d'un segment observé au cours du trafic (du client au serveur).

Mots clés : *trafic internet*, *trafic réseau*, *classification du trafic*, *apprentissage automatique*

Abstract. The research of this work consists in exploring interpretable machine learning algorithms such as: decision tree, random forest and XGBoost to propose efficient models for traffic classification according to the service that generates it. The dataset used for model building contains 377,526 traffics. Each traffic is described by 248 features which are information such as port numbers, statistics on packets exchanged, etc. Another dataset consisting of 19,626 traffics was also provided for evaluating the models. The study mainly consists in building models and then analyzing the importance of the features used in the construction in order to remove unnecessary features over many iterations to improve the performance. During the study, we found that the decision tree was the most effective in detecting attack traffic. The random forest gave us very good results with the least number of features. Finally, XGBoost can adapt to data deficits and gives very good results despite missing data. One of the best models built from 12 features achieved an overall accuracy of 98.40% on the evaluation dataset. Also, we identify a set of 14 important features for Internet traffic classification. From these 14 features, two of them were identified as being critical. These are the port number involved at the server side and the minimum size of a segment observed during the traffic (from the client to the server).

Keys words: *internet traffic*, *network traffic*, *traffic classification*, *machine learning*

Introduction

Le trafic internet a considérablement augmenté au cours de la dernière décennie en raison de l'avènement de nouvelles technologies, industries et applications [1]. Un tel niveau de croissance continue, reste un défi pour la gestion du réseau. Une classification précise du trafic internet est fondamentale pour une meilleure gestion des trafics du réseau ; de la surveillance à la sécurité, de la qualité de service à la fourniture de la ressource adéquate.

La classification automatique du trafic peut se faire à différents niveaux tels que : soit on détermine la catégorie du trafic (Exemples : Web, multimédia, base de données, mail, jeux, transfert de fichiers) [2][3][4], soit on détermine le protocole impliqué au niveau de l'application (FTP, HTTP, HTTPS, etc.) [5][6]. On peut également déterminer l'application exacte qui génère le trafic (Skype, uTorrent, etc.) [7]. En général, la classification automatique du trafic part des informations telles que les ports utilisés au niveau du client et du serveur, le contenu des payloads, les caractéristiques du flux ou le comportement des hôtes [8].

Depuis l'avènement de l'apprentissage automatique, la classification du trafic réseau à l'aide de techniques d'apprentissage automatique, supervisées et non supervisées, a suscité un intérêt croissant. En considérant l'état de l'art, nous avons pu relever des limites, dont certaines sont récurrentes. Nous avons abordé quelques-unes de ces limites auxquelles nous tentons d'apporter une solution.

La plupart des travaux d'apprentissage automatique appliqué à la classification du trafic internet, travaillent sur des ensembles de données presque parfaits, sans prendre en compte des cas où parfois certaines caractéristiques du trafic ne sont pas capturées. Plus simplement certaines informations peuvent être présentes pour un trafic donné tandis que pour un autre trafic, nous ne les avons pas. En plus de cela, les performances obtenues jusque-là pourraient être améliorées, car des études poussées sur les caractéristiques exploitées ne sont généralement pas effectuées et peuvent être à la base de certaines mauvaises performances. Malgré les bonnes performances de certains travaux, les modèles de classification utilisés sont efficaces sur certaines classes de trafic, mais sont inefficaces pour d'autres classes de trafic [3].

Notre travail consiste à mettre en place des modèles de classification du trafic internet avec de bonnes performances sur toutes les classes de trafics considérés. Également, nous nous as-

surons de proposer des modèles capables de s'adapter aux manques de données. Au fur et à mesure dans l'évolution de notre étude, nous faisons des analyses sur nos modèles afin d'éliminer les caractéristiques superflues à nos modèles qui apportent de l'information presque nulle et qui empêchent nos modèles d'atteindre de bonnes performances. Ceci nous permet à la fin de retenir un certain nombre de caractéristiques importantes dans la classification du trafic internet.

Problématique

L'explosion du trafic internet demande de plus en plus une gestion minutieuse et automatique du réseau. Un élément important dans la gestion du trafic est de pouvoir classifier chaque trafic du réseau internet afin de garantir la sécurité, la qualité de service, etc. Cependant, à travers les travaux sur l'utilisation de l'apprentissage automatique pour la classification automatique du flux internet, nous remarquons ceci :

- À notre connaissance, il n'y a pas d'étude effectuée sur les caractéristiques à utiliser pour construire des modèles performants pour la classification du trafic internet. En apprentissage automatique, certaines caractéristiques, lorsqu'elles sont utilisées, peuvent apporter du bruit diminuant ainsi les performances des modèles. D'autres également, n'apportent aucune information et rendent le modèle plus complexe et le temps d'apprentissage très long. Nous pensons que cette étude pourrait être nécessaire pour améliorer les performances des modèles ;
- Certains modèles obtenus ne sont pas efficaces pour la détection de certaines classes de flux ;
- À notre connaissance, il n'y a pas eu d'étude sur un modèle qui se construit à partir de données d'entraînement contenant certaines informations manquantes, c'est-à-dire des flux internet pour lesquels certaines caractéristiques n'ont pas pu être capturées. Ainsi, le modèle final aura du mal à traiter de nouveaux flux dont peut-être certaines caractéristiques n'ont pas pu être capturées.

Notre travail consiste à construire de nouveaux modèles de classification automatique avec de bonnes performances permettant de détecter n'importe quelle classe de flux. Pour ce travail, nous utilisons des techniques d'apprentissage automatique (XGBoost par exemple qui à notre connaissance, n'a pas encore été testé sur ce type de problème) capables de nous faire ressortir l'importance des caractéristiques exploitées. Ce qui nous permet d'éliminer les caractéristiques superflues au fur et à mesure. En exploitant également les forces de XGBoost, on pourra proposer des modèles capables de s'adapter au manque de certaines données tout en assurant une bonne performance sur ces modèles.

Objectifs

L'objectif général de ce travail est de pouvoir exploiter plusieurs algorithmes d'apprentissage automatique afin de prédire la catégorie (web, multimédia, base de données, mail, jeux, transfert de fichiers, attaque, P2P, connexion distante) d'un trafic internet. De façon spécifique, il s'agira de :

- proposer différents modèles de classification du trafic internet en exploitant plusieurs algorithmes d'apprentissage automatique à partir d'un ensemble de données relatives aux trafics internet ;
- proposer des modèles performants, capables de s'adapter aux manques de données ;
- dégager un certain nombre de caractéristiques incontournables dans la classification du trafic internet.

Méthodologie d'étude

La démarche que nous adoptons pour mener à bien cette étude peut se décliner en étapes comme suit :

- faire une étude bibliographique des travaux effectués dans la classification du trafic internet ;
- étudier différents algorithmes d'apprentissage automatique qu'on pourrait exploiter dans le cadre de ce travail ;
- identifier et prendre en main les outils nécessaires pour l'étude ;
- trouver des données d'apprentissage sur lesquelles nos algorithmes d'apprentissage automatique pourront s'entraîner ;
- mettre en place plusieurs modèles de classification à partir de plusieurs algorithmes d'apprentissage automatique afin d'en dégager les meilleurs ;
- étudier l'impact des caractéristiques exploitées sur les modèles sélectionnés afin d'éliminer les caractéristiques superflues et améliorer les performances des dits modèles ;
- discuter par rapport aux résultats obtenus et les comparer à ceux des travaux existants.

Plan du document

Notre document est subdivisé en trois chapitres. Le premier chapitre pour donner les généralités sur la classification du trafic à travers plusieurs approches en définissant également les algorithmes d'apprentissage automatique qu'on utilise pour construire les modèles de classification. Dans le second chapitre, nous parlons essentiellement des outils utilisés et nous présentons la méthodologie d'étude qui nous permet de bien exploiter ces outils afin d'atteindre les objectifs fixés sans oublier de décrire les bases de données utilisées pour la mise en place des modèles et leurs évaluations. Enfin, le troisième chapitre présente l'ensemble des résultats obtenus tout au long de ce travail, une discussion générale sur ce qu'on peut retenir du travail effectué, une évaluation des résultats obtenus par rapport aux objectifs fixés et une comparaison avec les travaux de l'état de l'art.

Chapitre 1

Généralités sur la classification du trafic

Introduction

Le sujet de la classification de trafic réseau est d'une grande importance pour la planification de réseau, la gestion de trafic, la gestion de priorité d'applications et le contrôle de sécurité. L'objectif étant de pouvoir construire de nouveaux modèles de classification encore plus performants, nous devons mener une étude sur l'existant. C'est ainsi que tout d'abord, nous définissons la classification du trafic, nous donnons ses objectifs et les approches de classification. Ensuite, nous abordons les algorithmes d'apprentissage automatique qui sont des algorithmes utilisés pour construire des modèles de classification du trafic. Enfin, nous faisons une critique de l'existant en donnant des approches d'amélioration à travers notre travail.

1.1 Définition de la classification du trafic

La classification du trafic est un domaine qui nous permet de déterminer la nature du trafic dans un réseau. Elle consiste à examiner des paquets IP (Internet Protocol) pour en extraire certaines caractéristiques spécifiques afin de répondre à certaines questions liées à son origine, le contenu transporté ou les intentions des utilisateurs. Généralement, elle traite des flux de paquets définis comme des séquences de paquets identifiés de manière unique par la même adresse IP source, le port source, l'adresse IP de destination, le port de destination et le protocole utilisé au niveau de la couche de transport et bien d'autres [9]. Cependant, les paquets peuvent être regroupés de plusieurs manières en fonction des classes que l'on voudrait obtenir pour la classification.

Formalisons maintenant le problème de classification du trafic. Soit p un trafic à analyser. Chaque trafic est décrit par un ensemble de n caractéristiques issues de son analyse. Ainsi, il peut être interprété par une variable X à n dimensions qui correspondent à un ensemble précis de caractéristiques : $p \rightarrow X = (x_1, x_2, x_3, ..., x_n)$. Pour le problème de la classification de flux, où p est représenté par un flux, nous essayons d'affecter à chacun de ces flux l'une des classes

données parmi un ensemble de c classes définies par une variable $Y = (y_1, y_2, y_3, ..., y_c, y_{c+1})$. $Y = y_{c+1}$ signifie que le flux analysé n'est reconnu comme aucune des classes données, c'est-à-dire qu'il est inconnu. Dans ce travail, nous nous intéressons aux applications internet telles que :

- **Web** : permet de consulter via un navigateur des pages regroupées sur des sites via le réseau Internet ; il utilise également le protocole websocket visant à créer des canaux de communication full-duplex par-dessus une connexion TCP pour les navigateurs web.
- **P2P** : un système dans lequel chaque ordinateur est à la fois client et serveur. Ce modèle de réseau informatique permet à chaque ordinateur ou entité de recevoir et de distribuer des fichiers ou données ;
- **Mail** : un service de transmissions de messages écrits et de documents envoyés électroniquement via le réseau Internet dans la boîte aux lettres électronique d'un destinataire choisi par l'émetteur ;
- **Transfert de fichier (FTP)** : un service permettant de transférer des fichiers qui peuvent être très lourds, d'un ordinateur à un serveur ou d'un serveur à un ordinateur ;
- **Base de données** : un système qui permet de stocker et de retrouver l'intégralité de données brutes ou d'informations en rapport avec un thème ou une activité ; celles-ci peuvent être de natures différentes et plus ou moins reliées entre elles ;
- **Multimédia** : ensemble des techniques et des applications qui présentent l'information sous forme combinée de sons, d'images, d'animations et de vidéos ;
- **Attaque** : est l'ensemble des techniques utilisées pour l'exploitation d'une faille d'un système informatique (système d'exploitation, logiciel ou bien même de l'utilisateur) à des fins non connues par l'exploitant du système et généralement préjudiciables ;
- **Service** : est un protocole d'interface informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.

1.2 Objectifs de classification du trafic

Bien que la recherche sur la classification du trafic soit assez spécifique, les motivations des chercheurs ne sont pas identiques [10]. Sur la figure 1.1, nous présentons des objectifs de classification typiques ou, en d'autres termes, trois domaines différents, où les méthodes proposées fonctionnent. Certaines approches classent le trafic en fonction de sa catégorie, c'est-à-dire si le trafic représente un transfert de fichier, un jeu en ligne, du multimédia, du Web, des attaques,

etc., ([2, 3, 4, 11, 12, 13, 14, 15, 16]). D'autres essaient d'identifier le protocole impliqué au niveau de l'application, tel que FTP, HTTP, SSH, Telnet ([5, 6, 17, 18, 19, 20, 21]). Le dernier groupe de méthodes classe le trafic en fonction de l'application exacte qui génère le trafic.

Par ailleurs, il y a souvent une confusion entre la classification des applications et la classification des protocoles au niveau de l'application (cf. figure 1.1) [7]. Par exemple, la classification du trafic Skype illustre le problème. Il s'appuie sur une infrastructure P2P tandis que son objectif principal est la prestation de services de voix sur IP (VoIP). De plus, pour la transmission de données, il utilise son protocole propriétaire Skype, mais les protocoles HTTP ou HTTPS (HTTP sur SSL) peuvent également être utilisés. De même, en raison des politiques strictes appliquées par les pare-feu et des restrictions pour certaines applications de streaming et P2P, les utilisateurs utilisent généralement des tunnels pour le trafic restreint avec les protocoles SSH ou SSL. Par rapport aux deux exemples abordés, les résultats de la classification sont différents selon les objectifs de la classification.

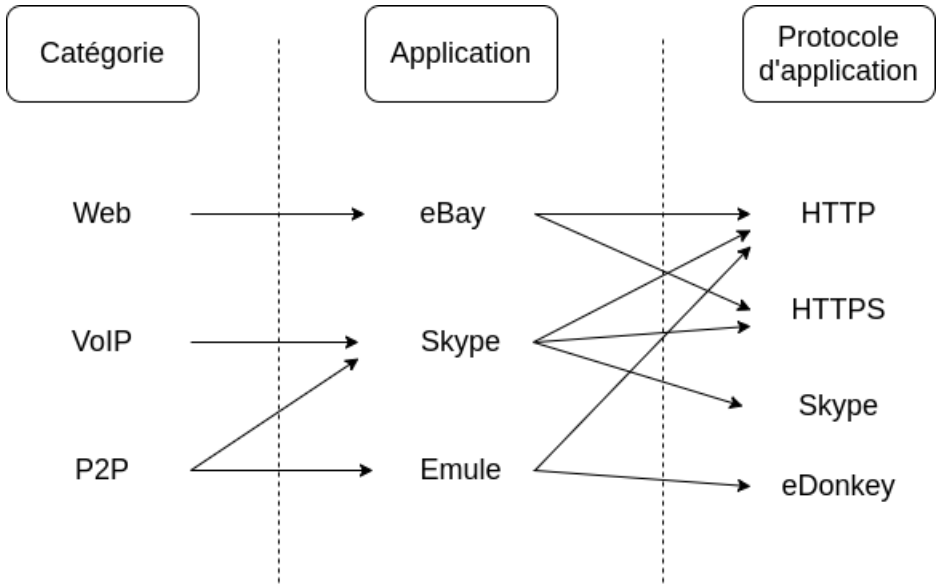

FIGURE 1.1 – Objectifs de la classification [9]

1.3 Les approches de classification du trafic internet

La sélection d'une approche appropriée pour la classification du trafic implique qu'on sache à l'avance l'application qu'on veut en faire. La variété des nouvelles applications d'Internet, y compris des services tels que le streaming, les jeux en ligne, le P2P ou la visioconférence, a intensifié les efforts de recherche afin de pouvoir identifier chacun d'eux à travers les caractéristiques de leurs flux. Ce qui a permis de pouvoir mettre en place quatre types d'approches de classifications en fonctions des informations sur lesquelles on se base pour réaliser la classification.

Dans cette section, nous présentons en détail chacune des quatre approches en plus des tra-

vaux qui ont été menés suivant chacune des approches. Il existe quatre approches étudiées dans le domaine de la classification du trafic sur deux décennies, à savoir deux approches basées sur le contenu consistant en l'inspection des ports et du payload ainsi que des approches basées sur les caractéristiques du flux et sur le comportement des hôtes [8]. Notons également par la suite que certaines études [22, 23] ont fourni des méthodes de classification pour les trafics cryptés, ce qui était difficile à réaliser dans le passé.

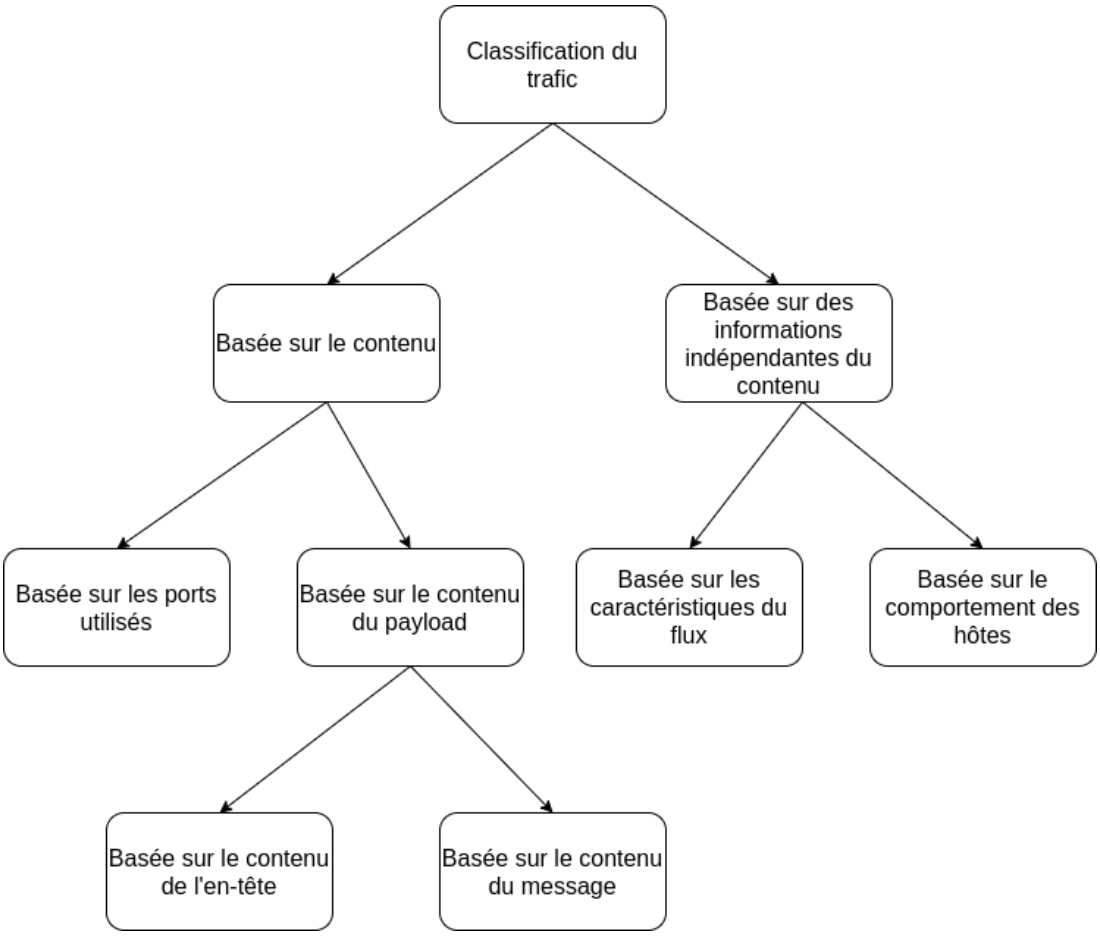

FIGURE 1.2 – Approches de la classification du trafic [9]

1.3.1 Approche basée sur les ports utilisés

Avant les années 2000, les FAI (Fournisseur d'Accès Internet) et les administrateurs de réseaux pouvaient classer avec précision le trafic réseau à l'aide des numéros de port TCP et UDP [10, 24]. Un protocole d'utilisation des ports en fonction de chaque application avait été établi, ce qui facilitait l'identification de n'importe quel trafic. Mais de nos jours, les nouvelles applications internet ont tendance à utiliser des ports de manière imprévisible pour échapper au contrôle du trafic. Un exemple concret est Skype qui met en place des mécanismes afin de contourner les pare-feu restrictifs. Il sélectionne au hasard les ports et peut basculer vers le port 80 ou 443 s'il ne parvient pas à établir une connexion sur des ports choisis dynamiquement. Par

conséquent, une simple inspection des numéros de port n'est plus un mécanisme de classification fiable [4, 25], en particulier lors de l'identification des applications.

Certaines études récentes revoient de manière critique la classification du trafic pour les méthodes basées sur les ports de la couche transport [26]. L'une des conclusions de leurs études est que les ports restent un discriminateur important, en particulier lorsqu'ils sont combinés avec d'autres caractéristiques telles que la taille des paquets, les drapeaux TCP et les informations sur le protocole. Les méthodes évaluées dans leurs études visent à classer les protocoles d'application plutôt que la classification détaillée de l'application de manière plus précise.

Des travaux de classification basée sur une approche consistant à l'analyse des ports se sont révélés très satisfaisants à l'exemple des travaux menés par Maier et al. [27] qui consistent à faire la distinction entre le P2P et le HTTP. Leur travail a également permis de conclure que le P2P n'est plus un trafic dominant en termes d'octets et que HTTP semble transporter la majeure partie du trafic.

1.3.2 Approche basée sur le payload

La deuxième approche basée sur le contenu consiste à inspecter le payload des paquets et pendant des années, elle a été considérée comme la méthode la plus précise. Dès que c'est possible d'identifier une signature unique grâce au payload, cette technique peut produire des résultats de classification fiables [4, 12]. De plus, les classifieurs basés sur le payload sont souvent utilisés en tant que vérificateurs pour d'autres méthodes [26, 28]. Néanmoins, en raison de problèmes de confidentialité et de chiffrement du payload, d'autres techniques ont reçu plus d'attention, car les méthodes basées sur le payload échouent toujours lorsque le trafic est chiffré [10, 7].

Risso et al. [7] ont travaillé sur des approches de classification basées sur le payload concernant les méthodes de vérification et de traitement du payload. La première définit quatre niveaux de vérification. Le premier niveau vise à localiser certaines signatures du message, le second de nature syntaxique, vérifie si le message est bien formé, par exemple, le payload HTTP doit contenir des en-têtes HTTP. Le troisième concerne la conformité des protocoles, le contrôle des échanges de messages serveur, tandis que le quatrième de nature sémantique vérifie le type d'objet envoyé par le protocole de la couche application. Une autre approche également traite des méthodes d'analyse du payload, à savoir les techniques basées sur les paquets et les messages, qui partent d'une simple vérification de certaines informations de base de l'en-tête du paquet à des traitements sophistiqués qui consistent à inspecter et à interpréter exactement ce que chaque application transmet.

De plus, Sen et al. [12] ont présenté une approche pour identifier le protocole eDonkey sur la base de l'analyse d'en-tête de la couche application. Plus précisément, les auteurs ont découvert que la signalisation et le téléchargement des paquets TCP ont un en-tête eDonkey particulier au-dessus de l'en-tête TCP. Dans le même article, les auteurs ont mis en évidence une signature

simple qui permet de révéler le trafic Kazaa basé sur l'analyse du protocole HTTP.

Bonfiglio et al. ont étudié le trafic Skype transporté par le protocole UDP [28]. Ils ont conclu que les messages UDP Skype chiffrés peuvent être identifiés en examinant la partie initiale du payload, qui est le début de message : *Start of Message (SoM)*, situé en haut de l'en-tête.

1.3.3 Approche basée sur le comportement des hôtes

Les approches basées sur le comportement des hôtes [2, 29] peuvent potentiellement résoudre certaines limites des méthodes basées sur le payload. L'approche est basée sur l'analyse du comportement des hôtes du réseau et peut être efficace même lorsque le payload est chiffré. Plus spécifiquement, les interactions entre les hôtes communicants sont représentées par des graphiques qui interprètent la relation «qui communique à qui». La classification dans ce cas, consiste à faire correspondre des graphiques de relation précédemment abordés avec des graphiques résultant du comportement d'un hôte après analyse [2].

BLINC, par exemple, propose une méthode intéressante basée sur l'observation et la reconnaissance des modèles de comportement de l'hôte puis sur la classification de ses flux selon les modèles [2]. Il analyse les modèles à trois niveaux. Au niveau interaction, il inspecte l'interaction avec d'autres hôtes. Au niveau fonctionnel, il vérifie si un hôte agit en tant que consommateur ou fournisseur du service (ou les deux). Au niveau application, il enregistre les ports de la couche de transport pour identifier l'origine de l'application.

Iliofotou et al. ont introduit l'idée des graphiques de dispersion du trafic (Traffic Dispersion Graphs TDGs) en tant qu'outil de surveillance et de classification [29]. Leurs travaux sur les TDGs représentent une extension de l'approche précédente. Plus précisément, ils proposent une autre façon de voir le trafic réseau : ils se concentrent sur les interactions des hôtes à l'échelle du réseau au lieu de modéliser le comportement d'un hôte unique. Les mêmes auteurs ont étendu leurs travaux antérieurs et développé une preuve du concept pour détecter le trafic P2P [30]. Leur outil de classification des applications, évalué sur des données réelles, peut identifier 90 % des flux P2P avec une précision de 95 %.

1.3.4 Approche basée sur les caractéristiques du flux

Les méthodes basées sur les informations relatives aux flux exploitent des caractéristiques telles que la taille moyenne des paquets, les temps d'arrivée des paquets, les temps de transmission des flux, etc. Les caractéristiques sont calculées sur plusieurs paquets regroupés en flux et utilisés dans le processus d'entraînement d'un modèle qui associe des ensembles de caractéristiques à des classes de trafic connues. Cette approche utilise principalement des techniques d'exploration de données et des algorithmes d'apprentissage automatique.

Moore et al. [3] ont proposé une approche statistique pour classer le trafic en différentes classes d'applications internet sur la base d'une combinaison de caractéristiques de flux telles que la longueur du flux, le temps entre les flux consécutifs, les temps entre les arrivées, etc.

Ce processus de classification utilisant un classifieur bayésien combiné à une méthode d'estimation de la densité du noyau a donné une exactitude générale allant jusqu'à 95 % (nombre total de classifications réussies par rapport au nombre total de classifications soumises au test). Cependant, les modèles obtenus sont généralement peu efficaces pour certains types de trafics tels que les attaques, le Pair à pair (P2P). Cependant, ils sont surtout très efficaces pour les trafics de type Web et Mail qui représentent à eux seuls plus de 94 % des données exploitées pour l'étude. Les différents résultats obtenus pendant la phase de test et d'évaluation sont consignés dans les tableaux 1.1 et 1.2

TABLEAU 1.1 – Récapitulatif des meilleures performances obtenues pendant la phase de test avec l'approche de Moore et al. [3]

Web	**Mail**	**Transfert de fichiers**	**Services**
99,27%	94,78%	82,25%	63,68%
Base de données	**P2P**	**Attaque**	**Multimédias**
86,91%	36,45%	13,46%	80,75%
Exactitude générale : 96,29%			

TABLEAU 1.2 – Récapitulatif des meilleures performances obtenues pendant la phase d'évaluation avec l'approche de Moore et al. [3]

Web	**Mail**	**Transfert de fichiers**	**Services**
98,06%	87,54%	90,03%	44,54%
Base de données	**P2P**	**Attaque**	**Multimédias**
68,63%	55,18%	N/A	N/A
Exactitude générale : 93%			

Auld et al. [31] utilisent une approche de classification basée sur les réseaux de neurones bayésiens pour classer le trafic sur 8 classes. La base de données de flux internet utilisée est identique à celle de Moore et al. [3] mais en supprimant les informations relatives aux hôtes et aux ports de la source et de la destination. Un modèle est d'abord construit sur l'ensemble de la base de données avec une exactitude générale de 99,3 %. Les résultats obtenus sont consignés dans le tableau 1.3. Il a été conclu par la suite que moins une classe possède de trafics sur l'ensemble des données et moins ses performances sont élevées. Ainsi la classe web qui représente 86,9 % du flux total a donné la plus grande exactitude qui est de 99,8 %. Afin de donner une chance égale à toutes les classes de flux, un autre modèle a été construit en limitant le nombre de

trafics par classe à 2441. Cela a permis d'augmenter considérablement les performances sur les classes de trafic qui paraissaient moins performantes. Mais le modèle obtenu est moins robuste, car il s'est conçu à partir d'un nombre de données limitées.

TABLEAU 1.3 – Récapitulatif des meilleures performances obtenues avec l'approche de Auld et al. [31]

Web	**Mail**	**Transfert de fichiers**	**Services**
99,8%	99,6%	82,25%	97,4%
Base de données	**P2P**	**Attaque**	**Multimédias**
97,6%	62,0%	68,6%	67,0%
Exactitude générale : 99,3%			

Zhong F. et al. [32] utilisent la machine à vecteur de support pour la classification du trafic internet. Plusieurs noyaux de SVM ont été testés et le plus intéressant a été le noyau radial. Plusieurs combinaisons de caractéristiques ont été effectuées à partir de 30 caractéristiques pour créer des modèles. Celui qui s'est montré le plus intéressant est un modèle issu d'une combinaison de 13 caractéristiques qui a permis d'obtenir une exactitude générale de 98 % (voir tableau 1.4). Afin de s'assurer de la stabilité du modèle, une phase d'évaluation a été effectuée avec de nouvelles données issues de flux capturés quelque temps plus tard en utilisant le modèle obtenu. Les résultats obtenus pour cette phase d'évaluation sont consignés dans le tableau 1.4.

TABLEAU 1.4 – Récapitulatif des performances obtenues avec l'étude de de Zhong F. et al. [32]

	Phase de test			Phase d'évaluation		
Classe de trafic	Précision	Rappel	F-mesure	Précision	Rappel	F-mesure
Web	99,24%	99,93%	99,58%	96,03%	98,72%	97,36%
Mail	96,68%	98,68%	97,67%	75,93%	97,10%	85,22%
Transfert de fichiers	93,23%	83,58%	87,69%	63,58%	71,91%	69,53%
Services	99,45%	91,5%	95,31%	93,05%	55,37%	69,43%
P2P	88,46%	92%	90,19%	52,82%	54,55%	53,67%
Base de données	97,12%	98,54%	97,83%	83,97%	51,52%	63,86%
Multimédias	96,09%	90,44%	93,18%	-	-	-
Attaque	12,5%	2,43%	4,08%	-	-	-

Il est important de notifier que les travaux [3], [31] et [32] utilisent tous les trois une même base de données et qu'une comparaison pourrait donc se faire entre eux.

Erman et al. [33] ont proposé une approche de classification semi-supervisée du trafic qui

combine des méthodes non supervisées et supervisées. Cette méthode a permis d'obtenir une exactitude de 94 %. Li et al. [34], utilisent la machine à vecteur de support dans la classification multi-classes du trafic réseau. Ainsi, à partir de neufs caractéristiques, ils ont pu construire un modèle capable de prédire six classes de trafics avec une exactitude générale de 99,4 %. Les performances obtenues sur le modèle sont résumées dans le tableau 1.5. Este et al. [35] ont proposé une approche pour la classification multi-classes du trafic, basée sur la machine à vecteur de support. Cette approche a permis d'obtenir une exactitude autour de 90 % pour chaque catégorie de classe (http, smtp, pop3, ftp, bittor, msn).

TABLEAU 1.5 – Récapitulatif des meilleures performances obtenues pendant la phase de test avec l'approche de Li et al. [34]

Classe de trafic	Taux de Faux Négatifs	Taux de Faux Positifs	Nombres de trafics
Web	0,31%	0,5%	25864
Mail	9,34%	2,44%	902
Transfert de fichiers	1,79%	0,94%	2237
Services	0,05%	0,03%	7348
Interactive	10,86%	4,23%	71
P2P	2%	2,82%	2021
Autres	0,26%	0,46%	8533

Cette sous-section a présenté l'ensemble des travaux basés sur l'approche d'exploitation des caractéristiques du flux pour construire des modèles de classification de trafics dont la plupart reposent sur l'apprentissage automatique. Nous exploitons également des méthodes d'apprentissage automatique sur des ensembles de caractéristiques du flux pour prédire la classe du trafic. Afin d'avoir plusieurs éléments de comparaison, nous utilisons la même base de données que les travaux [3], [31] et [32]. Ainsi, ces derniers restent nos principaux éléments de comparaison. Nous nous comparons également aux autres, mais en considérant d'autres aspects.

1.4 L'apprentisage automatique

1.4.1 Généralités

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle qui se concentre sur la conception de systèmes qui apprennent sur des ensembles de données qui leur sont fournis afin d'effectuer des prédictions. En gros, il s'agit de donner du sens aux données que possèdent les machines, de la même manière que les humains ou les animaux le font [36]. L'apprentissage automatique est un type d'intelligence artificielle par lequel un algorithme ou une méthode extrait des motifs à partir de données. Les algorithmes d'apprentissage automatique utilisent des méthodes de calcul pour «apprendre» des informations directement à partir de

données sans utiliser une équation prédéterminée comme modèle [37]. Les algorithmes améliorent de manière adaptative leurs performances lorsque le nombre d'échantillons disponibles pour l'apprentissage augmente.

Les algorithmes d'apprentissage automatique recherchent des modèles naturels dans les données qui génèrent des informations et aident à prendre de meilleures décisions. Ils sont utilisés quotidiennement pour prendre des décisions critiques en matière de diagnostic médical, d'échange d'actions, de prévision de la charge énergétique, etc. Les sites de médias font appel à l'apprentissage automatique pour passer en revue des millions d'options et proposer des recommandations de chansons ou de films. Les détaillants l'utilisent pour mieux comprendre le comportement d'achat de leurs clients [37]. Avec l'essor du Big Data, l'apprentissage automatique est devenu particulièrement important pour résoudre des problèmes dans des domaines tels que :

- Les finances, pour l'attribution de crédit et le trading algorithmique ;
- le traitement d'images et la vision par ordinateur, pour la reconnaissance faciale, la détection de mouvement et la détection d'objets ;
- la biologie numérique, pour la détection des tumeurs, la découverte de médicaments et le séquençage de l'ADN ;
- la production d'énergie, pour la prévision des prix et de la charge ;
- l'automobile, l'aérospatiale et la fabrication, pour la maintenance prédictive ;
- le traitement du langage naturel ;
- etc.

L'apprentissage automatique est subdivisé en deux principales classes : l'apprentissage supervisé, qui forme un modèle sur des données d'entrée et de sortie connues afin de prédire les sorties futures ; l'apprentissage non supervisé, qui trouve des modèles cachés ou des structures intrinsèques dans les données d'entrée. Nous allons détailler ces deux types d'apprentissage.

L'apprentissage supervisé a pour objectif de créer un modèle permettant de faire des prédictions à partir de données probantes en présence d'incertitude. Un algorithme d'apprentissage supervisé prend un ensemble connu de données d'entrée et de réponses connues aux données (sortie) et entraîne un modèle afin de générer des prédictions raisonnables sur la sortie pour de nouvelles données [38]. L'apprentissage supervisé utilise des techniques de classification et de régression.

- **Les techniques de classification** permettent de prédire des réponses discrètes. Par exemple, dire si un courrier électronique est authentique ou est un spam, ou si une tumeur est cancéreuse ou non. Les modèles de classification classent les données d'entrée en catégories.

Des applications typiques incluent l'imagerie médicale, la reconnaissance vocale et la notation de crédit. Comme algorithmes permettant de faire de la classification, nous pouvons citer : la machine à vecteur de support, Naïve Bayes, la recherche des plus proches voisins, l'arbre de décision, la forêt aléatoire, XGBoost, etc.

- **Les techniques de régression** prédisent des réponses continues, par exemple, des changements de température ou des fluctuations de la demande de puissance. Des applications typiques incluent la prévision de la charge d'électricité et le trading algorithmique. Comme algorithmes permettant de faire la régression, nous pouvons citer : la régression linéaire, Naïve Bayes, la recherche des plus proches voisins, l'arbre de décision, la forêt aléatoire, XGBoost, les réseaux de neurones, etc.

L'apprentissage non supervisé trouve des modèles cachés ou des structures intrinsèques dans les données. Il est utilisé pour tirer des inférences à partir d'ensemble de données non étiquetées [38]. Il existe deux méthodes d'apprentissage non supervisé que sont :

- **Le clustering ou regroupement** qui est la technique d'apprentissage non supervisée la plus courante. Il est utilisé pour l'analyse exploratoire des données afin de trouver des modèles ou des regroupements cachés dans les données. Les applications de clustering incluent l'analyse de la séquence des gènes, les études de marché et la reconnaissance d'objets. Quelques exemples d'algorithme permettant de faire du clustering sont : les K-moyennes, le mélange gaussien, le réseau de neurones, etc.

- **Les règles d'association** qui consistent à découvrir des relations intéressantes entre des variables dans de grandes bases de données. Par exemple, les personnes qui achètent une nouvelle maison ont aussi tendance à acheter de nouveaux meubles. Il découvre la probabilité de co-occurrence d'éléments dans une collection. Un algorithme qu'on retrouve très souvent pour les règles d'association est l'algorithme de l'apriori.

Choisir le bon algorithme peut sembler fastidieux : il existe des dizaines d'algorithmes d'apprentissage automatique supervisés et non supervisés, et chacun adopte une approche différente en matière d'apprentissage. À notre connaissance, il n'y a pas de meilleure méthode ni de solution unique. Trouver le bon algorithme repose sur un grand nombre d'essais. Notons que la sélection des algorithmes dépend également de la taille et du type de données avec lesquelles on travaille, des informations qu'on souhaite obtenir des données et de la manière dont ces informations seront utilisées.

Pour nos travaux, nous faisons de la classification. C'est-à-dire qu'on part sur des données de trafic avec des caractéristiques bien définies et des classes connues pour mettre en place un modèle capable de détecter plus tard la classe d'un trafic en ayant juste des informations sur ses caractéristiques. Cependant, en fonction des objectifs qu'on veut atteindre, les algorithmes à utiliser doivent respecter des critères spécifiques bien définis que sont :

- La capacité de l'algorithme à pouvoir traiter d'énormes quantités de données car la base de données à utiliser au cours de l'étude est constituée de plusieurs centaines de milliers de données.
- La capacité de l'algorithme à utiliser le moins de ressources possibles afin de faciliter la conception des modèles de classification.
- La rapidité de l'algorithme à effectué l'apprentissage. Car plus il apprend vite et plus on peut faire des combinaisons de paramètres afin de faire une recherche plus exhaustive et tendre vers des solutions optimales.
- La capacité de l'algorithme à être interprétable au regard des caractéristiques sur lesquelles il se base pour construire le modèle. Car si on connaît l'impact d'une caractéristique, on peut donc décider soit de la conserver ou de la retirer ce qui peut influencer sur les performances du modèle également sur la quantité de ressources exploitées par la machine lors de l'entraînement avec l'algorithme.
- La capacité de l'algorithme à s'adapter automatiquement aux manques de certaines données est un critère primordial à prendre en compte vu qu'elle se rapproche plus de ce qu'on pourrait observer dans des situations réelles. En travaillant sur plus de trois cent mille données il y a de fortes chances que des données manquent. Dans notre cas, seul l'algorithme de XGBoost possède cette capacité, ce qui justifie donc notre choix.

TABLEAU 1.6 – Quelques algorithmes en fonction des critères de choix

Critères de sélection de l'algorithme d'apprentissage automatique	**Exemple d'algorithmes d'apprentissage automatique respectant le critère**
La capacité de l'algorithme à pouvoir traiter d'énormes quantités de données. Dans notre cas, près de 400 000 observations	Réseaux de neurones, arbre de décision, forêt aléatoire,XGBoost, SVM, Naive Bayes, K-NN, etc.
La capacité de l'algorithme à utiliser le moins de ressources possibles	arbre de décision, XGBoost, SVM, Naive Bayes, K-NN, Gradient boosting, etc.
La rapidité de l'algorithme à effectuer l'apprentissage	arbre de décision, SVM, Naive Bayes, K-NN, etc.
La capacité de l'algorithme à être interprétable au regard des caractéristiques.	arbre de décision, forêt aléatoire, XGBoost, Gradient boosting, etc.
La capacité de l'algorithme à s'adapter aux manques de données automatiquement	XGBoost

Au vu des critères et du tableau comparatif 1.6, trois algorithmes d'apprentissage automatique répondent très bien aux critères à savoir : l'**arbre de décision**, **XGBoost** (Extreme Gra-

dient Boosting) et **la forêt aléatoire**. Nous utilisons donc ces algorithmes pendant l'étude pour la mise en place de nos modèles. Dans la suite, nous décrivons chacun des algorithmes retenus en expliquant également leur principe d'apprentissage.

1.4.2 L'arbre de décision

Un arbre de décision est une représentation d'un algorithme de classification de données suivant différents critères qu'on appellera décisions (ou nœuds) [39]. La figure 1.3 présente un exemple d'arbre de décision.

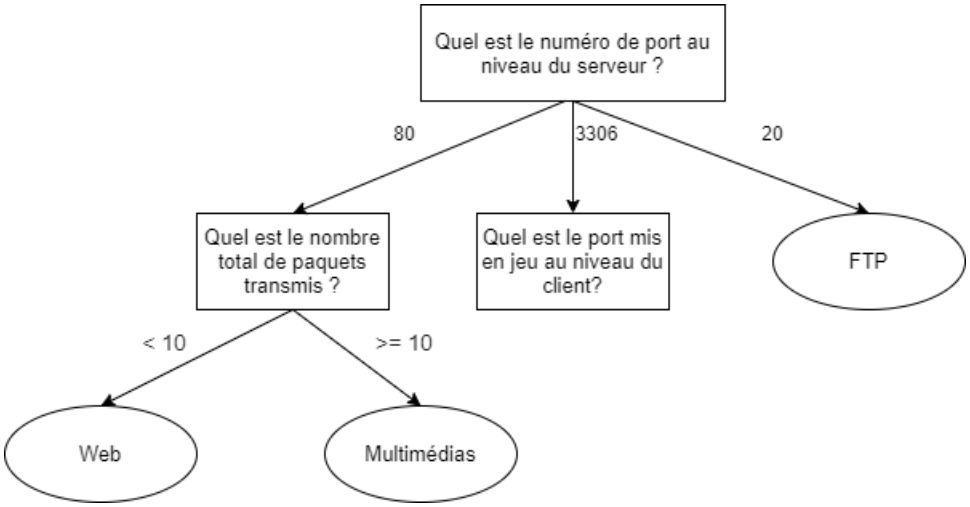

FIGURE 1.3 – Exemple d'un arbre de décision pour la classification d'un trafic

Cet arbre de décision permet en fonction de quelques questions de déterminer la classe d'un trafic en fonction de ses caractéristiques. Mais pour arriver à de telles règles de décisions, il faut synthétiser la connaissance et l'historique de l'ensemble d'un certain nombre de trafics internet donnés ayant été capturés. Il s'agit donc de trouver dans une grande quantité de données, les questions qui sont judicieuses d'être posées afin de prédire la classe d'un trafic.

Nous allons détailler la construction d'un tel arbre de décision. Posons d'abord quelques mots de vocabulaire.

- Chaque trafic de la donnée historique (ensemble d'entraînement) dispose d'une classe bien définie.
- Chaque trafic dispose de caractéristiques, de propriétés bien définies elles aussi. Par exemple : quel est le port mis en jeu au niveau du serveur ? Quel est le port mis en jeu au niveau du client ? Quel est le nombre total de paquets échangés durant le trafic ? etc.
- Chaque propriété ou caractéristique a un ensemble de valeurs possibles. Par exemple, pour le port mis en jeu au niveau du serveur, on peut avoir : 80, 8080, 20, 21, etc.

Dans notre exemple de figure 1.4 représentant l'arbre de décision, on a les éléments suivants :

- Un nœud de décision (représenté par un rectangle). À ce niveau, on pose une question afin de faire une répartition des trafics la plus homogène possible.
- Un bloc de fin qui est représenté par un ovale signifie qu'on a trouvé la classe du trafic.

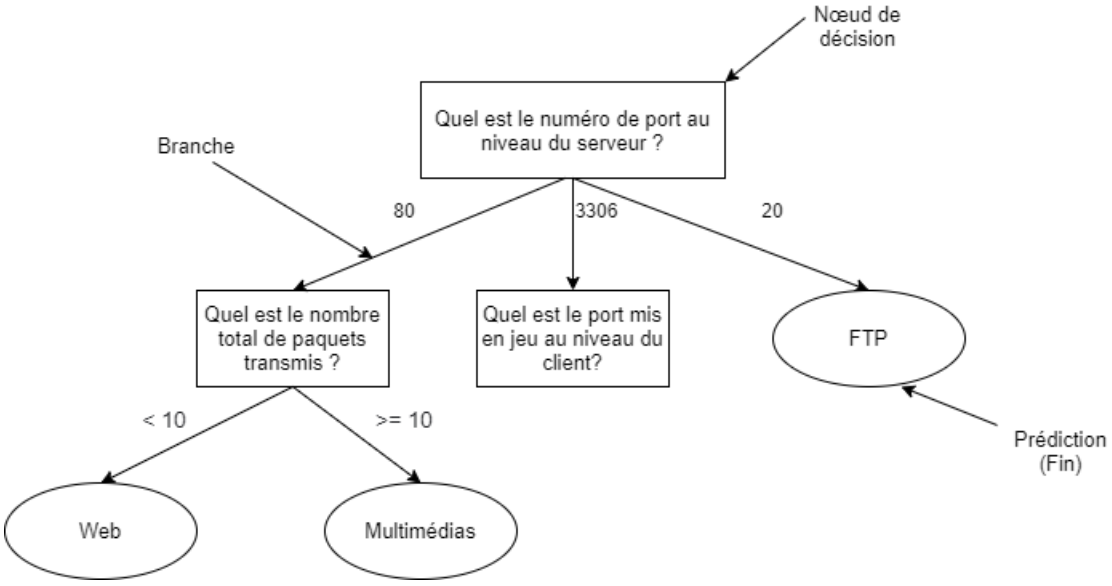

FIGURE 1.4 – Vocabulaire de l'arbre de décision

Dans les faits, on aura en entrée de notre algorithme une série de données qui représente de nombreux objets (ici des trafics internet) ayant chacun un ensemble de propriétés. Ces objets de l'ensemble d'entraînement sont déjà classés (ex : Mail, Web, Multimédias).

L'algorithme va choisir une première question à poser à l'objet. Pour cela il doit choisir la caractéristique (ou propriété) qui permet de découper nos trafics en deux ensembles les plus homogènes possibles, c'est à dire deux ensembles regroupant des trafics qui sont en grande partie d'une même classe. Par exemple, l'algorithme va tester la caractéristique du port mis en jeu au niveau du serveur et va ensuite répartir les trafics dans deux ensembles : les trafics ayant pour port au niveau du serveur 20 et les autres. Si dans un des ensembles on ne trouve que des trafics ayant la même classe alors l'algorithme s'arrêtera pour cette branche. Par exemple, si on constate que tous les trafics de port 20 au niveau du serveur sont de classe FTP, on peut s'arrêter là et dire que tous les trafics mettant en jeu le port 20 au niveau du serveur sont de classes FTP. Si on n'a pas obtenu un ensemble composé de flux qui ont la même classe après ce premier découpage, on va ensuite refaire le même travail pour les caractéristiques suivantes. Nous allons sélectionner la caractéristique permettant un classement le plus homogène possible. Ce processus est mené jusqu'à ce qu'on obtienne des classes homogènes. Cet algorithme est très intéressant pour extraire de l'information d'une source de données très grande. Il permet d'isoler

les propriétés ou caractéristiques qui apportent le plus d'information pour déterminer la classe de chaque objet.

1.4.3 La forêt aléatoire

Les forêts aléatoires sont composées (comme le terme "forêt" l'indique) d'un ensemble d'arbres décisionnels dans lequel a été introduit de l'aléatoire. Ces arbres se distinguent les uns des autres par le sous-échantillon de données sur lequel ils sont entraînés. Ces sous-échantillons sont tirés au hasard (d'où le terme "aléatoire") dans un jeu de données. Ils sont donc un ensemble d'arbres de décisions entraînés individuellement, légèrement différents les uns des autres. Pour prédire une nouvelle valeur, on effectue la classification pour chaque arbre de cette forêt [40]. La forêt choisit la valeur ayant le plus de votes parmi tous ses arbres.

L'idée derrière les forêts aléatoires est que chaque arbre de décision apprend sur une partie des données, tout en possédant une bonne capacité de prédiction. Toute la subtilité réside, dans la sélection de l'échantillon à utiliser pour créer chaque arbre, car on veut qu'ils soient tous différents [40]. Pour cela, on va découper le jeu de données d'entraînement, c'est à dire que si on a un jeu de données de N entrées, alors on crée un échantillon aléatoire de N' entrées. Ce sera le jeu données qui nous servira pour cet arbre en question. La forêt aléatoire se construit suivant quatre étapes :

- Prendre un nombre X d'observations du jeu de données de départ (avec remise).
- Prendre un certain nombre K de caractéristiques des M caractéristiques disponibles , par exemple : seulement les caractéristiques concernant les ports mis en jeu au niveau du serveur et du client pour le trafic.
- Entraîner un arbre de décision sur ce jeu de données.
- Répéter les étapes n fois de sorte à obtenir n arbres.

Pour une nouvelle observation dont on cherche la classe on descend les n arbres. Chaque arbre propose une classe différente. La classe retenue est celle qui est la plus représentée parmi tous les arbres de la forêt. La performance de la forêt aléatoire dépend alors de deux paramètres :

- **La corrélation** entre n'importe quelle paire de deux arbres de la forêt. Si la corrélation augmente, l'erreur augmente (puisque la variance est plus forte).
- **La performance** de chaque arbre pris individuellement. Plus les performances de l'arbre individuel sont importantes, plus la forêt aura de chance d'être globalement performante.

La forêt aléatoire est un algorithme possédant de très bonnes performances sur la plupart des problèmes. De plus, elles fonctionnent bien sur les grosses bases de données, car elles ne possèdent pas une si grande complexité pour l'entraînement.

1.4.4 Le gradient boosting

Avant de parler du gradient *boosting*, définissons d'abord le *boosting*. C'est une méthode qui permet de transformer les apprenants faibles en apprenants forts. Dans la construction des modèles, le Boosting travaille de manière séquentielle. Partons de notre exemple de classification du trafic pour montrer comment le Boosting fonctionne. Il commence par construire un premier modèle qu'il va évaluer. À partir de cette évaluation, chaque trafic va être pondéré en fonction de la performance de la prédiction. L'objectif est de donner un poids plus important aux trafics pour lesquels la valeur a été mal prédite pour la construction du modèle suivant. Le fait de corriger les poids au fur et à mesure permet de mieux prédire les valeurs difficiles [41]. Le *gradient boosting* est alors un algorithme qui va utiliser le **gradient de la fonction de perte** pour le calcul des poids des trafics lors de la construction de chaque nouveau modèle [41]. Techniquement parlant, on peut dire qu'une fonction de perte est une erreur, c'est-à-dire la différence entre la valeur prédite et la valeur réelle. Moins il y a d'erreur, meilleur est le modèle d'apprentissage. Le gradient boosting utilise généralement l'arbre de décision et on peut personnaliser l'algorithme en utilisant différents paramètres et différentes fonctions. C'est un algorithme capable de fonctionner sur de grandes quantités de données et qui nous permet d'évaluer l'impact des caractéristiques exploitées sur notre modèle.

1.4.5 XGBoost

XGBoost est le sigle retenu pour Extreme Gradient Boosting. Il s'agit d'une autre désignation de l'approche Gradient Boosting Machine qui a été introduite par Friedman en 1999 dans «Greedy Function Approximation : A Gradient Boosting Machine »(Jerome H. Friedman [42]). Son implémentation a été initialement effectuée par Tianqi Chen avant la contribution de nombreux développeurs. XGBoost est une implémentation spécifique de la méthode de *Gradient Boosting* qui fournit des prédictions plus précises en utilisant les forces de la dérivée du second ordre de la fonction de perte, la régularisation et le calcul parallèle [43]. Il utilise une régularisation avancée, qui améliore les capacités de généralisation du modèle. Il offre des performances élevées par rapport au Gradient Boosting. Il est très rapide et peut être parallélisé ou distribué entre plusieurs clusters. XGBoost calcule les gradients de second ordre, c'est-à-dire les dérivées partielles de second ordre de la fonction de perte, ce qui fournit plus d'informations sur la direction des gradients et sur la manière d'atteindre le minimum de notre fonction de perte le plus rapidement possible. XGBoost est également capable de gérer les valeurs manquantes en interne. Les avantages de XGBoost ne sont pas uniquement liés à ses performances, mais aussi aux divers paramètres que celui-ci propose. En effet, XGBoost propose un panel d'hyperparamètres très important ; il est ainsi possible grâce à cette diversité de paramètres, d'avoir un contrôle total sur l'implémentation de l'algorithme.

1.4.6 Évaluation d'un modèle de classification

Pour évaluer un modèle de classification, prenons l'exemple d'un classifieur binaire, c'est-à-dire, qui prédit deux classes notées classe 0 et classe 1. Pour mesurer les performances de ce classifieur, il est d'usage de distinguer 4 types d'éléments classés pour la classe voulue :

- Vrai positif VP : Elément de la classe 1 à prédire et qui est correctement prédit ;
- Vrai négatif VN : Elément de la classe 0 à prédire et qui est correctement prédit ;
- Faux positif FP : Elément de la classe 0 à prédire et qui est mal prédit ;
- Faux négatif FN : Elément de la classe 1 à prédire et qui est mal prédit ;

Ces informations peuvent être rassemblées et visualisées sous forme de tableau dans une matrice de confusion. Dans le cas d'un classifieur binaire, on obtient :

TABLEAU 1.7 – Matrice de confusion à deux classes

		Classe Prédite	
		Classe 0	Classe 1
Classe Réelle	Classe 0	VN	FN
	Classe 1	FP	VP

En particulier, si la matrice de confusion est diagonale (c'est-à-dire si FP = FN = 0) alors le classifieur est parfait. Notons que la matrice de confusion est aussi généralisable lorsqu'il y a k > 2 classes à prédire (cas de notre étude). Supposons que nous avons un modèle qui permet de prédire trois classes de trafic telles que : Attaque, Mail et P2P. On obtient

TABLEAU 1.8 – Matrice de confusion à trois classes

		Classe Prédite		
		Attaque	Mail	Web
Classe Réelle	Attaque	P_{AA}	P_{MA}	P_{WA}
	Mail	P_{AM}	P_{MM}	P_{WM}
	Web	P_{AW}	P_{MW}	P_{WW}

Définissons ensuite les différentes variables présentes sur la figure :

- P_{AA} : nombre de prédictions où les trafics d'un attaque ont été correctement prédits (comme un trafic attaque). C'est également le vrai positif pour la classe des attaques ;
- P_{MA} : nombre de prédictions où les trafics d'une attaque ont été mal prédits (comme étant, un trafic mail) ;

- P_{WA} : nombre de prédictions où les trafics d'une attaque ont été mal prédits (comme étant, un trafic web) ;
- P_{AM} : nombre de prédictions où les trafics d'un mail ont été mal prédits (comme étant, un trafic attaque) ;
- P_{MM} : nombre de prédictions où les trafics d'un mail ont été correctement prédits (comme un trafic mail). C'est également le vrai positif pour la classe des mails ;
- P_{WM} : nombre de prédictions où les trafics d'un mail ont été mal prédits (comme étant un trafic web) ;
- P_{AW} : nombre de prédictions où les trafics web ont été mal prédits (comme étant un trafic attaque) ;
- P_{MW} : nombre de prédictions où les trafics web ont été mal prédits (comme étant un trafic mail) ;
- P_{WW} : nombre de prédictions où les trafics web ont été correctement prédits (comme un trafic web). C'est également le vrai positif pour la classe web.

Comme pour la classification binaire, les vrais positifs pour une classe représentent le nombre de prédictions où les données étiquetées comme appartenant à une classe particulière ont été correctement classées comme ladite classe. Dans notre exemple, P_{AA}, P_{MM}, P_{WW} sont les vrais positifs respectivement pour les classes de trafic attaque, mail et web.

Les vrais négatifs pour une classe particulière sont calculés en prenant la somme des valeurs de chaque ligne et colonne, sauf la ligne et la colonne de la classe pour laquelle nous essayons de trouver les vrais négatifs. Par exemple, pour la classe attaque, les vrais négatifs sont obtenus à partir du calcul suivant : $VN = P_{MM} + P_{WM} + P_{MW} + P_{WW}$.

Pour déterminer les faux positifs pour une classe particulière, on prend la somme de toutes les valeurs dans la colonne correspondante à cette classe, à l'exception de la valeur des vrais positifs. Par exemple, pour la classe attaque, les faux positifs sont obtenus à partir du calcul suivant : $FP = P_{AM} + P_{AW}$.

Enfin, pour déterminer les faux négatifs pour une classe, on prend la somme de toutes les valeurs de la ligne correspondant à cette classe, à l'exception de la valeur des vrais positifs. Par exemple, pour la classe attaque, les faux négatifs sont obtenus à partir du calcul suivant : $FN = P_{MA} + P_{WA}$.

L'ensemble de ces calculs peuvent donc s'étendre sur des modèles de classification avec beaucoup plus de classes. Il est possible de calculer plusieurs indicateurs résumant la matrice de confusion. Par exemple, si nous souhaitons rendre compte de la qualité de la prédiction sur une classe donnée, on définit :

- **la précision** : le nombre de vrais positifs sur le nombre d'exemples prédits comme vrais par le modèle. Comme son nom l'indique, elle permet de mesurer la précision du modèle, soit la confiance que l'on peut attribuer au modèle. Plus la précision est forte, plus une prédiction 'vrai' du modèle peut être considérée comme exacte.

$$precision = \frac{VP}{VP + FP}[44] \tag{1.1}$$

- **le rappel** : est le nombre de vrais positifs sur le nombre d'exemples effectivement positifs dans l'ensemble de données. Le rappel permet de mettre en évidence l'exhaustivité du modèle.

$$rappel = \frac{VP}{VP + FN}[44] \tag{1.2}$$

- **la F-mesure** : est une combinaison de la précision et du rappel.

$$F - mesure = 2 \times \frac{precision \times rappel}{precision + rappel}[45] \tag{1.3}$$

- **la F_β-mesure** : un facteur permettant d'indiquer à quel point le rappel est plus important que la précision. Par exemple, si nous considérons que le rappel est deux fois plus important que la précision, nous pouvons mettre β à 2. La F-mesure standard équivaut à mettre β à 1 [45].

$$F_\beta - mesure = (1 + \beta^2) \times \frac{precision \times rappel}{(\beta^2 \times precision) + rappel}[45] \tag{1.4}$$

1.5 Critiques de l'existant et contribution de notre travail

1.5.1 Critiques de l'existant

Les résultats obtenus à travers les travaux existants présentés dans ce chapitre, ne sont pas parfaits. Il convient de souligner quelques limites qui situent et justifient notre travail.

La caractérisation du phénomène à comprendre ou de l'objet à apprendre dans un système d'apprentissage est une étape critique pour avoir un classifieur performant. Notons que certains travaux existants, ne présentent pas une étude formelle permettant de s'assurer que les caractéristiques utilisées sont informatives et discriminantes. D'un autre côté, certains travaux ne se basent que sur un nombre limité de caractéristiques comme les travaux de Li et al. [34].

En conséquence,

- il est parfois difficile d'identifier les caractéristiques et les propriétés du phénomène observé ou de l'objet à apprendre qui influencent effectivement les résultats obtenus ;

- il est parfois difficile de savoir quelles caractéristiques peuvent être combinées pour arriver à un modèle performant qui apprend bien.

De plus, on remarque que certains travaux malgré le fait d'obtenir de bonnes performances en général, n'arrivent pas à obtenir de bonnes performances sur certaines classes spécifiques comme les travaux de Moore et al. [3], Zhong F. et al. [32]. Généralement, les classes de trafics les plus concernées sont : les trafics de type transfert de fichiers (FTP) et attaque. Certains travaux ne partent que sur peu de classes de trafics comme les travaux de Li et al. [34] qui mettent en place un modèle basé uniquement sur six classes de trafics.

Il est important de souligner également la marge de progression potentielle avec les performances obtenues. L'amélioration des performances reste un défi constant lorsqu'on met en place des modèles de classification avec les algorithmes d'apprentissage automatique.

1.5.2 Contribution

Notre travail s'inscrit dans la logique de pallier ces limites de l'existant ainsi que certaines perspectives envisagées de certains travaux. Ainsi notre contribution par rapport au sujet de la classification du trafic internet se fait à travers les points suivants :

- Nous faisons une étude des caractéristiques pour ne retenir que celles qui sont pertinentes et informatives, tout en s'assurant de choisir les paramètres adéquats de l'algorithme d'apprentissage pour la conception du modèle de classification ;

- Nous veillons également au fait de pouvoir être capables de détecter avec efficacité n'importe quelle classe de trafic, c'est-à-dire être performant sur tous les types de trafics que nous considérons ;

- Nous proposons aussi des modèles capables de s'adapter aux manques de données. C'est-à-dire exploiter les caractéristiques à disposition pour un flux donné afin de prédire sa classe sans pour autant avoir la totalité des caractéristiques. Ceci nous permet de faire une ouverture sur la classification temps réel qui est le fait de pouvoir prédire la classe d'un trafic avant que celui-ci ne s'achève et donc ne nous fournit pas la totalité des informations requises. Il faudrait donc pouvoir exploiter le peu de caractéristiques reçues pendant que le trafic s'effectue et ainsi prédire sa classe ;

- Nous veillons à obtenir de meilleures performances par rapport aux travaux menés jusque-là ;

- Pour finir, nous dégageons les caractéristiques les plus discriminantes qui sont communes aux algorithmes considérés. Ces caractéristiques pourraient être considérées comme les plus importantes dans la classification du trafic internet.

Conclusion

De ce chapitre, nous retenons que la classification du trafic est un sujet qui mérite une attention continue, car les applications internet deviennent de plus en plus sophistiquées et les méthodes précédentes ne pourront donc pas s'appliquer facilement. Nous retenons également que l'étude sur la classification du trafic se fait généralement suivant quatre approches : celle basée sur les ports utilisés, celle basée sur le contenu du payload, celle basée sur le comportement des hôtes et enfin celle basée sur les caractéristiques du flux. Celle sur laquelle nous nous appuyons dans notre travail est l'approche basée sur les caractéristiques du flux. En partant d'un ensemble de caractéristiques du flux, nous appliquons des algorithmes d'apprentissage automatique sur ces caractéristiques afin d'obtenir des modèles de classification du trafic.

Chapitre 2

Matériel et méthodes

Introduction

L'objectif étant d'avoir des modèles performants de classification du flux internet, nous avons besoin d'un certain nombre d'outils, accompagné d'une méthodologie d'étude adéquate pour atteindre cet objectif. C'est ainsi que dans un premier temps, nous abordons ces différents outils à utiliser, ensuite nous mettons en place une méthodologie permettant de combiner ces différents outils afin d'atteindre nos résultats.

2.1 Matériel

2.1.1 Environnement de développement

Kit système

Pour nos travaux d'étude et d'application, nous avons utilisé un ordinateur portatif dont les caractéristiques sont les suivantes :

- **Modèle** : OMEN HP Laptop
- **Système d'exploitation** : Ubuntu 18.04 LTS
- **Architecture** : 64 bits
- **Processeur** : Intel(R) Core(TM) i7-7700HQ CPU @ 2,80GHz
- **Mémoire RAM** : 8 Go

Kit d'apprentissage

Langage de programmation : alors que l'apprentissage automatique gagne de plus en plus de terrain, presque tous les langages populaires ajoutent des modules et des bibliothèques pour

faciliter les tâches d'apprentissage machine. Les quatre langages que nous considérons être les meilleurs sont : python, Java, Matlab et R. Cependant nous allons nous baser sur les caractéristiques telles que :

- la communauté, afin de pouvoir trouver rapidement des solutions en cas de dysfonctionnements dans les codes développés ;
- la disponibilité des bibliothèques d'apprentissage automatique, afin d'avoir un choix très large et de pouvoir choisir la meilleure ;
- la rapidité d'exécution, car les algorithmes d'apprentissage machine mettent beaucoup de temps à s'exécuter pendant la phase d'entraînement ;
- le coût d'utilisation, car nous ne disposons pas de beaucoup de moyens financiers.

Le Tableau 2.1 nous montre un comparatif de ces quatre langages de programmation.

TABLEAU 2.1 – Comparatif des langages de programmation Python, Java, Matlab et R [46][47]

	Python	**Java**	**Matlab**	**R**
Coût d'utilisation	Gratuit	Gratuit	Payant	Gratuit
Communauté	Très large	Très large	Moyenne	Petite
Rapidité d'exécution	Rapide	Très rapide	Rapide	Rapide
Outils d'apprentissage	Très nombreux	Nombreux	Nombreux	Nombreux

Pour nos travaux, *Python* reste un bon compromis par rapport à nos critères. C'est un langage de programmation orienté objet et qui a un caractère multi-plateforme. C'est un langage puissant, à la fois facile à apprendre et riche en possibilités. Il est en outre, très facile d'étendre les fonctionnalités existantes, ainsi, il existe des bibliothèques qui aident le développeur à travailler sur des projets particuliers. Plusieurs bibliothèques peuvent ainsi être installées. Il est bien évidemment très adapté pour des applications d'apprentissage automatique.

Anaconda : est une distribution libre et gratuite des langages de programmation Python et R ainsi que de nombreux autres outils. Parmi ces outils, on retrouve spyder (un environnement de développement intégré abrégé EDI en français pour écrire du code python), matplotlib (pour visualiser des données à travers les courbes), etc. Ces outils sont très utiles pour des applications liées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique, etc.) [48].

Scikit-learn : est une bibliothèque libre en Python dédiée à l'apprentissage automatique. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support. Elle est conçue pour s'harmoniser avec d'autres bibliothèques libres en Python, notamment Numpy et Scipy [49].

XGBoost : est une bibliothèque libre en Python dédiée à la mise en place des algorithmes XGboost. Elle comprend un ensemble de fontions et de paramètres variés (la profondeur de l'arbre, le nombre d'estimateurs, etc.) permettant d'adapter l'algorithme à notre ensemble de données. Elle possède une documentation bien fournie qui permet une prise en main facile [50].

Numpy : est une bibliothèque libre en Python permet d'effectuer des calculs numériques. Elle introduit une gestion facilitée des tableaux de nombres et permet de manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux [51]. Plus précisément, cette bibliothèque open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier (excel, texte, etc.) ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes.

Matplotlib : est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous forme de graphiques. Elle peut être combinée avec les bibliothèques Python de calcul scientifique NumPy et SciPy [52].

Spyder est un environnement de développement scientifique gratuit et open source écrit en Python, pour Python, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données. Il offre un ensemble de fonctionnalités avancées d'édition, d'analyse, de débogage,etc. Il offre également un outil de développement complet avec les capacités d'exploration de données, d'exécution interactive, d'inspection approfondie et de visualisation de graphes [53] .

2.1.2 Les données

Tout au long de cette étude, nous avons utilisé les données collectées par un moniteur réseau de haute performance décrit dans [54] à l'Université de Londres. Le site expérimental pour la collecte des données est une grande installation de recherche accueillant environ 1 200 administrateurs, personnel technique et chercheurs. Un réseau Ethernet en duplex est utilisé sur ce site pour se connecter à l'Internet. L'ensemble des données de trafic est obtenu à partir des traces de trafic en duplex du centre de recherche sur une période de 24 heures.

Afin de construire les ensembles de flux, dix blocs d'environ 1680 secondes (28 minutes) ont été pris au hasard sur l'ensemble des 24 heures de flux à plusieurs moments de la journée.

Le début de chaque échantillon a été sélectionné au hasard (uniformément réparti sur toute la trace de la journée). La section A.1.1 en annexe illustre cette répartition et il en ressort qu'il existe un nombre de flux différent dans chaque bloc de données, en raison d'une densité de trafic variable pendant certaines périodes de la journée.

Chaque ensemble de données est représenté sous forme de fichier texte et se compose de plusieurs lignes. Chaque ligne représente un objet (flux). Sur Internet, un flux peut être défini comme un ou plusieurs paquets transitant entre deux adresses d'ordinateurs à l'aide d'un protocole particulier (par exemple, TCP, UDP, etc.) et le cas échéant, une paire de ports particulière (définie pour chaque extrémité du flux). Cet ensemble d'informations (IP source, IP destination, port source, port destination, protocole) est présent dans chaque paquet. Les informations source et destination dans l'ensemble sont donc inversées pour les paquets allant dans la direction opposée. De cette façon, un flux de paquets peut être soit en semi-duplex, soit en duplex intégral.

Afin d'avoir des captures de flux de qualité, la collecte a été faite en se concentrant uniquement sur les flux TCP car le TCP est un protocole avec état et ses flux ont un début et une fin bien définis. L'étude des données UDP est prévue dans les travaux futurs. Des informations supplémentaires ont été renseignées sur cette base de données dans l'article [55]. L'ensemble de données que nous allons utiliser est public, libre d'utilisation pour un cadre académique et disponible à travers un lien web public [56]. Le tableau 2.2 nous montre un récapitulatif de chaque classe de flux et leurs nombres sur l'ensemble des dix blocs de flux. L'ensemble de ces dix blocs est constitué au total de 377 526 flux et chaque flux est caractérisé par 248 caractéristiques. Nous devrions alors avoir 377 526 * 248 = 93 626 448 valeurs possibles, mais il y a 1 105 574 valeurs manquantes. Cette base de données est utilisée uniquement pour la phase d'entraînement et de test c'est-à-dire, lors de la conception du modèle de classification du flux. Afin de mieux évaluer nos modèles, un 11^{eme} bloc de données est prévu à cet effet et a été obtenu de la même manière que les 10 premiers blocs, mais un an après l'obtention de ces derniers. Ce 11^{eme} bloc, constitué de 19 626 flux, n'est utilisé uniquement que lors de la phase d'évaluation afin d'estimer les performances de nos modèles et de voir leurs efficacités sur des données réelles indépendantes des données d'entraînement. Le tableau 2.3 nous présente un récapitulatif de chaque classe de flux et leurs nombres sur l'ensemble des flux du 11^{eme} bloc.

TABLEAU 2.2 – Récapitulatif du nombre de flux par classe sur les 10 blocs prévus pour l'entraînement

Web	**Mail**	**FTP**	**Services**	**Base de données**
328091	28567	11539	2099	2648
Interactif	**P2P**	**Attaque**	**Multimédias**	**Jeux**
110	2094	1793	1152	8
Nombre de flux total : 377526				

TABLEAU 2.3 – Récapitulatif du nombre de flux par classe sur le 11^{eme} bloc prévu pour la phase d'évaluation

Web	**Mail**	**FTP**	**Services**	**Base de données**
15597	1799	1513	121	295
Interactif	**P2P**	**Attaque**	**Multimédias**	**Jeux**
4	297	0	0	0
Nombre de flux total : 19626				

2.1.3 Les caractéristiques du flux présentes dans les données

Cet ensemble d'informations est destiné à fournir une grande variété d'attributs pour caractériser les flux. Cela inclut des statistiques simples sur la longueur des paquets et les durées de transmission des paquets, ainsi que des informations dérivées du protocole de transport (TCP) : telles que le nombre de segments SYN (demande de synchronisation ou établissement de connexion) et ACK (signale que le paquet est un accusé de réception : *acknowledgement*). Ces informations sont fournies sur la base de tous les paquets (dans les deux sens) et dans chaque direction individuellement (serveur → client et client → serveur) [55].

De nombreuses statistiques de paquets sont obtenues en comptant directement les paquets également à travers les tailles d'en-tête des paquets. Un nombre important de caractéristiques (telles que les estimations du temps d'aller-retour, la taille des segments TCP et le nombre total de retransmissions) sont dérivées des en-têtes TCP.

Au total, nous avons 249 caractéristiques (avec la classe du trafic) qui se résument par : les durées de transmission du flux, les ports TCP mis en jeu, les durées de transmission des paquets (moyenne, variance, etc.), les tailles des segments (moyenne, variance, etc.), la bande passante efficace basée sur l'entropie [57], la transformée de Fourier du temps de transmission des paquets, la classe de chaque flux, etc. La section A.1.2 en annexe montre une description détaillée de chacune des 249 caractéristiques comme décrit dans l'article original [55] traitant de l'ensemble de données.

2.2 Méthodes d'étude

2.2.1 Procédure de travail générale

Tout au long de nos travaux, nous effectuons la même procédure d'étude en suivant un certain nombre d'étapes bien définies mais en faisant varier l'algorithme d'apprentissage qui représente le cœur de nos travaux. Nous utilisons trois algorithmes d'apprentissage que sont : *l'arbre de décision*, *XGBoost*, *la forêt aléatoire*. En effet, les algorithmes considérés à l'exception de XGBoost nécessitent une phase de pré-traitement des données qui consistera à combler les

manques de données avant de pouvoir exploiter les données en question. Cependant, l'algorithme de XGBoost a la capacité de s'adapter au manque de données, ce qui nous permet par la même occasion de répondre à une problématique principale de notre étude qui est de pouvoir construire des modèles capables de s'adapter au manque de données. Ainsi, pour l'algorithme de XGBoost, l'étude est effectuée sans combler les données manquantes.

Définissons donc la procédure de recherche à suivre pour chaque algorithme considéré à travers les points ci-dessous.

- Nettoyage, homogénéisation et adaptation de la base de données à l'algorithme d'apprentissage à utiliser afin de pouvoir rendre les données exploitables ;
- Lancement de la phase d'apprentissage par l'algorithme d'apprentissage considéré en combinant plusieurs paramètres de l'algorithme afin d'obtenir plusieurs modèles de performances variées ;
- À la fin de l'apprentissage, relever les métriques pouvant nous permettre d'évaluer les performances des modèles et l'importance des caractéristiques utilisées pour l'apprentissage ;
- Mener une discussion autour des résultats obtenus ;
- Réduire l'espace des caractéristiques en éliminant les caractéristiques les moins importantes, suivant un critère bien défini à partir des résultats sur l'importance des caractéristiques relevées précédemment *(Ex : En fixant un seuil de 1 %, nous éliminons donc toutes les caractéristiques qui ont une importance en dessous de 1 %)*. Pour choisir notre seuil, nous veillons généralement à ne pas aller en dessus de 1 %. Nous avons supposé que toutes les importances en dessous de 1 % peuvent être négligeables. De plus, avant de choisir ce seuil, nous faisons un travail préliminaire, qui consiste à afficher dans un repère *(0, x)*, c'est-à-dire sur une dimension, l'importance de toutes les caractéristiques exploitées pour concevoir le modèle. Généralement, on a des groupements qui se forment. Ensuite, on essaye d'identifier le groupement le plus à gauche, c'est-à-dire le plus proche de zéro. Une fois ce groupement identifié, on peut définir notre seuil et éliminer toutes les caractéristiques dont les importances sont inférieures ou égales à ce seuil ;
- Reprendre le processus en considérant notre nouvel espace de caractéristiques réduit jusqu'à ce qu'on remarque une baisse drastique de la performance des modèles obtenus. Ce qui fait qu'à la première itération, nous partons de 248 caractéristiques et nous réduisons ce nombre au fur et à mesure dans nos itérations jusqu'à ce qu'on remarque une très grande baisse dans les performances du modèle.

2.2.2 Procédure d'apprentissage par un algorithme

Il est impératif d'expliquer les étapes de l'apprentissage, car la phase d'apprentissage par les algorithmes représente l'essence même du travail de recherche dans notre étude. La figure 2.1 est une représentation schématique de tout le processus d'apprentissage en partant de l'ensemble de données à utiliser pour l'apprentissage jusqu'à l'obtention du modèle.

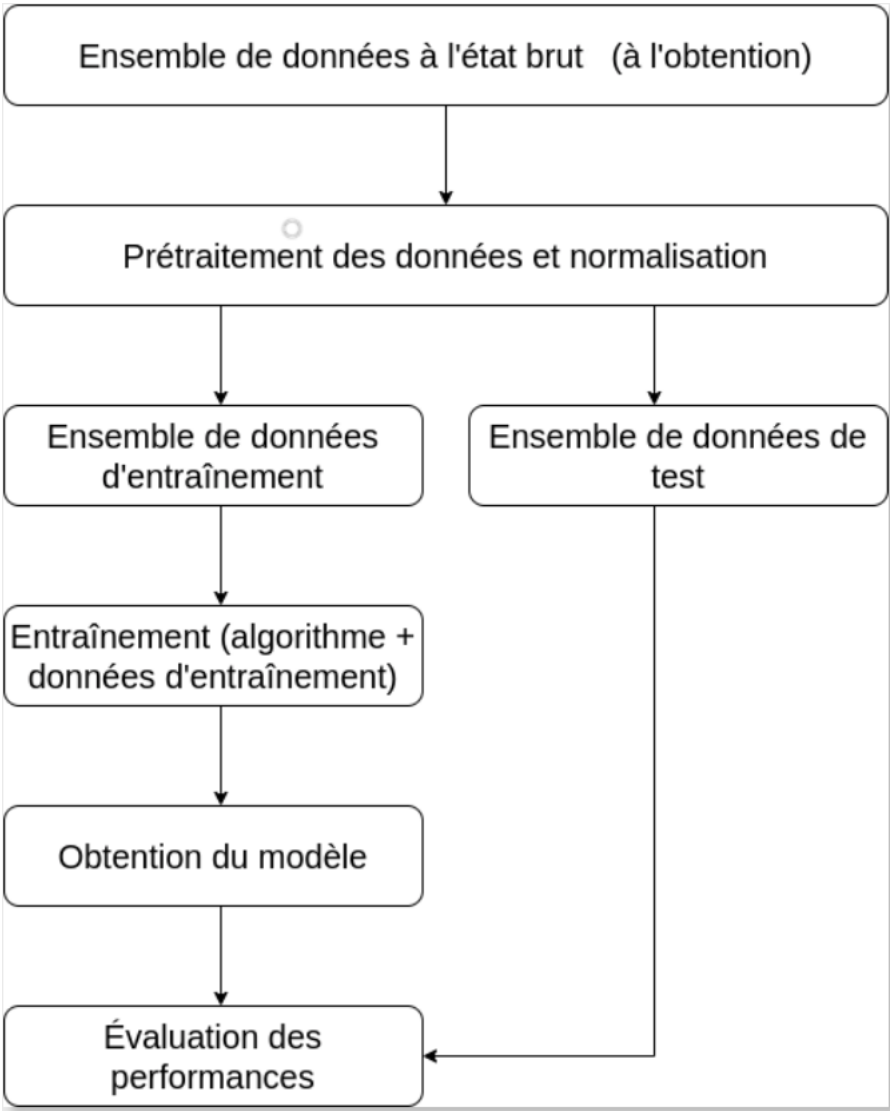

FIGURE 2.1 – Les différentes phases de l'apprentissage

Ensemble de données brutes : L'ensemble des données obtenues à partir du site [56] est un ensemble de 10 fichiers *excel* contenant des milliers de trafics comme décrit dans la section 2.1.2. Chaque ligne du fichier est représentée par un trafic et chaque colonne représente une des 248 caractéristiques qui décrivent le trafic suivi d'une dernière colonne qui représente la classe trafic. Il faut également noter le manque de plusieurs valeurs de caractéristiques à certains endroits pour lesquelles il faudra trouver une solution.

Pré-traitement des données : Comme souligné précédemment, les données obtenues directement à l'état brut sont des données avec beaucoup de défauts et de manques qui requièrent un certain nombre de traitements avant d'être exploitées par l'algorithme d'apprentissage (rappelons encore une fois ici que pour l'étude avec XGBoost nous conservons l'ensemble des données avec les manques). Une fois ces défauts éliminés, on doit procéder à une normalisation de toutes les données afin de les rendre exploitables par l'algorithme d'apprentissage. Les étapes mises en œuvre pour parer à ces anomalies et procéder à la normalisation sont les suivantes :

- Pour toutes les données quantitatives manquantes, remplacer chaque case manquante par la moyenne de la colonne. C'est-à-dire si on prend un trafic et qu'on suppose que la valeur pour la caractéristique "2" *(temps d'arrivée minimal pour tous les paquets)* n'est pas présente, on la remplace par la moyenne des valeurs pour la caractéristique "2" de tous les trafics qui ont une valeur bien déterminée pour cette caractéristique ;
- Pour toutes les données qualitatives manquantes, remplacer chaque case manquante par le mode de la colonne (la valeur dominatrice de cette colonne) ;
- Attribuer des valeurs quantitatives aux valeurs qualitatives. Car les algorithmes d'apprentissage automatique ne fonctionnent qu'avec des valeurs numériques ;
- Supprimer tous les trafics de type *Jeux* et *Interactif* au vu de leur faible quantité en données. Notre étude ne concernera donc pas ces deux classes de trafic.

Une fois que des données propres et homogènes sont obtenues, diviser les données en deux groupes. C'est-à-dire en un ensemble de données d'entraînement et en un ensemble de données de test.

Ensemble de données d'entraînement : L'ensemble de données d'entraînement est l'ensemble constitué de la plus grande partie de l'ensemble de données. Dans notre cas, elle représente 70 % (soit 264 268 flux) de l'ensemble de données et elle servira à l'entraînement de l'algorithme. C'est à partir de ces données que l'algorithme essaiera de mettre en place un modèle de prédiction du trafic internet.

Ensemble de données de test : L'ensemble de données de test est l'ensemble constitué de la plus petite partie de l'ensemble de données. Dans notre cas, elle représente 30 % (soit 113 258 flux) de l'ensemble de données et elle servira à tester le modèle. C'est à partir de ces données qu'on pourra évaluer le modèle obtenu après entraînement et déduire si oui ou non, il est performant.

Entraînement : C'est la phase pendant laquelle l'algorithme d'apprentissage automatique utilisé effectue l'entraînement sur les données d'entraînement. Mais afin d'obtenir de meilleurs

résultats, il faut choisir efficacement les hyperparamètres (c'est-à-dire les paramètres modifiables à savoir : la profondeur des arbres, le nombre d'estimateurs, l'algorithme d'optimisation, etc.). Tester plusieurs hyperparamètres pour trouver la parfaite combinaison est coûteux en temps et en puissance de calcul. Nous avons utilisé dans le cadre de notre travail le *Grid Search*. C'est une méthode simple qui permet de tester toutes les combinaisons d'hyperparamètres ajoutés. Elle consiste à mettre en place une grille de valeurs de ces hyperparamètres, et pour chaque combinaison, de former un modèle et de tester le modèle sur l'ensemble de données de test.

Obtention du modèle : À l'issue de l'entraînement et au vu de la méthode de *Grid Search* utilisée dans notre cas précis, on aura donc à la fin de l'entraînement plusieurs modèles avec des performances variées et on pourra retenir uniquement le modèle avec les meilleures performances.

Évaluation des performances : Une fois le modèle obtenu, la métrique principale qui va nous permettre d'évaluer la performance est **l'exactitude** (Accuracy). En effet, l'exactitude (A) de manière générale dans notre cas, est le nombre de trafics correctement classifiés ($N_{correctes}$) divisé par le nombre total de trafics soumis à la classification (N_{total}) (voir equation 2.1). Pour calculer cette métrique, nous utilisons les résultats obtenus par le modèle sur l'ensemble de données de test. Ainsi plus le résultat de cette division est proche de 1 et plus le modèle est performant. De manière spécifique, on peut également calculer cette métrique pour une seule classe spécifique de trafic. Dans ce cas, l'exactitude (A_i) est égale au nombre de trafics prédits comme étant de la dite classe et qui sont réellement de cette classe ($N_{classe-correctes}$) divisé par le nombre total de trafics de cette dite classe soumis à la classification ($N_{classe-total}$) (voir equation 2.2). Cependant, il peut arriver des cas pour lesquels l'exactitude ne soit pas un bon indicateur pour faire l'évaluation. On fait donc recours à la **somme des carrés des résidus** (SCR) sur les exactitudes pour chaque classe obtenues par le modèle. Ici, l'exactitude obtenue pour une classe donnée est A_i et comme on peut le déduire l'exactitude attendue pour cette classe si le modèle prédit correctement est 100. Moins le résultat de la somme des carrés des résidus est élevé et plus le modèle est performant. L'équation 2.3 nous permet de calculer cette métrique.

$$A = \frac{N_{correctes}}{N_{total}} * 100 \tag{2.1}$$

$$A_i = \frac{N_{classe-correctes}}{N_{classe-total}} * 100 \tag{2.2}$$

$$SCR = \sum_{i=1}^{n} (100 - A_i)^2 \tag{2.3}$$

Conclusion

Nous avons présenté dans ce chapitre les différents outils et matériels physiques comme logiciels ainsi que les technologies et la base de données qui vont nous permettre d'effectuer nos travaux tout en justifiant certains de nos choix. Nous avons également abordé la méthodologie d'étude permettant de combiner l'ensemble de ces ressources afin d'atteindre les objectifs fixés.

Chapitre 3

Résultats et discussions

Introduction

Une fois le matériel de travail à notre disposition et la méthode d'étude clairement définie, passons ensuite à la réalisation de l'étude proprement dite qui est de développer des modèles performants de classification du flux internet. Dans un premier temps, nous menons une étude suivant les algorithmes considérés à savoir *l'arbre de décision*, *XGBoost* et *la forêt aléatoire* sur notre base de données. Dans un second temps, nous faisons une étude comparative avec quelques travaux de l'état de l'art. Pour finir, nous menons une discussion afin de faire ressortir les aboutissements de notre étude.

3.1 Étude menée avec l'arbre de décision

L'arbre de décision est implémenté dans la bibliothèque *scikit-learn* de la version 3 du langage *Python*. La fonction permettant de mettre en place des modèles basés sur l'arbre de décision possède plusieurs paramètres qui influencent sur la performance. Cependant, nous nous basons sur trois paramètres que nous avons jugés importants pour l'étude.

- **criterion** : ou choix de mesure de qualité d'une répartition (les nœuds d'un arbre de décision sont divisés en utilisant un taux d'impureté). Les critères pris en charge sont le *gini* pour l'indice d'impureté de Gini et l'*entropy* pour le gain d'information.
- **max_depth** : ou la profondeur maximale de l'arbre. Si elle n'est pas précisée, les nœuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins du minimum d'échantillons requis pour diviser un nœud interne.
- **min_samples_split** : ou nombre minimum d'échantillons requis pour diviser un nœud interne.

Les détails sur chacun de ces paramètres ont été abordés dans l'annexe du document à la section A.2. Rappelons que pour la conception de notre modèle, 70 % de la base de données d'entraînement est utilisé pour l'entraînement et 30 % pour le test. Les résultats obtenus son disponible en annexe à la section A.3.1.

3.1.1 Étude de l'arbre de décision sur notre base de données en considérant 248 caractéristiques

Afin de mieux présenter les résultats et faire une meilleure analyse, nous faisons des études indépendantes suivant le paramètre "criterion" (choix de mesure de qualité d'une répartition) qui peut être soit "gini" ou "entropy". Ensuite nous faisons une analyse générale.

Étude suivant le critère "gini"

Les résultats des modèles obtenus en fixant le paramètre *criterion="gini"* sont reportés dans le tableau A.1 en annexe A.3.1.

On remarque à partir du tableau A.1 que l'exactitude générale pour l'ensemble des modèles obtenus tourne autour de 99,70 %. Cela se traduit par le fait que dans notre échantillon de test, il y a une très grande quantité de trafic web. De plus, vu que les modèles obtenus sont tous efficaces pour la détection de trafic web, l'impact de la performance des autres types de trafic ne ressort pas clairement sur l'exactitude générale. On note alors l'importance d'avoir calculé la somme des carrés des résidus pour chaque modèle. Ce qu'on remarque directement lorsqu'on considère le critère *gini* est que, quelle que soit la profondeur fixée pour l'arbre, plus on augmente le paramètre *min_samples_split*, et plus la SCR augmente aussi. Ceci se traduit alors par une dégradation des performances du modèle. On constate aussi que les performances sur les trafics de types base de données et web ont tendance à augmenter, mais les performances sur les autres types restent constantes ou se dégradent. Fixer le paramètre *min_samples_split* à 2 est donc un meilleur compromis.

Lorsqu'on fait une analyse sur la profondeur maximale des arbres, on constate qu'en fixant la profondeur à 100, on a une meilleure stabilité au niveau des modèles, car la SCR minimale obtenue est de 411,62 et la SCR maximale est de 493,53 ce qui fait donc un écart de 81,91 qui est largement en deçà des autres. Par contre cette profondeur de 100 ne nous a pas permis d'obtenir le meilleur modèle. Le meilleur modèle ayant la plus basse SCR (387,57) a été obtenu en laissant l'algorithme fixer la profondeur de l'arbre et en mettant *min_samples_split* à 2. Les paramètres ayant permis d'obtenir le meilleur modèle sont : *min_samples_split = 2* et *max_depth = None*.

Étude suivant le critère "entropy"

Les résultats des modèles obtenus en fixant le paramètre *criterion="entropy"* sont reportés dans le tableau A.2 en annexe A.3.1.

Le tableau A.2 nous montre très rapidement l'efficacité du paramètre fixé *criterion = "entropy"*, car on a obtenu des exactitudes générales supérieures à celles obtenues lorsqu'on fixe le critère sur *"gini"*. On peut donc retenir que pour le problème qu'on aborde, il serait plus efficace de se baser sur l'entropie minimale pour pouvoir créer les branches de nos arbres plutôt que de partir sur le gain d'information maximal (le paramètre "gini"). De plus, on remarque une amélioration au niveau des performances sur les trafics de type attaque et P2P. Quelle que soit la profondeur maximale de l'arbre, les bonnes performances sont obtenues pour le paramètre *min_samples_split* fixé à 10. Le meilleur modèle ayant la plus basse SCR (335,01) a été obtenu en laissant l'algorithme fixer la profondeur de l'arbre et en mettant *min_samples_split* à 10. Les paramètres ayant permis d'obtenir le meilleur modèle sont : *min_samples_split = 10* et *max_depth = None*.

Analyse suivant les deux critères *gini* et *entropy*

Les tests réalisés précédemment ont permis de conclure que le critère de l'entropie (*criterion="entropy"*) pour la formation des branches de l'arbre est le plus efficace pour notre étude. Nous nous basons uniquement sur le critère *"entropy"* pour la suite de nos tests en raison de son efficacité. Cependant, les performances de l'arbre reposent essentiellement sur la manière de paramétrer le minimum d'échantillons à prendre en considération pour créer les branches (*min_samples_split*) et la profondeur maximale de l'arbre (*max_depth*). Les paramètres ayant permis d'obtenir le meilleur modèle, soit une exactitude générale de 99,75 % et un SCR de 335,01 sont les suivants :

- **criterion** : *"entropy"*
- **max_depth** : *None* (défini par l'algorithme lui même)
- **min_samples_split** : *10*

Afin de s'assurer qu'on a un modèle bien conçu qui n'a pas sur-appris ou sous-appris et qui peut s'adapter en situation réelle, nous utilisons notre modèle pour prédire les trafics du bloc d'évaluation décrits au niveau de la section 2.1.2 du chapitre 2. C'est un ensemble d'échantillons complètement indépendant de l'ensemble d'échantillons utilisé pendant la conception du modèle. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.1.

TABLEAU 3.1 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 248 caractéristiques

Web	**Mail**	**TDF**	**Services**
98,72%	99,77%	83,21%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,98%	90,90%	N/A	N/A
Exactitude générale : 97,52%			

On note en général sur l'ensemble des classes, de très bonnes performances (>90 %). Par contre sur la classe transfert de fichier (TDF), on a une exactitude de 83,21 %. On peut dire que notre modèle s'adapte moins aux classes TDF, mais nous jugeons quand même que 83,21 % est un résultat satisfaisant. Alors le modèle retenu dans cette section s'adapte très bien à de nouveaux flux et nous donne des résultats très proches de ceux obtenus pendant la phase de test. On juge donc ce modèle performant.

Cependant, il convient de faire une analyse par rapport aux caractéristiques utilisées. Lorsqu'on analyse le taux d'importance donné par le modèle à chacune des 248 caractéristiques (dont la somme des taux d'importance des 248 caractéristiques est égale à 100 %) lors de l'apprentissage, on remarque que *129* caractéristiques ont une importance de 0 %. De plus, beaucoup d'autres caractéristiques ont des importances quasi-nulles (très proches de 0). Toutes ces caractéristiques peuvent être qualifiées de superflues, car en plus de rallonger le temps d'apprentissage, elles compliquent le modèle obtenu qui devient moins interprétable. Ceci peut alors influencer sur la performance du modèle. Pour cela, nous décidons de nous débarrasser de ces caractéristiques qui n'apportent que de l'information nulle ou quasi-nulle. Pour la suite, nous fixons un seuil de *0.1 %*, afin de supprimer toutes les caractéristiques ayant une importance en dessous de 0.1 %. Après cette opération, nous nous retrouvons à *12* caractéristiques dont la somme des taux d'importance est égale à *98,62 %* contre *236* caractéristiques ne réunissant que *1,38 %*. On voit tout de suite que notre modèle a été conçu à partir d'une quantité énorme de caractéristiques n'apportant presque aucune information qui auraient pu influencer sur ses performances. Ces caractéristiques peuvent également influencer sur le modèle en termes de simplicité, d'espace occupé en mémoire et en terme de rapidité d'exécution à classifier un trafic. Par la suite, nous reprenons notre étude en ne considérant que les 12 caractéristiques retenues.

3.1.2 Étude de l'arbre de décision sur notre base de données en considérant 12 caractéristiques

Comme nous l'avons souligné dans les tests précédents, nous partons uniquement sur les 12 caractéristiques qui ont permis d'obtenir une importance de 98,62 %. De plus, pour les nou-

veaux tests à venir, nous considérons uniquement le critère d'entropie (*criterion= "entropy"*), au vu des résultats satisfaisants qu'on a obtenus grâce à ce critère dans l'expérience précédente. Les performances des modèles obtenus sont résumées dans le tableau A.3 en annexe A.3.1.

En comparant les SCR pour les différentes profondeurs maximales, on remarque que le fait de limiter la profondeur à 5 et 10 n'ont pas permis à l'arbre de décision d'étendre ses branches plus en profondeur pour mieux apprendre. En quelques sortes, l'arbre de décision sous-apprend. Cependant, lorsqu'on laisse l'algorithme définir la profondeur adéquate, on commence par obtenir de meilleurs résultats. Le meilleur modèle obtenu après cette expérience, a une SCR de *346,02* et une exactitude générale de *99,76 %*. Il a été obtenu à partir du paramétrage suivant :

- **criterion** : *"entropy"*
- **max_depth** : *None* (défini par l'algorithme lui-même)
- **min_samples_split** : *20*

Il convient de noter que la performance obtenue pendant la phase de test en terme d'exactitude générale pour ce modèle à 12 caractéristiques, est légèrement au dessus de celle obtenue pour le modèle avec 248 caractéristiques pendant la même phase (99,76% contre 99,74%). De plus, on note d'autres points positifs pour le modèle à 12 caractéristiques avec des performances un peu au dessus pour des classes de trafic telles que : web, mail, TDF et multimédias. Par ailleurs, il est encore tôt pour affirmer que ce nouveau modèle à 12 caractéristiques est plus efficace sans passer par l'évaluation de ses performances sur des échantillons de trafic totalement indépendants de ceux utilisés pour l'apprentissage. Cette phase nous permet d'apprécier l'efficacité du modèle sur de nouvelles données.

Afin de s'assurer qu'on a un modèle bien conçu qui n'a pas sur-appris ou sous-appris et qui peut s'adapter en situation réelle, nous utilisons notre modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.2

TABLEAU 3.2 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 12 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,76%	99,94%	83,28%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,98%	93,93%	N/A	N/A
Exactitude générale : 98,40%			

On note en général de bonnes performances sur chaque classe avec les performances du TDF qui ne sont pas aussi bonnes que les autres. On peut dire que notre modèle s'adapte moins aux classes TDF comme cela avait été le cas dans l'étude avec les 248 caractéristiques même si on note une petite amélioration. Ce modèle s'adapte très bien aussi à de nouveaux flux et nous donne des résultats très proches de ce qu'on a espéré pendant la phase d'entraînement. On peut alors juger ce modèle performant. En plus de cela, il convient de noter que ce nouveau modèle a une performance un peu plus au dessus de celui obtenu avec les 248 caractéristiques pendant la phase d'évaluation en terme d'exactitude générale et d'exactitude spécifique à chaque classe. Il serait donc plus efficace en situation réelle par rapport au modèle à 248 caractéristiques. On peut alors dire que le fait d'enlever les caractéristiques superflues était une bonne idée car elle a permis d'améliorer les performances.

Pour la suite, nous fixons un seuil de 1 % et nous supprimons les caractéristiques dont l'importance est en dessous de 1 %. Après cette opération, on se retrouve avec 6 caractéristiques avec une importance totale de 98 % sur lesquelles nous faisons également une analyse afin de voir comment évolue les performances.

3.1.3 Étude de l'arbre de décision sur notre base de données en considérant 6 caractéristiques

Comme nous l'avons souligné dans les tests précédents, nous partons uniquement sur les 6 caractéristiques qui ont permis d'obtenir une importance de 98 %. Les performances des modèles obtenus sont résumées dans le tableau A.4 en annexe A.3.1.

Le meilleur modèle obtenu possède un SCR de $393,25$ (voir tableau 3.3). Ce modèle a été obtenu à partir des paramètres suivants :

- **criterion** : *"entropy"*
- **max_depth** : *None* (défini par l'algorithme lui même)
- **min_samples_split** : *10*

TABLEAU 3.3 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 6 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,85%	99,94%	99,38%	99,12%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,76%	96,08%	81,02%	95,95%
Exactitude générale : 99,72%			

Évaluons l'efficacité de notre modèle à partir des données d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.4

TABLEAU 3.4 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 6 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,90%	95,83%	83,61%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,98%	52,52%	N/A	N/A
Exactitude générale : 97,53%			

Le modèle obtenu donne des performances satisfaisantes pendant la phase d'évaluation. Mais ses performances sont moins bonnes que celles obtenues par les modèles à 12 et à 248 caractéristiques. Il est donc important de noter que les performances commencent à se détériorer en voulant supprimer le plus de caractéristiques possibles.

3.1.4 Discussion générale sur les résultats obtenus avec l'arbre de décision

De toutes les expériences menées à partir de l'arbre de décision, on peut dire que c'est une technique simple (sans trop de paramétrages) et efficace pour résoudre des problèmes de classification du trafic. Bien que les données manquantes aient été comblées par des moyennes qui ne sont pas vraiment des données fiables, l'arbre de décision a su s'adapter et nous donner des performances satisfaisantes. Grâce au critère d'*entropie* qui s'est révélé très efficace dès le départ pour la construction des arbres, on a pu éviter des expériences supplémentaires. Le premier modèle construit avec l'arbre de décision à partir des 248 caractéristiques a révélé que plus de la moitié (129) des caractéristiques apportaient de l'information nulle. De plus, en considérant les taux d'importance des 248 caractéristiques évalués par le modèle, 236 parmi elles ont une importance totale de *1,38 %* tandis que les 12 autres caractéristiques apportent un total de *98,62 %*. On a alors supposé que ces 236 caractéristiques étaient du superflu et qu'elles pouvaient empêcher l'arbre de bien apprendre sur la base de données. Par la suite, on a considéré uniquement ces 12 caractéristiques. Le tableau 3.5 nous montre un comparatif des performances des deux modèles pendant la phase d'évaluation. On remarque que le modèle à 12 caractéristiques est meilleur par rapport au modèle à 248 caractéristiques en terme d'exactitude générale et aussi spécifiquement sur des classes de trafic telles que : web, mail, TDF, P2P. En plus de cela, ce modèle consomme moins d'espace mémoire au vu de la petite quantité de caractéristiques qu'il exploite et s'avère très rapide dans sa tâche de classification d'un trafic. Néanmoins, en voulant réduire les caractéristiques à 6, on remarque une baisse de performance sur trois trafics dont la

plus importante est celle du P2P qui passe de 93,93 % à 52,52 %. Il est donc important de ne pas se passer de trop de caractéristiques même si elles ont un taux d'importance faible. Dans notre cas, s'arrêter à 12 caractéristiques sur 248 s'est révélé le meilleur choix.

En faisant une analyse sur l'importance des caractéristiques, lorsqu'on considère le modèle à douze caractéristiques, (voir tableau 3.6), on remarque que deux caractéristiques sur les douze, ont des poids d'importances très élevés que sont : le port mis en jeu au niveau du serveur pour le trafic avec une importance de 39,96% (caractéristique No 1) et le nombre total d'octets envoyés dans la fenêtre initiale avant la réception du premier paquet ACK du serveur avec une importance de 52,81% (caractéristique No 95).

L'importance des 10 autres caractéristiques n'est pas non plus négligeable car elles apportent 7,23 % de l'information (se référer à l'annexe A.1.2 pour voir la correspondance des caractéristiques avec leurs numéros).

TABLEAU 3.5 – Comparatif des exactitudes obtenues pendant la phase de test et d'évaluation pour les modèles à 248 caractéristiques, 12 caractéristiques et 6 caractéristiques à partir de l'arbre de décision

	Phase de test			**Phase d'évaluation**		
classe de trafic	248 C	12 C	6 C	248 C	12 C	6 C
Web	99,85%	99,86%	99,85%	98,72%	99,76%	99,90%
Mail	99,93%	99,97%	99,94%	99,77%	99,94%	95,83%
TDF	99,47%	99,83%	99,38%	83,21%	83,28%	83,61%
Services	99,41%	99,17%	99,12%	99,17%	99,17%	99,17%
Base de données	99,88%	99,88%	99,76%	98,98%	98,98%	98,98%
P2P	98,53%	97,39%	96,08%	90,90%	93,93%	52,52%
Attaque	82,92%	82,54%	81,02%	-	-	-
Multimédias	93,64%	94,21%	95,95%	-	-	-
Exactitude générale	**99,74%**	**99,76%**	**99,72%**	**97,52%**	**98,40%**	**97,53%**

TABLEAU 3.6 – Taux d'importances pour les caractéristiques du modèle à 12 caractéristiques de l'arbre de décision

No caractéristique : Taux d'importance				
1 : 39,96%	**2** : 01,32%	**15** : 00,50%	**30** : 00,44%	**59** : 01,51%
83 : 00,20%	**85** : 00,20%	**86** : 00,23%	**95** : 52,81%	**99** : 01,10%
100 : 01,30%	**23** : 00,43%			

3.2 Étude menée avec la forêt aléatoire

Comme nous l'avons dit plus haut, la forêt aléatoire est un algorithme qui va construire plusieurs modèles de l'arbre de décision et ensuite prendre sa décision en fonction de la prédiction faite par le plus grand nombre de modèles d'arbre de décision.

Elle est implémentée dans la bibliothèque *scikit-learn* de la version 3 du langage *Python*. La fonction permettant de mettre en place les modèles basés sur la forêt aléatoire possède plusieurs paramètres qui influencent sur la performance. Cependant, nous nous basons sur trois paramètres que nous avons jugé important que sont :

- **n_estimators** : Ce paramètre spécifie le nombre d'arbres séquentiels que nous voulons réaliser pour la forêt.
- **max_depth** : ou la profondeur maximale de chaque arbre de la forêt. Si elle n'est pas précisée, les nœuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins du minimum d'échantillons requis pour diviser un nœud interne.
- **min_samples_split** : ou nombre minimum d'échantillons requis pour diviser un nœud interne.

Pour la conception de notre modèle, 70 % de la base de données est utilisée pour l'entraînement et 30 % pour le test afin d'évaluer le modèle à chaque étape. Pour toutes les expériences qu'on va réaliser avec la forêt aléatoire, nous partons sur la même méthode de travail qu'avec les alogirthmes précédents. C'est-à-dire on fait plusieurs combinaisons de paramètres afin de dégager le meilleur modèle autour duquel se fait notre analyse.

3.2.1 Étude de la forêt aléatoire sur notre base de données en considérant 248 caractéristiques

Les tests réalisés ont permis de réaliser que la forêt aléatoire éprouve de la difficulté à apprendre efficacement surtout sur la classe de trafic de type *Attaque*. Nous n'arrivons pas à obtenir une exactitude supérieure à 72 %. Les paramètres ayant permis d'obtenir le meilleur modèle d'une exactitude générale de 99,71 % sont les suivants :

- **n_estimators** : *60*
- **max_depth** : *None* (défini par l'algorithme lui-même)
- **min_samples_split** : *2*

Ainsi grâce à ce modèle, on a pu obtenir les résultats récapitulés dans le tableau 3.7 pendant la phase de test du modèle après entraînement.

TABLEAU 3.7 – Récapitulatif des performances sur le meilleur modèle obtenu avec 248 caractéristiques pendant la phase de test

Web	**Mail**	**TDF**	**Services**
99,89%	99,94%	99,66%	99,27%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,16%	95,92%	71,53%	88,44%
Exactitude générale : 99,71%			

Afin de s'assurer qu'on a un modèle bien conçu qui n'a pas sur-appris ou sous-appris et qui peut s'adapter à de nouveaux trafics, nous utilisons notre modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.8

TABLEAU 3.8 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 248 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,93%	90,88%	99,34%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
38,30%	95,62%	N/A	N/A
Exactitude générale : 97,52%			

On note une mauvaise performance sur les classes de type base de données qui se traduit par le fait qu'avec le temps, le modèle perd en efficacité sur les classes de trafic de type base de données au vu du fait que les données d'évaluation ont été recueillies un an plus tard après les données d'entraînement. Néanmoins, le modèle s'adapte très bien sur les autres types de trafics en offrant une exactitude autour de 90 % pour les autres types de classes.

Lorsqu'on analyse l'importance donnée par le modèle à chacune des 248 caractéristiques lors de l'apprentissage, on remarque que *14* caractéristiques ont une importance de 0 %. De plus, beaucoup d'autres caractéristiques ont des importances quasi-nulles(très proches de 0). Toutes ces caractéristiques peuvent être qualifiées de superflues, car en plus de rallonger le temps d'apprentissage, elles compliquent le modèle obtenu qui devient moins interprétable ce qui peut influencer sur la performance du modèle. Pour la suite, nous fixons un seuil de *0.2 %*, afin de supprimer toutes les caractéristiques ayant une importance en dessous de 0.2 %. Après cette opération, nous nous retrouvons à *78* caractéristiques dont la somme des importances est égale à *90,86 %* contre *170* caractéristiques ne réunissant que *9,14 %* d'importance. On voit

ainsi que le modèle a été conçu à partir d'une quantité énorme de caractéristiques n'apportant presque aucune information qui aurait pu influencer sur ses performances. Par la suite, nous nous limitons à 78 caractéristiques pour effectuer le test.

3.2.2 Étude de la forêt aléatoire sur notre base de données en considérant 78 caractéristiques

Comme nous l'avons souligné dans les tests précédents, nous partons uniquement sur les 78 caractéristiques qui ont permis d'obtenir un taux d'importance total de 90,86 %. Pour les tests que nous allons effectuer, nous consignons uniquement le meilleur modèle.

D'après les tests effectués, on constate une amélioration des performances par rapport aux modèles à 248 caractéristiques. Tout d'abord, on a une exactitude qui est passée de 99,71 % à 99,75 % pendant la phase de test. De plus on a une amélioration sur les trafics de type *multimédias* et *attaque*. On peut déjà dire que la suppression du superflue a été efficace. Les paramètres du modèle obtenus sont :

- **n_estimators** : *100*
- **max_depth** : *35*
- **min_samples_split** : *2*

Ce modèle, a permis d'obtenir les résultats récapitulés dans le tableau 3.9 pendant la phase de test du modèle après entraînement.

TABLEAU 3.9 – Récapitulatif des performances sur le meilleur modèle obtenu avec 78 caractéristiques pendant la phase de test

Web	**Mail**	**TDF**	**Services**
99,91%	99,93%	99,83%	99,41%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,64%	96,90%	72,48%	94,21%
Exactitude générale : 99,75%			

Afin de s'assurer qu'on a un modèle bien conçu qui n'a pas sur-appris ou sous-appris et qui peut s'adapter en situation réelle, nous utilisons notre modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.10

TABLEAU 3.10 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 78 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,97%	92,77%	99,67%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
55,93%	98,65%	N/A	N/A
Exactitude générale : 98,61%			

On note en général de bonnes performances sur chaque classe (exactitude supérieure à 90 % pour la plupart des classes). Même si la performance n'est toujours pas satisfaisante sur la classe Base de données, on constate quand même une amélioration par rapport au modèle à 248 caractéristiques. Ce qui nous permet de confirmer une fois de plus que réduire le nombre de caractéristiques en enlevant celles qui apportent très peu ou pas d'information a eu un impact positif sur la performance.

Essayons encore une fois de réduire les caractéristiques en supprimant celles qui ont des importances presque négligeables. Pour cela, nous fixons un seuil de 0.5 % et nous supprimons celles qui ont une importance en dessous de ce seuil. Après suppression, nous nous retrouvons à 41 caractéristiques d'une importance totale de *89,62 %*.

3.2.3 Étude de la forêt aléatoire sur notre base de données en considérant 41 caractéristiques

Pour les nouveaux tests à faire, nous partons de 41 caractéristiques. Nous consignons uniquement le meilleur modèle.

D'après les tests effectués, on constate une petite amélioration des performances par rapport aux deux précédents modèles avec une exactitude de *99,76 %*. On peut dire que la suppression des caractéristiques continue d'être bénéfique. Les paramètres du modèle obtenus sont :

- **n_estimators** : *55*
- **max_depth** : *30*
- **min_samples_split** : *2*

Ce modèle, a permis d'obtenir les résultats récapitulés dans le tableau 3.11 pendant la phase de test du modèle après entraînement.

TABLEAU 3.11 – Récapitulatif des performances sur le meilleur modèle obtenu avec 41 caractéristiques pendant la phase de test

Web	**Mail**	**TDF**	**Services**
99,92%	99,92%	99,66%	99,41%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,52%	98,04%	72,86%	94,80%
Exactitude générale : 99,76%			

Utilisons le modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.12

TABLEAU 3.12 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 41 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,97%	99,72%	99,80%	99,17%
Base de données	**P2P**	**Attaque**	**Multimédias**
64,41%	96,97%	N/A	N/A
Exactitude générale : 99,35%			

On note de bonnes performances sur chaque classe (exactitude supérieure à 90 % pour la plupart des classes) sauf sur la classe Base de données. De plus, l'exactitude pour la phase d'évaluation a connu une croissance et passe à 99,35 %. Cette hausse se remarque plus au niveau de la classe de trafic base de données qui ne cesse d'augmenter en performance au fur et à mesure de la suppression des caractéristiques superflues.

Afin de supprimer le superflu, nous fixons un seuil à 1% et supprimons toutes les caractéristiques avec une importance en dessous. Après cette opération, il nous reste 23 caractéristiques totalisant une importance de 88,57 %. Ces 23 caractéristiques sont exploitées par la suite dans l'étude.

3.2.4 Étude de la forêt aléatoire sur notre base de données en considérant 23 caractéristiques

Pour les nouveaux tests à faire, nous partons de 23 caractéristiques en consignant uniquement le meilleur modèle.

D'après les tests effectués, le meilleur modèle obtenu a permis d'avoir une exactitude de 99,77 %. Une petite augmentation qui permet de dire que notre modèle s'améliore peu à peu au fur et à mesure qu'on se débarrasse de certaines caractéristiques. Les paramètres ayant permis d'obtenir ce modèle sont :

- **n_estimators** : *50*
- **max_depth** : *20*
- **min_samples_split** : *2*

Ce modèle, a permis d'obtenir les résultats récapitulés dans le tableau 3.13 pendant la phase de test du modèle après entraînement.

TABLEAU 3.13 – Récapitulatif des performances sur le meilleur modèle obtenu avec 23 caractéristiques pendant la phase de test

Web	**Mail**	**TDF**	**Services**
99,93%	99,93%	99,72%	99,41%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,76%	98,04%	72,86%	95,37%
Exactitude générale : 99,77%			

Utilisons le modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.14

TABLEAU 3.14 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 23 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,97%	99,94%	99,73%	99,73%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,30%	94,61%	N/A	N/A
Exactitude générale : 99,85%			

On note en général de bonnes performances sur chaque classe (exactitude supérieure à 90 % pour toutes les classes). On remarque que la performance pour les trafics de classe base de données prend enfin sa marque et nous donne enfin une exactitude très satisfaisante de 98,30

%. De plus, l'exactitude générale est passée à 99,85 %. Éliminer des caractéristiques superflues au fur et à mesure s'avère donc très avantageux.

Avec 23 caractéristiques, nous trouvons que c'est moins dense en espace mémoire. Cependant, nous restons dans notre logique de départ afin de voir si nous connaîtrons une amélioration dans les performances. Nous réduisons alors les caractéristiques en supprimant les moins importantes. Pour la suite, nous fixons un seuil de 2 % et supprimons les caractéristiques dont l'importance est en dessous de ce seuil. Après cette opération, nous nous retrouvons à 17 caractéristiques restantes d'une importance totale de 90,68 %.

3.2.5 Étude de la forêt aléatoire sur notre base de données en considérant 17 caractéristiques

D'après les tests effectués sur les 17 caractéristiques, le meilleur modèle obtenu a permis d'avoir une exactitude de 99,75 %. Il faut remarquer la petite baisse survenue par rapport à l'exactitude. On peut dire que les caractéristiques supprimées précédemment n'étaient pas forcément superflues, car leur suppression commence à se faire ressentir sur nos performances. Les paramètres ayant permis d'obtenir ce modèle sont :

- **n_estimators** : *50*
- **max_depth** : *12*
- **min_samples_split** : *2*

Ce modèle, a permis d'obtenir les résultats récapitulés dans le tableau 3.15 pendant la phase de test du modèle après entraînement.

TABLEAU 3.15 – Récapitulatif des performances sur le meilleur modèle obtenu avec 17 caractéristiques pendant la phase de test

Web	**Mail**	**TDF**	**Services**
99,91%	99,94%	99,86%	99,41%
Base de données	**P2P**	**Attaque**	**Multimédias**
99,76%	97,23%	71,92%	94,22%
Exactitude générale : 99,75%			

Utilisons le modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.16

TABLEAU 3.16 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 17 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,97%	99,94%	98,67%	99,73%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,64%	96,30%	N/A	N/A
Exactitude générale : 99,80%			

On note en général de bonnes performances sur chaque classe (exactitude supérieure à 90 % pour toutes les classes). Comme on l'avait anticipé pendant la phase de test, la performance a connu une petite baisse également pendant la phase d'évaluation avec une exactitude générale qui est passée à 99,80 %. À ce stade, on peut conclure qu'en essayant de diminuer encore plus les caractéristiques on pourrait avoir des baisses encore plus importantes dans les performances.

3.2.6 Discussion générale sur les résultats obtenus avec la forêt aléatoire

À partir des expériences menées à base de la forêt aléatoire, on arrive à en conclure que par rapport aux autres algorithmes la forêt aléatoire a du mal à s'adapter lorsqu'on fonctionne sur un grand nombre de caractéristiques. Les performances augmentaient au fur et à mesure qu'on réduisait les caractéristiques. De plus, paramétrer le nombre d'estimateurs a été fastidieux parce qu'une petite variation du nombre d'estimateurs pouvait engendrer une baisse de performance conséquente. En partant d'un modèle de 248 caractéristiques qui a donné une exactitude générale de 99,71 % on a pu obtenir avec un modèle de 23 caractéristiques une exactitude de 99,77 % qui représente la plus haute exactitude obtenue pendant la phase de test. En plus de cela, on a la performance sur la classe attaque qui ne tourne qu'entre 71 % et 72 % d'exactitude à travers les modèles. Il a fallu construire un modèle sur 23 caractéristiques pour avoir une très bonne performance sur la classe de trafic base de données pendant la phase d'évaluation(exactitude de 98,30 %) sachant qu'au départ avec 248 caractéristiques, on avait une exactitude de 38,30 %. En reconsidérant le modèle obtenu à partir de 17 caractéristiques, on remarque 3 caractéristiques qui se détachent avec des taux d'importance élevés. Ces trois caractéristiques sont : le port mis en jeu au niveau du serveur pour le trafic avec une importance de 18,78 % (caractéristique No 1), le nombre d'octets de la fenêtre initiale (du serveur au client) (caractéristiques No 96) et la taille minimale d'un segment (du client au serveur) (caractéristique No 83) avec une importance de 13,61 %. On remarque toujours nos deux premières caractéristiques qui reviennent encore, ce qui ne fait que confirmer leur importance dans la classification du trafic internet. Le tableau 3.18 montre les taux d'importance des différentes caractéristiques exploitées pour ce modèle (se référer à l'annexe A.1.2 pour voir la correspondance des caractéristiques avec leurs numéros).

TABLEAU 3.17 – Comparatif des exactitudes obtenues pendant la phase de test et d'évaluation pour les modèles à 248 caractéristiques, 78 caractéristiques, 41 caractéristiques et 23 caractéristiques à partir de la forêt aléatoire

	Phase de test				**Phase d'évaluation**			
classe de trafic	248 C	78 C	23 C	17 C	248 C	78 C	23 C	17 C
Web	99,89%	99,91%	99,93%	99,91%	99,93%	99,97%	99,97%	99,97%
Mail	99,94%	99,93%	99,93%	99,94%	90.88%	92,77%	99,94%	99,94%
TDF	99,66%	99,83%	99,72%	99,86%	99,34%	99,67%	99,73%	98,67%
Services	99,27%	99,41%	99,41%	99,41%	99,17%	99,17%	99,73%	99,73%
Base de données	99,16%	99,64%	99,76%	99,76%	38,30%	55,93%	98,30%	98,64%
P2P	95,92%	96,90%	98,04%	97,23%	95,62%	98,65%	94,61%	96,30%
Attaque	71,53%	72,48%	72,86%	71,92%	-	-	-	-
Multimédias	88,44%	94,21%	95,37%	94,22%	-	-	-	-
Exactitude générale	**99,71%**	**99,75%**	**99,77%**	**99,75%**	**97,52%**	**98,61%**	**99,85%**	**99,80%**

TABLEAU 3.18 – Taux d'importances pour les caractéristiques du modèle à 17 caractéristiques de la forêt aléatoire

No caractéristique : Taux d'importance				
1 : 18,78%	**44** : 03,47%	**46** : 01,56%	**82** : 02,65%	**83** : 13,61%
86 : 07,16%	**95** : 20,32%	**96** : 05,32%	**100** : 02,13%	**107** : 01,23%
113 : 01,66%	**158** : 02,40%	**165** : 02,13%	**179** : 04,07%	**180** : 05,53%
186 : 03,07%	**187** : 04,90%			

3.3 Étude menée avec XGboost

Dans cette section, nous exploitons une capacité particulière de XGBoost qui est de pouvoir concevoir des modèles de prédiction même lorsqu'il y a des valeurs manquantes pour certaines caractéristiques dans les données d'apprentissage. Cette particularité est également valable lorsqu'une fois le modèle conçu, nous l'utilisons pour prédire de nouvelles observations comportant des manques. Ainsi, tout au long de l'étude, les valeurs manquantes pour les caractéristiques des flux ne sont pas comblées, ni pour les données d'entraînement, ni pour les données de test et ni pour les données d'évaluation. XGBoost est implémentée dans la bibliothèque *XGBoost* de la version 3 du langage *Python*. Pour notre étude, nous considérons principalement les trois paramètres suivants qu'on fait varier afin d'atteindre de bonnes performances :

- **n_estimators** : Ce paramètre spécifie le nombre d'arbres séquentiels que nous voulons réaliser pour tenter de corriger les arbres précédents.
- **max_depth** : Il représente la profondeur de chaque arbre, qui est le nombre maximum de caractéristiques différentes utilisées dans chaque arbre.
- **colsample_bytree** : proportion de caractéristiques à utiliser par chaque arbre. Afin d'éviter que certaines caractéristiques ne s'attribuent trop de crédit pour la prédiction.

Les détails sur chacun de ces paramètres ont été abordés dans l'annexe du document à la section A.2. Pour la conception de notre modèle, 70 % de la base de données d'entraînement est utilisé pour l'entraînement et 30 % pour le test. Les résultats obtenus son disponible en annexe à la section A.3.2.

3.3.1 Étude de XGBoost sur notre base de données en considérant 248 caractéristiques

Commençons tout d'abord par réaliser notre étude en considérant l'ensemble des 248 caractéristiques de départ qui caractérisent chaque flux. Afin de mieux présenter les résultats et de faire une meilleure analyse, nous faisons des études indépendantes suivant le paramètre *n_estimators* qui est soit de 50 (Tableau A.5) ou de 100 (Tableau A.6). Ces résultats sont disponibles en annexe à la section A.3.2.

À partir des résultats obtenus, on remarque que les modèles à 100 estimateurs viennent améliorer les modèles à 50 estimateurs. De plus, les meilleurs modèles ont été obtenus lorsqu'on limite la profondeur des estimateurs à 15. Les paramètres ayant permis d'obtenir le meilleur modèle soit une exactitude générale de 99,86 % et un SCR de 389,02 sont les suivants :

- **n_estimators** : *100*
- **max_depth** : *15*
- **colsample_bytree** : *0,8*

Afin de s'assurer qu'on a un modèle bien conçu qui n'a pas sur-appris ou sous-appris et qui peut s'adapter à de nouvelles données, nous utilisons notre modèle pour prédire les trafics du bloc d'évaluation. Les exactitudes obtenues pour chaque classe de trafic sont consignées dans le tableau 3.19

TABLEAU 3.19 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 248 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,99%	99,94%	99,93%	95,04%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,64%	96,96%	N/A	N/A
Exactitude générale : 99,89%			

On constate de très bonnes performances sur les différentes classes de trafic. Notre modèle s'adapte donc très bien aux nouvelles données.

Cependant, en faisant une analyse sur les caractéristiques, on remarque que pour la construction du modèle, 181 caractéristiques ont des importances presque négligeables c'est-à-dire en dessous de 0,1 %. La somme de leur importance est évaluée à $5,14\%$ contre une importance de $94,86\%$ pour les 67 autres caractéristiques. Nous formons l'hypothèse que cet ensemble de 181 caractéristiques représentent un superflu qui empêche nos modèles de mieux apprendre et qu'en se débarrassant d'elles, nous pouvons obtenir les mêmes résultats ou faire mieux (réduire l'espace mémoire occupé par le modèle et avoir un modèle beaucoup plus simple et facilement interprétable). Par la suite, on considère uniquement les 67 caractéristiques retenues pour construire nos modèles.

3.3.2 Étude de XGBoost sur notre base de données en considérant 67 caractéristiques

Réalisons notre étude en considérant les 67 caractéristiques retenues précédemment. Afin de mieux présenter les résultats, nous faisons des tests suivant le paramètre *n_estimators* qui est soit de 50 (Tableau A.7) ou de 100 (Tableau A.8). Ces résultats sont disponibles en annexe à la section A.3.2.

On remarque à l'analyse des tableaux A.7 et A.8 que les modèles à 100 estimateurs sont des versions améliorées pour la plupart des modèles à 50 estimateurs. Le meilleur modèle obtenu à partir de cette expérience a une SCR estimée à 380,21 et a été obtenu grâce au paramétrage suivant :

- **n_estimators** : *100*
- **max_depth** : *15*
- **colsample_bytree** : *0.8*

Utilisons notre modèle pour prédire les trafics du bloc d'évaluation. Les précisions obtenues pour chaque classe de trafic sont consignées dans le tableau 3.20

TABLEAU 3.20 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 67 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,99%	99,97%	99,87%	97,52%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,64%	97,31%	N/A	N/A
Exactitude générale : 99,90%			

On constate de très bonnes performances sur les différentes classes de trafic. Notre modèle est efficace et s'adapte très bien à de nouvelles données de flux. De plus, nous avons une petite amélioration par rapport au modèle à 248 caractéristiques, car nous sommes passés d'une exactitude générale de 99,89 % à 99,90 % pour la phase d'évaluation. De plus, les performances se sont améliorées pour les classes de trafic Mail, Service et P2P et sont restées constantes sur les autres classes. On peut dire alors que le fait de supprimer le superflu de 181 caractéristiques a permis non seulement de simplifier le modèle, de gagner en espace mémoire occupé par le modèle mais aussi de gagner en performance.

En fixant un seuil de 0,5 % d'importance, on remarque que pour 44 caractéristiques, chacune d'elle a une importance en dessous de 0.5 % avec une importance totale de 9,27 % pour les 44 caractéristiques. Pour les 23 autres caractéristiques qui ont chacune une importance en dessus du seuil, elles totalisent ensemble un taux d'importance de 90,73 %. Nous formons l'hypothèse que l'ensemble des 44 caractéristiques représente un superflu que nous pouvons supprimer afin d'avoir les mêmes performances que précédemment ou de les améliorer.

3.3.3 Étude de XGBoost sur notre base de données en considérant 23 caractéristiques

Réalisons notre étude en considérant les 23 caractéristiques retenues précédemment. Pour cela, nous faisons une étude uniquement sur 100 estimateurs au vu de son efficacité depuis le début de l'expérience (A.9). Ces résultats sont disponibles en annexe à la section A.3.2.

On remarque du tableau A.9 que tous nos modèles ont une exactitude générale de 99,82 % contre 99,86 % pour la plupart des modèles à 67 caractéristiques. Une autre remarque toujours dans le même sens est la chute des performances sur le trafic de type *attaque*. La baisse des performances commence donc à se faire remarquer à cause de certaines caractéristiques qui n'y sont plus. Le meilleur modèle ayant la SCR la plus basse estimée à 721,63 a été obtenu grâce aux paramètres suivants :

- **n_estimators** : *100*
- **max_depth** : *10*
- **colsample_bytree** : *0.8*

Évaluons notre modèle à 23 caractéristiques afin de mesurer l'impact de cette baisse de performance sur les données d'évaluation.

TABLEAU 3.21 – Récapitulatif des performances sur le meilleur modèle obtenu pour la base de données d'évaluation avec 23 caractéristiques

Web	**Mail**	**TDF**	**Services**
99,99%	99,97%	96,16%	96,69%
Base de données	**P2P**	**Attaque**	**Multimédias**
98,64%	96,30%	N/A	N/A
Exactitude générale : 99,60%			

Après la phase d'évaluation, on constate de très bonnes performances sur les différentes classes de trafic. On peut alors dire que notre modèle est efficace et s'adapte très bien à de nouvelles données de flux. Même si notre modèle a connu des baisses de performances par rapport au modèle à 67 caractéristiques sur certaines classes, nous avons quand même des performances satisfaisantes (exactitude sur chaque classe supérieure à 95 %). Il faut aussi noter l'exactitude générale qui est passée de 99,90 % (modèle à 67 caractéristiques) à 99,60 % (modèle à 23 caractéristiques.) On peut donc dire que même si on a gagné en simplicité, le fait d'avoir supprimé certaines caractéristiques a fait baisser nos performances. Si nous décidons de supprimer des caractéristiques par la suite, nous remarquerons sûrement une dégradation de la performance à celle qu'on vient d'obtenir et cela se justifie, car nous avons des données manquantes pour certains flux et le fait de réduire encore les caractéristiques réduit les possibilités de XGBoost lors de la conception du modèle.

3.3.4 Discussion générale sur les résultats obtenus avec XGBoost

Des expériences menées avec XGBoost, on peut retenir que XGBoost est un algorithme très simple à paramétrer, mais en même temps puissant qui d'une part arrive à construire des modèles avec des données incomplètes et d'autre part, il arrive à nous donner des modèles performants malgré les données manquantes. Cependant, c'est un algorithme qui requiert énormément de caractéristiques pour pouvoir être performant lorsqu'on a affaire aux manques de données. Le meilleur modèle avec XGBoost a été obtenu en considérant 67 caractéristiques.

Mais, lorsqu'on a essayé de réduire en passant de 67 caractéristiques à 23 caractéristiques (caractéristiques les plus importantes selon l'algorithme), on a observé une baisse sur l'exactitude générale qui est passée de 99,87 % à 99,82 % pendant la phase de test et de 99,90 % à 99,60 % pour la phase d'évaluation. Une autre baisse de performance qui saute également à l'œil, est celle de la classe *attaque* qui est passée de 80,45 % d'exactitude à 73,43 % pendant la phase de test. Cependant, le modèle à 23 caractéristiques ne nous donne uniquement pas que du négatif puisque grâce à lui quelle que soit la classe considérée pendant la phase de test ou d'évaluation, nous avons des exactitudes supérieures à 96 % sauf sur la classe attaque qui ne dépasse pas 73 % d'exactitude. Il est donc important de notifier que pour une meilleure adaptation de l'algorithme de XGBoost à la classification du trafic avec des manques de données, il faut lui fournir plusieurs caractéristiques du trafic. Le tableau 3.22 fait une synthèse des résultats obtenus pour les modèles à 248, 67 et 23 caractéristiques

Nous pouvons également former l'hypothèse que si nos modèles étaient utilisés pour faire de la classification temps réel, l'efficacité des modèles s'amélioreraient au fur et à mesure que le trafic s'effectue et que de nouvelles caractéristiques se rajoutent. Néanmoins, n'oublions pas de dire que même avec peu de caractéristiques, les modèles conçus avec XGBoost demeurent très efficaces. Des travaux futurs pourraient se pencher sur ces aspects afin de confirmer nos hypothèses.

En considérant le modèle à 23 caractéristiques, on remarque une fois de plus les caractéristiques que sont : le port mis en jeu au niveau du serveur pour le trafic avec une importance de 18,05 % (caractéristique No 1) et le nombre total d'octets envoyés dans la fenêtre initiale avant la réception du premier paquet ACK du serveur avec une importance de 55,81 % (caractéristique No 95). Ces deux caractéristiques se distinguent une fois de plus avec de forts taux d'importance. Le taux d'importance des autres caractéristiques est résumé dans le tableau 3.23 (se référer à l'annexe A.1.2 pour voir la correspondance des caractéristiques avec leurs numéros).

TABLEAU 3.22 – Comparatif des exactitudes obtenues pendant la phase de test et d'évaluation pour les modèles à 248 caractéristiques, 67 caractéristiques et 23 caractéristiques à partir XGBoost

	Phase de test			**Phase d'évaluation**		
Classe de trafic	248 C	67 C	23 C	248 C	67 C	23 C
Web	99,97%	99,96%	99,97%	99,99%	99,99%	99,99%
Mail	99,97%	99,98%	99,98%	99,94%	99,77%	99,97%
TDF	99,97%	99,97%	99,86%	99,93%	99,87%	96,16%
Services	99,56%	99,56%	99,41%	95,04%	97,52%	96,69%
Base de données	99,88%	99,88%	99,88%	98,64%	98,64%	98,64%
P2P	98,86%	98,86%	98,20%	96,96%	97,31%	96,30%
Attaque	80,45%	80,45%	73,43%	-	-	-

Multimédias	97,69%	98,26%	96,53%	-	-	-
Exactitude générale	**99,86%**	**99,87%**	**99,82%**	**99,89%**	**99,90%**	**99,60%**

TABLEAU 3.23 – Taux d'importances pour les caractéristiques du modèle à 23 caractéristiques de XGBoost

No caractéristique : Taux d'importance				
1 : 18,05%	**10** : 01,10%	**15** : 00,47%	**20** : 00,64%	**43** : 02,51%
45 : 01,96%	**68** : 01,32%	**83** : 02,57%	**85** : 02,27%	**95** : 55,81%
96 : 01,31%	**99** : 02,27%	**100** : 01,85%	**102** : 02,09%	**108** : 00,60%
114 : 00,48%	**158** : 00,65%	**163** : 01,19%	**165** : 00,40%	**166** : 00,44%
176 : 00,47%	**179** : 01,13%	**230** : 00,38%		

3.4 Discussion générale

Les résultats des différentes expériences réalisées montrent que chaque algorithme possède ses particularités, ses faiblesses et ses forces. Par exemple, l'arbre de décision s'est révélé l'algorithme le plus efficace sur le trafic de classe attaque. Mais il a donné des résultats peu satisfaisants pour le trafic transfert de fichier, pendant la phase de test et a généralement obtenu de moins bons résultats par rapport aux autres algorithmes. La forêt aléatoire est l'algorithme qui a donné de bons résultats pendant les phases de test et d'évaluation avec le plus petit nombre de caractéristiques considérés même si elle est restée peu efficace sur le trafic d'attaque. Lorsqu'on considère XGBoost, ses performances sont réduites au fur et à mesure que l'on réduit les caractéristiques. Cela peut se refléter par le fait que vu qu'il y a des manques de données, il a besoin d'avoir le plus de caractéristiques possibles pour combler ces manques. Globalement, XGBoost a donné les meilleures performances au cours de notre étude, car grâce au modèle de 67 caractéristiques nous avons obtenu une exactitude globale de 99,87 % pour la phase de test et de 99,90 % pour la phase d'évaluation, ce qui représente les meilleurs résultats obtenus au cours de l'étude. Comme mentionné ci-dessus, chaque algorithme fonctionne sur la base des caractéristiques qu'il considère comme les plus discriminantes. Néanmoins, un certain nombre de caractéristiques se retrouvent très souvent dans les 20 premières caractéristiques les plus discriminantes de deux ou des trois algorithmes considérés. Ces caractéristiques, résumées dans le tableau 3.24, peuvent être donc considérées comme des caractéristiques essentielles pour la classification du trafic Internet. Cela n'est qu'une hypothèse et des études futures, nous permettrons de confirmer ou non cette hypothèse. Cependant, deux caractéristiques se distinguent de ce lot, car tout au long de l'étude, elles ont eu des taux d'importance très élevés quel que soit

l'algorithme. Il s'agit du numéro de port mis en jeu au niveau du serveur et la taille minimale d'un segment observé au cours du trafic (du client au serveur). On peut donc affirmer que ces deux caractéristiques sont indispensables dans la tâche de classification du trafic Internet.

TABLEAU 3.24 – Les 14 caractéristiques les plus importantes communes aux algorithmes considérés

Caractéristiques	**Caractéristiques**
Numéro de port (serveur)	Moyenne du nombre total d'octets dans les paquets IP
Nombre d'octets de données uniques (du client au serveur)	Si le point de terminaision requiert l'option Window Scaling (du serveur au client)
Taille minimale d'un segment (du client au serveur)	La taille moyenne des segments observée pendant la durée de vie de la connexion (du client au serveur)
Nombre d'octets de la fenêtre initiale (du client au serveur)	Nombre d'octets de la fenêtre initiale (du serveur au client)
La taille moyenne des segments observée pendant la durée de vie de la connexion (du serveur au client)	Maximum d'octets de données Ethernet (du client au serveur)
Maximum d'octets dans un paquet IP (du client au serveur)	Moyenne du nombre total d'octets dans un paquet IP (du client au serveur)
La variance du nombre total d'octets dans un paquet IP (du client au serveur)	Maximum d'octets de données Ethernet (du serveur au client)

3.5 Comparaison avec l'état de l'art

Nous comparons nos résultats à ceux de Zhong et al., car le contexte d'étude est le même. De plus, ils ont également obtenu de bonnes performances. Pour les comparaisons, nous utilisons les modèles suivants : le modèle d'arbre de décision à 12 caractéristiques, le modèle XGBoost à 23 caractéristiques et le modèle de la forêt aléatoire à 17 caractéristiques. Zhong et al. utilisent les SVM pour la classification du trafic. Plusieurs noyaux SVM ont été testés et le plus intéressant était le noyau radial. Plusieurs combinaisons de caractéristiques ont été faites à partir de 30 caractéristiques pour créer des modèles. La plus intéressante était une combinaison de 13 caractéristiques qui a donné une exactitude générale de 98 %. Cependant, avec notre modèle à 12 caractéristiques obtenu à partir de l'arbre de décision, nous avons une exactitude générale de 99,76 %, ce qui représente moins de caractéristiques pour plus de performances. Nous avons également obtenu des exactitudes générales plus élevées avec les autres modèles, bien qu'ils nécessitent un peu plus de caractéristiques (99,75 % pour le modèle de la forêt aléatoire

à partir de 17 caractéristiques et 99,82 % pour le modèle de XGBoost à partir de 23 caractéristiques). Les tableaux 3.25, 3.26 et 3.27 résument leurs résultats et les nôtres. En général, nous remarquons que, quel que soit le modèle considéré, nos résultats sont meilleurs. De plus, pour le trafic P2P, nous avons une meilleure performance avec nos modèles. Il est important de mentionner que certaines des caractéristiques qui ont été retenues comme indispensables pour la classification du trafic dans la section *Discussion générale* (section 3.4) se retrouvent parmi les 13 caractéristiques les plus discriminantes considérées dans l'étude de Zhong et al. Ces caractéristiques sont entre autres : le numéro de port (serveur), la taille minimale d'un segment (du client au serveur), le nombre d'octets de la fenêtre initiale (du client au serveur), le nombre d'octets de la fenêtre initiale (du serveur au client), la taille moyenne des segments observée pendant la durée de vie de la connexion (du serveur au client).

TABLEAU 3.25 – Comparaison de la métrique *précision* avec les travaux de Zhong et al. pour la phase d'évaluation

Classe de trafic	Zhong et al.	Arbre de décision	Forêt aléatoire	XGBoost
WWW	96,03%	99,75%	**99,97**%	99,65%
Mail	75,93%	**99,94**%	99,78%	99,83%
TDF	69,58%	**100**%	99,40%	99,52%
Services	93,05%	99,97%	99,97%	**100**%
P2P	52,82%	92,08%	**97,91**%	94,35%
Base de données	83,97%	97,33%	100%	**100**%

TABLEAU 3.26 – Comparaison de la métrique *rappel* avec les travaux de Zhong et al. pour la phase d'évaluation

Classe de trafic	Zhong et al.	Arbre de décision	Forêt aléatoire	XGBoost
WWW	98,72%	99,76%	99,97%	**99,99**%
Mail	97,10%	100%	99,94%	**100**%
TDF	71,91%	82,94%	**99,73**%	96,03%
Services	55,37%	**99,97**%	**99,97**%	96,69%
P2P	54,55%	93,34%	94,61%	**95,62**%
Base de données	51,52%	**98,98**%	98,30%	98,30%

TABLEAU 3.27 – Comparaison de la métrique *f1-score* avec les travaux de Zhong et al. pour la phase d'évaluation

Classe de trafic	Zhong et al.	Arbre de décision	Forêt aléatoire	XGBoost
WWW	97,36%	99,76%	**99,97**%	99,82%
Mail	85,22%	**99,97**%	99,86%	99,92%
TDF	69,53%	90,68%	99,57%	**97,75**%
Services	69,43%	**99,97**%	**99,97**%	98,32%
P2P	53,67%	93,00%	**96,23**%	94,98%
Base de données	63,86%	98,15%	97,97%	**99,14**%

Conclusion

Tout au long de ce chapitre, nous avons pu tester les algorithmes d'*arbre de décision*, de *XGBoost* et de *la forêt aléatoire* sur notre base de données de trafics internet. On a pu constater les forces et les faiblesses de chacun de ces algorithmes. Par exemple, l'arbre de décision a été le plus efficace sur le trafic attaque, XGBoost a pu s'adapter aux manques de données et nous fournir de bonnes performances en général. Enfin la forêt aléatoire n'a nécessité que très peu de caractéristiques pour être performant. L'utilisation de ces algorithmes a également permis de révéler un ensemble de *quatorze caractéristiques* indispensables dans la classification du trafic internet dont deux incontournables à savoir : **le port mis en jeu par le trafic au niveau du serveur** et **la taille minimale de segment observé au cours du trafic**.

Conclusion et perspectives

Dans ce travail, nous avons exploité des algorithme d'apprentissage automatiques tels que l'arbre de décision, la forêt aléatoire et XGBoost, pour proposer des modèles performants pour la classification du trafic. XGBoost, de par sa capacité d'adaptation aux manques de données, a permis de mettre en place des modèles adaptables aux déficits de données. La forêt aléatoire nous a également permis d'avoir des modèles performants sur la classe de trafic attaque. L'arbre de décision quant à lui, nous donne des modèles performants avec très peu de caractéristiques. De façon générale, on est parti de 248 caractéristiques pour essayer d'obtenir des modèles performants avec le moins de caractéristiques possibles. Ainsi, on a obtenu respectivement pour les algorithmes de l'arbre de décision, de la forêt aléatoire et de XGBoost des modèles respectifs de 12 caractéristiques avec une exactitude de 98,40 %, de 17 caractéristiques avec une exactitude de 99,80 % et de 23 caractéristiques avec une exactitude de 99,60 % pendant la phase d'évaluation. Les algorithmes utilisés sont également des algorithmes interprétables dans la mesure où ils permettent d'évaluer l'importance des caractéristiques exploitées. Cette capacité nous a ainsi permis de dégager un ensemble de 14 caractéristiques considérées comme les plus importantes par la plupart de nos algorithmes.

Néanmoins, il faut noter que tout au long de notre étude, nous exploitons les données des trafics TCP qui sont facilement exploitables par rapport à celles des trafics UDP. Des études futures se pencheront alors sur ce dernier type de trafic. De plus, notre étude ne se base que sur 8 classes de trafic. Pour le moment, nous jugeons le nombre de classes de trafics considérés encore faible, mais de futurs travaux sont en cours afin de mettre en place une base de données de plusieurs classes de trafics que nous laisserons en libre accès. Nous exploiterons également cette base de données pour mettre en place des modèles de classification basés sur un plus grand nombre classes de trafic. La capacité d'adaptation de XGBoost sur les manques de données nous permet également d'étudier comment on pourrait l'exploiter pour mettre en place un système de classification en temps réel.

Annexe A

Annexe

A.1 Détails sur l'ensemble de données

A.1.1 Ensemble des dix blocs de données formant les données d'entraînement

Data-set	Start-time	End-time	Duration	Flows (Objects)
entry01	2003-Aug-20 00:34:21	2003-Aug-20 01:04:43	1821.8	24863
entry02	2003-Aug-20 01:37:37	2003-Aug-20 02:05:54	1696.7	23801
entry03	2003-Aug-20 02:45:19	2003-Aug-20 03:14:03	1724.1	22932
entry04	2003-Aug-20 04:03:31	2003-Aug-20 04:33:15	1784.1	22285
entry05	2003-Aug-20 04:39:10	2003-Aug-20 05:09:05	1794.9	21648
entry06	2003-Aug-20 06:07:28	2003-Aug-20 06:35:06	1658.5	19384
entry07	2003-Aug-20 09:42:17	2003-Aug-20 10:11:16	1739.2	55835
entry08	2003-Aug-20 11:52:40	2003-Aug-20 12:20:26	1665.9	55494
entry09	2003-Aug-20 13:45:37	2003-Aug-20 14:13:21	1664.5	66248
entry10	2003-Aug-20 14:55:44	2003-Aug-20 15:22:37	1613.4	65036

FIGURE A.1 – Ensembles des dix blocs de données de flux formés sur 24 heures [55]

A.1.2 Description des 249 caractéristiques de l'ensemble de données

Number	Short	Long
1	Server Port	Port Number at server; we can establish server and client ports as we limit ourselves to flows for which we see the initial connection set-up.
2	Client Port	Port Number at client
3	min_IAT	Minimum packet inter-arrival time for all packets of the flow (considering both directions).
4	q1_IAT	First quartile inter-arrival time
5	med_IAT	Median inter-arrival time
6	mean_IAT	Mean inter-arrival time
7	q3_IAT	Third quartile packet inter-arrival time
8	max_IAT	Maximum packet inter-arrival time
9	var_IAT	Variance in packet inter-arrival time
10	min_data_wire	Minimum of bytes in (Ethernet) packet, using the size of the packet *on the wire.*
11	q1_data_wire	First quartile of bytes in (Ethernet) packet
12	med_data_wire	Median of bytes in (Ethernet) packet
13	mean_data_wire	Mean of bytes in (Ethernet) packet
14	q3_data_wire	Third quartile of bytes in (Ethernet) packet
15	max_data_wire	Maximum of bytes in (Ethernet) packet
16	var_data_wire	Variance of bytes in (Ethernet) packet
17	min_data_ip	Minimum of total bytes in IP packet, using the size of payload declared by the IP packet
18	q1_data_ip	First quartile of total bytes in IP packet
19	med_data_ip	Median of total bytes in IP packet
20	mean_data_ip	Mean of total bytes in IP packet
21	q3_data_ip	Third quartile of total bytes in IP packet
22	max_data_ip	Maximum of total bytes in IP packet
23	var_data_ip	Variance of total bytes in IP packet
24	min_data_control	Minimum of control bytes in packet, size of the (IP/TCP) packet header
25	q1_data_control	First quartile of control bytes in packet
26	med_data_control	Median of control bytes in packet
27	mean_data_control	Mean of control bytes in packet
28	q3_data_control	Third quartile of control bytes in packet
29	max_data_control	Maximum of control bytes in packet
30	var_data_control	Variance of control bytes packet
31	total_packets_*a b*	The total number of packets seen (client→server).
32	total_packets_*b a*	” (server→client)

FIGURE A.2 – Caractéristiques du trafic (1-32) [55]

Number	Short	Long
33	ack_pkts_sent_*a b*	The total number of ack packets seen (TCP segments seen with the ACK bit set) (client→server).
34	ack_pkts_sent_*b a*	" (server→client)
35	pure_acks_sent_*a b*	The total number of ack packets seen that were not piggy-backed with data (just the TCP header and no TCP data payload) and did not have any of the SYN/FIN/RST flags set (client→server)
36	pure_acks_sent_*b a*	" (server→client)
37	sack_pkts_sent_*a b*	The total number of ack packets seen carrying TCP SACK [6] blocks (client→server)
38	sack_pkts_sent_*b a*	" (server→client)
39	dsack_pkts_sent_*a b*	The total number of sack packets seen that carried duplicate SACK (D-SACK) [7] blocks. (client→server)
40	dsack_pkts_sent_*b a*	" (server→client)
41	max_sack_blks/ack_*a b*	The maximum number of sack blocks seen in any sack packet. (client→server)
42	max_sack_blks/ack_*b a*	" (server→client)
43	unique_bytes_sent_*a b*	The number of unique bytes sent, i.e., the total bytes of data sent excluding retransmitted bytes and any bytes sent doing window probing. (client→server)
44	unique_bytes_sent_*b a*	" (server→client)
45	actual_data_pkts_*a b*	The count of all the packets with at least a byte of TCP data payload. (client→server)
46	actual_data_pkts_*b a*	" (server→client)
47	actual_data_bytes_*a b*	The total bytes of data seen. Note that this includes bytes from retransmissions / window probe packets if any. (client→server)
48	actual_data_bytes_*b a*	" (server→client)
49	rexmt_data_pkts_*a b*	The count of all the packets found to be retransmissions. (client→server)
50	rexmt_data_pkts_*b a*	" (server→client)
51	rexmt_data_bytes_*a b*	The total bytes of data found in the retransmitted packets. (client→server)
52	rexmt_data_bytes_*b a*	" (server→client)
53	zwnd_probe_pkts_*a b*	The count of all the window probe packets seen. (Window probe packets are typically sent by a sender when the receiver last advertised a zero receive window, to see if the window has opened up now). (client→server)
54	zwnd_probe_pkts_*b a*	" (server→client)
55	zwnd_probe_bytes_*a b*	The total bytes of data sent in the window probe packets. (client→server)
56	zwnd_probe_bytes_*b a*	" (server→client)
57	outoforder_pkts_*a b*	The count of all the packets that were seen to arrive out of order. (client→server)

FIGURE A.3 – Caractéristiques du trafic (33-57) [55]

Number	Short	Long
58	outoforder_pkts_*b a*	" (server→client)
59	pushed_data_pkts_*a b*	The count of all the packets seen with the PUSH bit set in the TCP header. (client→server)
60	pushed_data_pkts_*b a*	" (server→client)
61	SYN_pkts_sent_*a b*	The count of all the packets seen with the SYN bits set in the TCP header respectively (client→server)
62	FIN_pkts_sent_*a b*	The count of all the packets seen with the FIN bits set in the TCP header respectively (client→server)
63	SYN_pkts_sent_*b a*	The count of all the packets seen with the SYN bits set in the TCP header respectively (server→client)
64	FIN_pkts_sent_*b a*	The count of all the packets seen with the FIN bits set in the TCP header respectively (server→client)
65	req_1323_ws_*a b*	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323[8] a 'Y' is printed on the respective field. If the option was not requested, an 'N' is printed. For example, an "N/Y" in this field means that the window-scaling option was not specified, while the Time-stamp option was specified in the SYN segment. (client→server)
66	req_1323_ts_*a b*	...
67	req_1323_ws_*b a*	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323[8] a 'Y' is printed on the respective field. If the option was not requested, an 'N' is printed. For example, an "N/Y" in this field means that the window-scaling option was not specified, while the Time-stamp option was specified in the SYN segment. (client→server)
68	req_1323_ts_*b a*	...
69	adv_wind_scale_*a b*	The window scaling factor used. Again, this field is valid only if the connection was captured fully to include the SYN packets. Since the connection would use window scaling if and only if both sides requested window scaling [8], this field is reset to 0 (even if a window scale was requested in the SYN packet for this direction), if the SYN packet in the reverse direction did not carry the window scale option. (client→server)
70	adv_wind_scale_*b a*	" (server→client)
71	req sack_*a b*	If the end-point sent a SACK permitted option in the SYN packet opening the connection, a 'Y' is printed; otherwise 'N' is printed. (client→server)
72	req sack_*b a*	" (server→client)
73	sacks_sent_*a b*	The total number of ACK packets seen carrying SACK information. (client→server)
74	sacks_sent_*b a*	" (server→client)

FIGURE A.4 – Caractéristiques du trafic (58-74) [55]

Number	Short	Long
75	urgent_data_pkts_*a b*	The total number of packets with the URG bit turned on in the TCP header. (client→server)
76	urgent_data_pkts_*b a*	" (server→client)
77	urgent_data_bytes_*a b*	The total bytes of urgent data sent. This field is calculated by summing the urgent pointer offset values found in packets having the URG bit set in the TCP header. (client→server)
78	urgent_data_bytes_*b a*	" (server→client)
79	mss_requested_*a b*	The Maximum Segment Size (MSS) requested as a TCP option in the SYN packet opening the connection. (client→server)
80	mss_requested_*b a*	" (server→client)
81	max_segm_size_*a b*	The maximum segment size observed during the lifetime of the connection. (client→server)
82	max_segm_size_*b a*	" (server→client)
83	min_segm_size_*a b*	The minimum segment size observed during the lifetime of the connection. (client→server)
84	min_segm_size_*b a*	" (server→client)
85	avg_segm_size_*a b*	The average segment size observed during the lifetime of the connection calculated as the value reported in the actual data bytes field divided by the actual data pkts reported. (client→server)
86	avg_segm_size_*b a*	" (server→client)
87	max_win_adv_*a b*	The maximum window advertisement seen. If the connection is using window scaling (both sides negotiated window scaling during the opening of the connection), this is the maximum window-scaled advertisement seen in the connection. For a connection using window scaling, both the SYN segments opening the connection have to be captured in the dumpfile for this and the following window statistics to be accurate. (client→server)
88	max_win_adv_*b a*	" (server→client)
89	min_win_adv_*a b*	The minimum window advertisement seen. This is the minimum window-scaled advertisement seen if both sides negotiated window scaling. (client→server)
90	min_win_adv_*b a*	" (server→client)
91	zero_win_adv_*a b*	The number of times a zero receive window was advertised. (client→server)
92	zero_win_adv_*b a*	" (server→client)
93	avg_win_adv_*a b*	The average window advertisement seen, calculated as the sum of all window advertisements divided by the total number of packets seen. If the connection endpoints negotiated window scaling, this average is calculated as the sum of all window-scaled advertisements divided by the number of window-scaled packets seen. Note that in the window-scaled case, the window advertisements in the SYN packets are excluded since the SYN packets themselves cannot have their window advertisements scaled, as per RFC 1323 [8]. (client→server)

FIGURE A.5 – Caractéristiques du trafic (75-93) [55]

Number	Short	Long
94	avg_win_adv_*b a*	” (server→client)
95	initial_window-bytes_*a b*	The total number of bytes sent in the initial window i.e., the number of bytes seen in the initial flight of data before receiving the first ack packet from the other endpoint. Note that the ack packet from the other endpoint is the first ack acknowledging some data (the ACKs part of the 3-way handshake do not count), and any retransmitted packets in this stage are excluded. (client→server)
96	initial_window-bytes_*b a*	” (server→client)
97	initial_window-packets_*a b*	The total number of segments (packets) sent in the initial window as explained above. (client→server)
98	initial_window-packets_*b a*	” (server→client)
99	ttl_stream_length_*a b*	The Theoretical Stream Length. This is calculated as the difference between the sequence numbers of the SYN and FIN packets, giving the length of the data stream seen. Note that this calculation is aware of sequence space wrap-arounds, and is printed only if the connection was complete (both the SYN and FIN packets were seen). (client→server)
100	ttl_stream_length_*b a*	” (server→client)
101	missed_data_*a b*	The missed data, calculated as the difference between the ttl stream length and unique bytes sent. If the connection was not complete, this calculation is invalid and an “NA” (Not Available) is printed. (client→server)
102	missed_data_*b a*	” (server→client)
103	truncated_data_*a b*	The truncated data, calculated as the total bytes of data truncated during packet capture. For example, with tcpdump, the snaplen option can be set to 64 (with -s option) so that just the headers of the packet (assuming there are no options) are captured, truncating most of the packet data. In an Ethernet with maximum segment size of 1500 bytes, this would amount to truncated data of 1500 64 = 1436bytes for a packet. (client→server)
104	truncated_data_*b a*	” (server→client)
105	truncated_packets_*a b*	The total number of packets truncated as explained above. (client→server)
106	truncated_packets_*b a*	” (server→client)
107	data_xmit_time_*a b*	Total data transmit time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload. (client→server)

FIGURE A.6 – Caractéristiques du trafic (94-107) [55]

Number	Short	Long
108	data_xmit_time_*b a*	" (server→client)
109	idletime_max_*a b*	Maximum idle time, calculated as the maximum time between consecutive packets seen in the direction. (client→server)
110	idletime_max_*b a*	" (server→client)
111	throughput_*a b*	The average throughput calculated as the unique bytes sent divided by the elapsed time i.e., the value reported in the unique bytes sent field divided by the elapsed time (the time difference between the capture of the first and last packets in the direction). (client→server)
112	throughput_*b a*	" (server→client)
113	RTT_samples_*a b*	The total number of Round-Trip Time (RTT) samples found. tcptrace is pretty smart about choosing only valid RTT samples. An RTT sample is found only if an ack packet is received from the other endpoint for a previously transmitted packet such that the acknowledgment value is 1 greater than the last sequence number of the packet. Further, it is required that the packet being acknowledged was not retransmitted, and that no packets that came before it in the sequence space were retransmitted after the packet was transmitted. Note : The former condition invalidates RTT samples due to the retransmission ambiguity problem, and the latter condition invalidates RTT samples since it could be the case that the ack packet could be cumulatively acknowledging the retransmitted packet, and not necessarily ack-ing the packet in question. (client→server)
114	RTT_samples_*b a*	" (server→client)
115	RTT_min_*a b*	The minimum RTT sample seen. (client→server)
116	RTT_min_*b a*	" (server→client)
117	RTT_max_*a b*	The maximum RTT sample seen. (client→server)
118	RTT_max_*b a*	" (server→client)
119	RTT_avg_*a b*	The average value of RTT found, calculated straightforward-ly as the sum of all the RTT values found divided by the total number of RTT samples. (client→server)
120	RTT_avg_*b a*	" (server→client)
121	RTT_stdv_*a b*	The standard deviation of the RTT samples. (client→server)
122	RTT_stdv_*b a*	" (server→client)
123	RTT_from_3WHS_*a b*	The RTT value calculated from the TCP 3-Way Hand-Shake (connection opening) [9], assuming that the SYN packets of the connection were captured. (client→server)

FIGURE A.7 – Caractéristiques du trafic (108-123) [55]

Number	Short	Long
124	RTT_from_3WHS_*b a*	” (server→client)
125	RTT_full_sz_smpls_*a b*	The total number of full-size RTT samples, calculated from the RTT samples of full-size segments. Full-size segments are defined to be the segments of the largest size seen in the connection. (client→server)
126	RTT_full_sz_smpls_*b a*	” (server→client)
127	RTT_full_sz_min_*a b*	The minimum full-size RTT sample. (client→server)
128	RTT_full_sz_min_*b a*	” (server→client)
129	RTT_full_sz_max_*a b*	The maximum full-size RTT sample. (client→server)
130	RTT_full_sz_max_*b a*	” (server→client)
131	RTT_full_sz_avg_*a b*	The average full-size RTT sample. (client→server)
132	RTT_full_sz_avg_*b a*	” (server→client)
133	RTT_full_sz_stdev_*a b*	The standard deviation of full-size RTT samples. (client→server)
134	RTT_full_sz_stdev_*b a*	” (server→client)
135	post-loss_acks_*a b*	The total number of ack packets received after losses were detected and a retransmission occurred. More precisely, a post-loss ack is found to occur when an ack packet acknowledges a packet sent (acknowledgment value in the ack pkt is 1 greater than the packet's last sequence number), and at least one packet occurring before the packet acknowledged, was retransmitted later. In other words, the ack packet is received after we observed a (perceived) loss event and are recovering from it. (client→server)
136	post-loss_acks_*b a*	” (server→client)
137	segs_cum_acked_*a b*	The count of the number of segments that were cumulatively acknowledged and not directly acknowledged. (client→server)
138	segs_cum_acked_*b a*	” (server→client)
139	duplicate_acks_*a b*	The total number of duplicate acknowledgments received. (client→server)
140	duplicate_acks_*b a*	” (server→client)
141	triple_dupacks_*a b*	The total number of triple duplicate acknowledgments received (three duplicate acknowledgments acknowledging the same segment), a condition commonly used to trigger the fast-retransmit/fast-recovery phase of TCP. (client→server)
142	triple_dupacks_*b a*	” (server→client)
143	max_#_retrans_*a b*	The maximum number of retransmissions seen for any segment during the lifetime of the connection. (client→server)

FIGURE A.8 – Caractéristiques du trafic (124-143) [55]

Number	Short	Long
144	max_#_retrans_*b a*	” (server→client)
145	min_retr_time_*a b*	The minimum time seen between any two (re)transmissions of a segment amongst all the retransmissions seen. (client→server)
146	min_retr_time_*b a*	” (server→client)
147	max_retr_time_*a b*	The maximum time seen between any two (re)transmissions of a segment. (client→server)
148	max_retr_time_*b a*	” (server→client)
149	avg_retr_time_*a b*	The average time seen between any two (re)transmissions of a segment calculated from all the retransmissions. (client→server)
150	avg_retr_time_*b a*	” (server→client)
151	sdv_retr_time_*a b*	The standard deviation of the retransmission-time samples obtained from all the retransmissions. (client→server)
152	sdv_retr_time_*b a*	” (server→client)
153	min_data_wire_*a b*	Minimum number of bytes in (Ethernet) packet (client→server)
154	q1_data_wire_*a b*	First quartile of bytes in (Ethernet) packet
155	med_data_wire_*a b*	Median of bytes in (Ethernet) packet
156	mean_data_wire_*a b*	Mean of bytes in (Ethernet) packet
157	q3_data_wire_*a b*	Third quartile of bytes in (Ethernet) packet
158	max_data_wire_*a b*	Maximum of bytes in (Ethernet) packet
159	var_data_wire_*a b*	Variance of bytes in (Ethernet) packet
160	min_data_ip_*a b*	Minimum number of total bytes in IP packet
161	q1_data_ip_*a b*	First quartile of total bytes in IP packet
162	med_data_ip_*a b*	Median of total bytes in IP packet
163	mean_data_ip_*a b*	Mean of total bytes in IP packet
164	q3_data_ip_*a b*	Third quartile of total bytes in IP packet
165	max_data_ip_*a b*	Maximum of total bytes in IP packet
166	var_data_ip_*a b*	Variance of total bytes in IP packet
167	min_data_control_*a b*	Minimum of control bytes in packet
168	q1_data_control_*a b*	First quartile of control bytes in packet
169	med_data_control_*a b*	Median of control bytes in packet
170	mean_data_control_*a b*	Mean of control bytes in packet
171	q3_data_control_*a b*	Third quartile of control bytes in packet
172	max_data_control_*a b*	Maximum of control bytes in packet
173	var_data_control_*a b*	Variance of control bytes packet
174	min_data_wire_*b a*	Minimum number of bytes in (Ethernet) packet (server→client)
175	q1_data_wire_*b a*	First quartile of bytes in (Ethernet) packet
176	med_data_wire_*b a*	Median of bytes in (Ethernet) packet
177	mean_data_wire_*b a*	Mean of bytes in (Ethernet) packet

FIGURE A.9 – Caractéristiques du trafic (144-177) [55]

Number	Short	Long
178	q3_data_wire_*b a*	Third quartile of bytes in (Ethernet) packet
179	max_data_wire_*b a*	Maximum of bytes in (Ethernet) packet
180	var_data_wire_*b a*	Variance of bytes in (Ethernet) packet
181	min_data_ip_*b a*	Minimum number of total bytes in IP packet
182	q1_data_ip_*b a*	First quartile of total bytes in IP packet
183	med_data_ip_*b a*	Median of total bytes in IP packet
184	mean_data_ip_*b a*	Mean of total bytes in IP packet
185	q3_data_ip_*b a*	Third quartile of total bytes in IP packet
186	max_data_ip_*b a*	Maximum of total bytes in IP packet
187	var_data_ip_*b a*	Variance of total bytes in IP packet
188	min_data_control_*b a*	Minimum of control bytes in packet
189	q1_data_control_*b a*	First quartile of control bytes in packet
190	med_data_control_*b a*	Median of control bytes in packet
191	mean_data_control_*b a*	Mean of control bytes in packet
192	q3_data_control_*b a*	Third quartile of control bytes in packet
193	max_data_control_*b a*	Maximum of control bytes in packet
194	var_data_control_*b a*	Variance of control bytes packet
195	min_IAT_*a b*	Minimum of packet inter-arrival time (client→server)
196	q1_IAT_*a b*	First quartile of packet inter-arrival time
197	med_IAT_*a b*	Median of packet inter-arrival time
198	mean_IAT_*a b*	Mean of packet inter-arrival time
199	q3_IAT_*a b*	Third quartile of packet inter-arrival time
200	max_IAT_*a b*	Maximum of packet inter-arrival time
201	var_IAT_*a b*	Variance of packet inter-arrival time
202	min_IAT_*b a*	Minimum of packet inter-arrival time (server→client)
203	q1_IAT_*b a*	First quartile of packet inter-arrival time
204	med_IAT_*b a*	Median of packet inter-arrival time
205	mean_IAT_*b a*	Mean of packet inter-arrival time
206	q3_IAT_*b a*	Third quartile of packet inter-arrival time
207	max_IAT_*b a*	Maximum of packet inter-arrival time
208	var_IAT_*b a*	Variance of packet inter-arrival time
209	Time_since_last_connection	Time since the last connection between these hosts
210	No._transitions_bulk/trans	The number of transitions between transaction mode and bulk transfer mode, where bulk transfer mode is defined as the time when there are more than three successive packets in the same direction without any packets carrying data in the other direction
211	Time_spent_in_bulk	Amount of time spent in bulk transfer mode
212	Duration	Connection duration
213	%_bulk	Percent of time spent in bulk transfer
214	Time_spent_idle	The time spent idle (where idle time is the accumulation of all periods of 2 seconds or greater when no packet was seen in either direction)

FIGURE A.10 – Caractéristiques du trafic (178-214) [55]

Number	Short	Long
215	%_idle	Percent of time spent idle
216	Effective_Bandwidth	Effective Bandwidth based upon entropy [10] (both directions)
217	Effective_Bandwidth_*a b*	” (client→server)
218	Effective_Bandwidth_*b a*	” (server→client)
219	FFT_all	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (all traffic) (Frequency #1)
220	FFT_all	” (Frequency #2)
221	FFT_all	” . . .
222	FFT_all	” . . .
223	FFT_all	” . . .
224	FFT_all	” . . .
225	FFT_all	” . . .
226	FFT_all	” . . .
227	FFT_all	” . . .
228	FFT_all	” (Frequency #10)
229	FFT_*a b*	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (client→server) (Frequency #1)
230	FFT_*a b*	” (Frequency #2)
231	FFT_*a b*	” . . .
232	FFT_*a b*	” . . .
233	FFT_*a b*	” . . .
234	FFT_*a b*	” . . .
235	FFT_*a b*	” . . .
236	FFT_*a b*	” . . .
237	FFT_*a b*	” . . .
238	FFT_*b a*	” (Frequency #10)
239	FFT_*b a*	FFT of packet IAT (arctan of the top-ten frequencies ranked by the magnitude of their contribution) (server→client) (Frequency #1)
240	FFT_*b a*	” (Frequency #2)
241	FFT_*b a*	” . . .
242	FFT_*b a*	” . . .
243	FFT_*b a*	” . . .
244	FFT_*b a*	” . . .
245	FFT_*b a*	” . . .
246	FFT_*b a*	” . . .
247	FFT_*b a*	” . . .
248	FFT_*b a*	” (Frequency #10)
249	Classes	Application class, as assigned in [1]

FIGURE A.11 – Caractéristiques du trafic (215-249) [55]

A.2 Détails sur quelques paramètres des algorithmes utilisés

criterion

Fonction permettant de mesurer la qualité d'une répartition. Les critères pris en charge sont "gini" pour l'impureté de Gini et "entropy" pour le gain d'information.

Les nœuds d'un arbre de décision sont divisés en utilisant une impureté. L'impureté est une mesure de l'homogénéité des étiquettes sur un nœud. Il existe de nombreuses façons de mettre en œuvre la mesure de l'impureté, dont deux que *Scikit-learn* a implémenté qui sont le gain d'information et l'impureté de Gini ou l'indice de Gini. Le gain d'information utilise la mesure de l'entropie comme mesure de l'impureté et divise un nœud de telle sorte qu'il donne le plus de gain d'information. Alors que l'impureté de Gini mesure les divergences entre les distributions de probabilité des valeurs de l'attribut cible et divise un nœud de manière à donner le moins d'impuretés possibles. De nombreux chercheurs soulignent que dans la plupart des cas, le choix des critères de répartition ne fera pas une grande différence dans la performance de l'arbre de décision [58]. Chaque critère est efficace dans certains cas et moins performant dans d'autres.

splitter

La stratégie utilisée pour choisir la répartition à chaque nœud. Les stratégies prises en charge sont "best" pour choisir la meilleure répartition et "random" pour choisir la meilleure répartition aléatoire.

Supposons que nous avons des centaines de caractéristiques, alors la répartition "best" serait idéale, car elle calculera les meilleures caractéristiques à diviser en fonction de la mesure d'impureté et les utilisera pour diviser les nœuds, alors que si on choisissait "random", on a une forte chance de se retrouver avec des caractéristiques qui ne nous donnent pas vraiment beaucoup d'informations, ce qui conduirait à un arbre plus profond et moins précis. D'un autre côté, le séparateur "random" présente certains avantages, notamment parce qu'il sélectionne un ensemble de caractéristiques de manière aléatoire et les sépare, ce qui lui évite le temps de calcul énorme nécessaire pour obtenir la répartition optimale. Ensuite, il est également moins sujet au sur-apprentissage, car on ne calcule pas essentiellement la meilleure répartition avant chaque répartition, donc si notre modèle est sujet à un sur-apprentissage, nous pouvons changer le séparateur en "random".

max_depth : Arbre de décision

La profondeur maximale de l'arbre. Si aucune profondeur n'est définie, les nœuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins du minimum d'échantillons requis pour diviser un nœud interne .

La profondeur maximale théorique qu'un arbre de décision peut atteindre est inférieure d'une unité au nombre d'échantillons d'entraînement, mais aucun algorithme ne nous permettra d'atteindre l'arbre jusqu'à ce point pour des raisons évidentes dont l'une des principales étant le sur-apprentissage. Notons ici qu'il s'agit du nombre d'échantillons d'entraînement et non du nombre de caractéristiques, car les données peuvent être réparties plusieurs fois sur la même caractéristique.

En général, plus nous laissons notre arbre pousser en profondeur, plus notre modèle deviendra complexe parce que nous aurons plus de découpage et qu'il capture plus d'informations sur les données. C'est l'une des causes principales du sur-apprentissage des arbres de décision parce que notre modèle s'adaptera parfaitement aux données d'entraînement et ne pourra pas bien généraliser sur l'ensemble des tests. Donc, si notre modèle subi un sur-apprentissage, la réduction de la profondeur de l'arbre est une façon de le combattre.

Il est également mauvais d'avoir une profondeur très faible parce que notre modèle va sous-apprendre pour trouver la meilleure valeur, il faudrait donc expérimenter plusieurs valeurs afin de déterminer la profondeur parfaite. Généralement, on laisse le modèle décider de la profondeur maximale, puis, en comparant les résultats de l'entraînement et des tests, on recherche un sur-apprentissage ou un sous-apprentissage afin de mieux adapter la profondeur maximale.

min_samples_split

Le nombre minimum d'échantillons requis pour diviser un nœud interne

Ce nombre représente le nombre minimum d'échantillons requis pour diviser un nœud interne. Cela peut varier entre le fait de considérer au moins un échantillon à chaque nœud et le fait de considérer tous les échantillons à chaque nœud. Lorsque nous augmentons ce paramètre, l'arbre devient plus contraint, car il doit prendre en compte plus d'échantillons à chaque nœud.

Selon une étude empirique sur le réglage des hyperparamètres des arbres de décision [59], les valeurs idéales de *min_samples_split* ont tendance à être comprises entre 1 et 40 pour l'algorithme *CART* qui est l'algorithme mis en œuvre dans scikit-learn. Ce paramètre est utilisé pour contrôler le sur-apprentissage.

n_estimators

En utilisant l'algorithme de XGBoost, les modèles sont formés de manière séquentielle, où chaque arbre de sous-séquence tente de corriger les erreurs commises par la séquence d'arbres précédente.

Le paramètre "n_estimators" spécifie le nombre d'arbres séquentiels que nous voulons réaliser pour tenter de corriger les arbres précédents. Le nombre d'estimateurs requis dépendra de la taille et de la complexité de notre ensemble de données.

max_depth : XGBoost

Ce paramètre détermine la profondeur à laquelle chaque estimateur est autorisé à construire un arbre. En général, l'augmentation de la profondeur de l'arbre peut entraîner un sur-apprentissage. Pour les ensembles de données à structure complexe, un arbre profond peut être nécessaire.

subsample

Ce paramètre détermine la part de l'ensemble de données d'entraînement sur lequel chaque estimateur (arbre) va se baser pour se construire.

Il s'agit d'un autre paramètre utile pour contrôler le sur-apprentissage. Pourquoi ce contrôle du sur-apprentissage ? Rappelons-nous de la section sur les *n_estimators* que chaque étape du processus d'apprentissage tente de corriger les erreurs de la séquence d'arbres précédente. Si chaque arbre de la séquence s'entraîne sur un ensemble de données légèrement différent de celui des arbres précédents, le processus de correction des erreurs commence à se généraliser sur les "nouveaux" échantillons pendant la phase d'entraînement. Ce qui fait que l'algorithme explore différents cas possibles.

A.3 Tableaux récapitulatifs des résultats obtenus

A.3.1 Quelques résultats obtenus avec l'arbre de décision

Pour mieux suivre les tableaux récapitulatifs des résultats obtenus, les abréviations suivantes ont été faites : *m_d* : "max_depth" ; *m_s_s* : "min_samples_split" ; *ATT* : Attaque ; *BDD* : Base de données ; *TDF* : Transfert de Fichier ; *MUL* : Multimédia ; *SERV* : Services ; *WEB* : Web ; *Acc Gén* : Exactitude générale du modèle ; *SCR* : Somme des carrés des résidus ; *N/A* : Défini par l'algorithme.

TABLEAU A.1 – Résultats des modèles obtenus à partir de l'arbre de décision en considérant les 248 caractéristiques avec le paramètre criterion="gini"

m_d	m_s_s	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
50	2	99,69	ATT : 80,64 BDD : 99,76 TDF : 99,05 MAIL : 99,91 MUL : 94,80 P2P : 96,74 SERV : 99,12 WEB : 99,82	414,25
	10	99,70	ATT : 80,26 BDD : 99,88 TDF : 99,47 MAIL : 99,90 MUL : 91,91 P2P : 96,90 SERV : 99,12 WEB : 99,83	465,83
	20	99,70	ATT : 79,88 BDD : 99,88 TDF : 99,60 MAIL : 99,90 MUL : 90,75 P2P : 96,74 SERV : 98,39 WEB : 99,84	503,81
	40	99,70	ATT : 77,80 BDD : 98,92 TDF : 99,36 MAIL : 99,70 MUL : 89,59 P2P : 96,90 SERV : 98,38 WEB : 99,87	615,12

100	2	99,70	ATT : 81,40 BDD : 99,76 TDF : 99,10 MAIL : 99,94 MUL : 92,48 P2P : 97,23 SERV : 99,27 WEB : 99,83	411,62
	10	99,70	ATT : 80,83 BDD : 99,88 TDF : 99,55 MAIL : 99,87 MUL : 90,75 P2P : 97,39 SERV : 99,12 WEB : 99,82	460,90
	20	99,70	ATT : 79,88 BDD : 99,88 TDF : 99,52 MAIL : 99,85 MUL : 91,32 P2P : 96,74 SERV : 98,38 WEB : 99,83	493,70
	40	99,70	ATT : 79,88 BDD : 99,88 TDF : 99,52 MAIL : 99,85 MUL : 91,33 P2P : 96,74 SERV : 98,38 WEB : 99,83	493,53
N/A	2	99,70	ATT : 81,59 BDD : 99,76 TDF : 99,27 MAIL : 99,85 MUL : 94,22 P2P : 96,25 SERV : 99,27 WEB : 99,83	387,57
	10	99,70	ATT : 80,83 BDD : 99,88 TDF : 99,55 MAIL : 99,90 MUL : 91,33 P2P : 97,23 SERV : 99,12 WEB : 99,82	451,36
	20	99,70	ATT : 79,88 BDD : 99,88 TDF : 99,44 MAIL : 99,88 MUL : 91,91 P2P : 96,90 SERV : 98,39 WEB : 99,84	482,83
	40	99,68	ATT : 77,79 BDD : 98,92 TDF : 99,38 MAIL : 99,67 MUL : 89,59 P2P : 96,90 SERV : 98,39 WEB : 99,86	614,37

TABLEAU A.2 – Résultats des modèles obtenus à partir de l'arbre de décision en considérant les 248 caractéristiques avec le paramètre criterion="entropy"

m_d	m_s_s	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
50	2	99,74	ATT : 82,16 BDD : 99,88 TDF : 99,27 MAIL : 99,95 MUL : 94,22 P2P : 98,20 SERV : 99,56 WEB : 99,85	355,68
	10	99,74	ATT : 82,92 BDD : 99,88 TDF : 99,50 MAIL : 99,93 MUL : 93,64 P2P : 98,36 SERV : 99,41 WEB : 99,84	335,51
	20	99,76	ATT : 81,59 BDD : 99,95 TDF : 99,47 MAIL : 99,93 MUL : 94,80 P2P : 98,36 SERV : 99,41 WEB : 99,87	369,31
	40	99,75	ATT : 82,54 BDD : 99,95 TDF : 99,55 MAIL : 99,93 MUL : 92,48 P2P : 98,69 SERV : 96,63 WEB : 99,87	374,70
100	2	99,76	ATT : 81,59 BDD : 99,95 TDF : 99,35 MAIL : 99,94 MUL : 95,95 P2P : 97,71 SERV : 99,41 WEB : 99,86	361,37
	10	99,75	ATT : 82,35 BDD : 99,88 TDF : 99,52 MAIL : 99,92 MUL : 95,37 P2P : 98,53 SERV : 99,41 WEB : 99,84	335,74
	20	99,76	ATT : 81,40 BDD : 99,95 TDF : 99,44 MAIL : 99,93 MUL : 94,80 P2P : 98,53 SERV : 99,41 WEB : 99,87	375,85
	40	99,74	ATT : 82,73 BDD : 99,95 TDF : 99,55 MAIL : 99,93 MUL : 92,48 P2P : 98,85 SERV : 96,63 WEB : 99,86	367,71

N/A	2	99,75	ATT : 82,35 BDD : 99,88 TDF : 99,41 MAIL : 99,94 MUL : 94,21 P2P : 98,04 SERV : 99,56 WEB : 99,86	349,47
	10	99,74	ATT : 82,92 BDD : 99,88 TDF : 99,47 MAIL : 99,93 MUL : 93,64 P2P : 98,53 SERV : 99,41 WEB : 99,85	335,01
	20	99,75	ATT : 81,59 BDD : 99,95 TDF : 99,52 MAIL : 99,93 MUL : 93,06 P2P : 98,69 SERV : 99,41 WEB : 99,86	389,41
	40	99,75	ATT : 82,54 BDD : 99,95 TDF : 99,55 MAIL : 99,93 MUL : 92,48 P2P : 98,85 SERV : 96,63 WEB : 99,87	374,31

TABLEAU A.3 – Résultats des modèles obtenus à partir de l'arbre de décision en considérant 12 caractéristiques

m_d	m_s_s	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	2	99,51	ATT : 71,16 BDD : 99,88 TDF : 98,41 MAIL : 99,82 MUL : 88,44 P2P : 89,88 SERV : 87,99 WEB : 99,83	1214,64
	10	99,51	ATT : 71,16 BDD : 99,88 TDF : 98,38 MAIL : 99,82 MUL : 88,44 P2P : 89,88 SERV : 87,99 WEB : 99,83	1214,73
	20	99,51	ATT : 71,16 BDD : 99,88 TDF : 98,41 MAIL : 99,82 MUL : 88,44 P2P : 89,88 SERV : 87,99 WEB : 99,83	1214,63
	40	99,51	ATT : 71,16 BDD : 99,88 TDF : 98,38 MAIL : 99,82 MUL : 88,44 P2P : 89,88 SERV : 87,99 WEB : 99,83	1274,73
10	2	99,79	ATT : 77,80 BDD : 99,88 TDF : 99,61 MAIL : 99,96 MUL : 94,80 P2P : 97,55 SERV : 99,12 WEB : 99,93	526,83
	10	99,79	ATT : 77,99 BDD : 99,88 TDF : 99,64 MAIL : 99,91 MUL : 94,80 P2P : 97,55 SERV : 99,27 WEB : 99,93	518,17
	20	99,79	ATT : 77,42 BDD : 99,88 TDF : 99,66 MAIL : 99,98 MUL : 94,22 P2P : 97,39 SERV : 99,12 WEB : 99,92	550,99
	40	99,77	ATT : 77,04 BDD : 99,88 TDF : 99,55 MAIL : 99,93 MUL : 91,91 P2P : 97,88 SERV : 97,36 WEB : 99,93	604,30
N/A	2	99,75	ATT : 81,40 BDD : 99,88 TDF : 99,44 MAIL : 99,97 MUL : 96,53 P2P : 97,23 SERV : 99,12 WEB : 99,86	366,80
	10	99,75	ATT : 82,35 BDD : 99,88 TDF : 99,72 MAIL : 99,89 MUL : 94,80 P2P : 97,22 SERV : 99,12 WEB : 99,86	347,19
	20	99,76	ATT : 82,54 BDD : 99,88 TDF : 99,83 MAIL : 99,97 MUL : 94,21 P2P : 97,39 SERV : 99,12 WEB : 99,86	346,02
	40	99,75	ATT : 81,59 BDD : 99,88 TDF : 99,55 MAIL : 99,93 MUL : 91,90 P2P : 97,87 SERV : 97,36 WEB : 99,89	416,29

TABLEAU A.4 – Résultats des modèles obtenus à partir de l'arbre de décision en considérant 6 caractéristiques

m_d	m_s_s	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	2	99,48	ATT : 71,16 BDD : 99,76 TDF : 98,41 MAIL : 99,74 MUL : 88,44 P2P : 82,22 SERV : 87,99 WEB : 99,85	1428,52
	10	99,48	ATT : 71,16 BDD : 99,76 TDF : 98,41 MAIL : 99,74 MUL : 88,44 P2P : 82,22 SERV : 87,99 WEB : 99,85	1428,52
	20	99,48	ATT : 71,16 BDD : 99,76 TDF : 98,41 MAIL : 99,74 MUL : 88,44 P2P : 82,22 SERV : 87,99 WEB : 99,85	1428,42
	40	99,48	ATT : 71,16 BDD : 99,76 TDF : 98,38 MAIL : 99,74 MUL : 88,44 P2P : 82,22 SERV : 87,99 WEB : 99,85	1428,52
N/A	2	99,72	ATT : 79,88 BDD : 99,88 TDF : 99,33 MAIL : 99,94 MUL : 96,53 P2P : 95,92 SERV : 99,12 WEB : 99,84	434,77
	10	99,72	ATT : 81,02 BDD : 99,76 TDF : 99,38 MAIL : 99,94 MUL : 95,95 P2P : 96,08 SERV : 99,12 WEB : 99,85	393,25
	20	99,73	ATT : 79,69 BDD : 99,76 TDF : 99,50 MAIL : 99,96 MUL : 95,37 P2P : 96,25 SERV : 99,12 WEB : 99,86	449,10
	40	99,72	ATT : 77,80 BDD : 99,76 TDF : 99,72 MAIL : 99,94 MUL : 93,06 P2P : 96,41 SERV : 97,36 WEB : 99,88	561,01

A.3.2 Quelques résultats obtenus avec XGBoost

Pour mieux suivre les tableaux récapitulatifs des résultats obtenus, les abréviations suivantes ont été faites : *m_d* : "max_depth" ; *n_est* : "n_estimators" ; *colsample_bytree* : "c_bt" ; *ATT* : Attaque ; *BDD* : Base de données ; *TDF* : Transfert de Fichier ; *MUL* : Multimédia ; *SERV* : Services ; *WEB* : Web ; *Acc Gén* : Exactitude générale du modèle ; *SCR* : Somme des carrés des résidus ; N/A : Défini par l'algorithme.

TABLEAU A.5 – Résultats des modèles obtenus à partir de XGBoost en considérant 248 caractéristiques avec 50 estimateurs

m_d	c_bt	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	0.5	99,85	ATT : 77,04 BDD : 99,88 TDF : 99,94 MAIL : 99,96 MUL : 98,26 P2P : 99,02 SERV : 99,56 WEB : 99,97	531,36
	0,8	99,86	ATT : 77,80 BDD : 99,88 TDF : 99,97 MAIL : 99,99 MUL : 98,26 P2P : 99,02 SERV : 99,56 WEB : 99,97	497,04
	1	99,85	ATT : 77,23 BDD : 99,88 TDF : 99,97 MAIL : 99,95 MUL : 98,84 P2P : 98,53 SERV : 99,56 WEB : 99,97	522,19

15	0.5	99,86	ATT : 79,88 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 99,42 P2P : 98,53 SERV : 99,41 WEB : 99,97	407,68
	0.8	99,86	ATT : 80,26 BDD : 99,88 TDF : 99,97 MAIL : 99,99 MUL : 95,95 P2P : 98,37 SERV : 99,41 WEB : 99,96	409,09
	1	99,86	ATT : 80,45 BDD : 99,88 TDF : 99,97 MAIL : 99,96 MUL : 97,69 P2P : 98,86 SERV : 99,56 WEB : 99,97	389,05
N/A	0,5	99,85	ATT : 78,56 BDD : 99,88 TDF : 99,97 MAIL : 99,96 MUL : 96,53 P2P : 98,53 SERV : 99,41 WEB : 99,97	474,24
	0,8	99,86	ATT : 78,37 BDD : 99,88 TDF : 99,97 MAIL : 99,96 MUL : 98,26 P2P : 98,86 SERV : 99,56 WEB : 99,97	472,39
	1	99,84	ATT : 77,99 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,97	489,55

TABLEAU A.6 – Résultats des modèles obtenus à partir de XGBoost en considérant 248 caractéristiques avec 100 estimateurs

m_d	c_bt	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	0.5	99,86	ATT : 77,61 BDD : 99,88 TDF : 99,94 MAIL : 99,98 MUL : 98,84 P2P : 98,86 SERV : 99,56 WEB : 99,97	504,17
	0,8	99,86	ATT : 78,94 BDD : 99,88 TDF : 99,97 MAIL : 99,99 MUL : 98,26 P2P : 99,02 SERV : 99,41 WEB : 99,97	447,88
	1	99,85	ATT : 78,56 BDD : 99,88 TDF : 99,97 MAIL : 99,96 MUL : 99,42 P2P : 98,86 SERV : 99,41 WEB : 99,97	461,67
15	0.5	99,86	ATT : 79,88 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 99,42 P2P : 98,53 SERV : 99,41 WEB : 99,97	407,68
	0.8	99,86	ATT : 80,45 BDD : 99,88 TDF : 99,97 MAIL : 99,97 MUL : 97,69 P2P : 98,86 SERV : 99,56 WEB : 99,97	389,02
	1	99,86	ATT : 80,07 BDD : 99,88 TDF : 99,97 MAIL : 99,96 MUL : 98,84 P2P : 98,86 SERV : 99,56 WEB : 99,97	400,06
N/A	0,5	99,86	ATT : 79,51 BDD : 99,88 TDF : 99,94 MAIL : 99,98 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,97	424,95
	0,8	99,86	ATT : 78,37 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 98,84 P2P : 98,69 SERV : 99,56 WEB : 99,97	471,13
	1	99,84	ATT : 78,94 BDD : 99,88 TDF : 99,97 MAIL : 99,99 MUL : 99,42 P2P : 98,69 SERV : 99,56 WEB : 99,97	445,79

TABLEAU A.7 – Résultats des modèles obtenus à partir de XGBoost en considérant 67 caractéristiques avec 50 estimateurs

m_d	c_bt	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	0.5	99,85	ATT : 77,23 BDD : 99,88 TDF : 99,92 MAIL : 99,96 MUL : 97,11 P2P : 99,02 SERV : 99,41 WEB : 99,97	528,16
	0,8	99,85	ATT : 77,99 BDD : 99,88 TDF : 99,92 MAIL : 99,96 MUL : 98,84 P2P : 99,02 SERV : 99,56 WEB : 99,96	486,96
	1	99,86	ATT : 78,56 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 98,84 P2P : 98,86 SERV : 99,41 WEB : 99,97	468,68
15	0.5	99,86	ATT : 79,89 BDD : 99,88 TDF : 99,94 MAIL : 99,95 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,96	409,53
	0.8	99,86	ATT : 80,45 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 97,11 P2P : 98,86 SERV : 99,41 WEB : 99,96	392,21
	1	99,86	ATT : 80,64 BDD : 99,88 TDF : 99,94 MAIL : 99,98 MUL : 98,26 P2P : 98,53 SERV : 99,56 WEB : 99,97	386,74
N/A	0,5	99,86	ATT : 78,94 BDD : 99,88 TDF : 99,92 MAIL : 99,95 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,97	448,64
	0,8	99,86	ATT : 79,13 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 97,11 P2P : 98,69 SERV : 99,41 WEB : 99,97	445,99
	1	99,85	ATT : 77,80 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,97	497,95

TABLEAU A.8 – Résultats des modèles obtenus à partir de XGBoost en considérant 67 caractéristiques avec 100 estimateurs

m_d	c_bt	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	0.5	99,85	ATT : 77,99 BDD : 99,88 TDF : 99,89 MAIL : 99,98 MUL : 97,11 P2P : 98,69 SERV : 99,41 WEB : 99,97	494,88
	0,8	99,85	ATT : 79,70 BDD : 99,88 TDF : 99,94 MAIL : 99,99 MUL : 98,84 P2P : 99,02 SERV : 99,56 WEB : 99,97	414,61
	1	99,86	ATT : 79,32 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 99,42 P2P : 98,86 SERV : 99,41 WEB : 99,97	429,66
15	0.5	99,86	ATT : 79,89 BDD : 99,88 TDF : 99,94 MAIL : 99,95 MUL : 98,26 P2P : 98,69 SERV : 99,41 WEB : 99,96	409,53
	0.8	99,86	ATT : 80,45 BDD : 99,88 TDF : 99,97 MAIL : 99,98 MUL : 98,26 P2P : 98,86 SERV : 99,56 WEB : 99,96	380,21

	1	99,86	ATT : 80,26 BDD : 99,88 TDF : 99,94 MAIL : 99,98 MUL : 99,42 P2P : 98,69 SERV : 99,56 WEB : 99,96	391,93
N/A	0,5	99,86	ATT : 79,32 BDD : 99,88 TDF : 99,89 MAIL : 99,95 MUL : 97,69 P2P : 98,86 SERV : 99,56 WEB : 99,97	434,52
	0,8	99,86	ATT : 79,51 BDD : 99,88 TDF : 99,97 MAIL : 99,99 MUL : 97,69 P2P : 98,69 SERV : 99,41 WEB : 99,97	427,26
	1	99,85	ATT : 80,07 BDD : 99,88 TDF : 99,94 MAIL : 99,98 MUL : 98,26 P2P : 98,86 SERV : 99,41 WEB : 99,96	401,90

TABLEAU A.9 – Résultats des modèles obtenus à partir de XGBoost en considérant 23 caractéristiques avec 100 estimateurs

m_d	c_bt	Acc Gén en %	Exactitude en % du test effectué sur chaque classe	SCR
5	0.5	99,82	ATT : 73,43 BDD : 99,88 TDF : 99,86 MAIL : 99,96 MUL : 95,95 P2P : 97,06 SERV : 99,56 WEB : 99,97	73,24
	0,8	99,82	ATT : 72,86 BDD : 99,88 TDF : 99,86 MAIL : 99,96 MUL : 97,11 P2P : 97,88 SERV : 99,41 WEB : 99,97	749,81
	1	99,82	ATT : 73,05 BDD : 99,88 TDF : 99,83 MAIL : 99,98 MUL : 95,95 P2P : 98,53 SERV : 99,41 WEB : 99,97	745,26
10	0.5	99,82	ATT : 73,43 BDD : 99,88 TDF : 99,86 MAIL : 99,96 MUL : 95,95 P2P : 97,39 SERV : 99,41 WEB : 99,97	729,56
	0.8	99,82	ATT : 73,43 BDD : 99,88 TDF : 99,86 MAIL : 99,98 MUL : 96,53 P2P : 98,20 SERV : 99,41 WEB : 99,97	721,63
	1	99,82	ATT : 73,43 BDD : 99,88 TDF : 99,86 MAIL : 99,98 MUL : 96,53 P2P : 97,06 SERV : 99,41 WEB : 99,97	727,03
N/A	0,5	99,82	ATT : 73,05 BDD : 99,88 TDF : 99,86 MAIL : 99,94 MUL : 96,53 P2P : 98,37 SERV : 99,41 WEB : 99,97	741,39
	0,8	99,82	ATT : 73,05 BDD : 99,88 TDF : 99,93 MAIL : 99,96 MUL : 97,11 P2P : 98,04 SERV : 99,41 WEB : 99,97	738,87
	1	99,82	ATT : 73,43 BDD : 99,88 TDF : 99,83 MAIL : 99,95 MUL : 95,95 P2P : 97,06 SERV : 99,41 WEB : 99,97	731,40

Bibliographie

[1] Janine Morley, Kelly Widdicks and Mike Hazas, *Digitalisation, energy and data demand : The impact of Internet traffic on overall and peak electricity consumption*. Energy Research & Social Science, Volume 38, Pages 128-137, 2018.

[2] Thomas Karagiannis, Konstantina Papagiannaki and Michalis Faloutsos, *BLINC : Multilevel Traffic Classification in the Dark*. In Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'05), pages 229–240. ACM, 2005.

[3] Andrew W. Moore and Denis Zuev, *Internet Traffic Classification Using Bayesian Analysis Techniques*. In Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pages 50–60. ACM, 2005.

[4] Andrew W. Moore and Konstantina Papagiannaki, *Towards the Accurate Identification of Network Applications*. Proceedings of the Passive and Active Measurement Workshop, pages 41–54, 2005.

[5] Jeffrey Erman, Martin Arlitt and Anirban Mahanti, *Traffic Classification Using Clustering Algorithms*. In Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data (MineNet'06), pages 281–286. ACM, 2006.

[6] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian, *Traffic Classification on the Fly*. ACM SIGCOMM Computer Communication Review, 36(2) :23–26, 2006.

[7] Fulvio Risso, Mario Baldi, Olivier Morandi, Andrea Baldini and Pere Monclus, *Lightweight, Payload-Based Traffic Classification : An Experimental Evaluation*. In IEEE International Conference on Communications, ICC'08, pages 5869 – 5875, May 2008.

[8] Jingjing Zhao, Xuyang Jing, Zheng Yan and Witold Pedrycz, *Network traffic classification for data fusion : A survey*. Information Fusion, Volume 72, Pages 22-47, 2021.

[9] Maciej Korczynski, *Classification de flux applicatifs et détection d'intrusion dans le trafic Internet*. Autre [cs.OH]. Université de Grenoble. Français. NNT : 2012GRENM087. tel-00858571, 2012

[10] Min Zhang, Wolfgang John, KC Claffy and Nevil Brownlee, *State of the Art in Traffic Classification : A Research Review*. 10th International Conference on Passive and Active Network Measurement (PAM'09) Student Workshop, pages 1–2, April 2009.

[11] Anthony Mcgregor, Mark Hall, Perry Lorier and James Brunskill, *Flow Clustering Using Machine Learning Techniques*. In Proceedings of the 5th International Conference on Passive and Active Network Measurement (PAM'04), pages 205–214, 2004.

[12] Subhabrata Sen, Oliver Spatscheck and Dongmei Wang, *Accurate, Scalable in-Network Identification of P2P Traffic Using Application Signatures*. In Proceedings of the 13th International Conference on World Wide Web (WWW'04), pages 512 – 521. ACM, 2004.

[13] B. Park, Young J. Won, Myung-Sup Kim and James W. Hong, *Towards automated application signature generation for traffic identification*. NOMS 2008 - 2008 IEEE Network Operations and Management Symposium, pp. 160-167 Salvador, Brazil, 2008

[14] A. Callado, C. Kamienski, G Szabo, Balazs Peter Gero, J. Kelner, Stenio Fernandes and Djamel Sadok, *A Survey on Internet Traffic Identification*. In IEEE Communications Surveys & Tutorials, vol. 11, no. 3, pp. 37-52, 2009.

[15] N. B. Azzouna and F. Guillemin, *Analysis of ADSL Traffic on an IP Backbone Link*. IEEE Global Telecommunications Conference 2003, 2003-December.

[16] A. Madhukar and C. Williamson, *A Longitudinal Study of P2P Traffic Classification*. 14th IEEE International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems, 2006-September.

[17] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker, *Unexpected Means of Protocol Inference*. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06), pages 313–326. ACM, 2006.

[18] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck and Dongmei Wang, *ACAS : Automated Construction of Application Signatures*. In Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data (MineNet'05), pages 197–202. ACM, 2005.

[19] Christopher Kohnen, Christian Uberall, Florian Adamsky, Veselin Rakocevic, Muttukrishnan Rajarajan and Rudolf Jager, *Enhancements to Statistical Protocol IDentification (SPID) for Self-Organised QoS in LANs*. In Proceedings of the 19th IEEE International Conference on Computer Communications and Networks (ICCCN'10), pages 1–6. IEEE, 2010.

[20] S. K. Patel and A. Sonker, *Internet Protocol Identification Number Based Ideal Stealth Port Scan Detection Using Snort*. 8th International Conference on Computational Intelligence and Communication Networks (CICN), Tehri, India, 2016.

[21] Khandait Pratibha, Neminath Hubballi and Bodhisatwa Mazumdar, *Efficient keyword matching for deep packet inspection based network traffic classification*. International Conference on Communication Systems & NETworkS, IEEE, 2020.

[22] Mahdavi E., A. Fanian and H. Hassannejad, *Encrypted Traffic Classification Using Statistical Features*. ISeCure 10.1, 2018.

[23] Vladimir A. Muliukha, Leonid U. Laboshin, Alexey A. Lukashin and Nikolay V. Nashivochnikov, *Analysis and Classification of Encrypted Network Traffic Using Machine Learning*. International Conference on Soft Computing and Measurements, IEEE, 2020.

[24] Internet Assigned Numbers Authority, `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`, consulté le 25 juin 2020.

[25] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson and Robin Sommer, *Dynamic Application-layer Protocol Analysis for Network Intrusion Detection*. In Proceedings of the 15th Conference on USENIX Security Symposium. USENIX Association, 2006.

[26] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos and KiYoung Lee, *Internet Traffic Classification Demystified : Myths, Caveats, and the Best Practices*. In Proceedings of the 2008 ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'08), pages 1–12. ACM, 2008.

[27] Gregor Maier, Anja Feldmann, Vern Paxson and Mark Allman, *On Dominant Characteristics of Residential Broadband Internet Traffic*. In Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC'09), pages 90–102. ACM, 2009.

[28] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi and Paolo Tofanelli, *Revealing Skype Traffic : When Randomness Plays with You*. In Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'07), pages 37–48. ACM, 2007.

[29] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh and George Varghese, *Network Monitoring Using Traffic Dispersion Graphs (TDGS)*. In Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07), pages 315 – 320. ACM, 2007.

[30] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh and George Varghese, *Graption : A Graph-based P2P Traffic Classification Framework*

for the Internet Backbone. Computer Networks : The International Journal of Computer and Telecommunications Networking, 55(8) :1909–1920, June 2011.

[31] Tom Auld, Andrew W. Moore. and Stephen F. Gull, *Bayesian Neural Networks for Internet Traffic Classification*. IEEE Transactions On Neural Networks, vol. 18, No. 1, January 2007.

[32] Zhong Fan and Ran Liu, *Investigation of Machine Learning Based Network Traffic Classification*. International Symposium on Wireless Communication Systems (ISWCS) 2017.

[33] J. Erman, A. Mahanti, M. Arlitt, I. Cohen and C. Williamson, *Semi-Supervised Network Traffic Classification*. In Proceedings of ACM SIGMETRICS, June 2007.

[34] Z. Li, R. Yuan and X. Guan, *Accurate Classification of the Internet Traffic Based on the SVM Method*. In Proceedings of ICC, June 2007.

[35] A. Este, F. Gringoli and L. Salgarelli, *Support Vector Machines for TCP Traffic Classification*. In Elsevier Computer Networks (COMNET), September 2009.

[36] KIRK M., *Thoughtful Machine Learning with Python*. O'REILLY, First Ed., 2017.

[37] La gestion des services informatiques (ITSM) encore plus efficace grâce à l'apprentissage automatique, `https://weekly-geekly-es.github.io/articles/fr445504/index.html`, consulté le 29 juin 2020.

[38] MathWorks, *Introducing Machine Learning*, 2016

[39] Machine Learning : classification à l'aide des arbres de décisions : fonctionnement et application en NodeJS, `https://maximilienandile.github.io/2016/10/17/Machine-Learning-classification-a-l-aide-des-arbres-de-decisions-fonctionnement-et-application-en-NodeJS/`, consulté le 29 juin 2020.

[40] Réduisez la corrélation entre les apprenants faibles à l'aide des forêts aléatoires, `https://openclassrooms.com/fr/courses/4470521-modelisez-vos-donnees-avec-les-methodes-ensemblistes/4664688-reduisez-la-correlation-entre-les-apprenants-faibles-a-l-aide-des-forets-aleatoires`, consulté le 29 juin 2020.

[41] Gradient Boosting, comment ça marche ?, `https://www.lovelyanalytics.com/2016/09/12/gradient-boosting-comment-ca-marche/`, consulté le 12 novembre 2020.

[42] Jerome H. Friedman, *Greedy Function Approximation : A Gradient Boosting Machine, The Annals of Statistics*. Vol. 29, No. 5 , pp. 1189-1232, October 2001.

[43] Difference between GBM (Gradient Boosting Machine) and XGBoost (Extreme Gradient Boosting), http://theprofessionalspoint.blogspot.com/2019/02/difference-between-gbm-gradient.html, consulté le 12 novembre 2020.

[44] Eric Biernat and Michel Lutz *Data science : fondamentaux et études de cas. Machine learning avec Python et R*, Eyrolles, 2015.

[45] What is the F-score ?, https://deepai.org/machine-learning-glossary-and-terms/f-score, consulté le 12 novembre 2020.

[46] Java vs MATLAB vs Python | What are the differences, https://stackshare.io/stackups/java-vs-matlab-vs-python, consulté le 17 Juillet 2020.

[47] Which programming language is best for economic research : Julia, Matlab, Python or R ?, https://voxeu.org/article/which-programming-language-best-economic-research, consulté le 17 Juillet 2020.

[48] Anaconda Documentation, https://docs.anaconda.com/, consulté le 17 Juillet 2020.

[49] FORMATION PYTHON MACHINE LEARNING AVEC SCIKIT.LEARN, https://www.ambient-it.net/formation/formation-python-machine-learning/, consulté le 17 juillet 2020.

[50] XGBoost Documentation, https://xgboost.readthedocs.io/en/latest/, consulté le 17 juillet 2020.

[51] Numpy, https://numpy.org/, consulté le 17 juillet 2020.

[52] Matplotlib : Visualization with Python, https://matplotlib.org/, consulté le 17 juillet 2020.

[53] Spyder IDE, https://www.spyder-ide.org/, consulté le 28 janvier 2022.

[54] A. W. Moore, J. Hall, C. Kreibich, E. Harris and I. Pratt, *Architecture of a Network Monitor*. In Passive & Active Measurement Workshop 2003 (PAM2003), La Jolla, CA, April 2003.

[55] A. W. Moore and D. Zuev, *Discriminators for use in flow-based classification*. Technical report, Intel Research, Cambridge, 2005.

[56] Computer Laboratory Internet Traffic Classification Using Bayesian Analysis Techniques, https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/nprobe/data/papers/sigmetrics/, consulté le 17 juillet 2020.

[57] N. G. Duffield, J. T. Lewis, N. O'Connell, R. Russell and F. Toomey, *Entropy of ATM traffic streams*. IEEE Journal on Selected Areas in Communications, 13(6) :981–990, August 1995.

[58] Laura Elena Raileanu and Kilian Stoffel, *Theoretical comparison between the Gini Index and Information Gain criteria*. Annals of Mathematics and Artificial Intelligence 41 : 77–93, 2004.

[59] Rafael Gomes Mantovani, Tomáš Horváth, Ricardo Cerri, Sylvio Barbon Junior, Joaquin Vanschoren and André Carlos Ponce de Leon Ferreira de Carvalho, *An empirical study on hyperparameter tuning of decision trees*. arXiv :1812.02207, décembre 2018.